Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Clemens Ballarin

Vorlesung Semantik
Sommersemester 2007
Übungsblatt 3
Besprechung am 15. Mai 2007

Aufgabe 1 (H) (*Sichtbarkeitsregeln*)

Gegeben sei folgendes Programm

$\{$ **var** $x := 0$;
proc $(p = x := x + 1)$;
proc $(q =$ **call** $p; y := x + 1)$;
proc $(r = \{$**proc**$(p = x := x * 2);$ **call** $q\})$;
var $x := 3$;
call $r; z := x$;
proc $(q =$ **call** $p; y := x - 1)$;
call $r; y := x + z$ $\}$

Hierbei wurden die meisten Block-Klammern $\{\ldots\}$ weggelassen. Es gilt die Konvention, dass ein Variablen- oder Prozedurblock sich immer so weit als möglich erstreckt.

Geben Sie jeweils die Belegung der Variablen während des Programmlaufs und die Belegung von y am Ende der Ausführung an für:

a) Dynamische Sichtbarkeitsregeln für Variablen und Prozeduren

b) Statische Sichtbarkeitsregeln für Variablen und Prozeduren

Aufgabe 2 (H) (*Regelinduktion*)

In dieser Aufgabe betrachten wir Listen (über einem beliebigen Elementtyp). Die cons-Operation wird mit $x \cdot xs$ bezeichnet, $|xs|$ ist die Länge von xs und xs_i das $i$te Element.

Permutationen von Listen werden durch die folgenden vier Regeln definiert.

$$([], []) \in \textbf{Perm} \quad \text{(Nil)} \qquad (x \cdot y \cdot l, y \cdot x \cdot l) \in \textbf{Perm} \quad \text{(Swap)}$$

$$\frac{(xs, ys) \in \textbf{Perm}}{(z \cdot xs, z \cdot ys) \in \textbf{Perm}} \quad \text{(Cons)} \qquad \frac{(xs, ys) \in \textbf{Perm} \qquad (ys, zs) \in \textbf{Perm}}{(xs, zs) \in \textbf{Perm}} \quad \text{(Trans)}$$

Dabei enthält die definierte Menge **Perm** Paare von Listen, die sich nur durch die Reihenfolge der Elemente unterscheiden. Mit $\cdot$ wird der *Cons*-Operator bezeichnet, d.h. $x \cdot xs$ ist die Liste bestehend aus dem ersten Element x und der Restliste xs. Formulieren Sie die dazugehörige Induktionsregel und beweisen Sie damit die folgenden Aussagen:

a) Für $(xs, ys) \in$ **Perm** gilt: xs und ys haben die gleiche Länge.

b) Für $(xs, ys) \in$ **Perm** gilt: es gibt eine Permutation π der Zahlen $1 \ldots |xs|$, so dass $xs_i = ys_{\pi(i)}$ für alle $i = 1 \ldots |xs|$.

Aufgabe 3 (Ü) (*Äquivalenz von Big-Step- und Small-Step-Semantik*)

Für die Sprache WHILE wurde in der Vorlesung eine alternative operationelle Semantik definiert. Hier soll nun eine Richtung der Äquivalenz von Big-Step-Semantik (Auswertungssemantik) $\rightarrow$ und Small-Step-Semantik (Transitionssemantik) $\rightarrow_1$ gezeigt werden. Beweisen Sie:

$$\forall s, \sigma, \sigma'. \, [\langle s, \sigma \rangle \rightarrow \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow_1^* \sigma']$$

Aufgabe 4 (P) (*Small-Step-Semantik in PROLOG*)

a) Neben *while* und *for* gibt es in imperativen Programmiersprachen oft auch ein *repeat*-Konstrukt, bei dem die Abbruchbedingung erst nach Ausführung des Schleifenkörpers getestet wird.

Die Syntax der Sprache WHILE aus der Vorlesung sei nun um

repeat s **until** b

erweitert.

Geben Sie die entsprechenden Regeln für die Small-Step-Semantik an. Die neuen Regeln sollen dabei nicht auf das **while** der Sprache zurückgreifen.

b) Die Small-Step-Semantik der erweiterten WHILE-Sprache soll in PROLOG programmiert werden. Es gelten die gleichen Vereinfachungsregeln für die abstrakte Syntax wie in Aufgabe 5, Blatt 2. Geben Sie Ihr Programm und ein Protokoll der Beispiele in der Datei `dialog.txt` ab.

Einen Rahmen zur Lösung dieser Programmieraufgabe finden Sie im Verzeichnis

`/usr/proj/semantik/prog/prolog/small-step/`