Übungen zur Semantik von Programmiersprachen

Aufgabe 1 (H) (*Small-Step-Semantik und Typsicherheit*) **Abgabetermin: 3.11.2001**

In der Vorlesung wurde eine Big-Step-Semantik für $WHILE_T$ angegeben. Entwickeln Sie eine entsprechende Small-Step-Semantik und beweisen Sie bezüglich dieser Semantik die Typsicherheit von $WHILE_T$.

Aufgabe 2 (Ü) (*Arrays*)

Die Sprache $WHILE_T$ soll um Arrays erweitert werden.

- Wie muss sich die Syntax für Ausdrücke ändern?
- Geben Sie eine angepasste Big-Step-Semantik für $WHILE_T$ mit Arrays an.
- Geben Sie die Typregeln für die erweiterte Sprache an.

Aufgabe 3 (Ü) (*Bytecode Verification*)

In dieser Aufgabe geht es um ein Typsystem für eine Maschinsprache. Wir betrachten eine abstrakte Stack-Maschine ähnlich zu der aus Aufgabe 4, Blatt 5. Sie besitzt einen Operanden-Stack, auf dem Werte der Typen **int** und **real** abgelegt werden können. Der Befehlssatz besteht aus den folgenden Instruktionen:

- **Const c**: Lege die Konstante c auf den Stack. Die Konstante c kann dabei sowohl vom Typ **int** als auch **real** sein).
 - **Load a**: Lade den Inhalt der Adresse a auf den Stack. Der Inhalt von a kann dabei sowohl vom Typ **int** als auch **real** sein).
 - **IAdd**: Addiere die beiden obersten Stack-Elemente und schreibe das Ergebnis zurück. Die beiden obersten Stack-Elemente müssen vom Typ **int** sein.
 - **RAdd**: Analog zu **IAdd**, die beiden obersten Stack-Elemente müssen aber vom Typ **real** sein.
 - **ToReal**: Konvertiert das oberste Stack-Element vom Typ **int** in den Typ **real**.
- Definieren Sie Typregeln mit Hilfe von Ausdrücke der Form $\Gamma \vdash is : stk$, die in einem Kontext $\Gamma : loc \rightarrow typ$ überprüfen, ob die Ausführung einer Instruktionssequenz is mit einem Anfangsstack stk erfolgreich sein wird. Die Einträge in stk sind dabei nicht Werte, sondern Typen von Werten.
 - Wenden sie Ihre Regeln an, um im Kontext $\Gamma = \{y \mapsto \mathbf{real}\}$ die Typen der Stack-einträge für folgendes Beispielprogramm (gestartet im leeren Stack) zu berechnen:
[Const 2, Const 3, IAdd, ToReal, Load y, RAdd]

Aufgabe 4 (H/P) (*Subtypen*) **Abgabetermin: 3.11.2001 an kleing@in.tum.de**

Wir erweitern die Sprache $WHILE_T$ um den Typ **real** (neben den Typen **bool** und **int**). Hierbei soll **int** ein *Subtyp* von **real** werden, d.h. alle Werte vom Typ **int** sind auch Werte des Typs **real**.

- (a) (H)
Geben Sie die Typregeln für das neue Typsystem von $WHILE_T$ an (sowohl für Expressions als auch für Statements soweit sie sich ändern).
- (b) (P)
Implementieren Sie eine Funktion in Gofer, die aus Deklarationsinformationen über Variablen den Typ eines Ausdrucks der Sprache $WHILE_T$ berechnet.
- (c) (P)
Erweitern Sie die abstrakte Maschine aus Aufgabe 4, Blatt 5 auf die Befehle und Datentypen (**int** und **real**) aus Aufgabe 3.
- (d) (P)
Schreiben Sie nun einen Compiler (in Gofer), der Ausdrücke der Sprache $WHILE_T$ in die Maschinsprache aus Aufgabe 3 übersetzt. Sie können sich dabei auf arithmetische Ausdrücke beschränken, die sich direkt in diese Sprache übersetzen lassen (Variablen und Konstanten der Typen **int** und **real**, und der Operator $+$ auf diesen beiden Typen). Beachten Sie, dass in der Maschine aus Aufgabe 3 **int** kein Subtyp von **real** ist, und Integer-Zahlen deshalb ggf. explizit nach **real** konvertiert werden müssen.

Geben Sie Beispielabläufe von einigen compilierten Ausdrücken ab.

In der Datei

```
/usr/proj/semantik/prog/gofer/subtype/subtype.gs
```

wird Ihnen ein Rahmen für das Gofer-Programm zur Verfügung gestellt.