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 6: Erweiterungen zu While Besprechung: 06.06.2017

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

(a) c_1 `or` c_2 und c_2 `or` c_1 sind äquivalent bzgl. der Big-Step-Semantik.

(b) c_1 `or` c_2 und c_2 `or` c_1 sind äquivalent bzgl. der Small-Step-Semantik.

(c) `x := 0; y := 0; while (y == 0) do (x := x + 1 or y := 1)` terminiert immer.

(d) `(while (`b`) do` c_1`) or (while (`b`) do` c_2`)` und `while (`b`) do (`c_1 `or` c_2`)` sind äquivalent bzgl. der Big-Step-Semantik.

(e) `x := 5 or x := 6` und `x := 5 || x := 6` sind semantisch äquivalent.

(f) c_1 `||` `(`c_2 `||` c_3`)` $= $ `(`c_1 `||` c_2`)` `||` c_3

(g) c_1 `||` c_2 und c_2 `||` c_1 sind äquivalent bzgl. der Small-Step-Semantik.

(h) Die Big-Step-Semantik von While_B ist nicht deterministisch.

(i) Nach Aufführung von `{ var x = 1; y := x + 1; { var y = 3; x := y + 2; { var x = 6; z := x + y }; y := z }; z := x + y + z }` hat `z` den Wert 24.

(j) `{ var z = 142; { var x = x + 1; z := x }; x := z - 1 }` ist semantisch äquivalent zu `skip`.

2. Blöcke und Parallelität (H)

In dieser Aufgabe seien die Erweiterungen zur Parallelität While_{PAR} und zu lokalen Variablen mittels Blöcken While_B kombiniert. Was sind die möglichen Endzustände des folgenden Programms in der kombinierten Small-Step-Semantik für den Anfangszustand $[\mathtt{x} \mapsto 1]$?

```
({ var y = 1; x := x + 1; y := y + 1; x := x + 2; y := y + 2; z := y }) ||
({ var y = 1; x := x * 3; y := y * 3; x := x * 4; y := y * 4; z := y })
```

3. Exceptions, break und continue (H)

Exceptions wie in der Vorlesung vorgestellt, können verwendet werden, um `break` und `continue` für Schleifen zu simulieren. `break` beendet sofort die innerste umgebende Schleife, `continue` beendet den aktuellen Schleifendurchlauf und setzt mit der Prüfung der Schleifenbedingung fort.

Beschreiben Sie, wie sich While mit `break` und `continue` als Quellcodetransformation auf While mit Exceptions abbilden lässt. Wie sähe eine Implementierung von `break` mit Label aus?

4. Goto und Small-Step-Semantik mit Continuations (Ü)

In dieser Aufgabe soll eine Small-Step-Semantik für While mit `goto` definiert werden. Dazu sei Lab eine Menge von Labels, die typischerweise mit l bezeichnet werden. Mit diesen können beliebige Stellen im Programm markiert werden, dafür erweitern wir die Syntax von While:

$\mathsf{Com}\quad c \quad ::= \quad l\texttt{:} \mid \texttt{goto}\ l \mid \ldots$

(a) Unsere bisherige Small-Step-Semantik ist nicht geeignet, `goto` sauber abzubilden: Die Regel $\textsc{Seq1}_{\text{SS}}$ für $c_1\texttt{;}\ c_2$ erlaubt es nicht, dass c_1 wegspringt. Daher stellen wir die Semantik auf Continuations um. Ein Zustand unserer Semantik ist nun $\langle cs,\ \sigma\rangle$ und besagt, dass statt einem einzelnen Programm c die Liste von Programmen cs auszuführen ist.

Geben Sie die Regeln für While in dieser Semantik an. Dies ist ohne rekursive Regeln wie $\textsc{Seq1}_{\text{SS}}$ möglich! Was sind die blockierten Zustände?

(b) Ergänzen Sie diese Small-Step-Semantik um Regeln für $l\texttt{:}$ und `goto` l. Da ein Sprung irgendwo im Program landen kann, müssen alle Small-Step-Regeln nun auch das komplette Programm durchschleifen. Da es nicht verändert wird, schreibt man es vor die Relation: $c \vdash \langle cs_1,\ \sigma_1\rangle \to_1 \langle cs_2,\ \sigma_2\rangle$ besagt, dass während der Auswertung des Programms c die Programmfragmente cs_1 im Zustand σ_1 in einem Schritt zu cs_2 im Zustand σ_2 ausgewertet werden.

Für die Regel für `goto` werden Sie eine Funktion benötigen, die in einem Programm c nach dem Label l sucht und ein Programm $\mathcal{L}_l(c)$ zurückgibt, das die Ausführung von c ab dem Label l beschreibt. Definieren Sie diese Funktion. Sie können dabei das Prädikat $l \in c_1$ verwenden, das wahr ist, wenn im Programm(fragment) c_1 das Label l vorkommt. Gehen Sie davon aus, dass in jedem Programm jedes Label höchstens einmal gesetzt wurde, und beachten Sie nur Labels, die auch im Programm vorkommen.

(c) Gegeben sei das folgende While-Programm c. Geben Sie $\mathcal{L}_{\texttt{lab}}(c)$ und die Ableitungsfolge von c in einem Zustand σ an.

```
y := 0; (while (y == 1) do (lab:; y := 2)); if (y == 0) then goto lab else skip
```