11. Übungsblatt zu Programmierung WS 2002 / 03

Prof. Dr. Gert Smolka und Dipl.-Inform. Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws02/

Abgabe: Montag, 20. Januar 2003

Schicken Sie Ihre Lösungen per E-Mail an Ihren Übungsgruppenleiter. Geben Sie in der E-Mail zuerst ihren Namen und die Nummer Ihrer Übungsgruppe an. Bitte reichen Sie alle Lösungen mit nur einer E-Mail ein.

Um die elektronische Abgabe einfach zu halten, sollen für die theoretischen Teilaufgaben keine Lösungen eingereicht werden (ist jeweils vermerkt). Die jeweiligen Punkte werden Ihnen ohne Abgabe zuerkannt.

Aufgabe 11.1: Geschlossene Ausdrücke und freie Bezeichner $(16 = 8 + 8)$ Betrachten Sie die abstrakte Syntax von FR (F plus rekursive Prozeduren).

(a) Schreiben Sie eine Prozedur

```
closed : exp → bool
```

die testet, ob ein Ausdruck geschlossen ist. Verwenden Sie eine Hilfsprozedur

```
closed' : exp → id list → bool
```

die testet, ob alle freien Bezeichner eines Ausdrucks in einer Liste vorkommen.

(b) Schreiben Sie eine Prozedur

```
freeIds : exp → id list
```

die zu einem Ausdruck eine Liste liefert, die die in diesem Ausdruck frei auftretenden Bezeichner enthält. Die Liste darf denselben Bezeichner mehrfach enthalten. Verwenden Sie eine Hilfsprozedur

```
freeIds' : id list → exp → id list
```

die nur die frei auftretenden Bezeichner liefert, die nicht in einer Liste von gebundenen Bezeichnern enthalten sind.

Aufgabe 11.2: Umgebungen (5) Schreiben Sie eine Strukturdeklaration, die Umgebungen gemäß der Signatur im Skript implementiert. Realisieren Sie Umgebungen mit Prozeduren, die die Ausnahme `Unbound` werfen, wenn sie auf Bezeichner angewendet werden, denen sie keinen Wert zuordnen.

Aufgabe 11.3: Rekursive Prozeduren $(15 = 5 + 5 + 5)$ Erweitern Sie die im Skript angegebenen Prozeduren `elab` und `eval` um Regeln für rekursive Abstraktionen. Erweitern Sie die Regel der Prozedur `eval` für Prozeduranwendungen so, dass auch rekursive Prozeduren angewendet werden können.

Aufgabe 11.4: Erweiterung von F um Paare ($17 = 3 + 3 + (1 + 3 + 3 + 1 + 3)$) Sie sollen F um Paare erweitern. Die abstrakte Syntax und die Menge der Werte sollen dafür wie folgt erweitert werden:

$$\begin{aligned} t \in \mathit{Ty} &= \cdots \mid t_1 * t_2 \\ e \in \mathit{Exp} &= \cdots \mid (e_1, e_2) \mid \texttt{fst}\ e \mid \texttt{snd}\ e \\ v \in \mathit{Val} &= \mathbb{Z} \cup \mathit{Pro} \cup \mathit{Val} \times \mathit{Val} \end{aligned}$$

(a) Geben Sie die für die statische Semantik zusätzlich erforderlichen Inferenzregeln an (keine Abgabe).

(b) Geben Sie die für die dynamische Semantik zusätzlich erforderlichen Inferenzregeln an (keine Abgabe).

(c) Erweitern Sie die Deklarationen für `ty`, `exp`, `elab`, `value` und `eval` um Paare.

Aufgabe 11.5: Erweiterung von F um Listen ($27 = 5 + 5 + (1 + 5 + 5 + 1 + 5)$) Sie sollen F um Listen erweitern. Die abstrakte Syntax und die Menge der Werte sollen dafür wie folgt erweitert werden:

$$\begin{aligned} t \in \mathit{Ty} &= \cdots \mid t\ \texttt{list} \\ e \in \mathit{Exp} &= \cdots \mid \texttt{nil}\colon t \mid e_1 :: e_2 \mid \texttt{null}\ e \mid \texttt{hd}\ e \mid \texttt{tl}\ e \\ v \in \mathit{Val} &= \mathbb{Z} \cup \mathit{Pro} \cup (\mathit{Val} \times \mathit{Val}) \end{aligned}$$

Der Ausdruck `nil`: t beschreibt die leere Liste und gibt für diese den Elementtyp t vor. Die leere Liste soll durch die Zahl 0 dargestellt werden.

(a) Geben Sie die für die statische Semantik zusätzlich erforderlichen Inferenzregeln an (keine Abgabe).

(b) Geben Sie die für die dynamische Semantik zusätzlich erforderlichen Inferenzregeln an (keine Abgabe).

(c) Erweitern Sie die Deklarationen für `ty`, `exp`, `elab`, `value` und `eval` um Listen.

Aufgabe 11.6: Programme $(20 = 10 + 10)$ Ausgehend von F deklarieren wir die folgende abstrakte Syntax für Programme:

```
type declaration = id * exp

type program = declaration list
```

(a) Schreiben Sie eine Prozedur

```
elabPro : ty env → program → ty env  (* Error, Unbound *)
```

die zu einer Typumgebung und einem Programm die gemäß der Deklarationen des Programms veränderte Typumgebung liefert. Falls das Programm für die gegebene Typumgebung unzulässig ist, soll eine Ausnahme geworfen werden. Gehen Sie davon aus, dass eine Prozedur `elab` für Ausdrücke gegeben ist.

(b) Schreiben Sie eine Prozedur

```
evalPro : value env → program → value env  (* Error, Unbound *)
```

die zu einer Wertumgebung und einem Programm die gemäß der Deklarationen des Programms veränderte Wertumgebung liefert. Falls die Ausführung des Programms in der gegebenen Wertumgebung zu einem Fehler führt, soll eine Ausnahme geworfen werden. Gehen Sie davon aus, dass eine Prozedur `eval` für Ausdrücke gegeben ist.