15. Übungsblatt zu Programmierung (WS 1999/2000)

Andreas Rossberg, Prof. Gert Smolka

Abgabe: Keine

Dieses Übungsblatt muss nicht mehr abgegeben werden, sondern dient nur zur Klausurvorbereitung! Zur weiteren Vorbereitung sollten Sie sich die Aufgaben 9.2-4, 11.1-4, 12.1-2, 14.1-2 und die kompletten Blätter 10 und 13 noch einmal ansehen. Zu statischer und dynamischer Semantik von F sollten Sie die Inferenzregeln im Skript verstanden haben.

Vorgegegebene Module finden Sie unter `http://www.ps.uni-sb.de/courses/info1-ws99/uebung/sml/blatt15/`.

Aufgabe 15.1: F mit Rekursion

Erweitern Sie den Interpreter für F wie folgt um rekursive Prozeduren.

(a) Erweitern Sie den Typ `Token.token` um einen Konstruktor `REC` und passen Sie die Lexer-Komponente entsprechend an.

(b) Erweitern Sie den Typ `exp` der abstrakten Syntax um rekursive Abstraktion und modifizieren Sie den Parser geeignet.

(c) Setzen Sie die Inferenzregel zur statischen Semantik rekursiver Abstraktionen im Elaborator um.

(d) Erweitern Sie den Typ `Value.value` um einen passenden Konstruktor `RecProc`, der rekursive Prozeduren repräsentiert. Setzen Sie dann die beiden neuen Inferenzregeln der dynamischen Semantik im Evaluator um.

Aufgabe 15.2: F mit Referenzen

Erweitern Sie den Interpreter für F (die Version ohne rekursive Prozeduren) um Referenzen.

(a) Schreiben Sie zunächst eine Komponente `State`, die Zustand gemäß folgender Signatur modelliert:

```
signature STATE =
  sig
    type address = int
    type value = Value.value
    type state

    exception Address

    val empty:  state
    val alloc:  state * value -> address * state
    val deref:  state * address -> value          (* Address *)
    val assign: state * address * value -> state  (* Address *)
  end
```

Der initiale Zustand ist durch `empty` gegeben. Mit `alloc` wird eine neue Zelle alloziert, Resultat ist die Adresse der Zelle sowie der veränderte Zustand. Mit `deref` und `assign` kann der Inhalt einer Zelle eingesehen bzw. geändert werden. Beide werfen eine Ausnahme, wenn die Adresse ungültig ist.

Sie sollen `State` ohne Verwendung von Referenzen implementieren!

(b) Erweitern Sie nun den Interpreter um Referenzen. Sie können dabei völlig analog zu Aufgabe 15.1 vorgehen. Sie benötigen drei neue Tokens `IREF`, `BANG` (Ausrufezeichen) und `ASSIGN`. Bei der abstrakten Syntax müssen Sie sowohl `exp` als auch `typ` erweitern. Der Typ der Prozedur `Eval.eval` ändert sich in:

```
eval:  state * value env -> exp -> value * state
```

Entsprechend müssen Sie natürlich die Benutzung von `eval` in der Komponente `Main` anpassen.

Aufgabe 15.3: Inferenzregeln

Aus SML kennen Sie Ausdrücke der Form

a_1 `andalso` a_2

Nehmen Sie an, die abstrakte Syntax von F wäre um solche Ausrücke erweitert.

(a) Legen Sie die statische Semantik mit Hilfe einer geeigneten Inferenzregel fest.

(b) Definieren Sie die dynamische Semantik durch geeignete Inferenzregeln. Achten Sie darauf, dass wie in SML der zweite Ausdruck nur ausgewertet werden soll, wenn der erste `true` ergibt.

Aufgabe 15.4: Wiederholung: Laufzeit

Gegeben ist die Prozedur

```
fun f n = if n > 0 then f(n-1) * f(n-1) else n
```

(a) Geben Sie das Berechnungprotokoll für `f 1` an.

(b) Zeichnen Sie den Rekursionsbaum für `f 3`.

(c) Ist die Laufzeit von `f` n für $n \in \mathbb{N}$ in

(i) $O(1)$?	(vi) $\Theta(1)$?
(ii) $O(n)$?	(vii) $\Theta(n)$?
(iii) $O(2n)$?	(viii) $\Theta(2n)$?
(iv) $O(n^2)$?	(ix) $\Theta(n^2)$?
(v) $O(2^n)$?	(x) $\Theta(2^n)$?

(d) Welche Funktion $\mathbb{N} \to \mathbb{N}$ berechnet `f`?

Aufgabe 15.5: Wiederholung: Iteration

Schreiben Sie ein iterative Prozedur

```
sum: (int -> int) * int * int -> int
```

so dass

$$sum(f,a,b) \; = \; \sum_{i=a}^{b} f(i)$$

für $a, b \in \mathbb{Z}$.

Beweisen Sie die Korrektheit Ihrer Prozedur.

Aufgabe 15.6: Wiederholung: Schleifen

Schreiben Sie mit Hilfe von Schleifen eine nicht-rekursive Prozedur

```
append: 'a list * 'a list -> 'a list
```

die zwei Listen konkateniert.

Hinweis: Achten Sie darauf, dass die Prozedur wirklich obigen Typ hat!