

Semantik von Programmiersprachen – SS 2015

<http://pp.ipd.kit.edu/lehre/SS2015/semantik>

Blatt 6: Erweiterungen zu While

Besprechung: 18.05.2015

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

- (a) $c_1 \text{ or } c_2$ und $c_2 \text{ or } c_1$ sind äquivalent bzgl. der Big-Step-Semantik.
- (b) $c_1 \text{ or } c_2$ und $c_2 \text{ or } c_1$ sind äquivalent bzgl. der Small-Step-Semantik.
- (c) $x := 0; y := 0; \text{ while } (y == 0) \text{ do } (x := x + 1 \text{ or } y := 1)$ terminiert immer.
- (d) $(\text{while } (b) \text{ do } c_1) \text{ or } (\text{while } (b) \text{ do } c_2)$ und $\text{while } (b) \text{ do } (c_1 \text{ or } c_2)$ sind äquivalent bzgl. der Big-Step-Semantik.
- (e) $x := 5 \text{ or } x := 6$ und $x := 5 \parallel x := 6$ sind semantisch äquivalent.
- (f) $c_1 \parallel (c_2 \parallel c_3) = (c_1 \parallel c_2) \parallel c_3$
- (g) $c_1 \parallel c_2$ und $c_2 \parallel c_1$ sind äquivalent bzgl. der Small-Step-Semantik.
- (h) Die Big-Step-Semantik von While_B ist nicht deterministisch.
- (i) Nach Ausführung von $\{ \text{var } x = 1; y := x + 1; \{ \text{var } y = 3; x := y + 2; \{ \text{var } x = 6; z := x + y \}; y := z \}; z := x + y + z \}$ hat z den Wert 24.
- (j) $\{ \text{var } z = 142; \{ \text{var } x = x + 1; z := x \}; x := z - 1 \}$ 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 $[x \mapsto 1]$?

$(\{ \text{var } y = 1; x := x + 1; y := y + 1; x := x + 2; y := y + 2; z := y \}) \parallel$
 $(\{ \text{var } y = 1; x := x * 3; y := y * 3; x := x * 4; y := y * 4; z := y \})$

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`:

`Com` $c ::= l : | \text{goto } l | \dots$

- (a) Unsere bisherige Small-Step-Semantik ist nicht geeignet, `goto` sauber abzubilden: Die Regel `SEQ1SS` für $c_1; c_2$ erlaubt es nicht, dass c_1 wegspringt. Daher stellen wir die Semantik auf Continuations um. Ein Zustand unserer Semantik ist nun $\langle c, cs, \sigma \rangle$ und besagt, dass nach dem Programm c noch die Liste von Programmen cs auszuführen sind. Geben Sie die Regeln für `While` in dieser Semantik an. Dies ist ohne rekursive Regeln wie `SEQ1SS` möglich! Was sind die blockierten Zustände?
- (b) Ergänzen Sie diese Small-Step-Semantik um Regeln für $l:$ 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: $c \vdash \langle c_1, cs_1, \sigma_1 \rangle \rightarrow_1 \langle c_2, cs_2, \sigma_2 \rangle$ besagt, dass während der Auswertung des Programms c das Programmfragment c_1 im Zustand σ_1 und mit Continuations cs in einem Schritt zu c_2 im Zustand σ_2 mit Continuations cs_2 ausgewertet wird. 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 $\mathcal{L}_{\text{lab}}(c)$ 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
```