ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Solutions of Exercise Sheet 9: Big Step Semantics

Assignment 1

We use the following abbreviations: l is the statement `a := a+n; b := b*n; n := n-1` and w the statement `while n#0 do l end`. Moreover, we write a state σ that assigns the integers $n_1, \ldots, n_k$ to the variables $\mathtt{v}_1, \ldots, \mathtt{v}_k$, respectively, as $[\mathtt{v}_1, \ldots, \mathtt{v}_k \mapsto n_1, \ldots, n_k]$.

$$\dfrac{\dfrac{}{\langle \texttt{a := a+n}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 0,1,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,1,2]}\ \textsc{Ass} \quad \dfrac{\dfrac{}{\langle \texttt{b := b*n}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,1,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,2]}\ \textsc{Ass} \quad \dfrac{}{\langle \texttt{n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]}\ \textsc{Ass}}{\langle \texttt{b := b*n; n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,1,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]}\ \textsc{Seq}}{\langle \texttt{a := a+n; b := b*n; n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 0,1,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]}\ \textsc{Seq} \quad \dfrac{t_1}{\langle \texttt{while n\#0 do l end}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 0,1,2]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{WhT}$$

where t_1 is the following derivation tree:

$$\dfrac{\dfrac{}{\langle \texttt{a := a+n}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,1]}\ \textsc{Ass} \quad \dfrac{\dfrac{}{\langle \texttt{b := b*n}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,1]}\ \textsc{Ass} \quad \dfrac{}{\langle \texttt{n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{Ass}}{\langle \texttt{b := b*n; n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{Seq}}{\langle \texttt{a := a+n; b := b*n; n := n-1}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{Seq} \quad \dfrac{}{\langle w, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{WhF}}{\langle \texttt{while n\#0 do l end}, [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 2,2,1]\rangle \to [\mathtt{a},\mathtt{b},\mathtt{n} \mapsto 3,2,0]}\ \textsc{WhT}$$

Assignment 2

$$\frac{\langle s;\ s_2, \sigma\rangle \to \sigma_1 \quad \langle \texttt{for(skip, } b\texttt{, } s_2\texttt{) } s\texttt{ end}, \sigma_1\rangle \to \sigma'}{\langle \texttt{for(skip, } b\texttt{, } s_2\texttt{) } s\texttt{ end}, \sigma\rangle \to \sigma'}\ \mathcal{B}[\![b]\!]\sigma = tt$$

$$\frac{}{\langle \texttt{for(skip, } b\texttt{, } s_2\texttt{) } s\texttt{ end}, \sigma\rangle \to \sigma}\ \mathcal{B}[\![b]\!]\sigma = ff$$

$$\frac{\langle s_1, \sigma\rangle \to \sigma_1 \quad \langle \texttt{for(skip, } b\texttt{, } s_2\texttt{) } s\texttt{ end}, \sigma_1\rangle \to \sigma'}{\langle \texttt{for(}s_1\texttt{, } b\texttt{, } s_2\texttt{) } s\texttt{ end}, \sigma\rangle \to \sigma'}\ s_1 \neq \texttt{skip}$$

Assignment 3

For the direction from right to left, we consider the derivation tree for

$$\langle \texttt{if } b \texttt{ then } s; \texttt{ while } b \texttt{ do } s \texttt{ end else skip end}, \sigma\rangle \to \sigma''$$

The last applied rule in this derivation tree is a rule for the if-then-else statement. So, the derivation tree has either the form

$$\frac{\dfrac{\vdots}{\langle s; \texttt{ while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma''}}{\langle \texttt{if } b \texttt{ then } s; \texttt{ while } b \texttt{ do } s \texttt{ end else skip end}, \sigma\rangle \to \sigma''}\ \textsc{IfT} \tag{1}$$

or

$$\frac{\langle \texttt{skip}, \sigma\rangle \to \sigma''}{\langle \texttt{if } b \texttt{ then } s; \texttt{ while } b \texttt{ do } s \texttt{ end else skip end}, \sigma\rangle \to \sigma''}\ \textsc{IfF} \tag{2}$$

- Let us first consider the case (2). The rule is only applicable when $\mathcal{B}[\![b]\!]\sigma = ff$. Furthermore, with the rule for `skip`, we conclude that $\sigma = \sigma''$. We construct the following derivation tree:
$$\frac{}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma}\ \textsc{WhF}$$
- Let us now consider the case (1). The rule is only applicable when $\mathcal{B}[\![b]\!]\sigma = tt$. The next applied rule in the derivation tree must be for sequential composition. The last part of the derivation tree has the form
$$\frac{\dfrac{\dfrac{\vdots}{\langle s, \sigma\rangle \to \sigma'} \quad \dfrac{\vdots}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma'\rangle \to \sigma''}}{\langle s; \texttt{ while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma''}\ \textsc{Seq}}{\langle \texttt{if } b \texttt{ then } s; \texttt{ while } b \texttt{ do } s \texttt{ end else skip end}, \sigma\rangle \to \sigma''}\ \textsc{IfT}$$
Let T_1 be the derivation tree above $\langle s, \sigma\rangle \to \sigma'$ and let T_2 be the derivation tree above $\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma'\rangle \to \sigma''$. We construct the following derivation tree:
$$\frac{\dfrac{T_1}{\langle s, \sigma\rangle \to \sigma'} \quad \dfrac{T_2}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma'\rangle \to \sigma''}}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma''}\ \textsc{WhT}$$

Assignment 4

You find a solution of this assignment in the literate Haskell file `simpi_onlyns.lhs`.

Assignment 5 - Headache of the week

In order to support exceptions, we have to add to the set of states some "exceptional" states to represent computation in which an exception has been thrown. When an exception is thrown, we have to collect the state of the computation of the name of the thrown exception. We represent an "exceptional" state by an element in $State \times ExcNames$. Then we define the new set of states as the union of "normal" and "exceptional" states. Formally, $State' = State \cup (State \times ExcNames)$.

The natural semantics has to be extended with the following rules, where $\sigma, \sigma' \in State$ and $\tau \in State'$.

$$\frac{}{\langle \texttt{throw}\ \ ex, \sigma\rangle \to (\sigma, ex)}$$

$$\frac{\langle s, \sigma\rangle \to (\sigma', ex) \quad \langle s_1, \sigma'\rangle \to \tau}{\langle \texttt{try}\ \ s\ \ \texttt{catch(}ex\texttt{)}\ \ s_1, \sigma\rangle \to \tau}$$

$$\frac{\langle s, \sigma\rangle \to (\sigma', ex')}{\langle \texttt{try}\ \ s\ \ \texttt{catch(}ex\texttt{)}\ \ s_1, \sigma\rangle \to (\sigma', ex')}\ ex \neq ex'$$

$$\frac{\langle s, \sigma\rangle \to \sigma'}{\langle \texttt{try}\ \ s\ \ \texttt{catch(}ex\texttt{)}\ \ s_1, \sigma\rangle \to \sigma'}$$

The semantics of other statements (assignment, skip, concatenation, if-then-else, while loop) is the same of the one introduced in the lecture when dealing with states in *State*.

In the case of an exceptional state in $State \times ExcNames$ we have to carry on the exception. This is formalized by the following rule, where s is a statement other than `try-catch`.

$$\frac{}{\langle s, (\sigma, ex)\rangle \to (\sigma, ex)}$$