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

Assignment 7

Name: Ashish Kosana

UML ID: 02148256

Collaborators:

1. SHASHANK DOKURU
2. HARISHWAR REDDY ERRI

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

1 References (20 points)

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

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

Step 1: Allocate a reference l for the lambda function $\lambda x.\, x$.

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

Step 2: Update l to hold the new function $\lambda x.\, !l\ 4$.

$\rightarrow\langle !l\ 2,\ \{l \mapsto \lambda x.\, !l\ 4\}\rangle$

Step 3: Dereference l and apply the resulting function to 2.

$\rightarrow\langle (\lambda x.\, !l\ 4)\ 2,\ \{l \mapsto \lambda x.\, !l\ 4\}\rangle$

Step 4: Apply the function $\lambda x.\, !l\ 4$, leading to another dereference of l.

$\rightarrow\langle !l\ 4,\ \{l \mapsto \lambda x.\, !l\ 4\}\rangle$

Step 5: Repeat the process, which causes an infinite loop.

$\rightarrow\langle (\lambda x.\, !l\ 4)\ 4,\ \{l \mapsto \lambda x.\, !l\ 4\}\rangle$

$\rightarrow\langle !l\ 4,\ \{l \mapsto \lambda x.\, !l\ 4\}\rangle$

$\rightarrow \ldots$

Explanation: This program enters an infinite loop because each dereference of l retrieves a function that again dereferences l. This cycle repeats indefinitely, preventing the program from terminating or evaluating to a value.

2 Typing Derivation (30 points)

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

(i)

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

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

Explanation: The typing derivation shows that the term is well-typed. In the context y : **int**: 1. $y + 40$ is typed as **int** using the 'T-Add' rule. 2. λx : **int**. $y + 40$ is typed as **int** → **int** using the 'T-Abs' rule. 3. $(\lambda x$: **int**. $y + 40)$ y is typed as **int** using the 'T-App' rule.

(ii)

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

This term is not well-typed. Here's the partial typing derivation that shows why it is not well typed:

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Error: } y \text{ not in context}}{x : \textsf{int} \vdash y : \textsf{int}}\ \textsc{T-Var} \qquad \dfrac{}{x : \textsf{int} \vdash 40 : \textsf{int}}\ \textsc{T-Int}}{x : \textsf{int} \vdash y + 40 : \textsf{int}}\ \textsc{T-Add}}{\vdash \lambda x : \textsf{int}.\, y + 40 : \textsf{int} \to \textsf{int}}\ \textsc{T-Abs} \qquad \dfrac{\dfrac{}{\vdash 1 : \textsf{int}}\ \textsc{T-Int} \qquad \dfrac{}{\vdash 2 : \textsf{int}}\ \textsc{T-Int}}{\vdash 1 + 2 : \textsf{int}}\ \textsc{T-Add}}{\vdash (\lambda x : \textsf{int}.\, y + 40)\ (1 + 2) : \textsf{int}}\ \textsc{T-App}$$

Explanation: This term is not well-typed because y is a free variable in the body of the lambda abstraction but is not present in the typing context. The derivation fails when attempting to type y within the lambda abstraction.