ETH zürich

P. Müller and D. Basin

Formal Methods and Functional Programming

Optional Exercises 12: Small-Step Semantics

The solutions of the assignments can be found at the end of the file.

Assignment 3 (k-Step Execution Extension)

Task. Prove the following statement:

$$\forall s_1, s_2, \sigma, \sigma', k \cdot \langle s_1, \sigma\rangle \rightarrow_1^k \sigma' \implies \langle s_1; s_2, \sigma\rangle \rightarrow_1^k \langle s_2, \sigma'\rangle$$

Assignment 4 (Adding `revert-if` Statements to IMP)

In this assignment, we add a statement to **IMP** that is related to a very simple form of transaction management or conflict resolution as used, for instance, in databases: The statement

$$\texttt{revert}\ s\ \texttt{if}\ b$$

executes the statement s; but if the boolean expression b is true after the execution of s, all effects of executing s are reverted. For example, the statement `revert x := 0 if x = 0` has no effect while the statement `revert x := 0 if x > 0` is equivalent to `x := 0`.

Task. Provide derivation rules in the small-step semantics for the `revert-if` statement.

Hint: You might want to consider the definition of states.

Hint: Ideally, your solution should support the possibility of `revert-if` statements being nested. But you might find it easier to consider the non-nested case first.

Note: This exercise is a bit more involved.

Solution of assignment 3 (k-Step Execution Extension)

Let

$$P(k) \equiv \forall s_1, s_2, \sigma, \sigma' \cdot \langle s_1, \sigma\rangle \to_1^k \sigma' \implies \langle s_1; s_2, \sigma\rangle \to_1^k \langle s_2, \sigma'\rangle.$$

We prove $\forall k \cdot P(k)$ by strong induction on the length k of the derivation sequence. Thus, for some arbitrary k, we get as induction hypothesis $\forall k' < k \cdot P(k')$, and need to prove $P(k)$.

Let s_1, s_2, σ and σ' be arbitrary. In order to prove the implication, we assume $\langle s_1, \sigma\rangle \to_1^k \sigma'$ (A1) and seek to show $\langle s_1; s_2, \sigma\rangle \to_1^k \langle s_2, \sigma'\rangle$ (PO).

- **Case** $k = 0$: This contradicts our assumption (A1) because there doesn't exist a zero-length derivation sequence from a configuration into a final state.
- **Case** $k = 1$: To show: $\langle s_1; s_2, \sigma\rangle \to_1^1 \langle s_2, \sigma'\rangle$

  Our assumption (A1) gives us that there is some T_1 such that *root*$(T_1) \equiv \langle s_1, \sigma\rangle \to_1 \sigma'$. Now we can construct a derivation tree to justify $\langle s_1; s_2, \sigma\rangle \to_1^1 \langle s_2, \sigma'\rangle$:

$$\frac{\begin{array}{c} T_1 \\ \langle s_1, \sigma\rangle \to_1 \sigma' \end{array}}{\langle s_1; s_2, \sigma\rangle \to_1 \langle s_2, \sigma'\rangle}\ (\textsc{Seq1}_{SOS})$$

- **Case** $k \geq 2$: To show: $\langle s_1; s_2, \sigma\rangle \to_1^k \langle s_2, \sigma'\rangle$.

  From (A1) and the fact that $k \geq 2$, we have that there is some intermediate configuration $\langle s'', \sigma''\rangle$, such that

$$\langle s_1, \sigma\rangle \to_1^1 \langle s'', \sigma''\rangle \to_1^{k-1} \sigma' \tag{A2}$$

  We apply our induction hypothesis to the $(k-1)$-long derivation sequence of (A2), and we get

$$\langle s''; s_2\ , \sigma''\rangle \to_1^{k-1} \langle s_2, \sigma'\rangle \tag{S1}$$

  Thus, what remains to be proven is the first transition of our goal sequence:

$$\langle s_1; s_2, \sigma\rangle \to_1^1 \langle s''; s_2\ , \sigma''\rangle$$

  From the first transition of (A2), we can conclude that there has to be some tree T_2 with *root*$(T_2) \equiv \langle s_1, \sigma\rangle \to_1 \langle s'', \sigma''\rangle$. Now we can construct a derivation tree to justify $\langle s_1; s_2, \sigma\rangle \to_1^1 \langle s''; s_2, \sigma''\rangle$:

$$\frac{\begin{array}{c} T_2 \\ \langle s_1, \sigma\rangle \to_1 \langle s'', \sigma''\rangle \end{array}}{\langle s_1; s_2, \sigma\rangle \to_1 \langle s''; s_2, \sigma''\rangle}\ (\textsc{Seq2}_{SOS})$$

Concatenating this with (S1) yields our goal sequence and concludes the proof.

Solution of assignment 4 (Adding revert-if statement to IMP)

In order to maintain a backup of the state that existed before statement s in `revert` s `if` b `end` had been executed, we extend our state to be a list of regular states:

$$State' = [State]$$

The extended state is used as a stack of backups. The top-most state is the current state, i.e., the one that is used to evaluate expressions and that is changed by assigments, and the other states are used to rollback actions, if necessary. We decompose a state by Haskell-like pattern matching, $\sigma' = \sigma : \sigma_s$ means that the extended state σ' consists of a head state σ and tail states σ_s.

The first new rule pushes the current state as a backup onto the state stack, and inserts a marker statement that is used to conditionally trigger a rollback.

$$\frac{}{\langle \texttt{revert}\ \ s\ \ \texttt{if}\ \ b, \sigma : \sigma_s\rangle \to_1 \langle s\texttt{;rollback-if}\ \ b, \sigma : \sigma : \sigma_s\rangle}\ (\textsc{Rev}_{SOS}\ )$$

The semantics of the marker statement are captured by the next two rules. The first one performs a rollback by discarding the head state, whereas the second one keeps the head state and instead removes the backup state. Note that `rollback-if` is assumed to not occur in the source programs, i.e., it is only inserted into the program by $\textsc{Rev}_{SOS}$.

$$\frac{}{\langle \texttt{rollback-if}\ \ b, \sigma : \sigma_s\rangle \to_1 \sigma_s}\ (\textsc{RbiT}_{SOS}) \quad \mathcal{B}[\![b]\!]\sigma = tt$$

$$\frac{}{\langle \texttt{rollback-if}\ \ b, \sigma : \sigma' : \sigma_s\rangle \to_1 \sigma : \sigma_s}\ (\textsc{RbiF}_{SOS}) \quad \mathcal{B}[\![b]\!]\sigma = ff$$

All other rules have to be adapted to the extended state. For some rules it is sufficient to simply propagate the extended state, e.g. for $\textsc{Skip}_{SOS}$ or $\textsc{Seq}_{SOS}$. For others, such as $\textsc{Ass}_{SOS}$, we have to decompose the extended state in order to preserve the original semantics.

$$\frac{}{\langle \texttt{skip}, \sigma_s\rangle \to_1 \sigma_s}\ (\textsc{Skip}_{SOS})$$

$$\frac{}{\langle x \texttt{ := } e, \sigma : \sigma_s\rangle \to_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma] : \sigma_s} \ (\text{Ass}_{SOS})$$

$$\frac{\langle s_1, \sigma_s\rangle \to_1 \sigma_s'}{\langle s_1 \texttt{;} s_2, \sigma_s\rangle \to_1 \langle s_2, \sigma_s'\rangle} \ (\text{Seq1}_{SOS})$$

$$\frac{\langle s_1, \sigma_s\rangle \to_1 \langle s_1', \sigma_s'\rangle}{\langle s_1 \texttt{;} s_2, \sigma_s\rangle \to_1 \langle s_1' \texttt{;} s_2, \sigma_s'\rangle} \ (\text{Seq2}_{SOS})$$

$$\frac{}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma : \sigma_s\rangle \to_1 \langle s_1, \sigma : \sigma_s\rangle} \ (\text{IfT}_{SOS}) \quad \mathcal{B}[\![b]\!]\sigma = tt$$

$$\frac{}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma : \sigma_s\rangle \to_1 \langle s_2, \sigma : \sigma_s\rangle} \ (\text{IfF}_{SOS}) \quad \mathcal{B}[\![b]\!]\sigma = ff$$

$$\frac{}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma_s\rangle \to_1 \langle \texttt{if } b \texttt{ then } s \texttt{;while } b \texttt{ do } s \texttt{ end else skip end}, \sigma_s\rangle} \ (\text{While}_{SOS})$$