

Formal methods at work 2018

Renan Greca; Konstantin Prokopchik

November 22, 2018

1. Consider two statements, S_1 and S_2 , that are equivalent in big step semantics ($S_1 \approx_{BS} S_2$). For every program S and memory state σ , it is also true that $(S_1; S) \approx_{BS} (S_2; S)$. Consider the following transition: $(S_1; S, \sigma) \Rightarrow \sigma'$: by the definition of the big-step semantics for sequential operations, there is a memory state σ'' such that $(S_1, \sigma) \Rightarrow \sigma''$:

$$\frac{(S_1, \sigma) \Rightarrow \sigma'' (S, \sigma'') \Rightarrow \sigma'}{(S_1; S, \sigma) \Rightarrow \sigma'}$$

Since $S_1 \approx_{BS} S_2$, it is also true that $(S_2, \sigma) \Rightarrow \sigma''$, and therefore:

$$\frac{(S_2, \sigma) \Rightarrow \sigma'' (S, \sigma'') \Rightarrow \sigma'}{(S_2; S, \sigma) \Rightarrow \sigma'}$$

By the definition of equivalence, both (S_1, σ) and (S_2, σ) result in the same state σ'' , and therefore both $(S_1; S, \sigma)$ and $(S_2; S, \sigma)$ produce the same final state σ' .

2. We can define the small-step semantics of **assert b before S** using the following rule:

$$\frac{}{(\text{assert } b \text{ before } S, \sigma) \rightarrow (S, \sigma)} \text{eval}(b, \sigma) = \text{true}$$

In other words, S is executed if b evaluates to *true*. Otherwise, no transition is produced.

The difference between **assert *false* before S** and **while *true* do *skip*** is that computation finishes on the former case, despite the lack of a final state, while the latter case is an endless loop.

The difference between **assert *false* before S** and ***skip*** can be illustrated by the sequential operation of two statements. Consider a statement in the form of $(S_1; S_2)$ in which S_1 can be either **assert *false* before S** or ***skip***:

$$\overline{(\text{assert } \textit{false} \text{ before } S; S_2, \sigma)}$$

$$\overline{(\textit{skip}; S_2, \sigma) \rightarrow (S_2, \sigma)}$$

It is possible to observe that, in the first case, the computation is aborted and S_2 is not executed. However, in the second case, S_2 is always executed with the same memory state as the one used by ***skip***.

3. Consider the statement:

$x := -1$; **while $x \leq 0$ do $(x := x - 1$ or $x := (-1) \times x$)**. For the purposes of brevity, let us call the second part of the statement as S .

$$S = \text{while } x \leq 0 \text{ do } (x := x - 1 \text{ or } x := (-1) \times x)$$

The **or** construct contained within S leads to the existence of multiple possible final states for this statement. If the left-side path of the **or** is always chosen, no final state is produced and, therefore, there is no transition in big-step semantics.

However, we can prove the transition for S using big-step semantics by assuming the right-side path of the **or** is taken and achieving a final state, for example:

$$\frac{(x := -1, \sigma) \Rightarrow \sigma[x \mapsto -1] \quad \frac{(x := (-1) \times x, \sigma[x \mapsto -1]) \Rightarrow \sigma[x \mapsto 1] \quad (S; \sigma[x \mapsto 1]) \Rightarrow \sigma[x \mapsto 1]}{(S; \sigma[x \mapsto -1]) \Rightarrow \sigma[x \mapsto 1]}}{(x := -1; S, \sigma) \Rightarrow \sigma[x \mapsto 1]}$$

In other words, given an initial state where x is a negative integer n , following the right-side path of the **or** causes the following big-step transition: $(S, \sigma[x \mapsto n]) \Rightarrow \sigma[x \mapsto p]$, where $p = -n$.

It is possible to observe that, by following the left-side path instead of the right-side path, the value of x decreases and we find statement/state pairs such as $(S, \sigma[x \mapsto -2])$, $(S, \sigma[x \mapsto -3])$ and so forth. Since the left-side path can be taken an arbitrary number of times, for any negative integer n there exists a transition in the form of $(S, \sigma[x \mapsto n])$. Then, by following the right-side path at any point, x becomes positive and the computation ends.

Therefore, this program as a whole can be described in big-step semantics as:

$$\forall p \in \mathbb{N}, (x := -1; S, \sigma) \Rightarrow \sigma[x \mapsto p]$$

The development of the statement using small-step semantics is as follows:

$$\begin{aligned} & (x := -1; S, \sigma) \rightarrow \\ & (S, \sigma[x \mapsto -1]) \rightarrow \\ & (if \ x \leq 0 \ then \ (x := x - 1 \ \mathbf{or} \ x := (-1) \times x); S \ else \ skip, \sigma[x \mapsto -1]) \rightarrow \\ & (x := x - 1 \ \mathbf{or} \ x := (-1) \times x; S, \sigma[x \mapsto -1]) \rightarrow \end{aligned}$$

From here, the following steps depend on the path taken by the **or**. Following the left-side path:

$$\begin{aligned} & (x := x - 1; S, \sigma[x \mapsto -1]) \rightarrow \\ & (S, \sigma[x \mapsto -2]) \rightarrow \\ & (if \ x \leq 0 \ then \ (x := x - 1 \ \mathbf{or} \ x := (-1) \times x); S \ else \ skip, \sigma[x \mapsto -2]) \rightarrow \\ & (x := x - 1 \ \mathbf{or} \ x := (-1) \times x; S, \sigma[x \mapsto -2]) \rightarrow \\ & (x := x - 1; S, \sigma[x \mapsto -2]) \rightarrow \\ & (S, \sigma[x \mapsto -3]) \rightarrow \end{aligned}$$

The development continues endlessly for all values of $x < 0$. Following the right-side path, we achieve a different conclusion:

$$\begin{aligned} & (x := (-1) \times x; S, \sigma[x \mapsto -1]) \rightarrow \\ & (S, \sigma[x \mapsto 1]) \rightarrow \\ & (if \ x \leq 0 \ then \ (x := x - 1 \ \mathbf{or} \ x := (-1) \times x); S \ else \ skip, \sigma[x \mapsto 1]) \rightarrow \\ & (skip, \sigma[x \mapsto 1]) \rightarrow \\ & \sigma[x \mapsto 1] \end{aligned}$$

While in big-step semantics this program always results in a state in which x has the value of any natural number, in small-step semantics it is clear that the program might result in an endless loop. This illustrates the differences between the two approaches: the big-step semantics is useful to validate what the program does, but the small-step semantics allows us to see where execution could go wrong. Now, consider an extension to the language in the form of $\mathbf{random}(x)$, which produces a final state in which x is a random natural number. Using big-step and small-step semantics, the transition provided by $\mathbf{random}(x)$ is the same:

$$\text{In big-step, } \forall p \in \mathbb{N}, \overline{(\mathbf{random}(x), \sigma) \Rightarrow \sigma[x \mapsto p]}$$

$$\text{In small-step, } \forall p \in \mathbb{N}, \overline{(\mathbf{random}(x), \sigma) \rightarrow \sigma[x \mapsto p]}$$

We have already seen a statement that produces an arbitrary natural number in big-step semantics. Therefore, when considering big-step semantics, $\mathbf{random}(x)$ appears redundant.

However, to show that $\mathbf{random}(x)$ is redundant in small-step semantics, there needs to be a small-step transition that produces a random natural number without the risk of an endless loop. This is not possible using the **or** construct.

Without the $\mathbf{random}(x)$ function, the **or** construct is the only way to achieve nondeterminism in the While language. However, an **or** construct within a **while** loop effectively creates a binary tree, in which leaves are final states and nodes lead to more possible final states. Since there are infinite natural numbers, a statement that returns an arbitrary natural number must have infinite final states. Therefore, in small-step semantics, such a statement is represented by an infinite binary tree, which according to König's Lemma, always has an infinite path, making the statement not equivalent to $\mathbf{random}(x)$, which guarantees a final state is reached.

4. The denotational semantics for **while** $x \geq 0$ **do** *skip* are as follows:

$$g = \llbracket \mathbf{while} \ x \geq 0 \ \mathbf{do} \ \mathit{skip} \rrbracket$$

$$F_{x \geq 0, \mathit{skip}}(g) = \mathit{cond}(\mathit{eval}(x \geq 0, -), g \circ \llbracket \mathit{skip} \rrbracket, \mathit{id}) = \mathit{cond}(\mathit{eval}(x \geq 0, -), g, \mathit{id})$$

$$F_{x \geq 0, \mathit{skip}}(g)(\sigma) = \begin{cases} \sigma, & \text{if } \sigma(x) < 0 \\ \perp, & \text{if } \sigma(x) \geq 0 \end{cases}$$

Consider a function g_\perp such that $g_\perp(\sigma) = \perp$. Then:

$$F_{x \geq 0, \mathit{skip}}(g_\perp) = \mathit{cond}(\mathit{eval}(x \geq 0, -), g_\perp, \mathit{id})$$

$$F_{x \geq 0, \mathit{skip}}(g_\perp)(\sigma) = \begin{cases} \sigma, & \text{if } \sigma(x) < 0 \\ g_\perp(\sigma) = \perp, & \text{if } \sigma(x) \geq 0 \end{cases}$$

- 5.

$$g = \llbracket \mathbf{repeat} \ S \ \mathbf{until} \ b \rrbracket = (\mathit{FIX} \ F_{b,S}) \circ \llbracket S \rrbracket$$

$$F_{b,S}(g) = \mathit{cond}(\mathit{eval}(!b, -), g, \mathit{id})$$

$$F_{b,S}(g)(\sigma) = \begin{cases} \sigma, & \text{if } \mathit{eval}(b) = \mathit{true} \\ \perp, & \text{if } \mathit{eval}(b) = \mathit{false} \end{cases}$$