13. Übungsblatt zu Informatik 1 WS 98/99

Andreas Rossberg, Prof. Gert Smolka

Abgabe: 12. Februar 1999 vor der Vorlesung

Hinweis: Den Quelltext des MML-Interpreters finden Sie unter `http://www.ps.uni-sb.de/courses/info1-ws98/uebung/sml/blatt13/`. Er ist in Kapitel 12 des Skriptes ausführlich beschrieben. Der vollständige Interpreter ist als direkt ausführbares Programm auf den CIP-Rechnern verfügbar. Sie können ihn dort mit `~rossberg/mml` aufrufen.

Bitte alle Programme per Email abgeben.

Aufgabe 13.1: Schleifen (6 Punkte)

Schreiben Sie folgende nicht-rekursive Prozeduren mithilfe von Schleifen:

(a) `length: 'a list -> int`, berechnet die Länge einer Liste.

(b) `sqrsum: int -> int`, liefert für n die Summe der ersten n Quadratzahlen.

Aufgabe 13.2: Elaboration von MML (8 Punkte)

Im Web finden Sie eine unvollständige Implementation eines Interpreters für MML. Schauen Sie sich das Modul `Elab` an, welches die Elaboration durchführt, und vergleichen Sie es mit den Inferenzregeln im Skript (Anhang A und Kapitel 6).

Vervollständigen Sie die fehlenden Prozeduren `cond` und `decList`. Die Prozedur `decList` ist etwas abweichend von der entsprechenden Inferenzregel zu realisieren: anstatt nur die lokale Umgebung $T' + T''$ zu liefern, muss ihre Prozedur $T + T' + T''$ zurückgeben.

Testen Sie Ihren Elaborator, indem Sie das wirre MML-Programm aus Aufgabe 4.2 prüfen lassen.

Aufgabe 13.3: Evaluation von MML (12 Punkte)

Schauen Sie sich das Modul `Eval` des MML-Interpreters an. Implementieren Sie die noch fehlenden Prozeduren `cond`, `dec` und `decList`. Die Prozedur `decList` ist analog zu `Elab.decList` zu implementieren, also entsprechend dem in Aufgabe 13.2 gesagten.

Werten Sie das MML-Programm aus Aufgabe 4.2 aus.

Aufgabe 13.4: Auswertungsstatistiken im Interpreter (10 Punkte)

Erweitern Sie den MML-Interpreter so, dass er zu jedem Ergebnis eine kleine Statistik ausgibt, die folgende Angaben über den Evaluationsvorgang enthält:

- Wie oft wurde eine primitive Prozedur appliziert?
- Wie oft wurde eine deklarierte Prozedur appliziert?
- Wieviele Prozeduren wurden erzeugt?

Ein Beispiel:

```
- fun fac(n) = if n = 0 then 1 else op*(n,fac(op-(n,1)));
> fac = fn      [0 primitive apps, 0 other apps, 1 procs created]
- fac(10);
> it = 3628800  [20 primitive apps, 11 other apps, 0 procs created]
-
- fun iter(n) = if n = 0 then 1 else
-               let fun pred(n) = op-(n,1) in
-                   iter(pred(n))
-               end;
> iter = fn     [0 primitive apps, 0 other apps, 1 procs created]
- iter(20);
> it = 1        [20 primitive apps, 41 other apps, 20 procs created]
```

Aufgabe 13.5∗: Analyse von Endrekursion (0+6 Punkte)

Schreiben Sie ein Modul `TailRec`:

```
exception Invalid

val isTailRecursive: Phrase.declaration -> bool     (* Invalid *)
```

das für eine gegebene Prozedur-Deklaration ohne `let` ermittelt, ob sie endrekursiv ist. Falls das Argument eine Wert-Deklaration ist, oder eine Prozedur-Deklaration, die einen Let-Ausdruck enthält, wird die Ausnahme `Invalid` geworfen.