

### Aufgabe 1 (H) (Ausnahmebehandlung)

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

**raise** und  $s_1$  **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.

- Passen Sie die Small-Step-Semantik für WHILE an die neuen Konstrukte an.
- Es soll nun gezeigt werden, dass die Regeln für die obigen Statements sinnvoll gewählt wurden. Formulieren Sie zunächst formal, dass in Ihrer Semantik  $\langle \mathbf{raise}, \sigma \rangle$  die einzige blockierte (*stuck*) Konfiguration ist.
- 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) (Eigenschaften des Operators $\hat{R}$ )

Für eine Menge  $R \subseteq \mathcal{P}(M) \times M$  von Regelinstanzen (mit endlich vielen Prämissen) über der Grundmenge  $M$  ist der Operator  $\hat{R} : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$  definiert wie folgt:

$$\hat{R}(B) = \{y \mid \exists X \subseteq B. (X, y) \in R\}$$

- Beweisen Sie den Satz:

Die Menge  $B$  ist  $R$ -abgeschlossen genau dann, wenn  $\hat{R}(B) \subseteq B$ .

- Beweisen Sie die *Monotonie* von  $\hat{R}$  bzgl. der Mengeninklusion, d.h.:

$$A \subseteq B \implies \hat{R}(A) \subseteq \hat{R}(B)$$

- Beweisen Sie:

$$\bigcup_{n \in \mathbb{N}} \hat{R}(B_n) = \hat{R}\left(\bigcup_{n \in \mathbb{N}} B_n\right)$$

für alle  $\omega$ -Ketten  $B_0 \subseteq B_1 \subseteq \dots$

### Aufgabe 3 (Ü) (*Compiler für arithmetische Ausdrücke*)

Für die arithmetischen Ausdrücke der Sprache WHILE soll 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 (anfängs 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 4 (P) (*Ausnahmebehandlung*)

Erweitern Sie die Small-Step-Semantik von WHILE um die Konstrukte **raise** und **handle**. Im Programmrahmen

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

finden sie einen entsprechend erweiterten Parser für WHILE. Beachten Sie, dass der Parser für  $s_1$  **handle**  $s_2$  die Syntax `try  $s_1$  handle  $s_2$  end` erwartet. Testen Sie ihr Programm und geben Sie die Prolog-Datei und die geforderten Testläufe ab.