Karlsruher Institut für Technologie

Lehrstuhl für Programmierparadigmen

Denis Lohner denis.lohner@kit.edu

Sebastian Ullrich sebastian.ullrich@kit.edu

Semantik von Programmiersprachen – SS 2017

http://pp.ipd.kit.edu/lehre/SS2017/semantik

Blatt 2: Big-Step-Semantik Besprechung: 08.05.2017

1. Welche der folgenden Aussagen sind richtig, welche falsch? (H)

(a) `false` $= \mathbf{ff}$

(b) `skip; x := true` ist ein While-Programm.

(c) `x := y * z` $\neq$ `x := z * y`.

(d) `x + y` $=$ `x - (0 - y)`.

(e) `x := y + 1; y := x` und `y := x; x := y + 1` haben das gleiche Verhalten.

(f) `if (not` b`) then` c_0 `else` c_1 und `if (`b`) then` c_1 `else` c_0 haben das gleiche Verhalten.

(g) Es gibt a und σ, für die $\mathcal{A}[\![a]\!]\,\sigma$ nicht definiert ist.

(h) Es gibt keine σ und σ', so dass $\langle$`while (true) do skip`$,\ \sigma\rangle \Downarrow \sigma'$.

(i) Ersetzt man die Regeln $\textsc{IfTT}_{\text{BS}}$ und $\textsc{IfFF}_{\text{BS}}$ durch folgende kombinierte Regel, ändert sich die Big-Step-Semantik nicht:

$$\frac{\langle c_0,\ \sigma\rangle \Downarrow \sigma' \qquad \langle c_1,\ \sigma\rangle \Downarrow \sigma'' \qquad \mathcal{B}[\![b]\!]\,\sigma = \mathbf{tt} \longrightarrow \sigma''' = \sigma' \qquad \mathcal{B}[\![b]\!]\,\sigma = \mathbf{ff} \longrightarrow \sigma''' = \sigma''}{\langle \texttt{if (}b\texttt{) then } c_0 \texttt{ else } c_1,\ \sigma\rangle \Downarrow \sigma'''}$$

2. Big-Step-Semantik in Prolog (H)

Implementieren Sie die Big-Step-Semantik für While in Prolog. Ein While-Programm wird in der folgenden Syntax eingegeben: Für das syntaktisch repräsentative While-Programm

```
n := 42; while (true && not (n <= 1)) do if (n <= 1) then skip else n := n - (1 * 23)
```

schreiben wir in Prolog

```
n:=42; while(and(true, not(n <= 1)), cond(n <= 1, skp, n := n - (1*23))).
```

Die Datei `common.pl` auf der Webseite zur Übung definiert dazu den Operator <= und stellt unter anderem folgende Prädikate bereit, die Sie natürlich benutzen sollen (Sie importieren die Definition mit der Prolog-Klausel :- consult('common.pl'). in Ihre Datei):

- evalA(S,A,V) ist wahr, wenn der arithmetische Ausdruck A, im Zustand S ausgewertet, den Integer-Wert V ergibt.
- evalB(S,B,V) ist wahr, wenn der Bool'sche Ausdruck B, im Zustand S ausgewertet, den Bool'schen Wert V ergibt, wobei Bool'sche Werte entweder tt oder ff sind.
- set(S1,Var,Val,S2) ist wahr, wenn sich der Zustand S2 aus dem Zustand S1 ergibt, in dem die Variable Var auf den Wert Val gesetzt wird.

- get(S,Var,Val) ist wahr, wenn im Zustand S die Variable Var den Wert Val hat. Wenn die Variable nicht gesetzt ist, wird ein beliebiger[1] Wert zurückgegeben, da wir mit totalen Zustandsabbildungen arbeiten wollen.

Variablen sind Prolog-Atome, Zustände werden als Listen von Paaren dargestellt; [] ist der initiale Zustand, [n−3,m−4] entspricht $[\texttt{n} \mapsto 3, \texttt{m} \mapsto 4]$. Sie sollen nun ein Prolog-Prädikat evalBS(Prog, S1, S2) definieren, das genau dann erfüllt ist, wenn Prog dem Programm c, S1 dem Zustand σ_1 und S2 dem Zustand σ_2 entspricht und $\langle c,\ \sigma_1\rangle \Downarrow \sigma_2$ gilt.

Schreiben Sie folgendes While-Programm als Prolog-Term und werten Sie es aus. Was berechnet es, in Abhängigkeit von dem Initialwert von `n`?

```
m := 1; while (1 <= n) do m := m * n; n := n - 1
```

3. Ableitungsbäume (H)

Schreiben Sie ein Programm in While, das für eine Zahl n die kleinste Zahl m bestimmt, so dass $2^m \geq |n|$, wobei $|n|$ der Absolutbetrag von n ist.

Zeichnen Sie den Ableitungsbaum einer Ausführung des Programms gemäß der Big-Step-Semantik mit $n = -4$ im Startzustand. Hören Sie auf wenn der Baum fertig ist, oder Sie überzeugt sind, dass (a) Sie das Anwenden der induktiven Regeln jetzt können und (b) dass der Computer diese Aufgabe besser können müsste.

4. Zählschleife (Ü)

Viele Programmiersprachen bieten neben einer `while`-Schleife auch eine `for`-Schleife an. Erweitern Sie die Syntax und Big-Step-Semantik von While um eine Zählschleife mit der Syntax `for` x `=` a `to` a' `do` c. Verfolgen Sie dabei verschiedene semantische Modellierungen. Finden Sie Programme, die je nach semantischer Variante unterschiedliches Verhalten zeigen. Hilfreiche Überlegungen: Könnte man Ihre Zählschleife auch als syntaktischen Zucker mittels `while` definieren? Terminieren alle `for`-Schleife immer? Was ist der Wert des Schleifenzählers nach Ende der Schleife? Implementieren Sie die Semantiken in Prolog und experimentieren Sie damit!

[1] Für einen Informatiker-kompatiblen Wert von beliebig.