Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Gerwin Klein

WS 2001/2002
Übungsblatt 4
5. 11. 2001

Übungen zur Semantik von Programmiersprachen

Aufgabe 1 (H) (*Exceptions*) (Abgabetermin: Dienstag, 12.11.01)

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

$$\textbf{raise} \quad \text{und} \quad s \ \textbf{handle} \ s$$

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 $\langle \text{raise}, \sigma \rangle$ 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 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) (*Atomare Statements*)

Zusätzlich zum **par** 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) (*Scope Rules*)

Gegeben sei folgendes Programm

$$
\begin{array}{l}
\textbf{begin var } x := 0; \\
\quad \textbf{proc } p \textbf{ is } x := x + 1; \\
\quad \textbf{proc } q \textbf{ is call } p;\ y := x + 1; \\
\quad \textbf{proc } r \textbf{ is proc } p \textbf{ is } x := x * 2; \\
\qquad\qquad \textbf{call } q; \\
\quad \textbf{begin var } x := 3; \\
\qquad \textbf{call } r;\ z := x; \\
\qquad \textbf{proc } q \textbf{ is call } p;\ y := x - 1; \\
\qquad \textbf{call } r;\ y := x + z \\
\quad \textbf{end} \\
\textbf{end}
\end{array}
$$

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 (P) (*Exceptions*)

(Abgabe: Dienstag, 12.11.01, 12 Uhr an kleing@in.tum.de)

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

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

(c) 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 beide neuen Konstrukte vorkommen.