

Semantik von Programmiersprachen – SS 2012

<http://pp.info.uni-karlsruhe.de/lehre/SS2012/semantik>

Blatt 11: Kontexte

Besprechung: 03.07.2012

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

- (a) `(if (true) then x := 18 else []); (y := 42; [])` ist ein Kontext.
- (b) Wenn $\langle c, \sigma \rangle \xrightarrow{\infty}_1$, dann $\mathcal{D} \llbracket c \rrbracket \sigma = \perp$.
- (c) Wenn $K_1[c] = K_2[c]$, dann $K_1 = K_2$.
- (d) Wenn $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$, dann $\langle K[c], \sigma \rangle \rightarrow_1 \langle K[c'], \sigma' \rangle$.
- (e) $\mathcal{D} \llbracket _ \rrbracket$ ist surjektiv.

2. Programmäquivalenz (H)

Verschiedene Semantiken führen zu verschiedenen Äquivalenzbegriffen für Programme. c_1 und c_2 heißen *äquivalent bzgl. der*

Big-Step-Semantik wenn für alle σ, σ' gilt, dass $\langle c_1, \sigma \rangle \Downarrow \sigma'$ gdw. $\langle c_2, \sigma \rangle \Downarrow \sigma'$;

Small-Step-Semantik wenn für alle σ und σ' gilt,

dass $\langle c_1, \sigma \rangle \xrightarrow{*}_1 \langle \text{skip}, \sigma' \rangle$ gdw. $\langle c_2, \sigma \rangle \xrightarrow{*}_1 \langle \text{skip}, \sigma' \rangle$ und dass $\langle c_1, \sigma \rangle \xrightarrow{\infty}_1$ gdw. $\langle c_2, \sigma \rangle \xrightarrow{\infty}_1$;

denotationalen Semantik wenn $\mathcal{D} \llbracket c_1 \rrbracket = \mathcal{D} \llbracket c_2 \rrbracket$.

c_1 und c_2 heißen *beobachtungsgleich* bezüglich einer Menge M von Kontexten, falls $K[c_1]$ und $K[c_2]$ semantisch äquivalent sind für alle Kontexte $K \in M$.

- (a) Vergleichen Sie die verschiedenen Äquivalenzbegriffe der Semantiken.
- (b) Finden Sie eine (möglichst kleine) Menge von Kontexten, für die beobachtungsgleiche Programme auch semantisch äquivalent sind.
- (c) Für welche M sind äquivalente Programme beobachtungsgleich?
- (d) Zeigen Sie, dass folgende Programme äquivalent sind:

```
while (not x == 1) do if (odd(x)) then x := (x * 3) + 1 else x := x div 2
```

 und

```
while (not x == 1) do if (not odd(x)) then x := x div 2 else x := (x * 3) + 1
```

 wobei $a_1 \text{ div } a_2$ bzw. $\text{odd}(a)$ ein neuer arithmetischer bzw. boolescher Ausdruck mit folgender Semantik sei:

$$\mathcal{A} \llbracket a_1 \text{ div } a_2 \rrbracket \sigma = \begin{cases} 0 & \text{falls } \mathcal{A} \llbracket a_2 \rrbracket \sigma = 0 \\ \left\lfloor \frac{\mathcal{A} \llbracket a_1 \rrbracket \sigma}{\mathcal{A} \llbracket a_2 \rrbracket \sigma} \right\rfloor & \text{sonst} \end{cases}$$

$$\mathcal{B} \llbracket \text{odd}(a) \rrbracket \sigma = (\mathcal{A} \llbracket a \rrbracket \sigma \text{ ungerade})$$

3. Nichtdeterminismus (Ü)

Die Sprache While_{ND} erweitert While um den nichtdeterministischen Auswahloperator $c_1 \text{ or } c_2$. In der Vorlesung haben wir bereits die Big-Step- und Small-Step-Semantiken darauf erweitert, hier sollen Sie nun eine denotationale Erweiterung entwickeln.

Statt höchstens eines Endzustandes sind nun beliebig viele Endzustände möglich. Die Bedeutung eines Programms ändert sich dementsprechend von einer partiellen Funktion auf Zuständen ($\Sigma \rightarrow \Sigma$) zu einer Funktion, die einen Anfangszustand auf eine Menge möglicher Endzustände ($\Sigma \Rightarrow \mathfrak{P}(\Sigma)$) abbildet.

- (a) Passen Sie die Definition von $\mathcal{D}[\![_]\!]$ entsprechend an. Definieren Sie auch $\mathcal{D}[\![c_1 \text{ or } c_2]\!]$.
- (b) Für den Fixpunkt benötigen Sie auch eine neue Approximationsordnung \sqsubseteq :

$$f \sqsubseteq g \quad \text{gdw.} \quad \forall \sigma. f(\sigma) \subseteq g(\sigma)$$

Was müssen Sie neu zeigen, um die Existenz und Eindeutigkeit des Fixpunkts zu garantieren?

- (c) Berechnen Sie $\mathcal{D}[\![\text{skip or while (true) do skip}]\!]$.
- (d) Reformulieren Sie das Adäquatheitstheorem. Gilt es immer noch?
- (e) Wiederholen Sie Aufgabe 2a für die Semantiken von While_{ND} .
- (f) Definieren Sie eine Funktion $\mathcal{T}[\![_]\!] :: \text{Com} \Rightarrow \mathfrak{P}(\Sigma)$, die die Menge der Zustände berechnet, für die die übergebene Anweisung immer terminiert.