Homework 10: Implementing the Environment Model
CIS 352: Programming Languages
Version 1: 28 March 2018

Administrivia

- Turn in the Part II problems in the CIS 352 submissions box. If you trade ideas with another student, document it in your cover sheet.
- Turn in the Part I problems in via Blackboard. Include your:
  (i) source files, (ii) transcripts of test runs, and (iii) cover sheet.

Grading Criteria

- The homework is out of 100 points.
- Running the required tests (for problems 1 and 2: et1,...,et6 and for problem 3: et1,...,et5) suffices for testing.
- Omitting your name(s) in the source code looses you 5 points.

Background

In this assignment you will implement versions of LFP with
- call-by-value evaluation and dynamic scoping,
- call-by-name evaluation and lexical scoping, and
- call-by-name evaluation and dynamic scoping.

Figure 1 gives the big-step evaluation rules for each of these. Additionally, the version of LFP we’ll be working with has two new commands:

1. The return command works pretty much as it does in C.

\[
\begin{align*}
\rho \vdash (e, s) &\Downarrow (v, s') \\
\rho \vdash (\text{return } e, s) &\Downarrow (v, s')
\end{align*}
\]

2. The print command does what you expect — sort of.

\[
\begin{align*}
\rho \vdash (e_i, s_i) &\Downarrow (v_i, s_{i+1}) \quad i = 1, \ldots, n \\
\rho \vdash (\text{print } (e_1, \ldots, e_n), s_1) &\Downarrow (\text{skip}, s_{n+1}) \\
\text{(the values of } e_1, \ldots, e_n \text{ are printed)}
\end{align*}
\]

Part I: Programming Problems: Implementing Versions of LFP

You’ll need the files in http://www.cis.syr.edu/courses/cis352/code/LFP2/; LFP2bs.hs has the call-by-value/lexical-scoping version of LFP.

❗ **Problem 1 (16 points) ❗**

runAll runs all these tests for you.

❗ **Problem 2 (16 points) ❗**
Change the version of LFP of LFP2bs.hs to call-by-name/lexical-scoping.

runAll runs all these tests for you.
Call-by-value, lexical-scoping

\[
\begin{align*}
\rho \vdash (e_1, s) \Downarrow \langle (\lambda x. \tilde{e}_1) \tilde{\rho}_1, s' \rangle \\
\rho \vdash (e_2, s') \Downarrow \langle \tilde{e}_2, s'' \rangle \\
\tilde{\rho}_1 [x \mapsto v_2] \vdash (\tilde{e}_1, s'') \Downarrow \langle v, s''' \rangle \\
\rho \vdash (e_1 e_2, s) \Downarrow \langle v, s''' \rangle \\
\rho \vdash (x, s) \Downarrow \langle v, s \rangle (v = \text{lookup}(\rho, x)) \\
\rho \vdash (\lambda x. e, s) \Downarrow \langle (\lambda x. e) \rho, s \rangle
\end{align*}
\]

Call-by-value, dynamic-scoping

\[
\begin{align*}
\rho \vdash (e_1, s) \Downarrow \langle (\lambda x. \tilde{e}_1)' \tilde{\rho}_1, s' \rangle \\
\rho \vdash (e_2, s') \Downarrow \langle \tilde{e}_2, s'' \rangle \\
\tilde{\rho}_1 [x \mapsto v_2] \vdash (\tilde{e}_1, s'') \Downarrow \langle v, s''' \rangle \\
\rho \vdash (e_1 e_2, s) \Downarrow \langle v, s''' \rangle \\
\rho \vdash (x, s) \Downarrow \langle v, s \rangle (v = \text{lookup}(\rho, x)) \\
\rho \vdash (\lambda x. e, s) \Downarrow \langle (\lambda x. e) \rho, s \rangle
\end{align*}
\]

Call-by-name, lexical-scoping

\[
\begin{align*}
\rho \vdash (e_1, s) \Downarrow \langle (\lambda x. \tilde{e}_1) \tilde{\rho}_1, s' \rangle \\
\rho \vdash (e_2, s') \Downarrow \langle \tilde{e}_2, s'' \rangle \\
\tilde{\rho}_1 [x \mapsto e] \vdash (\tilde{e}_1, s'') \Downarrow \langle e, s''' \rangle \\
\rho \vdash (e_1 e_2, s) \Downarrow \langle e, s''' \rangle \\
\rho \vdash (x, s) \Downarrow \langle e, s \rangle (e = \text{lookup}(\rho, x)) \\
\rho \vdash (\lambda x. e, s) \Downarrow \langle (\lambda x. e) \rho, s \rangle
\end{align*}
\]

Call-by-name, dynamic-scoping

\[
\begin{align*}
\rho \vdash (e_1, s) \Downarrow \langle (\lambda x. \tilde{e}_1)' \tilde{\rho}_1, s' \rangle \\
\rho \vdash (e_2, s') \Downarrow \langle \tilde{e}_2, s'' \rangle \\
\tilde{\rho}_1 [x \mapsto e] \vdash (\tilde{e}_1, s'') \Downarrow \langle e, s''' \rangle \\
\rho \vdash (e_1 e_2, s) \Downarrow \langle e, s''' \rangle \\
\rho \vdash (x, s) \Downarrow \langle e, s \rangle (e = \text{lookup}(\rho, x)) \\
\rho \vdash (\lambda x. e, s) \Downarrow \langle (\lambda x. e) \rho, s \rangle
\end{align*}
\]

Note: I’ve put closures in boxes. Also, the hat in \( \tilde{\rho} \) is just a decoration.

Figure 1: Key operational semantics rules
Problem 3 (16 points)
Change the version of LFP of LFP2bs.hs to call-by-name/dynamic-scoping. Test your program on et1, et2, et3, et4, and et5. runAllbut6 runs all these tests for you.

Part II: Programming Problems: More Monadic IO

Problem 4 (12 points)
Write a Haskell program that takes the name of a text file from the command line, finds all the repeated words in the file, and prints a list of these words and the lines they occur on. By a repeated word, I mean a word that appears twice in succession. For example, on the input file:

hop.txt
WALK. WALK. We like to walk.
WALK. TALK. We like to talk.
HOP. POP. We
We like to hop. We like to hop
hop on top of Pop.
STOP STOP You must not hop on
on Pop.

my program, when run under Unix, does the following:

Post% ./reps hop.txt
"walk" beginning on line 1
"walk" beginning on line 1
"we" beginning on line 3
"hop" beginning on line 4
"stop" beginning on line 6
"on" beginning on line 6

Note that "pop" is not considered a repeated word as the occurrences of "pop" are separated by other words. Also note that the first line has two occurrences of repetitions of "walk": one that the front of the line and the other at the end of the line (with the repeat on the next line). A word here is a string of characters surrounded by white space, which we put in all lowercase and delete any punctuation.

For full credit, your program must be compiled and run (correctly) on hop.txt from a command line. Use the starter file reps.hs from http://www.cis.syr.edu/courses/cis352/code/reps/.

*Hint: Consider the subproblem on taking the input file and producing a list of word occurrences, each paired with its line number. For the hop.txt input, the result would be something like:
[("walk",1), ("walk",1), ("we",1), ("like",1), ("to",1), ("walk",1), ("walk",2), ("talk",2), ("we",2), ..., ("pop",7)]
Part III: Written Problems

For each of the following, figure out by hand what the LFP program prints or returns.

Important: You can use your code to check your work, but working out these by hand will give you some practice for the next quiz.

Problem 5 (8 points)
Consider:

\[
\text{let } x = 1 \\
\text{in let } f = (\lambda z. (x + z)) \\
\text{in let } x = 100 \text{ in } (f 40)
\]

What is returned under:
(a) (4 points) call-by-value/lexical scoping?
(b) (4 points) call-by-value/dynamic scoping?

Problem 6 (8 points)
Consider:

\[
\text{let } x = 10 \\
\text{in let } f = (\lambda y. (x * y)) \\
\text{in let } g = (\lambda x. (f 5)) \\
\text{in print } ((f 3), (g 100))
\]

What is printed under:
(a) (4 points) call-by-value/lexical scoping?
(b) (4 points) call-by-value/dynamic scoping?

Problem 7 (8 points)
Assume location X1 (or ℓ1 if you prefer) starts out with contents 0. Consider:

\[
\text{let } f = \lambda y. \{ X1 := !X1 + 30; \text{ return } y \} \\
\text{in let } y = (f 2) \\
\text{in } \{ X2 := y * y; \text{ return } (!X1+!X2) \}
\]

What is returned under:
(a) (4 points) call-by-value/lexical scoping?
(b) (4 points) call-by-name/lexical scoping?
Problem 8 (8 points) 
Assume X1 starts out with contents 0. Consider:

```
let f = \y. { X1 := !X1 + 1; return y }
  in let g = \z.10
  in let w = (g (f 100)) in (w+!X1)
```

What is returned under
(a) (4 points) call-by-value/lexical scoping?
(b) (4 points) call-by-name/lexical scoping?

Problem 9 (8 points) 
Assume X1 starts out with contents 0. Consider:

```
let tick = 10
  in let tock = \u. { X1 := !X1 + tick; return !X1 }
  in let tick = 100 in (tock (tock (tock 0)))
```

What is returned under:
(a) (2 points) call-by-value/lexical scoping?
(b) (2 points) call-by-value/dynamic scoping?
(c) (2 points) call-by-name/lexical scoping?
(d) (2 points) call-by-name/dynamic scoping?

References


**Sample Problem**

For the following program, what does it return under: (a) call-by-value/lexical-scoping? (b) call-by-value/dynamic-scoping?

```plaintext
let x = 100
  in let f = (λz.(x + z))
  in let x = 20
      in (f 3)
```

**An answer**

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ0:</td>
<td>∅</td>
<td>let x = 100 in ...</td>
</tr>
<tr>
<td>ρ1:</td>
<td>x ↦→ 100</td>
<td>let f = ...</td>
</tr>
<tr>
<td>ρ2:</td>
<td>f ↦→ λz.(x + z) ρ1</td>
<td>let x = 20 in ...</td>
</tr>
<tr>
<td>ρ3:</td>
<td>x ↦→ 20</td>
<td>(f 3)</td>
</tr>
<tr>
<td>ρ4:</td>
<td>z ↦→ 3</td>
<td>x + z  = ρ1(x) + ρ1(z) = 100 + 3 = 103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ0:</td>
<td>∅</td>
<td>let x = 100 in ...</td>
</tr>
<tr>
<td>ρ1:</td>
<td>x ↦→ 100</td>
<td>let f = ...</td>
</tr>
<tr>
<td>ρ2:</td>
<td>f ↦→ λz.(x + z)</td>
<td>let x = 20 in ...</td>
</tr>
<tr>
<td>ρ3:</td>
<td>x ↦→ 20</td>
<td>(f 3)</td>
</tr>
<tr>
<td>ρ4:</td>
<td>z ↦→ 3</td>
<td>x + z  = ρ4(x) + ρ4(z) = 20 + 3 = 23</td>
</tr>
</tbody>
</table>