1 Ephemeral Storage in the S Machine

Relating the S machine back to the K machine briefly works as follows: an eval object followed by threaded sequence of continuations

\[
\text{eval } e \text{ } d_n, \text{cont } d_n k_n d_{n-1}, \text{cont } d_{n-1} k_{n-1} d_{n-2}, \ldots, \text{cont } d_1 k_1 d_0
\]

is combined into a single stack and expression to be evaluated

\[
\epsilon \circ k_1 \circ \ldots \circ k_{n-1} \circ k_n \triangleright e
\]

and then we (recursively) substitute store contents for destinations until no destinations are left in the state of the K machine.

Conspicuously absent from this mapping is an S machine state corresponding to

\[
\epsilon \circ k_1 \circ \ldots \circ k_{n-1} \circ k_n \triangleleft v
\]

We address this by creating a new (ephemeral!) semantic object

\[
\text{retn } c \text{ } d
\]
where $d$ is a destination and $c$ is a valid cell contents. It expresses that $c$ is returned to destination $d$, and its intention is that there is a continuation waiting for a value at $d$. These new ephemeral semantic objects can replace some of the persistent objects !cell $d\ c$. Besides helping us in establishing a bisimulation between the K and S machines, it also generates less “junk” in the store that remains until the end of the computation.

We show three example rules for this improved version of the semantics.

\[
!\text{cell} \ d\ c, \ \text{eval} \ d \ d' \mapsto \text{retn} \ c \ d' \\
\text{eval} \ (\lambda x\ .\ e) \ d \mapsto \text{retn} \ (\lambda x\ .\ e) \ d \\
\text{retn} \ \langle\rangle \ d, \ \text{cont} \ d \ (\text{case} _\_ \{ \langle\rangle \Rightarrow e' \}) \ d' \mapsto \text{eval} \ e' \ d'
\]

Note that in the first rule, !cell $d\ c$ persists in the state, even though it is not explicitly mentioned on the right-hand side. On the other hand, in the last rule the object retn $\langle\rangle \ d$ is removed from the state, which is correct since the shown continuation should be the only reference to destination $d$.

**Task 1** (10 points, WB). Show all rules for handling binary sums $\tau_1 + \tau_2$. Be sure to use retn objects only where appropriate for a bisimulation (which you do not need to prove). For full credit, you should create appropriate persistent !cell objects only when necessary.

**Solution 1:**

\[
\begin{align*}
\text{eval} \ (i \cdot e) \ d &\mapsto \text{eval} \ e' \ d', \ \text{cont} \ d' \ (i \cdot _) \ d \\
\text{retn} \ c \ d', \ \text{cont} \ d' \ (i \cdot _) \ d &\mapsto !\text{cell} \ d' \ c, \ \text{retn} \ (i \cdot d') \ d \\
\text{eval} \ (\text{case} \ e_0 \ {l \cdot x \Rightarrow e_1 \ | \ r \cdot y \Rightarrow e_2}) \ d &\mapsto \text{eval} \ e_0 \ d_0, \ \text{cont} \ d_0 \ (\text{case} _\_ \{ l \cdot x \Rightarrow e_1 \ | \ r \cdot y \Rightarrow e_2 \}) \ d \\
&\mapsto \text{eval} \ (d_0 \text{ fresh}) \\
\text{retn} \ (l \cdot d_1) \ d_0, \ \text{cont} \ d_0 \ (\text{case} _\_ \{ l \cdot x \Rightarrow e_1 \ | \ r \cdot y \Rightarrow e_2 \}) \ d &\mapsto \text{eval} \ ([d_1/x]e_1) \ d \\
\text{retn} \ (r \cdot d_2) \ d_0, \ \text{cont} \ d_0 \ (\text{case} _\_ \{ l \cdot x \Rightarrow e_1 \ | \ r \cdot y \Rightarrow e_2 \}) \ d &\mapsto \text{eval} \ ([d_2/y]e_2) \ d
\end{align*}
\]

**Task 2** (10 points, WB). Show all rules for handling binary lazy products $\tau_1 \& \tau_2$, under the same instructions as the previous task.

**Solution 2:**

\[
\begin{align*}
\text{eval} \ (e_1, e_2) \ d &\mapsto \text{retn} \ (e_1, e_2) \ d \\
\text{eval} \ (e \cdot i) \ d &\mapsto \text{eval} \ e' \ d', \ \text{cont} \ d' \ (\_ \cdot i) \ d \\
&\mapsto \text{eval} \ e_1 \ d \\
\text{retn} \ (e_1, e_2) \ d', \ \text{cont} \ d' \ (\_ \cdot i) \ d &\mapsto \text{eval} \ e_1 \ d \\
\text{retn} \ (e_1, e_2) \ d', \ \text{cont} \ d' \ (\_ \cdot r) \ d &\mapsto \text{eval} \ e_2 \ d
\end{align*}
\]

\[\square\]
2 Compiling Abstract Types

In class, we implicitly handled type abstraction at the term level, i.e., our terms had no syntax to indicate that we were abstracting over types. Indeed, our rules were of the form:

\[
\begin{align*}
\Delta, \alpha \text{ type} ; \Gamma \vdash e : \tau & \quad & \Delta ; \Gamma \vdash e : \forall \alpha.\tau & \quad & \Delta ; \Gamma \vdash e : [\sigma/\alpha]\tau \\
\Delta ; \Gamma \vdash \Lambda \alpha. e : \forall \alpha.\tau &
\end{align*}
\]

More traditional presentations use term-level syntax \(\Lambda \alpha.e\) for abstracting over types and \(e[\sigma]\) for applying terms to types. In this section, we explore how to evaluate expressions for abstract types \(\exists \alpha.\tau\) and \(\forall \alpha.\tau\) on K and S\(_\eta\) machines. In particular, we are concerned with how to evaluate expressions typed by the following rules:

\[
\begin{align*}
\Delta, \alpha \text{ type} ; \Gamma \vdash e : \tau & \quad & \Delta ; \Gamma \vdash e : [\sigma/\alpha]\tau \\
\Delta ; \Gamma \vdash e : \forall \alpha.\tau & \quad & \Delta ; \Gamma \vdash e[\sigma] : [\sigma/\alpha]\tau \\
\Delta, \alpha \text{ type} ; \Gamma, x : \tau \vdash e' : \tau' & \quad & \Delta ; \Gamma \vdash \text{case } e \{\langle \alpha, x \rangle \Rightarrow e'\} : \tau'\\
\end{align*}
\]

The operational semantics for terms given by (I-\(\exists\)) and (E-\(\exists\)) are as on p. L14.5. The operational semantics for terms given by (I-\(\forall\)) and (E-\(\forall\)) are given by:

\[
\begin{align*}
\Lambda \alpha.e \text{ val} & \quad & e \mapsto e' & \quad & (C-\forall) \\
\end{align*}
\]

Task 3 (5 points, WB). The rule (E-\(\exists\)) on p. L14.5 is subject to several presuppositions. Why is it important that \(\alpha\) not appear in \(\tau'\), i.e., that \(\Delta \vdash \tau'\) type? Explain your answer in no more than 5 lines.

Solution 3: Without the premise, you could leak a value of abstract type out of its scope and get a term of variable type in an empty context. Consider for example the following derivation, where the term in the conclusion has variable type \(\alpha\) even though \(\alpha\) type is not in the empty type variable context “.":

\[
\begin{align*}
\vdots ; \cdot \vdash \langle 1, \langle \rangle \rangle ; \exists \alpha.\alpha \quad \alpha \text{ type} ; x : \alpha \vdash x : \alpha & \quad & \vdots ; \cdot \vdash \text{case } \langle 1, \langle \rangle \rangle \{\langle \alpha, x \rangle \Rightarrow x\} : \alpha
\end{align*}
\]
Task 4 (5 points, WB). Why would we be wrong to replace the rule (R-$\forall$) with the following rule?

$$
(\Lambda \alpha.e)[\sigma] \mapsto e \quad (X-$\forall$)
$$

Solution 4: It breaks the preservation theorem. Consider

$$
\vdash \Lambda \alpha.(\alpha, \lambda x.x) : \forall \alpha.\exists \beta.\beta \rightarrow \alpha.
$$

Then

$$
\vdash (\Lambda \alpha.(\alpha, \lambda x.x))[\mathbf{1}] : \exists \beta.\beta \rightarrow \mathbf{1}.
$$

But under (X-$\forall$), $(\Lambda \alpha.(\alpha, \lambda x.x))[\mathbf{1}] \mapsto (\alpha, \lambda x.x)$. Suppose to the contrary that

$$
\vdash \langle \alpha, \lambda x.x \rangle : \exists \beta.\beta \rightarrow \mathbf{1}.
$$

By inversion with (I-$\exists$), we get $\vdash \alpha$ type, a contradiction.

2.1 K machine

Let us warm up by considering evaluation on a stack machine.

Task 5 (5 points, WB). For each of the typing rules (I-$\exists$), (E-$\exists$), (I-$\forall$), and (E-$\forall$),

- if any new stack frames are required to evaluate the expression in the conclusion, give them along with their typing judgments (see pp. L15.8f.);
- if no new stack frames are required, explain why.

Solution 5: Evaluating expressions typed by (I-$\exists$) requires the frame $\langle \sigma, - \rangle$:

$$
k \vdash \exists \alpha.\tau \Rightarrow \rho
$$

(k o $\langle \sigma, - \rangle$) $\vdash [\sigma/\alpha]\tau \Rightarrow \rho$ (FRME)

Evaluating expressions typed by (E-$\exists$) requires the frame case $- \{ (\alpha, x) \Rightarrow e' \}$:

$$
k \vdash \tau' \Rightarrow \rho \quad \alpha \text{ type}; x : \tau \vdash e' : \tau'
$$

(k o case) $- \{ (\alpha, x) \Rightarrow e' \} \vdash \exists \alpha.\tau \Rightarrow \rho$ (FRMC)

Evaluating expressions typed by (I-$\forall$) requires no additional frames because expressions of the form $\Lambda \alpha.e$ are always values. Evaluating expressions typed by (E-$\forall$) requires the frame $-[\sigma]$:

$$
k \vdash [\sigma/\alpha]\tau \Rightarrow \rho
$$

(k o $- [\sigma]$) $\vdash \forall \alpha.\tau \Rightarrow \rho$ (FRMA)

Task 6 (5 points, WB). If any new K machine reduction rules are required, given them here. If none are needed, state this.
Remark. Your reduction rules must be sound and complete relative to the
dynamics of our language. You are not required to prove that this is the case.

Solution 6: The new reduction rules are:

\[
\begin{align*}
  k \triangleright \langle \sigma, e \rangle & \mapsto k \circ \langle \sigma, - \rangle \triangleright e \\
  k \circ \langle \sigma, - \rangle \triangleleft v & \mapsto k \triangleleft \langle \sigma, v \rangle \\
  k \triangleright \text{case } e \{ \langle \alpha, x \rangle \Rightarrow e' \} & \mapsto k \circ \text{case } - \{ \langle \alpha, x \rangle \Rightarrow e' \} \triangleright e \\
  k \triangleright e[\alpha] & \mapsto k \circ -[\sigma] \triangleright e \\
  k \circ -[\sigma] \triangleleft \Lambda \alpha.e & \mapsto k \triangleright [\sigma/\alpha]e
\end{align*}
\]

Recall what it means for a machine state \( s \) to be well-typed with answer type \( \sigma \), written \( s : \sigma \):

\[
\begin{align*}
  k \overset{\tau_1}{\Rightarrow} \sigma & \quad \vdash \vdash e : \tau_1 \\
  (k \triangleright e) : \sigma & \quad (S-E) \\
  k \overset{\tau_1}{\Rightarrow} \sigma & \quad \vdash \vdash v : \tau_1 \quad v \text{val} \\
  (k \triangleleft v) : \sigma & \quad (S-R)
\end{align*}
\]

The associated preservation theorem states that if \( s : \sigma \) and \( s \mapsto s' \), then \( s' : \sigma \).

Task 7 (10 points, WB). Check the preservation theorem for the reduction rules
you gave in task 6. You are only required to submit the solution for

- one rule of the form \( k \triangleright \langle \sigma, e \rangle \mapsto k' \triangleright e' \) (if you gave any such rules),
- one rule of the form \( k \triangleright \langle \sigma, e \rangle \mapsto k' \triangleleft e' \) (if you gave any such rules),
- one rule of the form \( k \triangleright \Lambda \alpha.e \mapsto k' \triangleright e' \) (if you gave any such rules), and
- one rule of the form \( k \triangleleft \Lambda \alpha.e \mapsto k' \triangleright e' \) (if you gave any such rules).

Hint. If you cannot prove the theorem in some of the cases, check your solution
to tasks 5 and 6!

Solution 7: We must show in each of the cases that if \( s : \rho \) and \( s \mapsto s' \), then \( s' : \rho \).

Case \( k \triangleright \langle \sigma, e \rangle \mapsto k \circ \langle \sigma, - \rangle \triangleright e \).

\[
\begin{align*}
  1. \quad (k \triangleright \langle \sigma, e \rangle) : \rho & \quad \text{Assumption} \\
  2. \quad k \overset{\tau_1}{\Rightarrow} \rho & \quad \text{Inversion on 1, (S-E)} \\
  3. \quad \vdash \vdash \langle \sigma, e \rangle : \tau_1 & \quad \text{Inversion on 1, (S-E)} \\
  4. \quad \tau_1 = \exists \alpha.\tau & \quad \text{Inversion on 3}
\end{align*}
\]
5. $\vdash \sigma$ type \hspace{1cm} Inversion on 3
6. $\cdot \vdash e : [\sigma/\alpha]\tau$ \hspace{1cm} Inversion on 3
7. $k \circ (\sigma, -) \div [\sigma/\alpha]\tau \Rightarrow \rho$ \hspace{1cm} (FRME), items 2 and 5
8. $(k \circ (\sigma, -) \triangleright e) : \rho$ \hspace{1cm} (S-E), items 6 and 7

Case There are no rules of the form $k \triangleright (\sigma, e) \mapsto k' \triangleleft e'$.

Case There are no rules of the form $k \triangleright \Lambda \alpha.e \mapsto k' \triangleright e$.

Case $k \circ [\sigma] \triangleleft \Lambda \alpha.e \mapsto k \triangleright [\sigma/\alpha]e$.

1. $(k \circ [\sigma] \triangleleft \Lambda \alpha.e) : \rho$ \hspace{1cm} Assumption
2. $k \circ [\sigma] \div \tau_1 \Rightarrow \sigma$ \hspace{1cm} Inversion on 1, (S-R)
3. $\cdot \vdash \Lambda \alpha.e : \tau_1$ \hspace{1cm} Inversion on 1, (S-R)
4. $\Lambda \alpha.e \ val$ \hspace{1cm} Inversion on 1, (S-R)
5. $\tau_1 = \forall \alpha.\tau$ \hspace{1cm} Inversion on 3, (I-\forall)
6. $k \div [\sigma/\alpha]\tau \Rightarrow \rho$ \hspace{1cm} Inversion on 2, (FRMA)
7. $\cdot \vdash \sigma$ type \hspace{1cm} Inversion on 2, (FRMA)
8. $\alpha$ type; $\cdot \vdash e : \tau$ \hspace{1cm} Inversion on 3, (I-\forall)
9. $\cdot \vdash e : [\sigma/\alpha]\tau$ \hspace{1cm} Substitution property, items 7 and 8
10. $(k \triangleright [\sigma/\alpha]e) : \rho$ \hspace{1cm} (S-E), items 6 and 9

2.2 $S_\eta$ machine

Let us now consider evaluation on a store machine with environments (see pp. L17.5f.). We observed that substituting values for variables was unrealistic in a lower-level implementation. To solve this, we introduced the concept of an environment which maps term variables to locations containing values.

Our development in this section will still use environments to map term variables to locations. Because we are now dealing with terms containing types, e.g. $(\Lambda \alpha.e)[\sigma]$, we might consider having our environment also map type variables to locations containing types. This introduces needless complexity because we never actually use the types at runtime. Indeed, we can prove a theorem showing that we get “the same result” from evaluating terms with types as from evaluating terms whose types have been “erased”\footnote{Type erasure is a translation $|\cdot|$ where, e.g., $|\Lambda \alpha.e| = \lambda \cdot |e|$, $|e[\sigma]| = |e[\cdot]|$, $|e(\cdot)| = |\langle \cdot, e \rangle|$, $|\text{case } e \{\langle \alpha, \cdot \rangle \Rightarrow e' \}| = \text{case } e \{\langle \cdot, \cdot \rangle \Rightarrow e' \}$, $|\lambda \alpha.e| = \lambda \cdot |e|$, $|e_1 e_2| = |e_1| |e_2|$, $|l \cdot e| = l \cdot |e|$, etc.}. In a more realistic setting, we would erase types before evaluating on the $S_\eta$ machine. For simplicity in this assignment, we will instead directly substitute types for type variables.
**Hint.** This subsection builds on the previous one!

**Task 8** (5 points, WB). Give judgments eval $\eta e d$ and associated dynamics to evaluate terms $e$ typed by (I-∀) and (E-∀). To do so,

1. extend the syntax of cells $c$ if needed, and
2. introduce new judgments of the form !cell $d c$ and cont $d k d'$ if needed.

Do not store the types in closures. Rather, directly substitute them for type variables. Explicitly, one should be able to reach a state with eval $\eta [\sigma/\alpha] e d$ from the state eval $\eta (\Lambda \alpha.e)[\sigma] d$. You are not expected to use ephemeral retn $c d$ judgments.

**Solution 8:** We extend the syntax of cells $c$ to include closures $\langle \eta, \Lambda \alpha.e \rangle$.

\[
\text{eval } \eta \Lambda \alpha.e d \mapsto \text{!cell } d \langle \eta, \Lambda \alpha.e \rangle \\
\text{eval } \eta e[\sigma] d \mapsto \text{eval } \eta e d_1, \text{cont } d_1 (-[\sigma]) d \quad (d_1 \ \text{fresh}) \\
\text{!cell } d_1 \langle \eta, \Lambda \alpha.e \rangle, \text{cont } d_1 (-[\sigma]) d \mapsto \text{eval } \eta [\sigma/\alpha] e d
\]

We validate the sanity check. Where we colour the judgments causing the rules to fire in red,

\[
\text{eval } \eta (\Lambda \alpha.e)[\sigma] d \\
\mapsto \text{eval } \eta \Lambda \alpha.e d_1, \text{cont } d_1 (-[\sigma]) d \\
\mapsto \text{!cell } d_1 \langle \eta, \Lambda \alpha.e \rangle, \text{cont } d_1 (-[\sigma]) d \\
\mapsto \text{!cell } d_1 \langle \eta, \Lambda \alpha.e \rangle, \text{eval } \eta [\sigma/\alpha] e d
\]

as desired. □

**Task 9** (5 points, WB). Give judgments eval $\eta e d$ and associated dynamics to evaluate terms $e$ typed by (I-∃) and (E-∃). To do so,

1. extend the syntax of cells $c$ if needed, and
2. introduce new judgments of the form !cell $d c$ and cont $d k d'$ if needed.

Again, do not store the types in closures. Rather, directly substitute them for type variables. You are not expected to use ephemeral retn $c d$ judgments.
Hint. Your rules should validate
\[
\text{eval } (\cdot) \left( \text{case } (1, \lambda y. \{\}) \right) \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} d \mapsto^* !\text{cell } d \{\}, \ldots
\]

Solution 9: We extend the syntax of continuations with the temporary closure
\(\langle \eta, \text{case} - \{\langle \alpha, x \rangle \Rightarrow e'\} \rangle\). The dynamics are given by:

\[
\begin{align*}
\text{eval } \eta \langle \sigma, e \rangle d & \mapsto \text{eval } \eta e \cdot d_1, \text{cont } d_1 \langle \sigma, - \rangle d \quad (d_1 \text{ fresh}) \\
!\text{cell } d_1 \langle 1, \text{cont} d_1 \langle \sigma, - \rangle d & \mapsto !\text{cell } d \langle \sigma, d_1 \rangle \\
\text{eval } \eta \left( \text{case } e \{\langle \alpha, x \rangle \Rightarrow e'\} \right) d & \mapsto \text{eval } \eta e \cdot d_1, \\
& \quad \text{cont } d_1 \langle \eta, \text{case} - \{\langle \alpha, x \rangle \Rightarrow e'\} \rangle d \quad (d_1 \text{ fresh}) \\
!\text{cell } d_1 \langle \sigma, d_2 \rangle, \text{cont } d_1 \langle \eta, \text{case} - \{\langle \alpha, x \rangle \Rightarrow e'\} \rangle d & \mapsto \text{eval } (\eta, d_2/x) [\sigma/\alpha]e' d
\end{align*}
\]

We would expect these rules to validate
\[
\text{eval } (\cdot) \left( \text{case } (1, \lambda y. \{\}) \right) \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} d \mapsto^* !\text{cell } d \{\}, \ldots
\]

This is the case:

\[
\begin{align*}
\text{eval } (\cdot) \left( \text{case } (1, \lambda y. \{\}) \right) \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} d \\
& \mapsto \text{eval } (\cdot) \{1, \lambda y. \{\} \} d_1, \text{cont } d_1 \langle (\cdot), \text{case} - \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} \rangle d \\
& \mapsto \text{eval } (\cdot) \lambda y. \{\} d_2, \text{cont } d_2 \langle 1, - \rangle d_1, \text{cont } d_1 \langle (\cdot), \text{case} - \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, \text{cont } d_1 \langle (\cdot), \text{case} - \{\langle \alpha, x \rangle \Rightarrow x(\{\}) \} \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, \text{eval } (d_2/x) x(\{\}) d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, \text{eval } (d_2/x) x d_3, \text{cont } d_3 \langle (d_2/x), (\{\}) \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, !\text{cell } d_3 \langle (\cdot), \lambda y. \{\} \rangle, \text{cont } d_3 \langle (d_2/x), (\{\}) \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, !\text{cell } d_3 \langle (\cdot), \lambda y. \{\} \rangle, \\
& \quad \text{eval } (d_2/x) (\cdot) d_4, \text{cont } d_4 \langle d_3, - \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, !\text{cell } d_3 \langle (\cdot), \lambda y. \{\} \rangle, \\
& \quad !\text{cell } d_4 \langle \cdot \rangle, \text{cont } d_4 \langle d_3, - \rangle d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, !\text{cell } d_3 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_4 \langle \cdot \rangle, \text{eval } (d_4/y) (\cdot) d \\
& \mapsto !\text{cell } d_2 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_1 \langle 1, d_2 \rangle, !\text{cell } d_3 \langle (\cdot), \lambda y. \{\} \rangle, !\text{cell } d_4 \langle \cdot \rangle, !\text{cell } d \langle \cdot \rangle \quad \Box
\end{align*}
\]