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

Vorlesung Semantik
Sommersemester 2004
Übungsblatt 4
Besprechung am 19. Mai 2004

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) (*Compiler für arithmetische Ausdrücke*)

Für die arithmetischen Ausdrücke der Sprache WHILE sollen ein Computer 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.

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

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

Hinweis: Überlegen Sie, wie man die Ausnahme in der Big-Step-Semantik durch die Ableitungsregeln weitergeben kann.

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**. Im Programmrahmen

```
/usr/proj/semantik/prog/prolog/raise/
```

finden sie einen entsprechend erweiterten Parser für WHILE. Testen Sie ihre Programme und geben Sie jeweils zwei Testläufe ab, in denen die beiden neuen Konstrukte vorkommen.