University of Massachusetts Lowell — Comp 3010: Organization of Programming Languages

Assignment 7

Name : Harishwar Reddy Erri

UML ID : 02148304

Collaborators : 1. Shashank Reddy Dokuru

2. Ashish Reddy Kosana

Make sure that the remaining pages of this assignment do not contain any identifying information.

1 References (20 points)

$$\begin{array}{l}\textsf{let } y = \textsf{ref } \lambda x.\, x \textsf{ in} \\ \textsf{let } z = (y := \lambda x.\, !y\ 4) \textsf{ in} \\ !y\ 2\end{array}$$

We will evaluate this program using small-step semantics under call-by-value (CBV) semantics.

1. Initial Expression:

$$\langle \textsf{let } y = \textsf{ref } \lambda x.\, x \textsf{ in let } z = (y := \lambda x.\, !y\ 4) \textsf{ in } !y\ 2, \rangle$$

2. Evaluate let $y =$ ref$\lambda x.x$: - ref $\lambda x.\, x$ creates a reference containing the function $\lambda x.\, x$. - Resulting expression:

$$\langle \textsf{let } z = (y := \lambda x.\, !y\ 4) \textsf{ in } !y\ 2,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

3. Evaluate `let z = (y :=` $\lambda x.!y\,4)$: - Assigns $y := \lambda x.\, !y\,4$, which updates the reference to a new function. - Resulting expression:

$$\langle !y\ 2,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

4. Evaluate $!y$: - Dereferences y, which now contains $\lambda x.\, !y\,4$. - Resulting expression:

$$\langle \lambda x.\, !y\,4\,2,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

5. Apply the function: - Substitute $x = 2$: - Resulting expression:

$$\langle !y\,4,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

6. Evaluate $!y$ again: - Dereference y, which still points to $\lambda x.\, !y\,4$. - Resulting expression:

$$\langle \lambda x.\, !y\,4\,4,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

7. Apply the function again: - Substitute $x = 4$: - Resulting expression:

$$\langle !y\,4,\ \{l_1 \mapsto \lambda x.\, x\}\rangle$$

8. Conclusion: The evaluation will continue indefinitely as each application of $!y$ returns back to the same function $!y\,4$ without reaching a value.

2 Typing Derivation (30 points)

Show the type-checking for the following terms using derivation trees to get credit.

(i)

$$y : \textbf{int} \vdash (\lambda x : \textbf{int}.\, y + 40)\; y : \textbf{int}$$

$$\text{T-App}\;\dfrac{\text{T-Abs}\;\dfrac{\text{T-Add}\;\dfrac{\text{T-Var}\;\dfrac{}{y : int \vdash y : int} \quad \text{T-Int}\;\dfrac{}{y : int \vdash 40 : int}}{y : int \vdash y + 40 : int}}{y : int \vdash \lambda x : int.\, (y + 40) : int \to int} \quad \text{T-Var}\;\dfrac{}{y : int \vdash y : int}}{y : int \vdash (\lambda x : int.\, y + 40)\, y : int}$$

The typing derivation for 2(a) is valid.This results in the entire expression being well-typed.

(ii)

$$\vdash (\lambda x : \textbf{int}.\, x + 40)\; (1 + 2) : \textbf{int}$$

$$\text{T-App}\;\dfrac{\text{T-Abs}\;\dfrac{\text{T-Add}\;\dfrac{\text{T-Var}\;\dfrac{}{\vdash x : int} \quad \text{T-Int}\;\dfrac{}{\vdash 40 : int}}{\vdash x + 40 : int}}{\vdash \lambda x : int.\, (x + 40) : int \to int} \quad \text{T-Add}\;\dfrac{\text{T-Int}\;\dfrac{}{\vdash 1 : int} \quad \text{T-Int}\;\dfrac{}{\vdash 2 : int}}{\vdash 1 + 2 : int}}{\vdash (\lambda x : int.\, x + 40)\, (1 + 2) : int}$$

The typing derivation for 2(b) is not valid because y is not in the context, making it impossible to type-check y + 40. This results in the entire expression being ill-typed.