Karlsruher Institut für Technologie

Lehrstuhl für Programmierparadigmen

Joachim Breitner
breitner@kit.edu

Denis Lohner
denis.lohner@kit.edu

Semantik von Programmiersprachen – SS 2015

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

Blatt 7: Prozeduren Besprechung: 01.06.2015

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

(a) $[(\texttt{p},\ \texttt{i := i * 1; call p})] \vdash \langle \texttt{call p},\ \sigma\rangle \Downarrow \sigma$

(b) $([(\texttt{p},\ \texttt{skip})],\ \texttt{call q})$ ist ein Programm der Sprache While_{PROC}.

(c) $\langle \texttt{skip},\ \sigma\rangle$ ist die einzige blockierte Konfiguration der Small-Step-Semantik für While_{PROC}.

(d) Wenn $P \vdash \langle c,\ \sigma\rangle \overset{\infty}{\to}_1$, dann enthält c eine `while`-Schleife.

(e) Sei $P \equiv [(\texttt{p},\ \texttt{y},\ \texttt{x := 4; result := 5 * y})]$, $E_0 \equiv [\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1]$ und $s \equiv [\mathsf{next} \mapsto 2]$.
Wenn $P, E_0, E_0 \vdash \langle \texttt{\{ var x = 3; y <- call p(x); y := y + x \}},\ s\rangle \Downarrow s'$,
dann $s'(E_0(\mathsf{x})) = 4$ und $s'(E_0(\mathsf{y})) = 18$.

(f) Wenn $P, E_0, E \vdash \langle c,\ s\rangle \Downarrow s'$, dann $s'(\mathsf{next}) = s(\mathsf{next})$.

2. Small-Step-Semantik für Prozeduren mit Parametern (H)

Für Prozeduren mit einem Parameter (While_{PROCP}) gibt es folgende Vorschläge für eine Aufrufregel der Small-Step-Semantik:

(a)
$$\frac{(p, x, c) \in P}{P \vdash \langle y\ \texttt{<- call}\ p(a),\ \sigma\rangle \to_1 \langle x\ \texttt{:=}\ a;\ c;\ y\ \texttt{:= result},\ \sigma\rangle}$$

(b)
$$\frac{(p, x, c) \in P}{P \vdash \langle y\ \texttt{<- call}\ p(a),\ \sigma\rangle \to_1 \langle \texttt{\{ var result = 0; \{ var}\ x\ \texttt{=}\ a;\ c\ \texttt{\}; }\ y\ \texttt{:= result \}},\ \sigma\rangle}$$

(c)
$$\frac{(p, x, c) \in P}{P \vdash \langle y\ \texttt{<- call}\ p(a),\ \sigma\rangle \to_1 \langle \texttt{\{ var}\ x\ \texttt{=}\ a\texttt{; \{ var result = 0;}\ c;\ y\ \texttt{:= result \} \}},\ \sigma\rangle}$$

(d)
$$\frac{(p, x, c) \in P \qquad z \text{ nicht verwendet in } P \text{ und } a}{P \vdash \langle y\ \texttt{<- call}\ p(a),\ \sigma\rangle \to_1 \langle \texttt{\{ var}\ z\ \texttt{=}\ 0\texttt{; \{ var}\ x\ \texttt{=}\ a\texttt{; \{ var result = 0;}\ c;\ z\ \texttt{:= result \} \};}\ y\ \texttt{:=}\ z\ \texttt{\}},\ \sigma\rangle}$$

Untersuchen Sie, in wie weit diese Regeln call-by-value Prozedurparameter und Rückgabewerte korrekt abbilden. Begründen Sie Ihre Antwort; ggf. mit Beispielprogrammen, die entsprechende Defizite aufzeigen. Welche dieser Regeln modellieren statische Variablenbindung?

3. Call by reference (Ü)

Die in der Vorlesung vorgestelle Big-Step-Semantik für Prozeduren mit einem Parameter wertet den übergebenen Parameter beim Aufruf aus und übergibt nur den Wert an die aufgerufene Prozedur. Insbesondere bleibt der Wert einer Variablen, die als Parameter übergeben wird, im

aufrufenden Kontext unverändert. Diese Parameterübergabeart heißt *call by value*. Daneben gibt es auch noch *call by reference*, bei der Änderungen am Parameterwert in der aufgerufenen Prozedur auch nach dem Ende des Aufrufs in der aufrufenden Prozedur sichtbar sind. Passen Sie in dieser Aufgabe die Big-Step-Semantik von While_{PROCP} entsprechend an:

(a) Passen Sie die Syntax-Definition von While_{PROCP} an, so dass nur noch Variablen als Parameter verwendet können.

(b) Ändern Sie die Big-Step-Semantik so, dass Parameter immer als call by reference übergeben werden.

(c) Ändern Sie die Regeln erneut so, dass auch die Rückgabewertvariable mit call by reference übergeben wird.

(d) Finden Sie ein Programm, bei dem sich das Verhalten in beiden Varianten von call by reference und call by value unterscheidet. Belegen Sie dies durch die Ableitungsbäume.