Karlsruher Institut für Technologie

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

Semantik von Programmiersprachen – SS 2019

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

Lösungen zu Blatt 4: Big-Step- und Small-Step-Semantik Besprechung: 20.05.2019

1. Semantik mit Ausführungszeiten (H)

Die Big-Step-Semantik für While berücksichtigt nicht, wie viele Schritte die Ausführung eines Programms benötigt. Dies sollen Sie in dieser Aufgabe modellieren:

(a) Definieren Sie, auf Papier oder in Prolog, eine Auswertungsrelation $\langle c,\ \sigma\rangle \Downarrow_t \sigma'$ mit der Bedeutung, dass die Ausführung von c im Anfangszustand σ im Endzustand σ' endet und dafür t Schritte benötigt. Finden Sie dafür eine geeignete Definition, was ein Schritt sein soll.

(b) Beschreiben Sie formal, in welcher Beziehung $\langle c,\ \sigma\rangle \Downarrow \sigma'$ und $\langle c,\ \sigma\rangle \Downarrow_t \sigma'$ stehen. Wie würde man diese Beziehung beweisen?

(c) Überprüfen Sie, welche der folgenden Eigenschaften der Big-Step-Semantik $\langle c,\ \sigma\rangle \Downarrow \sigma'$ sich auf $\langle c,\ \sigma\rangle \Downarrow_t \sigma'$ übertragen lassen. Formulieren Sie die Aussagen entsprechend. Wie müssten die Beweise angepasst werden?

- Schleifenabwicklungslemma (Lem. 9)
- Determinismus (Thm. 10)
- Äquivalenz zwischen Big-Step- und Small-Step-Semantik (Kor. 23)

Lösung:

(1a) Verschiedene Begriffe für einen Schritt sind denkbar:

- Jeder Ableitungsschritt im Baum ist ein Schritt. Dies ist der natürlichste Ansatz bezüglich der Big-Step-Semantik, erlaubt aber nur schwierige Vergleiche mit den Längen der Ableitungsfolgen der Small-Step-Semantik. Die Auswertung eines arithmetischen oder booleschen Ausdrucks kostet in diesem Modell keine Zeit.
- Man definiert auch Ausführungszeiten für arithmetische und boolesche Ausdrücke, z.B. mit einer rekursiven Ausführungszeitfunktion. Dann muss man allerdings neu über syntaktischen Zucker für arithmetische und boolesche Ausdrücke nachdenken: Eine Addition würde dann ggf. länger dauern als eine Subtraktion!
- Man definiert die Ausführungszeiten so, dass genau die Länge der Small-Step-Semantik herauskommt – oder die Zahl der Ausführungsschritte in ASM nach Übersetzung mit dem Compiler. Damit kann man dann zwar schöne Äquivalenz- bzw. Erhaltungseigenschaften zeigen, allerdings ist dieser Zeitbegriff sehr künstlich.
  Nielson und Nielson scheinen (Kap. 10.2, Tabelle 10.3) eine solche Definition bezüglich der Small-Step-Semantik anzugeben.

Hier zuerst die einfachste Version. Die Regeln für die Big-Step-Semantik ergeben sich quasi direkt aus den originalen Regeln:

$$\textsc{Skip}_{\text{TBS}}\colon \langle \texttt{skip},\ \sigma\rangle \Downarrow_1 \sigma \qquad \textsc{Ass}_{\text{TBS}}\colon \langle x\ \texttt{:=}\ a,\ \sigma\rangle \Downarrow_1 \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$$

$$\textsc{Seq}_{\text{TBS}}\colon \frac{\langle c_0,\ \sigma\rangle \Downarrow_t \sigma' \qquad \langle c_1,\ \sigma'\rangle \Downarrow_{t'} \sigma''}{\langle c_0\ \texttt{;}\ c_1,\ \sigma\rangle \Downarrow_{t+t'+1} \sigma''}$$

$$\textsc{IfTT}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \qquad \langle c_0,\ \sigma\rangle \Downarrow_t \sigma'}{\langle \texttt{if (}b\texttt{) then }c_0\texttt{ else }c_1,\ \sigma\rangle \Downarrow_{t+1} \sigma'}$$

$$\textsc{IfFF}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{ff} \qquad \langle c_1,\ \sigma\rangle \Downarrow_t \sigma'}{\langle \texttt{if (}b\texttt{) then }c_0\texttt{ else }c_1,\ \sigma\rangle \Downarrow_{t+1} \sigma'}$$

$$\textsc{WhileFF}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{ff}}{\langle \texttt{while (}b\texttt{) do }c,\ \sigma\rangle \Downarrow_1 \sigma}$$

$$\textsc{WhileTT}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \qquad \langle c,\ \sigma\rangle \Downarrow_t \sigma' \qquad \langle \texttt{while (}b\texttt{) do }c,\ \sigma'\rangle \Downarrow_{t'} \sigma''}{\langle \texttt{while (}b\texttt{) do }c,\ \sigma\rangle \Downarrow_{t+t'+1} \sigma''}$$

Oder als Prolog-Programm:

```
evalTBS(skp, S, S, 1).
evalTBS((P1;P2), S1, S3, T) :-
        evalTBS(P1, S1, S2, T1),
        evalTBS(P2, S2, S3, T2),
        T is T1 + T2 + 1.
evalTBS(Var := Aexp, S1, S2, 1) :-
        evalA(S1, Aexp, Value),
        set(S1, Var, Value, S2).
evalTBS(cond(Bexp, P1, _), S1, S2, T) :-
        evalB(S1, Bexp, tt),
        evalTBS(P1, S1, S2, T1),
        T is T1 + 1.
evalTBS(cond(Bexp, _, P2), S1, S2, T) :-
        evalB(S1, Bexp, ff),
        evalTBS(P2, S1, S2, T2),
        T is T2 + 1.
evalTBS(while(Bexp,_), S1, S1, 1) :-
        evalB(S1, Bexp, ff).
evalTBS(while(Bexp,P), S1, S3, T) :-
        evalB(S1, Bexp, tt),
        evalTBS(P, S1, S2, T1),
        evalTBS(while(Bexp,P), S2, S3, T2),
        T is T1 + T2 + 1.
```

Wenn man auch die Auswertungszeit der Ausdrücke einbeziehen möchte, bräuchte man noch folgende Funktionen:

$$\begin{aligned}
\mathcal{TA}[\![n]\!] &= 1\\
\mathcal{TA}[\![x]\!] &= 1\\
\mathcal{TA}[\![a_1\ \texttt{-}\ a_2]\!] &= \mathcal{TA}[\![a_1]\!] + \mathcal{TA}[\![a_2]\!] + 1\\
\mathcal{TA}[\![a_1\ \texttt{*}\ a_2]\!] &= \mathcal{TA}[\![a_1]\!] + \mathcal{TA}[\![a_2]\!] + 1
\end{aligned}$$

$$\begin{aligned}
\mathcal{TB}[\![\texttt{true}]\!] &= 1\\
\mathcal{TB}[\![\texttt{false}]\!] &= 1\\
\mathcal{TB}[\![a_1 \texttt{ <= } a_2]\!] &= \mathcal{TA}[\![a_1]\!] + \mathcal{TA}[\![a_2]\!] + 1\\
\mathcal{TB}[\![\texttt{not } b]\!] &= \mathcal{TB}[\![b]\!] + 1\\
\mathcal{TB}[\![b_1 \texttt{ \&\& } b_2]\!] &= \mathcal{TB}[\![b_1]\!] + \mathcal{TB}[\![b_2]\!] + 1
\end{aligned}$$

Damit sähen die Big-Step-Regeln dann so aus:

$$\textsc{Skip}_{\text{TBS}}\colon \langle \texttt{skip},\ \sigma\rangle \Downarrow_1 \sigma \qquad \textsc{Ass}_{\text{TBS}}\colon \langle x \texttt{ := } a,\ \sigma\rangle \Downarrow_{\mathcal{TA}[\![a]\!]+1} \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$$

$$\textsc{Seq}_{\text{TBS}}\colon \frac{\langle c_0,\ \sigma\rangle \Downarrow_t \sigma' \qquad \langle c_1,\ \sigma'\rangle \Downarrow_{t'} \sigma''}{\langle c_0 \texttt{;}\ c_1,\ \sigma\rangle \Downarrow_{t+t'+1} \sigma''}$$

$$\textsc{IfTT}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \qquad \langle c_0,\ \sigma\rangle \Downarrow_t \sigma'}{\langle \texttt{if (}b\texttt{) then } c_0 \texttt{ else } c_1,\ \sigma\rangle \Downarrow_{\mathcal{TB}[\![b]\!]+t+1} \sigma'}$$

$$\textsc{IfFF}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{ff} \qquad \langle c_1,\ \sigma\rangle \Downarrow_t \sigma'}{\langle \texttt{if (}b\texttt{) then } c_0 \texttt{ else } c_1,\ \sigma\rangle \Downarrow_{\mathcal{TB}[\![b]\!]+t+1} \sigma'}$$

$$\textsc{WhileFF}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{ff}}{\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow_{\mathcal{TB}[\![b]\!]+1} \sigma}$$

$$\textsc{WhileTT}_{\text{TBS}}\colon \frac{\mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \qquad \langle c,\ \sigma\rangle \Downarrow_t \sigma' \qquad \langle \texttt{while (}b\texttt{) do } c,\ \sigma'\rangle \Downarrow_{t'} \sigma''}{\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow_{\mathcal{TB}[\![b]\!]+t+t'+1} \sigma''}$$

(1b) Es gilt: $\langle c,\ \sigma\rangle \Downarrow \sigma'$ gdw. $\exists t.\ \langle c,\ \sigma\rangle \Downarrow_t \sigma'$. Beweis der einzelnen Richtungen durch Regelinduktion.

(1c) Alle drei Eigenschaften lassen sich über die Äquivalenz aus (1b) sogar ohne zusätzlichen Beweisaufwand übertragen, dann bekommt man aber recht schwache Aussagen, die keinerlei Informationen über die Ausführungszeiten beinhalten:

Schleifenabwicklungslemma
$\exists t.\ \langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow_t \sigma'$
gdw.
$\exists t.\ \langle \texttt{if (}b\texttt{) then (}c\texttt{; while (}b\texttt{) do } c\texttt{) else skip},\ \sigma\rangle \Downarrow_t \sigma'$.

Determinismus Wenn $\langle c,\ \sigma\rangle \Downarrow_t \sigma'$ und $\langle c,\ \sigma\rangle \Downarrow_{t'} \sigma''$, dann $\sigma' = \sigma'$.

Äquivalenz zu Small-Step $\exists t.\ \langle c,\ \sigma\rangle \Downarrow_t \sigma'$ gdw. $\langle c,\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma'\rangle$.

Es sind aber auch folgende stärkere Aussagen möglich – die aber von dem gewählten Schrittbegriff abhängen und hier nur für den einfachen Fall (ein Schritt = eine Ableitungsregelanwendung) angegeben sind:

Schleifenabwicklungslemma
$\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow_t \sigma'$
gdw.
$\langle \texttt{if (}b\texttt{) then (}c\texttt{; while (}b\texttt{) do } c\texttt{) else skip},\ \sigma\rangle \Downarrow_{t+1} \sigma'$.
Der Beweis dazu funktioniert analog zum Schleifenabwicklungslemma (Lem. 9) mittels „Ableitungsbaumtransformation“, nur dass hierbei auch die Zeiten mitberücksichtigt werden müssen.

Determinismus Wenn $\langle c,\ \sigma\rangle \Downarrow_t \sigma'$ und $\langle c,\ \sigma\rangle \Downarrow_{t'} \sigma''$, dann $\sigma' = \sigma''$ und $t = t'$.
Auch hier lässt sich der alte Beweis (Lem. 10) direkt übertragen: Induktion über eine Ableitung zusammen mit Regelinversion auf der anderen Ableitung.

Äquivalenz zu Small-Step Selbst wenn es verlockend wäre, aus der Zahl der Ableitungsschritte der Small-Step-Semantik die Ausführungdauer in der Big-Step-Semantik exakt berechnen zu wollen (oder umgekehrt), geht dies nicht, ohne die Struktur des Programms anzusehen – sofern man nicht von vornherein die Schrittanzahl in der Small-Step-Semantik zugrundegelegt hat.
Man kann allerdings Abschätzungen beweisen, die zeigen, dass die Ausführungszeiten zumindest asymptotisch gleich sind: Aus $\langle c,\ \sigma\rangle \Downarrow_{t_1} \sigma'$ folgt $\langle c,\ \sigma\rangle \overset{t_2}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ mit $t_2 \leq 3 \cdot t_1$ und aus $\langle c,\ \sigma\rangle \overset{t_2}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ folgt $\langle c,\ \sigma\rangle \Downarrow_{t_1} \sigma'$ mit $t_1 \leq 2 \cdot t_2 + 1$. Der Beweis folgt der Struktur der Äquivalenz der Semantiken ohne Ausführungszeiten.

2. Semantische Äquivalenz bei Small-Step (Ü)

Zwei Programme c und c' sind *äquivalent* bezüglich der Small-Step-Semantik, falls für alle Zustände σ, σ' gilt:

(i) $\langle c,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ genau dann, wenn auch $\langle c',\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ und
(ii) $\langle c,\ \sigma\rangle \overset{\infty}{\to}_1$ genau dann, wenn auch $\langle c',\ \sigma\rangle \overset{\infty}{\to}_1$.

Vergleichen Sie diesen Äquivalenzbegriff mit dem semantischen Äquivalenzbegriff für die Big-Step-Semantik.

Beweisen Sie, dass die folgenden Programme im Small-Step-Sinn äquivalent sind:

(a) `while (`b`) do` c und `if (`b`) then` c`; while (`b`) do` c `else skip`
(b) `while (true) do` c' und c`; while (true) do` c''
(c) Wenn c und c' äquivalent sind, dann auch `while (`b`) do` c und `while (`b`) do` c'.

Überlegen Sie sich einen sinnvollen Äquivalenzbegriff, für den (c) nicht gilt.

Lösung: Für unsere spezielle Small-Step-Semantik folgt (ii) bereits aus (i): Angenommen, dass $\langle c,\ \sigma\rangle \overset{\infty}{\to}_1$. Wegen der Eindeutigkeit maximaler Ableitungen (Kor. 18) gibt es dann kein σ' mit $\langle c,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$. Damit gilt wegen (i) auch $\langle c',\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ für kein σ'.

Wegen der Existenz und Eindeutigkeit maximaler Ableitung (Kor. 18) gibt es aber entweder σ' und ein blockierendes c'' mit $\langle c',\ \sigma\rangle \overset{*}{\to}_1 \langle c'',\ \sigma'\rangle$ oder $\langle c',\ \sigma\rangle \overset{\infty}{\to}_1$. Im ersten Fall ist aber $c'' = \texttt{skip}$, weil `skip` das einzige blockierende Programm ist (Fortschrittslemma 16), also gilt der zweite Fall, $\langle c',\ \sigma\rangle \overset{\infty}{\to}_1$.

Die Äquivalenz von c und c' bezüglich der Big-Step-Semantik ist definiert als: $\langle c,\ \sigma\rangle \Downarrow \sigma'$ genau dann, wenn auch $\langle c',\ \sigma\rangle \Downarrow \sigma'$ für alle σ, σ'.

Da aber $\langle c,\ \sigma\rangle \Downarrow \sigma'$ und $\langle c,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ äquivalent sind (Kor. 23), sind somit diese beiden Äquivalenzbegriffe die gleichen.

(a) Da Big-Step- und Small-Step-Äquivalenz gleich sind, folgt dies direkt aus dem Schleifenabwicklungslemma (Lem. 9) oder auch einfach nach Regelinversion auf der linken Seite.
(b) Beide Programme terminieren nie, damit sind sie äquivalent. Nichtterminationsbeweise:
   i. Angenommen, ein Programm der Form `while (true) do` c terminiert für (mindestens) einen Anfangszustand. Seien also σ, σ' die Zustände, unter denen die Schleife am schnellsten terminiert, also so dass $\langle \texttt{while (true) do}\ c,\ \sigma\rangle \overset{n}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle$ mit n minimal unter allen solchen Ableitungen. Dann gilt:

$$
\begin{aligned}
& \langle \texttt{while (true) do}\ c,\ \sigma\rangle \\
\to_1\ & \langle \texttt{if (true) then}\ c\texttt{; while (true) do}\ c\ \texttt{else skip},\ \sigma\rangle \\
\to_1\ & \langle c\texttt{; while (true) do}\ c,\ \sigma\rangle \\
\overset{n-2}{\to}_1\ & \langle \mathit{skip},\ \sigma'\rangle
\end{aligned}
$$

Nach dem Zerlegungslemma für Sequenz (Lem. 20) gibt es i, j und σ^* mit $\langle c,\ \sigma\rangle \xrightarrow{i}_1 \langle \texttt{skip},\ \sigma^*\rangle$, $\langle \texttt{while (true) do } c,\ \sigma^*\rangle \xrightarrow{j}_1 \langle \texttt{skip},\ \sigma'\rangle$ und $i + j + 1 = n - 2$. Damit gilt $j < n$, im Widerspruch zur Annahme.

ii. Die Hintereinanderausführung von Programmen, von denen eines nicht terminiert, terminiert auch nicht. Dies folgt aus dem Zerlegungslemma für Sequenz (Lem. 20) mit Widerspruchsbeweis.

(c) Es genügt aus Symmetriegründen, (i) nur in eine Richtung zu zeigen.
Wir zeigen durch vollständige Induktion nach n dass für beliebiges σ aus der Ableitung $\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \xrightarrow{n}_1 \langle \texttt{skip},\ \sigma'\rangle$ die Ableitung $\langle \texttt{while (}b\texttt{) do } c',\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma'\rangle$ folgt.

Damit erhalten wir als Induktionsannahme dass für alle $m < n$ und σ aus der Ableitung $\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \xrightarrow{m}_1 \langle \texttt{skip},\ \sigma'\rangle$ folgt, dass $\langle \texttt{while (}b\texttt{) do } c',\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma'\rangle$ gilt.
Falls $\mathcal{B}[\![b]\!]\sigma = \mathbf{ff}$, folgt die Behauptung trivial mit den Regeln WhileSS und IfFFSS in zwei Schritten. Sei also $\mathcal{B}[\![b]\!]\sigma = \mathbf{tt}$. Durch Regelinversion erhält man:

$$\begin{aligned}&\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \to_1 \langle \texttt{if (}b\texttt{) then } c\texttt{; while (}b\texttt{) do } c \texttt{ else skip},\ \sigma\rangle\\ \to_1\ &\langle c\texttt{; while (}b\texttt{) do } c,\ \sigma\rangle \xrightarrow{n-2}_1 \langle \texttt{skip},\ \sigma'\rangle\end{aligned}$$

Nach dem Zerlegungslemma für Sequenz (Lem. 20) gibt es damit i, j und σ^* mit $\langle c,\ \sigma\rangle \xrightarrow{i}_1 \langle \texttt{skip},\ \sigma^*\rangle$, $\langle \texttt{while (}b\texttt{) do } c,\ \sigma^*\rangle \xrightarrow{j}_1 \langle \texttt{skip},\ \sigma'\rangle$ und $i + j + 1 = n - 2$. Da $j < n$ folgt nach der Induktionsannahme, dass $\langle \texttt{while (}b\texttt{) do } c',\ \sigma^*\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma'\rangle$. Wegen $\langle c,\ \sigma\rangle \xrightarrow{i}_1 \langle \texttt{skip},\ \sigma^*\rangle$ gilt auch $\langle c,\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma^*\rangle$ und deswegen mit der semantischen Äquivalenz von c und c' auch $\langle c',\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip},\ \sigma^*\rangle$. Somit ergibt sich folgende maximale Ableitungfolge (unter Verwendung des Liftinglemma 19):

$$\begin{aligned}&\langle \texttt{while (}b\texttt{) do } c',\ \sigma\rangle \to_1 \langle \texttt{if (}b\texttt{) then } c'\texttt{; while (}b\texttt{) do } c' \texttt{ else skip},\ \sigma\rangle\\ \to_1&\langle c'\texttt{; while (}b\texttt{) do } c',\ \sigma\rangle \xrightarrow{*}_1 \langle \texttt{skip; while (}b\texttt{) do } c',\ \sigma^*\rangle \to_1 \langle \texttt{while (}b\texttt{) do } c',\ \sigma^*\rangle\\ \xrightarrow{*}_1&\langle \texttt{skip},\ \sigma'\rangle\end{aligned}$$

und damit die Behauptung.

In der Vorlesung wurde die Äquivalenz von Programmen bezüglich einer Menge V von Variablen eingeführt (Definition 24). Wenn $V = \{\texttt{tmp1}\}$, so sind die beiden Programme $c =$ `a := a - 1; tmp1 := 0` und $c' =$ `a := a - 1; tmp1 := a` bezüglich $\mathsf{Var} \setminus V$ äquivalent, nicht aber `while (0 <= tmp1) do` c und `while (0 <= tmp1) do` c'.