Types and Programming Languages (15-814), Fall 2018
Assignment 7: Machinations

Contact: 15-814 Course Staff

Due Tuesday, November 13, 2018, 11:59pm

This assignment is due by 11:59pm on the above date and it must be submitted electronically as a PDF file on Canvas. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID. As before, problems marked "WB" are subject to the whiteboard policy; all other problems must be done individually.

Task 0 (0 points)**.** How long did you spend on this assignment? Please list the questions that you discussed with classmates using the whiteboard policy.

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

$$\mathsf{eval}\ e\ d_n, \mathsf{cont}\ d_n\ k_n\ d_{n-1}, \mathsf{cont}\ d_{n-1}\ k_{n-1}\ d_{n-2}, \ldots, \mathsf{cont}\ d_1\ k_1\ d_0$$

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.

Conspicously 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

$$\mathsf{retn}\ c\ d$$

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 ephmeral 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.

$$\begin{array}{lcl}
\mathsf{!cell}\ d\ c, \mathsf{eval}\ d\ d' & \mapsto & \mathsf{retn}\ c\ d' \\
\mathsf{eval}\ (\lambda x.\, e)\ d & \mapsto & \mathsf{retn}\ (\lambda x.\, e)\ d \\
\mathsf{retn}\ \langle\rangle\ d, \mathsf{cont}\ d\ (\mathbf{case}\ _\ \{\langle\rangle \Rightarrow e'\})\ d' & \mapsto & \mathsf{eval}\ e'\ d'
\end{array}$$

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.

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.

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:

$$\frac{\Delta, \alpha\ \mathit{type}; \Gamma \vdash e : \tau}{\Delta; \Gamma \vdash e : \forall\alpha.\tau} \qquad \frac{\Delta; \Gamma \vdash e : \forall\alpha.\tau \quad \Delta \vdash \sigma\ \mathit{type}}{\Delta; \Gamma \vdash e : [\sigma/\alpha]\tau}$$

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_η machines. In particular, we are concerned with how to evaluate expressions typed by the followin g rules:

$$\frac{\Delta, \alpha\ \mathit{type}; \Gamma \vdash e : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}\ (\text{I-}\forall) \qquad \frac{\Delta; \Gamma \vdash e : \forall\alpha.\tau \quad \Delta \vdash \sigma\ \mathit{type}}{\Delta; \Gamma \vdash e[\sigma] : [\sigma/\alpha]\tau}\ (\text{E-}\forall)$$

$$\frac{\Delta \vdash \sigma \; \textit{type} \quad \Delta; \Gamma \vdash e : [\sigma/\alpha]\tau}{\Delta; \Gamma \vdash \langle \sigma, e \rangle : \exists\, \alpha.\tau} \text{ (I-}\exists\text{)}$$

$$\frac{\Delta; \Gamma \vdash e : \exists\, \alpha.\tau \quad \Delta, \alpha \; \textit{type}; \Gamma, x : \tau \vdash e' : \tau' \quad \Delta \vdash \tau' \; \textit{type}}{\Delta; \Gamma \vdash \textbf{case}\; e\; \{\langle \alpha, x \rangle \Rightarrow e'\} : \tau'} \text{ (E-}\exists\text{)}$$

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:

$$\frac{}{\Lambda\alpha.e \; \textit{val}} \text{ (V-}\forall\text{)} \qquad \frac{e \mapsto e'}{e[\sigma] \mapsto e'[\sigma]} \text{ (C-}\forall\text{)} \qquad \frac{}{(\Lambda\alpha.e)[\sigma] \mapsto [\sigma/\alpha]e} \text{ (R-}\forall\text{)}$$

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

Task 4 (5 points, WB)**.** Why would we be wrong to replace the rule (R-$\forall$) with the following rule?

$$\frac{}{(\Lambda\alpha.e)[\sigma] \mapsto e} \text{ (X-}\forall\text{)}$$

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.

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.

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

$$\frac{k \div \tau_1 \Rightarrow \sigma \quad \cdot; \cdot \vdash e : \tau_1}{(k \rhd e) : \sigma} \text{ (S-E)} \qquad \frac{k \div \tau_1 \Rightarrow \sigma \quad \cdot; \cdot \vdash v : \tau_1 \quad v \; \textit{val}}{(k \lhd v) : \sigma} \text{ (S-R)}$$

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

2.2 S_η 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"[1]. In a more realistic setting, we would erase types before evaluating on the S_η 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 $\mathsf{eval}\ \eta\ e\ d$ and associated dynamics to evaluate terms e typed by (I-$\forall$) and (E-$\forall$). To do so,

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

[1] Type erasure is a translation $|\cdot|$ where, e.g., $|\Lambda x.e| = \lambda\_.|e|$, $|e[\sigma]| = |e|\langle\rangle$, $|\langle\sigma, e\rangle| = |\langle\langle\rangle, e\rangle|$, $|\mathbf{case}\ e\ \{\langle\alpha, x\rangle \Rightarrow e'\}| = \mathbf{case}\ e\ \{\langle\_, x\rangle \Rightarrow e'\}$, $|\lambda x.e| = \lambda x.|e|$, $|e_1 e_2| = |e_1|\,|e_2|$, $|l \cdot e| = l \cdot |e|$, etc.

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

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

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

Again, do not store the types in closures. Rather, directly substitute them for type variables. You are not expected to use ephemeral $\mathsf{retn}\ c\ d$ judgments.

Hint. Your rules should validate

$$\mathsf{eval}\ (\cdot)\ (\mathbf{case}\ \langle \mathbf{1}, \lambda y.\langle\rangle\rangle\ \{\langle \alpha, x\rangle \Rightarrow x\langle\rangle\})\ d \mapsto^* !\mathsf{cell}\ d\ \langle\rangle, \ldots$$