Homework 6: Big- & small-step semantics for LC

CIS 352: Programming Languages

23 February 2018, Version 2

Typo corrections in red.

Administrivia

- Trade ideas with another student? *Document it* in your source file.
- Turn in the Part 1 problems (on paper) via the CIS 352 box on the 4th floor of SciTech. *Include a cover sheet.*
- Turn in the Part 2 problems via Blackboard. Include *(i)* the source files, *(ii)* the transcripts of test runs, and *(iii)* the cover sheet.

Grading Criteria

- The homework is out of 100 points.
- Each programming problem is $\approx$ 70% correctness and $\approx$ 30% testing.
- Omitting your name(s) in the source code looses you 5 points.

Background

This assignment involves using, extending, and implementing the big-step and small-step operational semantics of the LC language from (Pitts, 2002). Also see the rules reference on page 4 below and (Hennessy, 2014, chapter 3).

Part I: Problems on Paper

Notation: $s_{i,j,k}$ = the state $\{\, \ell_0 \mapsto i, \ell_1 \mapsto j, \ell_2 \mapsto k \,\}$. E.g., $s_{3,24,0}$ = the state $\{\, \ell_0 \mapsto 3, \ell_1 \mapsto 24, \ell_2 \mapsto 0 \,\}$.

Fair warning: Expect questions like Problems 1 and 2 on Quiz 3.

Advice: Do Problems 1 and 2 first by hand (for practice, understandings, and quiz-prep), then after finishing the next problem, use `stepRunA` and `stepRunC` to check (and correct) your work.

❖ ***Problem 1 (12 points)*** ❖

Give a full justification (i.e., proof) of each of the following small-step transitions:

(a) $\langle\, ((!\ell_2 + 3) * (1+4)), s_{1,0,5} \,\rangle \to \langle\, ((5+3) * (1+4)), s_{1,0,5} \,\rangle$

(b) $\langle\, \ell_0 {:=} 11;\ \mathbf{skip},\ s_{5,4,3} \,\rangle \to \langle\, \mathbf{skip};\ \mathbf{skip},\ s_{11,4,3} \,\rangle$

(c) $\langle\, \mathbf{if}\ !\ell_0 = !\ell_1\ \mathbf{then\ skip\ else}\ C_1,\ s_{3,4,5} \,\rangle \to \langle\, \mathbf{if}\ 3 \neq \ell_1\ \mathbf{then\ skip\ else}\ C_1,\ s_{3,4,5} \,\rangle$ where $C_1 = \{\, \ell_0 {:=} !\ell_0;\ \ell_1 {:=} !\ell_2 - 1 \,\}$.

❖ ***Problem 2 (8 points)*** ❖

For (a) and (b) below, give the complete transition sequence starting from the configuration.

(a) $\langle\, ((!\ell_2 + 3) * (1+4)), s_{1,0,5} \,\rangle$

(b) $\langle\, \mathbf{if}\ !\ell_0 \neq !\ell_1\ \mathbf{then}\ \{\, \ell_0 {:=} !\ell_0;\ \ell_1 {:=} !\ell_2 - 1 \,\}\ \mathbf{else\ skip},\ s_{3,4,5} \,\rangle$

Do not show the justification of each step, but, of course, each step must be justifiable from the small-step rules.

❖ ***Problem 3 (20 points)*** ❖

Suppose we add a new command to LC:

$\textbf{do } C \textbf{ whilst } B$

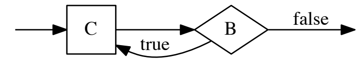

Figure 1: The do-whilst flow graph.

which is a version of the do-while loop from C, Java, etc. See Figure 1 for its flow graph. Formally:

Figure 2: A big-step semantics for `do-whilst`-commands

$$\Downarrow\text{-}\mathit{DoWhilst}_1\!: \quad \frac{\langle C, s\rangle \Downarrow \langle \textbf{skip}, s'\rangle \quad \langle B, s'\rangle \Downarrow \langle \text{false}, s''\rangle}{\langle \textbf{do } C \textbf{ whilst } B, s''\rangle \Downarrow \langle \textbf{skip}, s''\rangle}$$

$$\Downarrow\text{-}\mathit{DoWhilst}_2\!: \quad \frac{\langle C, s\rangle \Downarrow \langle \textbf{skip}, s'\rangle \quad \langle B, s'\rangle \Downarrow \langle \text{true}, s''\rangle \quad \langle \textbf{do } C \textbf{ whilst } B, s''\rangle \Downarrow \langle \textbf{skip}, s'''\rangle}{\langle \textbf{do } C \textbf{ whilst } B, s''\rangle \Downarrow \langle \textbf{skip}, s'''\rangle}$$

(a) ***(10 points)*** Give small-step transition rules for the `do-whilst`-command.

(b) ***(10 points)*** Give the full derivation of the small-step transition relation starting from: $\langle\, \textbf{do } \ell := !\ell - 1 \textbf{ whilst } !\ell > 0,\ [\ell \mapsto 2]\,\rangle$

Part II: Implementation Problems

The CEK interpreter for LC will serve as our reference implementation of LC (to check against the two interpreters you will implement). We'll make use of the following files:[1]

`LC.hs`	contains the abstract syntax of LC, utilities (e.g., `aApply`), and QuickCheck generators.
`State.hs`	contains the data structure for LC-states.
`LCParser.hs`	contains a parser for LC.
`LCCEK.hs`	contains a CEK interpreter (from class).
`LCbs.hs`	contains a big-step interpreter.
`LCss.hs`	contains a start at a small-step interpreter.

[1] From: `http://www.cis.syr.edu/courses/cis352/code/LC/`.
You will be modifying just `LCbs.hs` and `LCss.hs`.

LC phrases These are of three sorts: *(A)* arithmetic expressions, *(B)* boolean expressions, and *(C)* commands.

The big-step interpreter deals with these different sorts by having three different evaluators: `evalA`, `evalB`, and `evalC` for, respectively, **A**rithmetic expressions, **B**oolean expressions, and **C**ommands.

The small-step interpreter (and the CEK machine) deal with these three sorts of LC phrases via the data-type:

```
data Phrase = A AExp | B BExp | C Command
```

You can view `A`, `B`, and `C` as interpreter-specific tags. Whenever LC abstract syntax occurs in the interpreter, it *must* be tagged.

The `LCCEK.hs` file has lots of examples.

❖ ***Problem 4 (40 points)*** ❖

(a) ***(20 Points)*** Complete the small-step interpreter in `LCss.hs`.[2]

(b) ***(5 Points)*** TESTING 1. Try: `(stepRunC' fact state4)` on your small-step interpreter; it runs a LC command for computing 4! and its final state should be:

```
s[0]=4   s[1]=24   s[2]=0   s[3]=0   s[4]=0
```

(c) ***(5 Points)*** TESTING 2. Try: `(quickCheck ss_prop)`, which runs 100 random LC commands (sans **while**'s) on your small-step interpreter and on the CEK interpreter and compares the results. Your code should pass all 100 tests.

(d) ***(10 Points)*** TESTING 3. Devise and run your own set of tests to make sure your implementation of **while**-loops is correct.

[2] Just like in the big-step case, premises in rules show up as recursive-calls in the `where`-clauses in the implementation.

❖ ***Problem 5 (20 points)*** ❖

This is the continuation of Problem 3.

`LC.hs` and `LCParser.hs` already can handle `do-whilst`-commands and `LCbs.hs` has a "fix me" stub for your code.

(a) ***(10 points)*** Extend the LC big-step evaluator of `LCbs.hs` to handle `do-whilst`-commands according to the rules given in Figure 2.

(b) ***(10 points)*** Come up with some convincing tests that your implementation for part (a) is correct.

Challenge Problems

✪ ***Challenge Problem 1: (10 points).*** ✪

Do Exercise 3.4.2 on page 34 in (Pitts, 2002). This is not than hard, but you need to think things through *carefully!*

✪ ***Challenge Problem 2: (10 points).*** ✪

Modify the big-step interpreter to implement the rules of the previous problems. Come up with some convincing tests that your implementation is correct.

References

Matthew Hennessy. Semantics of programming languages (CS3017) Course notes 2014-2015. Technical report, Trinity College Dublin, 2014. URL https://www.scss.tcd.ie/Matthew.Hennessy/splexternal2015/LectureNotes/Notes14%20copy.pdf.

Andrew Pitts. Lecture notes on semantics of programming languages: For part IB of the Cambridge CS tripos. Technical report, University of Cambridge, 2002. URL http://www.cl.cam.ac.uk/teaching/2001/Semantics/.

Rules reference

LC: Big-steps rules

$$\Downarrow\text{-}\circledast\text{:}\ \frac{\langle E_1, s\rangle \Downarrow \langle n_1, s'\rangle \quad \langle E_2, s'\rangle \Downarrow \langle n_2, s''\rangle}{\langle E_1 \circledast E_2, s\rangle \Downarrow \langle c, s''\rangle}\ (c = n_1 \circledast n_2)$$

$$\Downarrow\text{-}Con\text{:}\ \frac{}{\langle c, s\rangle \Downarrow \langle c, s\rangle}\ (c \in \mathbb{Z} \cup \mathbb{B})$$

$$\Downarrow\text{-}Loc\text{:}\ \frac{}{\langle !\ell, s\rangle \Downarrow \langle s(\ell), s\rangle}\ (\ell \in dom(s))$$

$$\Downarrow\text{-}Skip\text{:}\ \frac{}{\langle \textbf{skip}, s\rangle \Downarrow \langle \textbf{skip}, s\rangle}$$

$$\Downarrow\text{-}Set\text{:}\ \frac{\langle E, s\rangle \Downarrow \langle n, s'\rangle}{\langle \ell \leftarrow E, s\rangle \Downarrow \langle \textbf{skip}, s'[\ell \mapsto n]\rangle}$$

$$\Downarrow\text{-}If_1\text{:}\ \frac{\langle B, s\rangle \Downarrow \langle \text{true}, s'\rangle \quad \langle C_1, s'\rangle \Downarrow \langle \textbf{skip}, s''\rangle}{\langle \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s\rangle \Downarrow \langle \textbf{skip}, s''\rangle}$$

$$\Downarrow\text{-}If_2\text{:}\ \frac{\langle B, s\rangle \Downarrow \langle \text{false}, s'\rangle \quad \langle C_2, s'\rangle \Downarrow \langle \textbf{skip}, s''\rangle}{\langle \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s\rangle \Downarrow \langle \textbf{skip}, s''\rangle}$$

$$\Downarrow\text{-}Seq\text{:}\ \frac{\langle C_1, s\rangle \Downarrow \langle \textbf{skip}, s'\rangle \quad \langle C_2, s'\rangle \Downarrow \langle \textbf{skip}, s''\rangle}{\langle C_1; C_2, s\rangle \Downarrow \langle \textbf{skip}, s''\rangle}$$

$$\Downarrow\text{-}While_2\text{:}\ \frac{\langle B, s\rangle \Downarrow \langle \text{false}, s'\rangle}{\langle \textbf{while}\ B\ \textbf{do}\ C, s\rangle \Downarrow \langle \textbf{skip}, s'\rangle}$$

$$\Downarrow\text{-}While_1\text{:}\ \frac{\langle B, s\rangle \Downarrow \langle \text{true}, s'\rangle \quad \langle C, s'\rangle \Downarrow \langle \textbf{skip}, s''\rangle \quad \langle \textbf{while}\ B\ \textbf{do}\ C, s''\rangle \Downarrow \langle \textbf{skip}, s'''\rangle}{\langle \textbf{while}\ B\ \textbf{do}\ C, s\rangle \Downarrow \langle \textbf{skip}, s'''\rangle}$$

LC: Small-steps rules

$$\rightarrow\text{-}op1\text{:}\ \frac{\langle E_1, s\rangle \rightarrow \langle E_1', s'\rangle}{\langle E_1 \circledast E_2, s\rangle \rightarrow \langle E_1' \circledast E_2, s'\rangle}$$

$$\rightarrow\text{-}op2\text{:}\ \frac{\langle E_2, s\rangle \rightarrow \langle E_2', s'\rangle}{\langle n_1 \circledast E_2, s\rangle \rightarrow \langle n_1 \circledast E_2', s'\rangle}$$

$$\rightarrow\text{-}op3\text{:}\ \frac{}{\langle n_1 \circledast n_2, s\rangle \rightarrow \langle c, s\rangle}\ (c = n_1 \circledast n_2)$$

$$\rightarrow\text{-}loc\text{:}\ \frac{}{\langle !\ell, s\rangle \rightarrow \langle s(\ell), s\rangle}\ (\ell \in dom(s))$$

$$\rightarrow\text{-}set1\text{:}\ \frac{\langle E, s\rangle \rightarrow \langle E', s'\rangle}{\langle \ell \leftarrow E, s\rangle \rightarrow \langle \ell \leftarrow E', s'\rangle}$$

$$\rightarrow\text{-}set2\text{:}\ \frac{}{\langle \ell \leftarrow n, s\rangle \rightarrow \langle \textbf{skip}, s[\ell \mapsto n]\rangle}$$

$$\rightarrow\text{-}seq1\text{:}\ \frac{\langle C_1, s\rangle \rightarrow \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \rightarrow \langle C_1'; C_2, s'\rangle}$$

$$\rightarrow\text{-}seq2\text{:}\ \frac{}{\langle \textbf{skip}; C, s\rangle \rightarrow \langle C, s\rangle}$$

$$\rightarrow\text{-}if1\text{:}\ \frac{\langle B, s\rangle \rightarrow \langle B', s'\rangle}{\langle \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s\rangle \rightarrow \langle \textbf{if}\ B'\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s'\rangle}$$

$$\rightarrow\text{-}if2\text{:}\ \frac{}{\langle \textbf{if}\ \text{true}\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s\rangle \rightarrow \langle C_1, s\rangle}$$

$$\rightarrow\text{-}if3\text{:}\ \frac{}{\langle \textbf{if}\ \text{false}\ \textbf{then}\ C_1\ \textbf{else}\ C_2, s\rangle \rightarrow \langle C_2, s\rangle}$$

$$\rightarrow\text{-}while\text{:}\ \frac{}{\langle \textbf{while}\ B\ \textbf{do}\ C, s\rangle \rightarrow \langle \textbf{if}\ B\ \textbf{then}\ \{C;\ \textbf{while}\ B\ \textbf{do}\ C\ \}\ \textbf{else}\ \textbf{skip}, s\rangle}$$

Testing Tools

Tools for `LCbs.hs`

- At the bottom of `LCbs.hs` you will find some sample integer expressions (`ie0`, ..., `ie3`), boolean expressions (`be0`, ..., `be3`), and commands (`cmd0`, ..., `cmd8`).
- Evaluate these use `runA` (for arithmetic expressions), `runB` (for boolean expressions), and `runC` (for commands). Each of these will return the final configuration of the evaluation. For example:

```
ghci> ie2
"val(x1)+2"

ghci> runA ie2 state0
(2,fromList [(0,1),(1,0),(2,3)])

ghci> cmd3
"{ x0 := (-2); x3 := ((-3)+val(x1)) }"

ghci> runC cmd3 state0
(skip,fromList [(0,-2),(1,0),(2,3),(3,-3)])
```

Tools for `LCss.hs`

- At the bottom of `LCss.hs` you will find our friends `ie0`, ..., `cmd8`.
- To run a command and just get the final configuration, use `run'`. For example:

```
ghci> cmd3
"{ x0 := (-2); x3 := ((-3)+val(x1)) }"
ghci> run' cmd3 state0
Step:  -1
 C skip
 s[0]=-2 s[1]=0 s[2]=3 s[3]=-3
```

Confession: The "`Step:  -1`" thing in the output of `run'` is me being lazy and reusing a function from the step-by-step output code.

- To get a trace of the computation, use `stepRunA'`, `stepRunB'`, and `stepRunC'`. For example:

```
ghci> stepRunC' cmd3 state0
Step:  0
 C { x0 := (-2); x3 := ((-3)+val(x1)) }
 s[0]=1 s[1]=0 s[2]=3 <tap return>
Step:  1
 C { skip; x3 := ((-3)+val(x1)) }
 s[0]=-2 s[1]=0 s[2]=3 <tap return>
Step:  2
 C x3 := ((-3)+val(x1))
 s[0]=-2 s[1]=0 s[2]=3 <tap return>
Step:  3
 C x3 := ((-3)+0)
 s[0]=-2 s[1]=0 s[2]=3 <tap return>
Step:  4
 C x3 := (-3)
 s[0]=-2 s[1]=0 s[2]=3 <tap return>
Step:  5
 C skip
 s[0]=-2 s[1]=0 s[2]=3 s[3]=-3 <tap return>
ghci>
```