Semantics of programming languages - Supplementary material

April 2019

1 Natural semantics of arithmetic expressions

The semantics of arithmetic expressions is given by function $\mathscr{E}$. We can also se an operational approach to define a natural semantics for the arithmetic expressions. It will have two kinds of configurations:

$\langle e, s\rangle$ denoting that e has ti be evaluated in state s, and
v denoting the final value, an element of $\mathbf{Z}$.

The transition relation $\rightarrow_{Expr}$ has the form

$$\langle e, s\rangle \rightarrow_{Expr} v$$

where the idea is that e evaluates to v in state s. An example axioms and rules are as follows

$$\langle n, s\rangle \rightarrow_{Expr} \mathscr{N}[\![n]\!]$$

$$\langle x, s\rangle \rightarrow_{Expr} s\ x$$

$$\frac{\langle e_1, s\rangle \rightarrow_{Expr} v_1 \quad \langle e_2, s\rangle \rightarrow_{Expr} v_2}{\langle e_1 + e_2, s\rangle \rightarrow_{Expr} v} \qquad \text{where } v = v_1 + v_2$$

Complete the specification of the transition system. Use structural induction on **Expr** to prove that the meaning of e defined by this relation is the same as that defined by function $\mathscr{E}$.

2 Natural semantics of Boolean expressions

In a similar way, we can specify a natural semantics for the Boolean expressions. The transitions will have the form

$$\langle b, s\rangle \rightarrow_{Bexpr} t$$

where $t \in \mathbf{B}$. Specify the transition system and prove that the meaning of b defined in this way is the same as that defined by function $\mathscr{B}$.

3 Natural semantics of statements

Consider the following statements

$$\texttt{while } \neg(x = 1) \texttt{ do } (y := y * x; x := x - 1)$$
$$\texttt{while } 1 \leq x \texttt{ do } (y := y * x; x := x - 1)$$
$$\texttt{while true do skip}$$

For each statement determine whether or not it always terminates and whether or not it always loops. Try to argue for your answers using the axioms and rules of Natural semantics.

4 Extension of a language

We extend the language *Jane* with the statement

$$\texttt{repeat } S \texttt{ until } b$$

and we define both transition relations for it - in natural and in structural operational semantics. We note, that the semantics of the `repeat`-construct is not allowed to rely on the existence of a `while`-construct in the language.

$$\frac{\langle S, s\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'} \ (\texttt{repeat}^{\mathbf{tt}}_{ns}) \quad \text{if } \mathscr{B}[\![b]\!]s' = \mathbf{tt}$$

$$\frac{\langle S, s\rangle \to s', \quad \langle \texttt{repeat } S \texttt{ until } b, s'\rangle \to s''}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s''} \ (\texttt{repeat}^{\mathbf{ff}}_{ns}) \quad \text{if } \mathscr{B}[\![b]\!]s' = \mathbf{ff}$$

$$(\texttt{repeat}_{os}) \quad \langle \texttt{repeat } S \texttt{ until } b, s\rangle \Rightarrow \langle S; \texttt{if } b \texttt{ then skip else } (\texttt{repeat } S \texttt{ until } b, s), s\rangle$$

5 Adding variable increment

Like in several main-stream programming languages, $++x$ increments the value of x in the state and evaluates to the incremented value. This way, the increment operation makes the evaluation of expressions to now have side effects.

Big-step SOS is one of the semantics which is the most affected by the inclusion of side effects in expressions, because the previous triples $\langle e, s\rangle \to_{Expr} v$ and $\langle b, s\rangle \to_{Bexpr} t$ need to change to four-tuples of the form $\langle e, s\rangle \to_{Expr} \langle v, s'\rangle$ and $\langle b, s\rangle \to_{Bexpr} \langle t, s'\rangle$. These changes are necessary to account for collecting the possible side effects generated by the evaluation of expressions (note that the evaluation of Boolean expressions, because of $\leq$, can also have side effects). The big-step operational semantics of almost all the language constructs needs to change as well.

Once all the changes to the existing semantics of a language are applied, the big-step semantics of increment is straightforward:

$$\langle ++x, s\rangle \rightarrow_{Expr} \langle s\ x \oplus \mathbf{1}, s\,[x \mapsto sx \oplus \mathbf{1}]\rangle$$

Indeed, the problem with big-step is not necessarily to define the semantics of variable increment, but what it takes to be able to do it. One needs to redefine configurations as explained above and, consequently, to change the semantics of all the already existing features of IMP to use the new configurations.