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

(a) The statement s stores 2^k in variable $\mathtt{y}$ where k is the initial value of variable $\mathtt{x}$.

(b) We use the following abbreviations: b is the statement `y := y*2; x := x-1` and w the statement `while x>0 do b 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 $[\mathtt{v}_1, \ldots, \mathtt{v}_k \mapsto n_1, \ldots, n_k]$. The values of the other variables in state σ are left implicit (their values are irrelevant).

$$\dfrac{\langle \mathrm{y}{:=}1, [\mathrm{x} \mapsto 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 2, 1] \qquad \dfrac{\dfrac{\langle \mathrm{y}{:=}\mathrm{y}{*}2, [\mathrm{x}, \mathrm{y} \mapsto 2, 1]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 2, 2] \quad \langle \mathrm{x}{:=}\mathrm{x}{-}1, [\mathrm{x}, \mathrm{y} \mapsto 2, 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 1, 2]}{\langle b, [\mathrm{x}, \mathrm{y} \mapsto 2, 1]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 1, 2]} \qquad \dfrac{\dfrac{\langle \mathrm{y}{:=}\mathrm{y}{*}2, [\mathrm{x}, \mathrm{y} \mapsto 1, 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 1, 4] \quad \langle \mathrm{x}{:=}\mathrm{x}{-}1, [\mathrm{x}, \mathrm{y} \mapsto 1, 4]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]}{\langle b, [\mathrm{x}, \mathrm{y} \mapsto 1, 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]} \quad \langle w, [\mathrm{x}, \mathrm{y} \mapsto 0, 4]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]}{\langle w, [\mathrm{x}, \mathrm{y} \mapsto 1, 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]}}{\langle w, [\mathrm{x}, \mathrm{y} \mapsto 2, 1]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]}}{\langle s, [\mathrm{x} \mapsto 2]\rangle \to [\mathrm{x}, \mathrm{y} \mapsto 0, 4]}$$

Assignment 2

The deduction rules for `repeat` s `until` b are

$$\frac{\langle s, \sigma\rangle \to \sigma'}{\langle \texttt{repeat}\ s\ \texttt{until}\ b, \sigma\rangle \to \sigma'}\ \mathcal{B}[\![b]\!]\sigma' = tt$$

and

$$\frac{\langle s, \sigma\rangle \to \gamma \quad \langle \texttt{repeat}\ s\ \texttt{until}\ b, \gamma\rangle \to \sigma'}{\langle \texttt{repeat}\ s\ \texttt{until}\ b, \sigma\rangle \to \sigma'}\ \mathcal{B}[\![b]\!]\gamma = f\!f$$

We can define the semantics of `repeat-until` relying on the semantics of the `while` loop.

$$\frac{\langle \texttt{while (not}\ b\texttt{) do}\ s\ \texttt{end;}\ s, \sigma\rangle \to \sigma'}{\langle \texttt{repeat}\ s\ \texttt{until}\ b, \sigma\rangle \to \sigma'}$$

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

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

or

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

- Let us first consider the case (2). The rule is only applicable when $\mathcal{B}[\![b]\!]\sigma = f\!f$. Furthermore, with the rule for `skip`, we conclude that $\sigma = \sigma''$. We construct the following derivation tree:

  $$\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma$$

- 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

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

  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:

  $$\frac{\dfrac{T_1}{\langle s, \sigma\rangle \to \sigma'} \quad \dfrac{T_2}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma'\rangle \to \sigma''}}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''}$$

Assignment 4

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

Assignment 5

The semantics of arithmetic expressions returns integer values. Since **IMP** does not have information about the type of variables and expressions (that is, we cannot distinguish if an expression is an integer value or a pointer), the references are represented by integer values.

The new state is composed by an environment ($Env : Var \to \mathbb{Z}$), that relates each variable to a reference, and a heap ($Heap : \mathbb{Z} \to \mathbb{Z}$), that relates each reference to an integer value. Then the new representation of states is $State = Env \times Heap$.

Note that in the slides integer values are represented by the set Val. Here we use a different notation since we want to underline that references and values are the same thing.

We extend the semantics of expressions with the two following rules:

$$\mathcal{A}[\![*\mathsf{e}]\!](env, h) = h(\mathcal{A}[\![\mathsf{e}]\!](env, h))$$

$$\mathcal{A}[\![\mathtt{x}]\!](env, h) = env(\mathtt{x})$$

$$\mathcal{A}[\![\&\mathtt{x}]\!](env, h) = env(\mathtt{x})$$

$$\frac{env' = env[\mathtt{x} \mapsto \mathcal{A}[\![e]\!](env, h)]}{\langle \mathtt{x\ :=}\ e, (env, h)\rangle \to (env', h)}$$

$$\frac{h' = h[\mathcal{A}[\![e_1]\!](env, h) \mapsto \mathcal{A}[\![e_2]\!](env, h)]}{\langle *\mathsf{e}_1 := e_2, (env, h)\rangle \to (env, h')}$$

Note that using this semantics is possible to assign any arithmetic expression as reference to a variable (for instance, `x := 1`). In addition, we modified the rule of the evaluation of variables, since this rule will be used when accessing the content of a reference through `*e`. For this reason, $\mathcal{A}[\![\mathtt{x}]\!](env, h)$ returns the reference pointed by the variable `x` (that is, it has the same semantics of `&x`).

For the other cases of the evaluation of arithmetic expressions and of the natural semantics, the semantics is the pointwise extension to states composed by an environment and a heap. For instance, the semantics of the concatenation of statements is redefined as follows:

$$\frac{\langle \mathtt{s1}, (env, h)\rangle \to (env', h'), \langle \mathtt{s2}, (env', h')\rangle \to (env'', h'')}{\langle \mathtt{s1;\ s2}, (env, h)\rangle \to (env'', h'')}$$