

Übungen zur Vorlesung  
„Grundlagen der Programm- und Systementwicklung“

*Diese Übung wird am 14. Februar ausnahmsweise nicht im -2770 diskutiert, sondern stattdessen im Hörsaal -2370!*

In der Vorlesung wurde die abstrakte Spezifikation des organisierten Speichers definiert. In den folgenden Aufgaben werden wir diese Spezifikation funktional implementieren. Diese Implementierung werden wir anschließend benutzen, um einen Interpreter für eine kleine imperative Sprache mit Zeigern und später ggf. mit Objekten zu definieren.

### **13.1. Partielle Funktionen und Speicher**

In Arbeitsblatt 2 haben Sie die Spezifikation GREX kennengelernt, die die Menge der partiellen Funktionen  $f: \text{Nat} \rightarrow \alpha$  beschreibt. Wird die Sorte Nat durch eine beliebige Sorte  $\beta$  ersetzt und der Sortenkonstruktor  $\text{GreX}(\alpha)$  zu  $\text{GreX}'(\beta, \alpha)$  erweitert, dann beschreibt GREX' die Menge der partiellen Funktionen  $f: \beta \rightarrow \alpha$ .

(H) Implementieren Sie die erweiterte Spezifikation GREX' mit Assoziationslisten, d.h. mit Listen von Paaren (Argument :  $\beta$ , Ergebnis :  $\alpha$ ) in funktionaler Notation.

(H) Falls  $\beta = \text{Nat}$  und  $\alpha = \text{Val}(\gamma) \mid \text{undef}$ , wobei undef einen zugewiesenen, nicht initialisierten Speicherplatz beschreibt, lassen sich zusätzlich zwei Funktionen definieren:

newloc:  $\text{GreX}(\alpha) \rightarrow \text{Nat}$   
extend:  $\text{GreX}(\alpha), \text{Nat} \rightarrow \text{GreX}(\alpha)$ .

newloc( $\sigma$ ) gibt die Adresse eines freien Speicherplatzes in  $\sigma$  zurück und extend( $\sigma, x$ ) weist den Speicherplatz mit Adresse  $x$  in  $\sigma$  zu. Definieren Sie die Funktion extend und eine Funktion newloc in funktionaler Notation. Wie hängen diese Funktionen mit der Einführung in die denotationelle Semantik zusammen, die in der 10. Übung gegeben wurde?

### **13.2 Pointer Talk**

Ein Interpreter ist eine Funktion  $\text{interp} : \text{Exp}, \text{Env}, \text{Store} \rightarrow (\text{Val}, \text{Env}, \text{Store})$ , die einen Ausdruck  $e \in \text{Exp}$  in einer Umgebung  $\eta \in \text{Env}$  und bezüglich eines Speichers  $\sigma \in \text{Store}$  auswertet. Das Ergebnis ist ein Tripel bestehend aus einem Wert  $v \in \text{Val}$ , einer neuen Umgebung  $\eta' \in \text{Env}$  und einem neuen Speicher  $\sigma' \in \text{Store}$ . Die Umgebung  $\eta$  bezeichnet die Symboltabelle für die Variablen in  $e$ , und der Speicher  $\sigma$  bezeichnet den Rechnerspeicher (vgl. Übung 10). Bei der Definition des Interpreters werden wir nur die Auswertung der Ausdrücke betrachten und die Typüberprüfung ignorieren.

Die Sprache *Pointer Talk* ist durch folgende Produktionen definiert:

$$\begin{aligned} x, y: & \text{ Var}, \quad c : \text{ Int}, \quad e : \text{ Exp}, \\ e ::= & \quad x \mid c \mid *x \mid \&x \mid \text{new}(x) \mid x := e \mid \mathbf{local} \ x ; e \ \mathbf{end} \mid e_1 ; e_2. \end{aligned}$$

Wie in der Sprache C bezeichnet  $*x$  den Wert des Speicherplatzes referenziert bei  $x$  und  $\&x$  die Adresse von  $x$ . Wie in der Sprache Pascal alloziert  $\text{new}(x)$  dynamisch einen Speicherplatz und aktualisiert  $x$  mit seiner Adresse.

1. Definieren Sie die Funktion  $\text{interp}$  in funktionaler Notation.
2. Berechnen Sie:  $\mathbf{local} \ p ; \mathbf{local} \ x ; \mathbf{local} \ y ; x := 1 ; \text{new}(p) ; p := \&x ; y := *p \ \mathbf{end} \ \mathbf{end} \ \mathbf{end}$ .