Before starting, be sure that you understand the course policy on Academic Integrity. Download code05.tar from the course website, which contains Standard ML files for a lambda-calculus interpreter.

The two parts of this assignment investigate adding a mutable heap to the simply-typed lambda calculus. Our syntax for creating, retrieving, and mutating the contents of a reference are like in SML. The first part examines a left-to-right small-step operational semantics, type system, and type soundness proof. The second part examines a left-to-right large-step environment-based operational semantics and implementation in SML.

**Part I**

Here is the syntax, small-step operational semantics, and type system for a simply-typed lambda calculus with a mutable heap. Important comments and definitions follow.

$$
\begin{align*}
  e & ::= c \mid x \mid \lambda x. e \mid e \ e \mid \text{ref } e \mid ! e \mid e := e \mid a \\
  \tau & ::= \text{int} \mid \tau \rightarrow \tau \mid \text{ref } \tau \\
  \Gamma & ::= \cdot \mid \Gamma, x : \tau \mid \Gamma, a : \tau
\end{align*}
$$

$$
\begin{align*}
  H & ::= c \mid \lambda x. a \mid H, a \mapsto v \\
  \vdash & e \rightarrow_{cbv} H': e'
\end{align*}
$$

**E-Alloc**

$$
\begin{align*}
  a \notin \text{Dom}(H) \\
  \vdash H; \text{ref } v \rightarrow_{cbv} H, a \mapsto v; a
\end{align*}
$$

**E-Get**

$$
\begin{align*}
  H(a) = v \\
  H; ! a \rightarrow_{cbv} H; v
\end{align*}
$$

**E-Set**

$$
\begin{align*}
  H[a] \leftarrow v = H' \\
  H; a := v \rightarrow_{cbv} H'; v
\end{align*}
$$

**E-App1**

$$
\begin{align*}
  H; e_f \rightarrow_{cbv} H'; e'_f \\
  H; e_f \ e_a \rightarrow_{cbv} H'; e'_f \ e_a
\end{align*}
$$

**E-App2**

$$
\begin{align*}
  H; e_a \rightarrow_{cbv} H'; e'_a \\
  H; v_f \ e_a \rightarrow_{cbv} H'; v_f \ e'_a
\end{align*}
$$

**E-Alloc1**

$$
\begin{align*}
  H; e_a \rightarrow_{cbv} H'; e'_a \\
  H; \text{ref } e_a \rightarrow_{cbv} H'; \text{ref } e'_a
\end{align*}
$$

**E-Alloc2**

$$
\begin{align*}
  H; e_r \rightarrow_{cbv} H'; e'_r \\
  H; ! e_r \rightarrow_{cbv} H'; ! e'_r
\end{align*}
$$

**E-Set1**

$$
\begin{align*}
  H; e_r := e_a \rightarrow_{cbv} H'; e'_r := e_a
\end{align*}
$$

**E-Set2**

$$
\begin{align*}
  H; v_r := e_a \rightarrow_{cbv} H'; v_r := e'_a
\end{align*}
$$

**T-Const**

$$
\begin{align*}
  \Gamma \vdash c : \text{int}
\end{align*}
$$

**T-Var**

$$
\begin{align*}
  \Gamma \vdash x : \tau
\end{align*}
$$

**T-Lam**

$$
\begin{align*}
  \Gamma, x : \tau_a \vdash e_b : \tau_r \\
  \Gamma \vdash \lambda x. e_b : \tau_a \rightarrow \tau_r
\end{align*}
$$

**T-App1**

$$
\begin{align*}
  \Gamma \vdash e_f : \tau_a \rightarrow \tau_r \\
  \Gamma \vdash e_a : \tau_a \\
  \Gamma \vdash e_f \ e_a : \tau_r
\end{align*}
$$

**T-App2**

$$
\begin{align*}
  \Gamma \vdash e_a : \tau_a \\
  \Gamma \vdash e_a : \tau_a \\
  \Gamma \vdash e_a : \tau_a
\end{align*}
$$

**T-Alloc**

$$
\begin{align*}
  \Gamma \vdash e_a : \tau_a \\
  \Gamma \vdash \text{ref } e_a : \text{ref } \tau_a
\end{align*}
$$

**T-Get**

$$
\begin{align*}
  \Gamma \vdash e_r : \tau_a \\
  \Gamma \vdash ! e_r : \tau_a
\end{align*}
$$

**T-Set**

$$
\begin{align*}
  \Gamma \vdash e_r : \text{ref } \tau_a \\
  \Gamma \vdash e_r : \tau_a \\
  \Gamma \vdash e_r : \tau_a
\end{align*}
$$

**T-Addr**

$$
\begin{align*}
  \Gamma(a) = \tau \\
  \Gamma \vdash a : \text{ref } \tau
\end{align*}
$$
Notes:

- The formal definition of substitution has been omitted. (It is the “obvious” extension of the definition presented in lecture.)
- The heap $H$ is “threaded through” the evaluation of the program, just like in IMP. The heap is a mapping of addresses $a$ to values $v$. The syntax $H(a) = v$ means lookup the $a$ in the heap $H$ yielding a value $v$. (Unlike IMP, there is no “default value”; an address that does not appear in a heap cannot be lookup-ed.) The syntax $H[a] ← v = H'$ means that $H'$ is the update of the heap $H$ so that $a$ maps to $v$. (Again, an address that does not appear in a heap cannot be updated.) Note, we could define $H(a) = v$ and $H[a] ← v = H'$ with inference rules, but this informal description will suffice.
- References can hold values of any type, but the type of a value held in a specific reference never changes.
- An address $a$ represents an allocated reference cell in a running program. Addresses are distinct from variables. Addresses would never exist in a source program, but the Preservation Lemma will require type-checking of addresses. Therefore, the definition of $\Gamma$ includes types for addresses. Note how the typing rule for addresses (T-Addr) is different from the typing rule for variables (T-VAR).
- Note that an allocation chooses a “fresh” address, one that does not appear in the current heap.
- Note that an assignment chooses a “fresh” address, one that does not appear in the current heap.
- The Progress and Preservation Lemmas will also require “type-checking a heap”. The judgment $\Gamma \vdash H : \Gamma$ holds if $H$ is $\cdot, a_1 \mapsto v_1, \ldots, a_n \mapsto v_n$ and $\Gamma$ is $\cdot, a_1 : \tau_1, \ldots, a_n : \tau_n$ (i.e., $\Gamma$ has no variables and exactly the same addresses as $H$) and for all $1 \leq i \leq n$, $\Gamma \vdash v_i : \tau_i$ (i.e., every value in the heap has the type $\Gamma$ says is contained at that address). Note, we could define $\Gamma \vdash H : \Gamma$ with inference rules, but this informal description will suffice.
- We say “$\Gamma_2$ extends $\Gamma_1$” if for all $a$ and $x$, (($\Gamma_1(a) = \tau$ implies $\Gamma_2(a) = \tau$) and ($\Gamma_1(x) = \tau$ implies $\Gamma_2(x) = \tau$)). That is, $\Gamma_2$ has all the variables and addresses that $\Gamma_1$ has and with the same types, but $\Gamma_2$ can have more addresses and variables. Note that every $\Gamma$ extends itself.

In the following, we will assume (i.e., use without proof) the following lemmas:

- **Lemma (Canonical Forms):**
  If $\Gamma \vdash H : \Gamma$ and $\Gamma \vdash v : \tau$, then
  - if $\tau = \text{int}$, then $v = c$ (for some $c$)
  - if $\tau = \tau_a \rightarrow \tau_r$, then $v = \lambda x. e_b$ (for some $\lambda x. e_b$)
  - if $\tau = \text{ref } \tau'$, then $v = a$ and $H(a) = v'$ (for some $a$ and $v'$).

- **Lemma (Substitution):**
  If $\Gamma, z : \tau_z \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau_z$, then $\Gamma \vdash e_1[e_2/z] : \tau$.

- **Lemma (Weakening):**
  If $\Gamma \vdash e : \tau$ and $\Gamma'$ extends $\Gamma$, then $\Gamma' \vdash e : \tau$.

- **Lemma (Heap Extension):**
  If $\Gamma \vdash H : \Gamma$ and $a \notin \text{Dom}(H)$ and $\Gamma \vdash v : \tau$, then $\Gamma \vdash H, a \mapsto v : \Gamma, a : \tau$.

- **Lemma (Heap Lookup):**
  If $\Gamma \vdash H : \Gamma$ and $H(a) = v$ and $\Gamma(a) = \tau$, then $\Gamma \vdash v : \tau$.

- **Lemma (Heap Update):**
  If $\Gamma \vdash H : \Gamma$ and $\Gamma(a) = \tau$ and $\Gamma \vdash v : \tau$ and $H[a] ← v = H'$, then $\Gamma \vdash H' : \Gamma$. 

2
1. Complete the proof of the following Progress Lemma. (You need to complete the T-Get, T-Set, and T-Addr cases.)

**Lemma (Progress):**

If $\vdash H : \Gamma$ and $\Gamma \vdash e : \tau$, then either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.

**Proof:**

By induction on (the derivation of) $\Gamma \vdash e : \tau$.

- **T-Const** concludes the derivation of $\Gamma \vdash e : \tau$:
  
  Therefore, $e = c$ and $\tau = \text{int}$.

  We must show that either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.

  We have $e = c$ is a value.

- **T-Var** concludes the derivation of $\Gamma \vdash e : \tau$:
  
  Therefore, $e = x$ and $\tau = \Gamma(x)$.

  We must show that either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.

  From $\vdash H : \Gamma$, we have that “$\Gamma$ has no variables and exactly the same addresses as $H$.”

  But, $\tau = \Gamma(x)$ is contradictory, since “$\Gamma$ has no variables.”

  Therefore, vacuously, either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.

- **T-Lam** concludes the derivation of $\Gamma \vdash e : \tau$:
  
  Therefore, $e = \lambda x. e_b$ and $\tau = \tau_a \rightarrow \tau_r$.

  We must show that either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.

  We have $e = \lambda x. e_b$ is a value.
• T-APPLY concludes the derivation of $\Gamma \vdash e : \tau$.
Therefore, $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$, $\Gamma \vdash e_a : \tau_a$, $e = e_f e_a$, and $\tau = \tau_r$.
We must show that either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \rightarrow_{cbv} H'; e'$.
By the induction hypothesis applied to $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$ with $\vdash H : \Gamma$, we have either

- $e_f$ is a value:
  Therefore, $e_f = v_f$.
  By the induction hypothesis applied to $\Gamma \vdash e_a : \tau_a$ with $\vdash H : \Gamma$, we have either

  * $e_a$ is a value:
    Therefore, $e_a = v_a$.
    From $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$ and $e_f = v_f$, we have $\Gamma \vdash v_f : \tau_a$. 
    By Canonical Forms applied to $\vdash H : \Gamma$ and $\vdash v_f : \tau_a$, we have $v_f = \lambda x. e_b$.
    From E-APPLY, we can construct the derivation $H; (\lambda x. e_b) v_a \rightarrow_{cbv} H; e_b[v_a/x]$.
    Therefore, we have $(\lambda x. e_b) v_a \rightarrow_{cbv} H; e_b[v_a/x]$.
    Take $H' = H$ and $e' = e_b[v_a/x]$.
    From $H; (\lambda x. e_b) v_a \rightarrow_{cbv} H; e_b[v_a/x]$, $e = e_f e_a$, $e_f = v_f$, $e_a = v_a$, $v_f = \lambda x. e_b$, $H' = H$, and $e' = e_b[v_a/x]$, we have $H; e \rightarrow_{cbv} H'; e'$.

  * there exists an $H^1$ and an $e'_a$ such that $H; e_a \rightarrow_{cbv} H^1; e'_a$:
    From E-APPLY2 and $H; e_a \rightarrow_{cbv} H^1; e'_a$,
    we can construct the derivation $H; e_a \rightarrow_{cbv} H^1; e'_a$.
    Therefore, we have $H; e_a \rightarrow_{cbv} H^1; e'_a$.
    Take $H' = H^1$ and $e' = v_f e'_a$.
    From $H; v_f e_a \rightarrow_{cbv} H^1; v_f e'_a$, $e = e_f e_a$, $v_f = e_f$, $H' = H^1$, and $e' = v_f e'_a$, we have $H; e \rightarrow_{cbv} H'; e'$.

- there exists an $H^1$ and an $e'_f$ such that $H; e_f \rightarrow_{cbv} H^1; e'_f$:
  From E-APPLY1 and $H; e_f \rightarrow_{cbv} H^1; e'_f$,
  we can construct the derivation $H; e_f \rightarrow_{cbv} H^1; e'_f$.
  Therefore, we have $H; e_f \rightarrow_{cbv} H^1; e'_f$.
  Take $H' = H^1$ and $e' = e'_f e_a$.
  From $H; e_f e_a \rightarrow_{cbv} H^1; e'_f e_a$, $e = e_f e_a$, $H' = H^1$, and $e' = e'_f e_a$, we have $H; e \rightarrow_{cbv} H'; e'$.
• **T-Alloc** concludes the derivation of $\Gamma \vdash e : \tau$:

Therefore, $\Gamma \vdash e_a : \tau_a$, $e = \text{ref } e_a$, and $\tau = \text{ref } \tau_a$.

We must show that either $e$ is a value or there exists an $H'$ and an $e'$ such that $H; e \to_{\text{cbv}} H'; e'$.

By the induction hypothesis applied to $\Gamma \vdash e_a : \tau_a$ with $\vdash H : \Gamma$, we have either

- $e_a$ is a value:

  Therefore, $e_a = v_a$.

  Because the set of addresses is infinite and the set of addresses in $\text{Dom}(H)$ is finite, there exists $a^\dagger$ such that $a^\dagger \notin \text{Dom}(H)$.

  From E-Alloc and $a^\dagger \notin \text{Dom}(H)$,

  we can construct the derivation $\quad a^\dagger \notin \text{Dom}(H) \quad \frac{H; \text{ref } v_a \to_{\text{cbv}} H, a^\dagger \mapsto v_a; a^\dagger}{H; \text{ref } v_a \to_{\text{cbv}} H, a^\dagger \mapsto v_a; a^\dagger}$.

  therefore, we have $H; \text{ref } v_a \to_{\text{cbv}} H, a^\dagger \mapsto v_a; a^\dagger$.

  Take $H' = H, a^\dagger \mapsto v_a$ and $e' = a^\dagger$.

  From $H; \text{ref } v_a \to_{\text{cbv}} H, a^\dagger \mapsto v_a; a^\dagger, e = \text{ref } e_a, e_a = v_a, H' = H, a^\dagger \mapsto v_a$, and $e' = a^\dagger$,

  we have $H; e \to_{\text{cbv}} H'; e'$.

- there exists an $H^\dagger$ and an $e' = a^\dagger$

  From E-Alloc1 and $H; e_a \to_{\text{cbv}} H^\dagger; e'_a$,

  we can construct the derivation $\quad H; e_a \to_{\text{cbv}} H^\dagger; e'_a \quad \frac{H; e_a \to_{\text{cbv}} H^\dagger; e'_a}{H; \text{ref } e_a \to_{\text{cbv}} H^\dagger; \text{ref } e'_a}$.

  therefore, we have $H; \text{ref } e_a \to_{\text{cbv}} H^\dagger; \text{ref } e'_a$.

  Take $H' = H^\dagger$ and $e' = \text{ref } e'_a$.

  From $H; \text{ref } e_a \to_{\text{cbv}} H^\dagger; \text{ref } e'_a, e = \text{ref } e_a, H' = H^\dagger$, and $e' = \text{ref } e'_a$,

  we have $H; e \to_{\text{cbv}} H'; e'$.

• **T-Get** concludes the derivation of $\Gamma \vdash e : \tau$:

  ***COMPLETE THE T-Get CASE.***

• **T-Set** concludes the derivation of $\Gamma \vdash e : \tau$:

  ***COMPLETE THE T-Set CASE.***

• **T-Addr** concludes the derivation of $\Gamma \vdash e : \tau$:

  ***COMPLETE THE T-Addr CASE.***
2. Complete the proof of the following Preservation Lemma.
(You need to complete the T-Get/E-Get, T-Set/E-Set2, T-Set/E-Set, and T-Addr cases.)

**Lemma (Preservation):**
If \( \vdash H : \Gamma, \Gamma \vdash e : \tau \), and \( H; e \rightarrow_{\text{cbv}} H'; e' \),
then there exists \( \Gamma' \) such that \( \Gamma' \) extends \( \Gamma \), \( \vdash H' : \Gamma' \), and \( \Gamma' \vdash e' : \tau \).

**Proof:**
By induction on (the derivation of) \( \Gamma \vdash e : \tau \).

- **T-Const** concludes the derivation of \( \Gamma \vdash e : \tau \):
  Therefore, \( e = c \) and \( \tau = \text{int} \).
  From \( H; e \rightarrow_{\text{cbv}} H'; e' \) and \( e = c \), we have \( H; c \rightarrow_{\text{cbv}} H'; e' \).
  But \( H; c \rightarrow_{\text{cbv}} H'; e' \) is contradictory; there is no derivation of such a judgement.
  Therefore, vacuously, there exists \( \Gamma' \) such that \( \Gamma' \) extends \( \Gamma \), \( \vdash H' : \Gamma' \), and \( \Gamma' \vdash e' : \tau \).

- **T-Var** concludes the derivation of \( \Gamma \vdash e : \tau \):
  Therefore, \( e = x \) and \( \tau = \Gamma(x) \).
  From \( H; e \rightarrow_{\text{cbv}} H'; e' \) and \( e = x \), we have \( H; x \rightarrow_{\text{cbv}} H'; e' \).
  But \( H; x \rightarrow_{\text{cbv}} H'; e' \) is contradictory; there is no derivation of such a judgement.
  Therefore, vacuously, there exists \( \Gamma' \) such that \( \Gamma' \) extends \( \Gamma \), \( \vdash H' : \Gamma' \), and \( \Gamma' \vdash e' : \tau \).

- **T-Lam** concludes the derivation of \( \Gamma \vdash e : \tau \):
  Therefore, \( \Gamma, x : \tau_a \vdash e_b : \tau_r \), \( e = \lambda x. e_b \), and \( \tau = \tau_a \rightarrow \tau_r \).
  From \( H; e \rightarrow_{\text{cbv}} H'; e' \) and \( e = \lambda x. e_b \), we have \( H; \lambda x. e_b \rightarrow_{\text{cbv}} H'; e' \).
  But \( \lambda x. e_b \rightarrow_{\text{cbv}} e' \) is contradictory; there is no derivation of such a judgement.
  Therefore, vacuously, there exists \( \Gamma' \) such that \( \Gamma' \) extends \( \Gamma \), \( \vdash H' : \Gamma' \), and \( \Gamma' \vdash e' : \tau \).
• T-APPLY concludes the derivation of $\Gamma \vdash e : \tau$: 
Therefore, $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$, $\Gamma \vdash e_a : \tau_a$, $e = e_f$, $e_a$, and $\tau = \tau_r$.
From $H; e \rightarrow_{cbv} H'; e'$ and $e = e_f$, $e_a$, we have $H; e_f e_a \rightarrow_{cbv} H'; e'$.
By cases on (the derivation of) $H; e_f e_a \rightarrow_{cbv} H'; e'$.

– E-APPLY concludes the derivation of $H; e_f e_a \rightarrow_{cbv} H'; e'$:
Therefore, $e_f = \lambda x. e_b$, $e_a = v_a$, $H' = H$, and $e' = e_b[v_a/x]$.
From $\Gamma \vdash e_a : \tau_a$ and $e_a = v_a$, we have $\Gamma \vdash v_a : \tau_a$.
From $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$ and $e_f = \lambda x. e_b$, we have $\Gamma \vdash \lambda x. e_b : \tau_a \rightarrow \tau_r$.
By inversion of $\Gamma \vdash \lambda x. e_b : \tau_a \rightarrow \tau_r$, we have $\Gamma, x : \tau_a \vdash e_b : \tau_r$.
By Substitution applied to $\Gamma, x : \tau_a \vdash e_b : \tau_r$ and $\Gamma \vdash v_a : \tau_a$, we have $\Gamma \vdash e_b[v_a/x] : \tau_r$.
Take $\Gamma' = \Gamma$.
From $\Gamma' = \Gamma$, we have $\Gamma'$ extends $\Gamma$.
From $\vdash H : \Gamma$, $H' = H$, and $\Gamma' = \Gamma$, we have $\vdash H' : \Gamma'$.
From $\Gamma \vdash e_b[v_a/x] : \tau_r$, $e' = e_b[v_a/x]$, $\tau = \tau_r$, and $\Gamma' = \Gamma$, we have $\Gamma' \vdash e' : \tau$.

– E-APPLY1 concludes the derivation of $H; e_f e_a \rightarrow_{cbv} H'; e'$:
Therefore, $H; e_f \rightarrow_{cbv} H'; e_f'$, $H' = H'$, and $e' = e_f e_a$.
By the induction hypothesis applied to $\vdash H : \Gamma$, $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$, and $H; e_f \rightarrow_{cbv} H'; e_f'$, we have there exists $\Gamma'$ such that $\Gamma'$ extends $\Gamma$, $\vdash H' : \Gamma'$, and $\Gamma' \vdash e'_f : \tau_a \rightarrow \tau_r$.
By Weakening applied to $\Gamma \vdash e_a : \tau_a$ and $\Gamma' \vdash e_a : \tau_a$,
we can construct the derivation $\Gamma' \vdash e'_f : \tau_a \rightarrow \tau_r$. 
$\Gamma' \vdash e'_f e_a : \tau_a$;
therefore, we have $\Gamma' \vdash e'_fe_a : \tau_r$.
Take $\Gamma' = \Gamma'$.
From $\Gamma'$ extends $\Gamma$ and $\Gamma' = \Gamma'$, we have $\Gamma'$ extends $\Gamma$.
From $\vdash H' : \Gamma'$, $H' = H'$, and $\Gamma' = \Gamma'$, we have $\vdash H' : \Gamma'$.
From $\Gamma' \vdash e'_f e_a : \tau_r$, $e' = e'_f e_a$, $\tau = \tau_r$, and $\Gamma' = \Gamma'$, we have $\Gamma' \vdash e' : \tau$.

– E-APPLY2 concludes the derivation of $H; e_f e_a \rightarrow_{cbv} H'; e'$:
Therefore, $H; e_a \rightarrow_{cbv} H'; e'_a$, $e_f = v_f$, $H' = H'$, and $e' = v_f e'_a$.
By the induction hypothesis applied to $\vdash H : \Gamma$, $\Gamma \vdash e_a : \tau_a$, and $H; e_a \rightarrow_{cbv} H'; e'_a$,
we have there exists $\Gamma'$ such that $\Gamma'$ extends $\Gamma$, $\vdash H' : \Gamma'$, and $\Gamma' \vdash e'_a : \tau_a$.
From $\Gamma \vdash e_f : \tau_a \rightarrow \tau_r$ and $e_f = v_f$, we have $\Gamma \vdash v_f : \tau_a \rightarrow \tau_r$.
By Weakening applied to $\Gamma \vdash v_f : \tau_a \rightarrow \tau_r$ and $\Gamma' \vdash e'_a : \tau_a$,
we have $\Gamma' \vdash v_f : \tau_a \rightarrow \tau_r$.
From T-APPLY, $\Gamma' \vdash v_f : \tau_a \rightarrow \tau_r$, and $\Gamma' \vdash e'_a : \tau_a$,
we can construct the derivation $\Gamma' \vdash v_f : \tau_a \rightarrow \tau_r$.
$\Gamma' \vdash v_f e'_a : \tau_r$;
therefore, we have $\Gamma' \vdash v_f e'_a : \tau_r$.
Take $\Gamma' = \Gamma'$.
From $\Gamma'$ extends $\Gamma$ and $\Gamma' = \Gamma'$, we have $\Gamma'$ extends $\Gamma$.
From $\vdash H' : \Gamma'$, $H' = H'$, and $\Gamma' = \Gamma'$, we have $\vdash H' : \Gamma'$.
From $\Gamma' \vdash v_f e'_a : \tau_r$, $e' = v_f e'_a$, $\tau = \tau_r$, and $\Gamma' = \Gamma'$, we have $\Gamma' \vdash e' : \tau$. 7
• T-Alloc concludes the derivation of \( \Gamma \vdash e : \tau \):

Therefore, \( \Gamma \vdash e_a : \tau_a \), \( e = \text{ref } e_a \), and \( \tau = \text{ref } \tau_a \).

From \( H ; e \rightarrow_{\text{cbv}} H' ; e' \) and \( e = \text{ref } e_a \), we have \( H ; \text{ref } e_a \rightarrow_{\text{cbv}} H' ; e' \).

By cases on (the derivation of) \( H ; \text{ref } e_a \rightarrow_{\text{cbv}} H' ; e' \):

- E-Alloc1 concludes the derivation of \( H ; \text{ref } e_a \rightarrow_{\text{cbv}} H' ; e' \):

  Therefore, \( H ; e_a \rightarrow_{\text{cbv}} H' ; e' \), \( e = \text{ref } e_a \), \( H' = H \), and \( e' = \text{ref } e_a' \).

  By the induction hypothesis applied to \( H : \Gamma \), \( \Gamma \vdash e_a : \tau_a \), and \( H ; e_a \rightarrow_{\text{cbv}} H' ; e_a' \), we have there exists \( \Gamma' \) such that \( \Gamma' \) extends \( \Gamma \), \( \vdash H' : \Gamma' \), and \( \Gamma' \vdash e_a' : \tau_a \).

  From T-Alloc and \( \Gamma' \vdash e_a' : \tau_a \),

  we can construct the derivation

  \[
  \Gamma' \vdash e_a' : \tau_a
  \]

  therefore, we have \( \Gamma' \vdash \text{ref } e_a' : \text{ref } \tau_a \).

  Take \( \Gamma' = \Gamma' \).

  From \( \Gamma' \) extends \( \Gamma \) and \( \Gamma' \vdash e_a' : \tau_a \), we have \( \Gamma' \vdash H' : \Gamma' \).

  From \( \Gamma' \vdash e_a' : \tau_a \), \( e' = \text{ref } e_a' \), \( \tau = \text{ref } \tau_a \), and \( \Gamma' = \Gamma' \), we have \( \Gamma' \vdash e' : \tau \).

- E-Alloc concludes the derivation of \( H ; \text{ref } e_a \rightarrow_{\text{cbv}} H' ; e' \):

  Therefore, \( e_a = v_a \), \( a \notin \text{Dom}(H) \), \( H' = H \), \( a \rightarrow v_a \), and \( e' = a \).

  From \( \Gamma \vdash e_a : \tau_a \) and \( \Gamma \vdash v_a : \tau_a \), we have \( \Gamma \vdash v_a : \tau_a \).

  By Heap Extension applied to \( H \), \( a \notin \text{Dom}(H) \), and \( \Gamma \vdash v_a : \tau_a \), we have \( \vdash H, a \rightarrow v_a : \Gamma, a : \tau_a \).

  From T-Addr, we can construct the derivation

  \[
  (\Gamma, a : \tau_a)(a) = \tau_a
  \]

  therefore, we have \( \Gamma, a : \tau_a \vdash a : \text{ref } \tau_a \).

  Take \( \Gamma' = \Gamma, a : \tau_a \).

  By definition of extends and \( \Gamma' = \Gamma, a : \tau_a \), we have \( \Gamma' \) extends \( \Gamma \).

  From \( \vdash H, a \rightarrow v_a : \Gamma, a : \tau_a \), \( H' = H, a \rightarrow v_a \), and \( \Gamma' = \Gamma, a : \tau_a \), we have \( \vdash H' : \Gamma' \).

  From \( \Gamma, a : \tau_a \vdash a : \text{ref } \tau_a \), \( e' = a \), \( \tau = \text{ref } \tau_a \), and \( \Gamma' = \Gamma, a : \tau_a \), we have \( \Gamma' \vdash e' : \tau \).
• T-Get concludes the derivation of $\Gamma \vdash e : \tau$:

Therefore, $\Gamma \vdash e_r : \text{ref } \tau_a$, $e = e_r$, and $\tau = \tau_a$.

From $H; e \rightarrow_{\text{cbv}} H'; e'$ and $e = e_r$, we have $H; e_r \rightarrow_{\text{cbv}} H'; e'$.

By cases on (the derivation of) $H; e_r \rightarrow_{\text{cbv}} H'; e'$:

- E-Get1 concludes the derivation of $H; e_r \rightarrow_{\text{cbv}} H'; e'$:

Therefore, $H; e_r \rightarrow_{\text{cbv}} H'; e_r'$, and $e' = e_r'$.

By the induction hypothesis applied to $\vdash H : \Gamma$, $\Gamma \vdash e_r : \text{ref } \tau_a$, and $H; e_r \rightarrow_{\text{cbv}} H'; e_r'$, we have there exists $\Gamma'$ such that $\Gamma'$ extends $\Gamma$, $\vdash H' : \Gamma'$, and $\Gamma \vdash e_r : \text{ref } \tau_a$.

From T-Get and $\Gamma \vdash e_r : \text{ref } \tau_a$,

we can construct the derivation $\Gamma \vdash e_r : \text{ref } \tau_a$;

therefore, we have $\Gamma' \vdash e_r : \tau_a$.

Take $\Gamma' = \Gamma^\uparrow$.

From $\Gamma'^\uparrow$ extends $\Gamma$ and $\Gamma' = \Gamma^\uparrow$, we have $\Gamma'$ extends $\Gamma$.

From $\vdash H' : \Gamma'$, $H' = H'$, and $\Gamma' = \Gamma^\uparrow$, we have $\vdash H' : \Gamma'$.

From $\Gamma' \vdash ! e_r' : \tau_a$, $e' = e_r'$, $\tau = \tau_a$, and $\Gamma' = \Gamma^\uparrow$, we have $\Gamma' \vdash e' : \tau$.

- E-Get concludes the derivation of $H; e_r \rightarrow_{\text{cbv}} H'; e'$:

***COMPLETE THE E-Get CASE.***

(Hint: Use the Heap Lookup Lemma.)

• T-Set concludes the derivation of $\Gamma \vdash e : \tau$:

Therefore, $\Gamma \vdash e : \text{ref } \tau_a$, $\Gamma \vdash e_a : \tau_a$, $e = e_a$, and $\tau = \tau_a$.

From $H; e \rightarrow_{\text{cbv}} H'; e'$ and $e = e_a$, we have $H; e_a \rightarrow_{\text{cbv}} H'; e'$.

By cases on (the derivation of) $H; e_a \rightarrow_{\text{cbv}} H'; e'$:

- E-Set1 concludes the derivation of $H; e_a \rightarrow_{\text{cbv}} H'; e'$:

Therefore, $H; e_a \rightarrow_{\text{cbv}} H'; e_a'$, and $e' = e_a'$.

By the induction hypothesis applied to $\vdash H : \Gamma$, $\Gamma \vdash e_r : \text{ref } \tau_a$, and $H; e_a \rightarrow_{\text{cbv}} H'; e_a'$, we have there exists $\Gamma'$ such that $\Gamma'$ extends $\Gamma$, $\vdash H' : \Gamma'$, and $\Gamma \vdash e_a : \text{ref } \tau_a$.

By Weakening applied to $\Gamma \vdash e_a : \tau_a$ and $\Gamma'$ extends $\Gamma$,

we have $\Gamma' \vdash e_a : \tau_a$.

From T-Set, $\Gamma' \vdash e_a' : \text{ref } \tau_a$, and $\Gamma' \vdash e_a : \tau_a$,

we can construct the derivation $\Gamma \vdash e_a' : \text{ref } \tau_a$;

therefore, we have $\Gamma' \vdash e_a' : \tau_a$.

Take $\Gamma' = \Gamma^\uparrow$.

From $\Gamma'^\uparrow$ extends $\Gamma$ and $\Gamma' = \Gamma^\uparrow$, we have $\Gamma'$ extends $\Gamma$.

From $\vdash H' : \Gamma'$, $H' = H'$, and $\Gamma' = \Gamma^\uparrow$, we have $\vdash H' : \Gamma'$.

From $\Gamma' \vdash e_a' : \tau_a$, $e_a' = e_a'$, $\tau = \tau_a$, and $\Gamma' = \Gamma^\uparrow$, we have $\Gamma' \vdash e' : \tau$.

- E-Set2 concludes the derivation of $H; e_a : \text{ref } \tau_a$:

***COMPLETE THE E-Set2 CASE.***

- E-Set concludes the derivation of $H; e_a : \text{ref } \tau_a$:

***COMPLETE THE E-Set CASE.***

(Hint: Use the Heap Update Lemma.)

• T-Addr concludes the derivation of $\Gamma \vdash e : \tau$:

***COMPLETE THE T-Addr CASE.***

3. Explain why the proof of the Preservation Lemma would not succeed if it did not state that $\Gamma'$ extends $\Gamma$. In other words, explain why the following two “simplifications” of the Preservation Lemma cannot be proven:

- If $\vdash H : \Gamma$, $\Gamma \vdash e : \tau$, and $H; e \rightarrow_{\text{cbv}} H'; e'$, then $\vdash H' : \Gamma$, and $\Gamma \vdash e' : \tau$.

- If $\vdash H : \Gamma$, $\Gamma \vdash e : \tau$, and $H; e \rightarrow_{\text{cbv}} H'; e'$, then $\Gamma'$ extends $\Gamma$, $\vdash H' : \Gamma'$, and $\Gamma' \vdash e' : \tau$. 

***COMPLETE THE T-Addr CASE.***
Part II

Here is the (revised) syntax and large-step environment-based operational semantics for a simply-typed lambda calculus with a mutable heap. Important comments and definitions follow.

\[ e ::= c | x | \lambda x. e | e e | \text{ref} \ e | ! e | e := e \]
\[ v ::= c | \langle E, x, e \rangle | a \]
\[ H ::= \cdot | H, a \mapsto v \]
\[ E ::= \cdot | E, x \mapsto v \]
\[ \tau ::= \text{int} | \tau \rightarrow \tau | \text{ref} \tau \]
\[ \Gamma ::= \cdot | \Gamma, x : \tau \]

\[
H; E; e \Downarrow_{\text{cbv}} H'; v'
\]

- **E-Const**
  \[
  H_0; E; c \Downarrow_{\text{cbv}} H_0; c
  \]
- **E-Var**
  \[
  E(x) = v \\
  H_0; E; x \Downarrow_{\text{cbv}} H_0; v
  \]
- **E-Lam**
  \[
  H_0; E; \lambda x. e \Downarrow_{\text{cbv}} H_0; (E, x, e)
  \]
- **E-Apply**
  \[
  H_0; E; e_f \Downarrow_{\text{cbv}} H_1; (E', x, e) \\
  H_1; E; e_a \Downarrow_{\text{cbv}} H_2; v_a \\
  H_2; E'; x \mapsto v_a; e \Downarrow_{\text{cbv}} H_3; v
  \]
- **E-Alloc**
  \[
  H_0; E; e_a \Downarrow_{\text{cbv}} H_1; v_a \\
  a \notin \text{Dom}(H_1) \\
  H; E; \text{ref} \ e_a \Downarrow_{\text{cbv}} H_1, a \mapsto v_a; a
  \]
- **E-Get**
  \[
  H_0; E; e_r \Downarrow_{\text{cbv}} H_1; a \\
  H_1(a) = v \\
  H; E; ! e_r \Downarrow_{\text{cbv}} H_1; v
  \]
- **E-Set**
  \[
  H_0; E; e_r \Downarrow_{\text{cbv}} H_1; a \\
  H_1; E; e_a \Downarrow_{\text{cbv}} H_2; v \\
  H_2[a] \leftarrow v = H_3
  \]

Notes:

- Note that the abstract syntax of values is not a subset of the abstract syntax of expressions. In particular, unlike the small-step semantics in Part 1, addresses are not part of the expression syntax.
- A large-step interpreter for a language with a heap must produce a heap and a value as results and “thread the heap through”.
- An environment-based interpreter does not use substitution. Instead, the evaluation judgement includes an environment \( E \), which maps variables to values. To implement lexical scope correctly, functions are not values — rather, they evaluate to closures \( \langle E, x, e \rangle \). Furthermore, variables have an evaluation rule (E-Var) — the value of the variable is lookup-ed in the environment. Note how the E-Apply rule works — the body of the function is evaluated using the environment in its closure! Notice we do not “thread the environment through;” evaluation does not produce an environment.
- We adopt the type-system from Part 1, except that we drop the T-Addr rule; addresses are not part of the expression syntax and we need only type-check expressions.

The code provided defines abstract syntax and a scanner/parser for the simply-typed lambda calculus with integers, addition, multiplication, greater-than, integer-based conditionals (0 is false, other integers are true), and a heap. To make type-checking easier, functions have the concrete syntax \( \text{fn} x : t => e \). Specifically, they must have explicit type arguments and the “:” and “\(\Rightarrow\)” must be present. To make parsing easier, some expressions may require more parentheses than would be required in SML; see the example files provided. The parser transforms \( \text{let} \ x : t = e \ in \ eb \) to \( \text{fn} x : t => \text{eb} \) \( e \); therefore, the abstract syntax does include a case for let-expressions.
4. Complete the type-checker `typecheck`. Your type-checker should raise `TypeError` if and only if the expression does not typecheck under the provided context. Use the little list library in the file for managing the context.

5. Complete the large-step interpreter `interpret`. Remember to “thread the heap through.” In particular, at a function call, the body is evaluated using the environment in the closure but the current heap. Use the little list library in the file for managing environments and heaps. Your interpreter should raise `RuntimeError` if and only if it gets stuck. (But expressions that type-check should not get stuck!)

6. The provided `factorial.lcr` file should produce the value 120. The provided `infinite.lcr` file should diverge. Explain how it is possible to write a recursive function without including a `fix` expression.

Debriefing

- How many hours did you spend on this assignment?
- Would you rate it as easy, moderate, or difficult?
- How deeply do you feel you understand the material it covers (0% – 100%)?
- If you have any other comments about the assignment, then please include them with your submission or send email to mtf@cs.rit.edu.

Submission

All components of the assignment:

- problems 1, 2, 3, 6, and debriefing in a file named `homework05.pdf`
- problems 4 and 5 in the file named `lcr.sml`

must be submitted to the Homework 5 Assignment on MyCourses by the due date.

For handwritten components, please use either a document scanner or a mobile scanning app like Adobe Scan or Microsoft Lens.