4. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 20. November 1997 **Abgabe:** 1. Dezember 1997

Aufgabe IV.1: Endapplikationen ($8 + 5$ Punkte)

Betrachten Sie die Sprache, die in Aufgabe III.4 definiert wurde, und die Beweisbäume, die deren Ausführung beschreiben. Sei t ein solcher Beweisbaum. Wir definieren die Menge der Knoten an *Endposition* wie folgt:

- Die Wurzel des Beweisbaums steht an Endposition.
- Ist $t = (\textbf{if-t}\ t_1\ t_2)$, so steht t_1 nicht an Endposition. t_2 steht an Endposition genau dann, wenn t an Endposition steht. (Analog gilt dies für **if-f**.)
- Ist $t = (\textbf{op}\ t_1\ t_2)$, so steht weder t_1 noch t_2 an Endposition.
- Ist $t = (\textbf{app}\ t_1\ t_2\ t_3)$, so stehen t_1 und t_2 nicht an Endposition. t_3 steht immer an Endposition.

a. Schreiben Sie eine Prozedur `mark-tail-calls`, die einen Beweisbaum als Eingabe erhält und eine Kopie dieses Beweisbaums zurückliefert, in der jeder $(\textbf{app}\ t_1\ t_2\ t_3)$-Knoten an Endposition durch einen $(\textbf{tail}\ t_1\ t_2\ t_3)$-Knoten ersetzt worden ist. (*Tail call* ist der englische Begriff für Endapplikationen. Endrekursion ist ein Spezialfall der Endapplikation.)

b. Es werde folgender Ausdruck ausgewertet:

```
(((lambda (x) (x x))
  (lambda (count)
    (lambda (n)
      (if ((< n) 1)
          1
          ((count count) ((- n) 1))))))
 5)
```

Wie sieht der zugehörige Beweisbaum vor und nach der Transformation aus?

Aufgabe IV.2: Ablauftiefe (6 + 2 + 4 Punkte)

a. Schreiben Sie eine Prozedur `max-stack-depth`, die einen Beweisbaum, eventuell mit **tail**-Knoten wie in Aufgabe IV.1, als Eingabe erhält und die maximale Kellertiefe als Integer zurückgibt, die zur Auswertung des zugehörigen Ausdrucks benötigt wird.

b. Wie tief wird der Keller maximal bei der Auswertung des folgenden Ausdrucks zur Berechnung von 2^{15}, wenn die **tail**-Transformation aus Aufgabe IV.1 durchgeführt wird?

```
((((lambda (x) (x x))
   (lambda (exp)
     (lambda (x)
       (lambda (y)
         (if ((< y) 2)
             x
             ((* x) (((exp exp) x) ((- y) 1))))))))
  2) 15)))
```

c. Transformieren Sie die obige Prozedur `exp`, die x^y berechnet, in eine endrekursive Form. Idee: Im i-ten Schritt ist bereits x^i berechnet und x^{y-i} muß noch dazumultipliziert werden. Übergeben Sie bei jedem Aufruf von `exp` das Teilergebnis x^i in einem Akkumulatorargument.

   Wie tief wird der Keller maximal bei der Auswertung des transformierten Ausdrucks bei der Berechnung von 2^{15}, wenn die **tail**-Transformation aus Aufgabe IV.1 durchgeführt wird?

Aufgabe IV.3: Listen in SML (4 + 4 Punkte)

Gegeben sei folgende SML-Typdefinition:

```
datatype myList = MyNil | MyCons of int * myList
```

a. Schreiben Sie eine zweistellige Funktion `append` in SML, die ihre beiden Argumentlisten aneinanderhängt und die resultierende Liste zurückgibt.

b. Implementieren Sie die einstellige Funktion `reverse` in SML, die eine Liste zurückgibt, die die Elemente ihrer Eingabeliste in umgekehrter Reihenfolge enthält.

Hinweise zum Starten von SML auf den CIP-Rechnern

Zum Starten von SML auf den CIP-Rechnern geben Sie einfach den Befehl `sml` ein. Sie erhalten eine ähnliche Oberfläche, wie sie Ihnen von SCM bekannt ist: Sie geben SML-Ausdrücke ein, die sofort evaluiert werden und deren Ergebnisse ausgegeben werden.

In Kürze werden wir auch für SML eine Emacs-Unterstützung bereitstellen. Weitere Informationen hierüber werden über die WWW-Seite zur Vorlesung bekanntgegeben werden.