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

Vorlesung Semantik
Wintersemester 2002/03
Übungsblatt 4
Besprechung am 14. November 2002

Aufgabe 1 (H) (*Ausnahmebehandlung*)

Die Sprache WHILE soll um Exception-Handling erweitert werden. Dazu führen wir zwei weitere Statements ein:

$$\textbf{raise} \quad \text{und} \quad s_1 \ \textbf{handle}\ s_2$$

Wenn in s_1 mit **raise** eine Exception erzeugt wird, soll das Statement s_1 **handle** s_2 den Exception-Handler s_2 ausführen. Ansonsten wird s_2 ignoriert.

a) Passen Sie die Small-Step-Semantik für WHILE an die neuen Konstrukte an.

b) Zeigen Sie nun, dass Sie obige Regeln sinnvoll gewählt haben. Formulieren Sie dazu formal, dass in ihrer Semantik $\langle \textbf{raise}, \sigma \rangle$ die einzige blockierte *(stuck)* Konfiguration ist. Beweisen Sie ihre Behauptung. Gehen Sie dabei analog zum entsprechenden Beweis in der Vorlesung vor.

Aufgabe 2 (Ü) (*Nicht-Determinismus*)

Gegeben sei folgendes Programm in der nicht-deterministischen Variante von WHILE:

$$x := -1;\ \textbf{while}\ x \leq 0\ \textbf{do}\ (x := x - 1\ \textbf{or}\ x := -1 * x)$$

a) Beschreiben Sie die Menge der möglichen Endzustände, die bei Anwendung der Big-Step-Semantik, ausgehend vom Startzustand σ, erreicht werden können.

b) Beschreiben Sie die Menge der Ableitungen, die aus der Anwendung der Small-Step-Semantik resultieren können.

c) Kann man Big-Step-Semantik und Small-Step-Semantik für die nicht-deterministische WHILE-Sprache als äquivalent bezeichnen?

Aufgabe 3 (Ü) (*Parallele Programme*)

Zusätzlich zum $\|$-Konstrukt soll die parallele Variante von WHILE um die Möglichkeit erweitert werden, bestimmte Statements atomar auszuführen, um dem Programmierer ein Mittel zur Synchronisation an die Hand zu geben:

$$\textbf{protect}\ s$$

Erweitern Sie die Small-Step-Semantik von WHILE entsprechend.

Aufgabe 4 (Ü) (*Sichtbarkeitsregeln*)

Gegeben sei folgendes Programm

```
begin var x := 0;
        proc p is x := x + 1 end proc
        proc q is call p; y := x + 1 end proc
        proc r is proc p is x := x * 2 end proc
                  call q
        end proc
        begin var x := 3;
                call r; z := x;
                proc q is call p; y := x − 1 end proc
                call r; y := x + z
        end var
end var
```

Die hier verwendete Syntax entspricht der Syntax aus der Vorlesung wie folgt:

$$\textbf{begin var } x := a; s \textbf{ end var} \quad \equiv \quad \{\textbf{var } x := a; s\}$$
$$\textbf{proc } p \textbf{ is } s_1 \textbf{ end proc } s_2 \quad \equiv \quad \{\textbf{proc } (p = s_1); s_2\}$$

Geben Sie jeweils die Belegung der Variablen y am Ende der Ausführung an für:

a) Dynamische Sichtbarkeitsregeln für Variablen und Prozeduren

b) Statische Sichtbarkeitsregeln für Prozeduren und dynamische für Variablen

c) Statische Sichtbarkeitsregeln für beide

Aufgabe 5 (H/P) (*Ausnahmebehandlung*)

a) (H) Passen Sie die Big-Step-Semantik für WHILE an die Konstrukte **raise** und **handle** an.

b) (H) Kann man Big-Step-Semantik und Small-Step-Semantik für die erweiterte WHILE-Sprache als äquivalent bezeichnen? Begründen Sie ihre Antwort.

c) (P) Erweitern Sie ihre Lösungen aus Blatt 2 und 3 jeweils um die Konstrukte **raise** und **handle**. Sie müssen dazu auch den Parser für WHILE erweitern. Testen Sie ihre Programme und geben Sie jeweils zwei Testläufe ab, in denen die beiden neuen Konstrukte vorkommen.

Allgemeines

- Ab 11. November findet die Vorlesung am Montag in Raum 00.07.011 statt.
- Ab dem nächsten Übungsblatt werden Programmieraufgaben in der funktionalen Sprache *Gofer* gestellt. Diese Sprache ist ähnlich zu Haskell und ML. Zu Gofer wird kein Einführungskurs gehalten. Einführungen finden sie unter den Links:

  http://www4.in.tum.de/lehre/vorlesungen/info1ws01/doc/goferdoc.ps
  http://www.cs.olemiss.edu/~hcc/reports/gofer_notes.pdf