

## Übungen zu „Prinzipien von Programmiersprachen“ Blatt 3

Auf Blatt 2 haben wir die Umgebungssemantik eingeführt, die präziser das Verhalten realer Maschinen modelliert. Um die Semantik von Funktionen definieren zu können, müssen wir die Menge der Werte um sogenannte Funktionsabschlüsse erweitern (engl. closures).

$$v ::= \dots \\ | \langle \rho; x : \tau; e \rangle$$

Ein Funktionsabschluss besteht aus dem formalen Parameter, dem Funktionsrumpf und den Bindungen aller freien Variablen. Abstraktionen und Applikationen werden dann wie folgt ausgewertet.

$$\frac{\rho \vdash \mathbf{fun}(\mathbf{val} x : \tau) \Rightarrow e \Downarrow \langle \rho; x : \tau; e \rangle}{\rho \vdash e_1 \Downarrow \langle \rho'; x : \tau; e \rangle \quad \rho \vdash e_2 \Downarrow v_2 \quad \rho', x \Downarrow v_2 \vdash e \Downarrow v} \rho \vdash e_1 e_2 \Downarrow v$$

Der Funktionsrumpf  $e$  wird in der ursprünglichen Umgebung  $\rho'$  ausgewertet.

**Aufgabe 10.** Die obige Regel für Funktionen berücksichtigt nur Wertebindungen. Wie lässt sich die Umgebungssemantik auf Namensbindungen erweitern? *Hinweis:* Nimm an, dass alle Variablenvorkommen mit dem Bindungsmodus (**val** oder **name**) annotiert sind.

**Aufgabe 11.** Wie lassen sich rekursive Funktionen der Form

$$\mathbf{rec}(f : \tau_1 \rightarrow \tau_2) \Rightarrow \mathbf{fun}(\mathbf{val} x : \tau_1) \Rightarrow e$$

behandeln?

**Aufgabe 12\*.** Der Lambda-Kalkül ist ungetypt oder präziser uni-getypt. Er lässt sich in eine getypte Sprache einbetten, sofern diese über Funktionstypen und rekursive Typen verfügt. Ein geeigneter Einbettungstyp lässt sich zum Beispiel in Haskell wie folgt definieren.

$$\mathbf{newtype} \mathbf{Val} = \mathbf{Fun}\{\mathbf{app} :: \mathbf{Val} \rightarrow \mathbf{Val}\}$$

Implementiere die Lambda-Terme aus Abschnitt §3.4 auf diese Weise in einer Sprache deiner Wahl.