Übungen zur Semantik von Programmiersprachen

Aufgabe 1 (H) (*Korrektheit des WHILE-Compilers*)

Abgabetermin: Montag, 26.11.01

In der Vorlesung wurde eine Richtung der Korrektheit des Compilers für WHILE mit struktureller Induktion über s und Induktion über die Länge der Ableitung $\rightarrow^*$ gezeigt.

$$\text{comp } s \vdash \langle 0, \sigma \rangle \rightarrow^* \langle |\text{comp } s|, \sigma' \rangle \implies \langle s, \sigma \rangle \rightarrow \sigma'$$

- An welcher Stelle und aus welchem Grund scheitert der Beweis, wenn man nur mit Induktion über die Länge der Ableitung arbeitet?
- Wie müsste man die Funktion `comp` und die abstrakte Maschine ändern, damit der Beweis nicht scheitert?

Aufgabe 2 (H) (*Definite Assignment*) **Abgabetermin: Montag, 26.11.01**

Wird in einer Programmiersprache *Definite Assignment* verlangt, bedeutet dies die Forderung, dass alle Variablen vor ihrer ersten Benutzung auch initialisiert werden. Das Statement $x := 1$ z.B. ist also auf jeden Fall zulässig, $x := y$ dagegen darf nur zugelassen werden, wenn feststeht, dass der Variablen y vorher ein Wert zugewiesen wurde. Definite Assignment lässt sich mit einem System von Typregeln beschreiben.

- Geben Sie Typregeln für Definite Assignment in der Sprache WHILE an. Benutzen Sie dazu die Schreibweise $V \vdash s \text{ init } W$ für “Wenn alle Variablen in V initialisiert sind, dann werden nach der Ausführung von s alle Variablen in W initialisiert sein”. Die Typregeln sollen überprüfen, ob nur initialisierte Variablen benutzt werden, und gleichzeitig die Menge der initialisierten Variablen berechnen.
- Modifizieren Sie die Big-Step-Semantik so, dass beim Benutzen einer nicht initialisierten Variable ein Fehler auftritt.
- Formulieren Sie formal die Korrektheitsaussage: “In typkorrekten Programmen tritt kein Fehler wegen uninitialisierten Variablen auf.”

Aufgabe 3 (Ü) (*Statische Modifikation von Typen*)

Das Typsystem der Sprache WHILE_T soll derartig erweitert werden, daß Typen von Variablen im Programmverlauf verändert werden können. Beispielsweise soll folgendes Programm als typkorrekt angesehen werden: $X := 5; X := \text{true}$.

- Entwickeln Sie Typisierungsregeln für ein solches Typsystem. Beachten Sie dabei, daß sich der Deklarationskontext Γ während des Programmverlaufs verändern kann. Deshalb sollen Typisierungsregeln die Form $\{\Gamma\} \text{konstrukt} \{\Gamma'\}$ besitzen mit der Bedeutung, daß *konstrukt* typkorrekt ist, falls Γ vor der Ausführung und Γ' nach der Ausführung gilt.

- (b) Geben Sie eine Big-Step-Semantik für die erweiterte Sprache an.
- (c) Beweisen Sie die Typsicherheit.

Aufgabe 4 (P) (*Typchecking und Typinferenz*)

Abgabetermin: Montag, 26.11.01 an kleing@in.tum.de

In der Vorlesung wurde die um Typen erweiterte Sprache $WHILE_T$ vorgestellt.

- (a) Programmieren Sie den Typüberprüfungsalgorithmus dieser Sprache in der Programmiersprache Prolog.
- (b) Testen Sie Ihr Programm als Typinferenzverfahren, indem Sie für ein geeignetes Beispiel die Typen ausrechnen lassen.