Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz/Softwaretechnologie

Fachbereich Informatik (20)
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in das funktionale Programmieren

Wintersemester 1995/96

Übungsblatt Nr. 6

Der Idoptyp bestimmt die Reaktionsnorm, die Umwelt die Position des Organismus innerhalb Reaktionsgrenzen
aus Schönmuth,Flade, Seeland Genetische und phylogenetische Grundlagen

Aufgabe 1:

a) Schreiben Sie eine Goferfunktion `maptree`, die analog wie die Funktion `map` auf Listen für dem Datentyp
`data (Tree a) = TCons (Tree a) a (Tree a) | TNil`
funktioniert.

b) Erklären Sie folgenden algebraischen Gofer-Datentypen: `data Foo a = Eins a (Foo a)| String |List a` Schreiben Sie eine Funktion, die analog zu `maptree` und `map` auf `(Foo a)` arbeitet.

Aufgabe 2:
Schreiben Sie `map` in CORE-FPT und leiten Sie dann den Typ von `map` mit den Typinferenzalgorithmus aus dem Skript ab.

Aufgabe 3:
(R.M.Burstall & J.Darlington, ACM 1977). Wir führen im folgenden Transformationen auf Gofer-Programmen ein, die durch Manipulation der rekursiven Struktur eines Programms, Effizienzvorteile in der Berechnung liefern sollen:

(a) **Definitionen** sind sind die üblichen Funktionsdefinitionsgleichungen in Gofer mit Pattern auf der linken Seite.

(b) **Instanziierung:** wir können vor einer Funktionsgleichung eine Instanz dieser Definition als weitere Gleichung einfügen.

(c) **Unfolding:** Seien $E = E'$ und $F = F'$ Definitionsgleichungen in einem Goferprogramm. Wenn es ein Redex von E in F' gibt, so können wir in F' reduzieren und ersetzen $F = F'$ dann durch eine neue Gleichung $F = F''$, wobei F'' das Redukt von F' ist.

(d) **Folding:** Als Folding bezeichnen wir die zum *Unfolding* inverse Operation.

(e) **Abstraktion:** Ausdrücke mit lokalen Definitionen der Form

$$E = E'[u_1/F_1, \ldots, u_n/F_n] \textbf{ where } u_1 = F \ldots\ u_n = F_n$$

(f) **arithmetische Gesetze:** Wir erlauben es, Kommutativität und Assoziativität der eingebauten arithmetischen Operationen auszunutzen.

Ein Beispiel

Seien a, b, c, d Listen die Vektoren darstellen. Das Skalarprodukt zweier Vektoren werde durch die Funktion

1. `dot x y 0 = 0`
2. `dot x y (n+1) = (dot x y n)+(x!!n)*(y!!n)`

berechnet. Wir wollen nun

3. `f a b c d n = (dot a b n) + (dot c d n)`

berechnen. Wir wenden die Transformationsregeln an:

4. (Instanziierung von 3):
   `f a b c d 0 = (dot a b 0) + (dot c d 0)`
5. (Unfolding mit 1):
   `f a b c d 0 = 0`
6. (Instanziierung von 3):
   `f a b c d (n+1) = (dot a b (n+1)) + (dot c d (n+1))`
7. (Unfolding mit 2):
   `f a b c d (n+1) = (dot a b n)+(a!!n)*(b!!n) + (dot c d n)+(c!!n)*(d!!n)`
8. (Unfolding mit 3, arithmetische Gesetze):
   `f a b c d (n+1) = (f a b c d n) + (a!!n)*(b!!n) + (c!!n)*(d!!n)`

(a) Transformieren sie die Fibonacci-Zahlen

$$\begin{array}{lll} 1. & f\ 0 & = & 1 \\ 2. & f\ 1 & = & 1 \\ 3. & f\ (n+2) & = & (f(n+1)) + (fn) \end{array}$$

entsprechend obigem Schema in eine äquivalente Definition mit linearer Laufzeit. Wichtig hierbei ist die Einführung einer geschickten Hilfsfunktion, die als Ergebnis das Tupel `(f (n+1),f n))` hat.

(b) Betrachten Sie folgendes Gofer-Programm:

```
data BTree = Tip Int |
             INode BTree BTree

1. f (Tip x) = x
2. f (INode x y) = f x + f y
3. g (Tip x) = x
4. g (INode x y) = g x * g y
```

Transformieren Sie nach obigem Schema die Funktion

```
5. both btree = (f btree, g btree)
```

(c) Testen Sie beide Programmbeispiele in Gofer und vergleichen Sie den Reduktionsaufwand jeweils vor und nach der Transformation. (Programm & Testlauf!!!)