Karlsruher Institut für Technologie

Lehrstuhl für Programmierparadigmen
Sebastian Ullrich sebastian.ullrich@kit.edu
Max Wagner maximilian.wagner@kit.edu

Semantik von Programmiersprachen – SS 2019

http://pp.ipd.kit.edu/lehre/SS2019/semantik

Blatt 11: Kontexte Besprechung: 08.07.2019

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

(a) `(if (true) then x := 18 else [])` ; `(y := 42; [])` ist ein Kontext.

(b) Wenn $\langle c,\ \sigma\rangle \overset{\infty}{\to}_1$, dann $\mathcal{D}[\![c]\!]\,\sigma = \bot$.

(c) Wenn $K_1[c] = K_2[c]$, dann $K_1 = K_2$.

(d) Wenn $\langle c,\ \sigma\rangle \to_1 \langle c',\ \sigma'\rangle$, dann $\langle K[c],\ \sigma\rangle \to_1 \langle K[c'],\ \sigma'\rangle$.

(e) $\mathcal{D}[\![\cdot]\!]$ 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 $\langle c_1,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ gdw. $\langle c_2,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ und dass $\langle c_1,\ \sigma\rangle \overset{\infty}{\to}_1$ gdw. $\langle c_2,\ \sigma\rangle \overset{\infty}{\to}_1$;

denotationalen Semantik wenn $\mathcal{D}[\![c_1]\!] = \mathcal{D}[\![c_2]\!]$.

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 `div` a_2 bzw. `odd(`a`)` ein neuer arithmetischer bzw. boolescher Ausdruck mit folgender Semantik sei:

$$\mathcal{A}[\![a_1 \texttt{ div } a_2]\!]\,\sigma = \begin{cases} 0 & \text{falls } \mathcal{A}[\![a_2]\!]\,\sigma = 0 \\ \left\lfloor \frac{\mathcal{A}[\![a_1]\!]\sigma}{\mathcal{A}[\![a_2]\!]\sigma} \right\rfloor & \text{sonst} \end{cases}$$

$$\mathcal{B}[\![\texttt{odd(}a\texttt{)}]\!]\,\sigma = (\mathcal{A}[\![a]\!]\,\sigma \text{ ungerade})$$

3. Nichtdeterminismus (Ü)

Die Sprache While_{ND} erweitert While um den nichtdeterministischen Auswahloperator c_1 `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 ($\Sigma \rightharpoonup \Sigma$) zu einer Funktion, die einen Anfangszustand auf eine Menge möglicher Endzustände ($\Sigma \to \mathfrak{P}(\Sigma)$) abbildet.

(a) Passen Sie die Definition von $\mathcal{D}[\![\cdot]\!]$ entsprechend an. Definieren Sie auch $\mathcal{D}[\![c_1$ `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 $\mathcal{D}[\![$`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 $\mathcal{T}[\![\cdot]\!] :: \mathsf{Com} \to \mathfrak{P}(\Sigma)$, die die Menge der Zustände berechnet, für die die übergebene Anweisung immer terminiert.