

# Language Semantics and Implementation Exercises

Ohad Kammar (ohad.kammar@ed.ac.uk)

and

Alex Simpson (als@ed.ac.uk)

March 4, 2010

1. Let  $P_1$  be the following LC program:

$$\mathbf{while} \ !\ell_0 > 0 \ \mathbf{do} \ \ell_1 := !\ell_1 + 1 ; \\ \ell_0 := !\ell_0 - 1$$

Consider  $P_1$ 's SMC machine semantics.

Find a non-iterative program  $P_2$  such that for all memory maps  $s_1, s_2$  defined for  $\ell_0$  and  $\ell_1$ , the following holds:

$$\langle P_1 \cdot \mathbf{nil}, \mathbf{nil}, s_1 \rangle \longrightarrow^* \langle \mathbf{nil}, \mathbf{nil}, s_2 \rangle \\ \Downarrow \\ \langle P_2 \cdot \mathbf{nil}, \mathbf{nil}, s_1 \rangle \longrightarrow^* \langle \mathbf{nil}, \mathbf{nil}, s_2 \rangle$$

Prove your answer.

You can use determinacy of the SMC machine in your proof:

For every LC program  $P$ , if  $\langle P \cdot \mathbf{nil}, \mathbf{nil}, s \rangle \longrightarrow^* \langle \mathbf{nil}, \mathbf{nil}, s_1 \rangle$   
and  $\langle P \cdot \mathbf{nil}, \mathbf{nil}, s \rangle \longrightarrow^* \langle \mathbf{nil}, \mathbf{nil}, s_2 \rangle$  then  $s_1 = s_2$ .

2. Show how LC integer expressions  $E$  are an inductively defined subset.
3. Let  $\longrightarrow$  be any relation over some set  $A$ . Show how the reflexive-transitive closure  $\longrightarrow^*$  is an inductively defined subset.
4. Let  $\rightsquigarrow$  be the semantics for simple expressions (slide 29 in lectures 3–4). Prove:

(a) For all derivations  $e \rightsquigarrow^* e'$  and for all simple expressions  $\hat{e}$ , integers  $n$  and integer operations  $iop \in \{+, \times\}$ :

- $e \ iop \ \hat{e} \rightsquigarrow^* e' \ iop \ \hat{e}$
- $n \ iop \ e \rightsquigarrow^* n \ iop \ e'$

- (b) For  $iop \in \{+, \times\}$ , if  $e_1 \rightsquigarrow^* n_1$  and  $e_2 \rightsquigarrow^* n_2$  then  $e_1 \text{ iop } e_2 \rightsquigarrow^* n_3$ , where  $n_3 = n_1 \text{ iop } n_2$ .
- (c) For all expressions  $e$ , there exists an  $n \in \mathbb{Z}$  such that  $e \rightsquigarrow^* n$ .
- (d) For all simple expressions  $e$  and  $n \in \mathbb{Z}$ , we have  $e \rightsquigarrow^* n$  if and only if for all control stacks  $C$ , result stacks  $S$  and memory maps  $M$ ,

$$\langle e \cdot C, M, s \rangle \longrightarrow^* \langle C, n \cdot M, s \rangle$$

5. Consider the rule for **while** in the small-step operational semantics for LC.

What is the main difference between this rule and the other rules?

Can you rephrase it in a manner similar to the others?

If not, what are the main difficulties?

6. Give small-step operational semantics to the following extensions of LC:
- (a) The language LC+E, that has two additional command constructs for exceptions:

$$C ::= \dots \mid \mathbf{handle} \ C \ \mathbf{with} \ C \mid \mathbf{raise}$$

These constructs are meant to be a simple exception mechanism, with only one type of exception.

- (b) (Claus Brabrand, 2005) The language LC+T, which extends LC with *transactions*:

$$C ::= \dots \mid \mathbf{transaction} \ C \ \mathbf{unroll} \ \mathbf{if} \ B$$

Informally, the command **transaction**  $C$  **unroll** **if**  $B$  first performs  $C$  and then checks  $B$ . If  $B$  is **true**, all of  $C$ 's side effects are rolled back, or undone, as if the command  $C$  did not take place.

7. (LSI 2009, exercise sheet 1, question 2) This question concerns the LC transition system semantics (the small-step).

- (a) Using rule induction prove the property that expressions of LC are side-effect free as formalised on Slide 23 (of Pitts's notes).
- (b) Suppose LC is extended with a new expression form which can have side effects,  $C \ \mathbf{result} \ E$ . The idea is that  $C$  is first executed and then the result  $E$  is evaluated.
- (c) Extend the definition of the LC transition relation to include this new expression form.
- (d) Let  $s$  be the state  $\{l' \mapsto 4\}$ . Write down the full sequence of transitions starting from the configuration

$$\langle l := (!l' + ((l' := !l' + 1 \ \mathbf{result} \ !l') + !l')), s \rangle$$

that are derivable using the definition of the transition relation.

8. (based on LSI resit exam 2006, questions 1(b)–(d)) Suppose we extend LC to a new language,  $LC^+$ , by adding a new command **for**  $E$  **do**  $C$ , which is executed at a given state by first evaluating  $E$  to a value  $n \geq 0$  and then executing  $C$   $n$  times.

- (a) Give evaluation rules for this new kind of command.  
 (b) Give the full derivation of the  $LC^+$  evaluation relation starting from the configuration

$$\langle \mathbf{for} \ !\ell' \ \mathbf{do} \ \ell := !\ell + 1, \{\ell \mapsto 1, \ell' \mapsto 2\} \rangle$$

- (c) It turns out that  $LC^+$  integer expressions are side-effect free. Prove this statement using rule induction.  
 (d) Given a command  $C$  and an expression  $E$ , give an  $LC^{\text{loc}}$  command  $C'$  such that:

$$\langle \mathbf{for} \ E \ \mathbf{do} \ C, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle \iff \langle C', s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$$

9. Evaluate  $\langle M, [] \rangle$ , where  $M$  is given by

```

begin loc  $\ell_0 := 1$  ;
  begin loc  $\ell_0 := 2$  ;
     $\ell_1 := !\ell_0$ 
  end
end

```

10. In the rule for blocks, we had three side conditions:

- (a)  $\ell' \notin \text{dom}(s')$ .  
 (b)  $\ell' \notin \text{dom}(s'')$ .  
 (c)  $\ell'$  doesn't occur in  $C$ .

Find three examples, each violating only one of the above conditions, for which using the derivation for blocks yields undesired semantics.

In other words, show that each of these conditions is necessary.

11. (adapted from LSI 2009, exercise 1) Prove or refute the following equivalences:

- (a)  $C_1 ; (\mathbf{if} \ B \ \mathbf{then} \ C_2 \ \mathbf{else} \ C_3) \cong (\mathbf{if} \ B \ \mathbf{then} \ C_1 ; C_2 \ \mathbf{else} \ C_1 ; C_3)$   
 (b)  $\mathbf{begin} \ \mathbf{loc} \ \ell_0 := E ; \ell_1 := !\ell_0 \ \mathbf{end} \cong \ell_1 := E$   
 (c)  $\mathbf{while} \ !\ell_0 > 0 \ \mathbf{do} \ (\ell_1 := !\ell_1 + 1 ; \ell_0 := !\ell_0 - 1)$   
 $\cong \mathbf{if} \ !\ell_0 > 0 \ \mathbf{then} \ \ell_1 := !\ell_1 + !\ell_0 ; \ell_0 := 0 \ \mathbf{else} \ \mathbf{skip}$

12. (a) Give an inductive definition of contexts,  $C[ ]$ , of LC.

- (b) Divide the contexts into “types”, according to their hole and the overall phrase.
  - (c) Prove that if  $C[ ]$  is a context whose hole is of the same type as  $P$ , then  $C[P]$  is a valid LC phrase.
  - (d) Let  $C_1[ ]$  be a context whose type is the same as the type of the hole of the context  $C_2[ ]$ . Show that  $C_2[C_1[ ]]$  is a context.
13. Let us define a new notion of equivalence: two phrases of the same type are *contextually equivalent*,  $P_1 \cong_{\text{ctx}} P_2$ , if and only if, for every context  $C[ ]$  whose hole is of the same type as  $P_i$ , and for every terminal configuration  $\langle V, s \rangle$ ,

$$\langle C[P_1], [] \rangle \Downarrow \langle V, s \rangle \iff \langle C[P_2], [] \rangle \Downarrow \langle V, s \rangle$$

- (a) Show that  $\cong_{\text{ctx}}$  satisfies the basic properties of equality.
- (b) Prove that if  $P_1 \cong P_2$  then  $P_1 \cong_{\text{ctx}} P_2$ .
- (c) Prove that for every phrase  $P$ , memory map  $s$ , location  $\ell$ , integer  $n$ , and terminal configuration  $\langle V, s' \rangle$ ,

$$\langle P, s[\ell \mapsto n] \rangle \Downarrow \langle V, s' \rangle \iff \langle \ell := n ; P, s \rangle \Downarrow \langle V, s' \rangle$$

- (d) Prove that if  $P_1 \cong_{\text{ctx}} P_2$  then  $P_1 \cong P_2$ .

14. (a) Consider the LC program,  $P$ :

$$z := !x ; y := 0 ; \mathbf{while} \ !z > 1 \ \mathbf{do} \ (z := z \div 2 ; y := !y + 1)$$

Here location  $x$  is the input,  $y$  is the output and  $z$  is a counter.  
Prove the Hoare triple:

$$\{!x > 0\} P \{2^{!y} \leq !x \ \text{and} \ !x < 2^{!y+1}\}$$

Hint, use the invariant:

$$!z > 0 \ \text{and} \ 2^{!y} \times !z \leq !x \ \text{and} \ !x < 2^{!y+1} \times !z$$

- (b) Prove rigorously (using evaluation semantics) that  $P$  terminates for all integer input values of  $!x$ . (Hint, first prove the while loop terminates for  $!z \leq 1$ ; then use complete induction to prove for  $!z \geq 2$ .)
- (c) Find a Hoare triple that correctly specifies the behaviour of  $P$  on all integer inputs  $!x$  (not just on  $!x > 0$ ). (No need to prove the Hoare triple.)

15. Give formal evaluations in LLC of:

- (a) **let**  $x = !\ell \times !\ell'$  **in**  $(\ell := x ; \ell' := x)$
- (b) **let**  $c = \ell := !\ell + !\ell'$  **in**  $(c ; c)$

starting in the state  $[\ell \mapsto 2, \ell' \mapsto 3]$ . For each of (a) and (b) do this twice: once using call-by-name and once using call-by-value.

Which of CBN and CBV seems more reasonable in each case?

16. Consider the definition of substitution  $P'[P/x]$  in the slides for lectures 11 and 12.

Give examples to show what can go wrong in the last clause if  $z$  occurs free either in  $P$  or in  $P_2$ .

Why is it alright for  $z$  to occur free in  $P_1$ ?

(compare to question 10)

17. (based on LSI exam 2009, question 3) This question concerns the functional/imperative language LFP from the lectures.

- (a) The general format of the relation used in the LFP evaluation semantics is

$$\langle M, s \rangle \Downarrow \langle V, s' \rangle$$

Briefly explain what each of  $M$ ,  $s$ ,  $V$  and  $s'$  are. In doing so, you may assume that the grammar of LFP expressions is known, but you should explain in detail the different possibilities for  $V$ .

- (b) The language  $\text{LFP}^+$  is obtained by adding the recursion construct

$$\mathbf{rec } x.M$$

to LFP. Give the evaluation rule for the recursion construct.

- (c) Describe the format of the general typing relation for the language  $\text{LFP}^+$ . Illustrate your answer by giving the typing rule for the recursion construct  $\mathbf{rec } x.M$ .

- (d) Give a full typing derivation for the expression  $F$ :

$$\mathbf{rec } f.(\lambda x.\mathbf{if } x \leq 1 \mathbf{ then } x \mathbf{ else } f(x - 1) + f(x - 2))$$

- (e) Give a full derivation of the  $\text{LFP}^+$  evaluation of the configuration  $\langle F(2), s \rangle$ , where  $s$  is any state.