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

**Vorlesung Semantik**
**Sommersemester 2005**
Übungsblatt 4
Besprechung am 13. Mai 2005

## 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) Es soll nun gezeigt werden, dass die Regeln für die obigen Statements sinnvoll gewählt wurden. Formulieren Sie zunächst formal, dass in Ihrer Semantik $\langle \textbf{raise}, \sigma \rangle$ die einzige blockierte (*stuck*) Konfiguration ist.

c) Beweisen Sie die Behauptung. Gehen Sie dabei analog zum entsprechenden Beweis in der Vorlesung vor. Beschränken Sie sich auf die Fälle **skip**, $s_1; s_2$, **raise** und $s_1$ **handle** $s_2$.

## Aufgabe 2 (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 $\{\dots\}$ 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 3 (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 $\sigma$, 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 4 (Ü) (*Compiler für arithmetische Ausdrücke*)**

Für die arithmetischen Ausdrücke der Sprache WHILE sollen ein Compiler angegeben werden. Der Compiler nimmt einen Ausdruck und übersetzt ihn in eine Liste von Instruktionen einer Stackmaschine. Auf der Stackmaschine stehen folgende Instruktionen zur Verfügung:

- `Const c`: Lege die Konstante $c$ auf den Stack.
- `Load a`: Lade den Inhalt der Variablen $a$ auf den Stack.
- `Apply f`: Wende die binäre Funktion $f$ auf die beiden obersten Stackelemente an und ersetze sie durch das Ergebnis.

Die Maschine, auf der die übersetzten Ausdrücke abgearbeitet werden sollen, nimmt eine Liste von Instruktionen, einen (anfangs leeren) Stack und einen Speicherbereich mit Variablenbelegungen als Argumente. Auf die Variablenbelegungen kann mit Hilfe von `Load` zugegriffen werden. Die Stackmaschine liefert einen Resultatkeller als Ergebnis. Am Ende der Abarbeitung soll nur noch das Ergebnis der Berechnung auf dem Stack liegen.

a) Geben Sie den Compiler für arithmetische Ausdrücke an.

b) Formulieren Sie die Aussage, dass der Compiler korrekt ist, formal.

c) Beweisen Sie die Korrektheitsaussage.