Formal Semantics of Programming Languages Exercise 3 (June 30)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

June 1, 2017

The exercise is to be submitted by the deadline stated above as a report with a decent cover page (title of the course, your name, Matrikelnummer, email address) in one of the following forms:

1. either as a single PDF file uploaded in Moodle (no emails, please), or
2. as a stapled paper report handed out to me (in class or in my mailbox).

Exercise 3: For Loops

1. Take an imperative programming language with a loop command

   $C ::= \ldots \mid \mathbf{for}(C_1; B; C_2)\ C_3$

   where the evaluation of a Boolean expression B may alter the store (see the previous exercises). The semantics of the **for** loop is analogous to that one of C/Java-like languages: first, C_1 is executed, then B is evaluated. If the result is "true", C_3 and C_2 are executed and then B is evaluated again.

   a) Give an operational semantics for this language assuming a judgment $\langle B, s\rangle \rightarrow \langle t, s'\rangle$ for the evaluation of boolean expression B in store s yielding a truth value t and a store s'.

   b) Give a denotational semantics for this language assuming a valuation function $\mathbf{B}$: $BoolExp \rightarrow Store \rightarrow (Truth \times Store)$. Please note that the evaluation of a command may not terminate.

   c) State for both commands and boolean expressions formally the equivalence of the operational semantics and the denotational semantics (you need not prove that statement).

2. Take an imperative programming language with a loop command

   $C ::= \ldots \mid \mathbf{for}\ I\ \mathbf{from}\ E_1\ \mathbf{to}\ E_2\ \mathbf{by}\ E_3\ \mathbf{do}\ C$

   where the evaluation of an expression E yields an integer and *cannot* alter the store. The **for** loop iteratively executes the loop body C with the value of variable I set subsequently to $i_1, i_1 + i_3, i_1 + 2 \cdot i_3, \ldots, i_1 + k \cdot i_3$ where i_1, i_2, i_3 are the values of E_1, E_2, E_3, respectively, and $i_1 + k \cdot i_3$ is the largest value less than or equal i_2 (if $i_1 > i_2$, the loop is not executed at all). After the execution of the loop, I has the same value that it had before the execution of the loop (i.e., I is only temporarily assigned).

   a) Give an operational semantics for this language assuming a judgment $\langle E, s\rangle \rightarrow i$ for the evaluation of expression E in store s yielding an integer i.

   b) Give a denotational semantics for this language assuming a valuation function $\mathbf{E}$: $Expression \rightarrow Store \rightarrow \mathbb{Z}$. Please note that the evaluation of a command may not terminate (since the language also contains general "while" loops).

   c) State for both commands and expressions formally the equivalence of the operational semantics and the denotational semantics (you need not prove that statement).

3. **Bonus 15% (Optional):** Apparently, for the second form of the **for** loop non-termination cannot arise from the execution of the **for** loop itself (but only from the execution of the loop body C). Therefore, for defining the semantics of the **for** loop it is not necessary to

resort to least fixed point semantics, but it suffices to use *primitive recursion*: any function $f : \mathbb{N} \times \ldots \to \ldots$ defined in the form

$$f(n, \ldots) := \begin{cases} \ldots & \text{if } n = 0 \\ \ldots f(n-1, \ldots) \ldots & \text{else} \end{cases}$$

(where the only recursive call is the single call $f(n-1, \ldots)$ denoted above) is uniquely defined for any argument $n \in \mathbb{N}$.

Define the semantics of the **for** loop by primitive recursion using as n the number of iterations of the loop.