3. Übungsblatt „Programmiersprachen“ SS 99

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 30. April 1999 **Abgabe:** 7. Mai 1999

Die Programmieraufgaben auf diesem Übungsblatt sind nicht mehr in F1, sondern in SML zu lösen.

Aufgabe III.1: Listen und Paare (3 Punkte)

Schreiben Sie eine endrekursive Prozedur `minimax: int list -> int * int`, die das kleinste und das größte Element einer Liste berechnet.

Aufgabe III.2: Listenfunktionale (4 + 3 Punkte)

a. Schreiben Sie eine Prozedur

```
foldr: ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
```

so dass ein Aufruf `foldr` f z `[`x_1`, ..., `x_n`]` den Wert

$$f\ (x_1, f\ (\ldots, f\ (x_{n-1}, (f\ (x_n, z)))\ldots))$$

berechnet. Beispielsweise soll

```
foldr (fn (x, y) => x + y) 0 [1, 2, 3]
```

zu $1 + (2 + (3 + 0)) = 6$ auswerten.

b. Implementieren Sie durch Rückführung auf `foldr` eine Prozedur

```
map: ('a -> 'b) -> 'a list -> 'b list
```

so dass `map` f `[`x_1`, ..., `x_n`]` die Liste `[`$f\ x_1$`, ..., `$f\ x_n$`]` berechnet.

Aufgabe III.3: Funktional zur Listenpartitionierung (4 Punkte)

Schreiben Sie eine Prozedur

```
partition: ('a -> bool) -> 'a list -> 'a list * 'a list
```

so dass `partition f xs` zwei Listen berechnet: Die erste enthält alle Elemente `x` aus `xs`, für die `f x = true` gilt, die zweite alle Elemente, für die `f x = false` gilt. Die Reihenfolge der Elemente in den Ergebnislisten soll dabei der Reihenfolge der Elemente in `xs` entsprechen.

Aufgabe III.4: Listenselektion (4 Punkte)

Schreiben Sie eine Prozedur `nthr: 'a list * int -> 'a`, die das n-te Element von hinten einer Liste bestimmt. Dabei wird das letzte Element mit 0 bezeichnet. Die Liste soll nur einmal durchlaufen werden. Tip: Die Position eines Elements ist erst auf dem „Rückweg“ der Rekursion bekannt. Eine Hilfsprozedur `nthr': 'a list -> 'a * int`, die ein Element mit seiner Positionsangabe zurückgibt, ist hilfreich.

Aufgabe III.5: Infix-Grammatiken $(2 + 2 + 2 + 2$ Punkte)

Diese Aufgabe verwendet Infix-Grammatiken, wie sie in Prof. Smolkas Info-1-Skript, Abschnitt 6.11, definiert werden (auf den Web-Seiten zur Vorlesung zu finden).

Geben Sie zu jedem der folgenden Ausdrücke eine Infix-Grammatik $G = (A, O, \mathrm{Pr}, \mathrm{As})$ mit $A = \{\mathtt{X}\}$ und $O = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ an, die es erlaubt, den jeweiligen Ausdruck äquivalent ohne Klammern zu schreiben:

a. `(((X a X) a X) b (X c (X c X)))`

b. `((X b (X b X)) c ((X a X) a X))`

c. `(X a (((X c X) b X) b X))`

d. `(X a (((X b (X c X)) b X) a X))`

Aufgabe III.6: Polymorphe Prozeduren $(2 + 2 + 2 + 1$ Punkte)

In SML kann man eine zweistellige Prozedur entweder geschönfinkelt $(\alpha \to \beta \to \gamma)$ oder kartesisch $(\alpha \times \beta \to \gamma)$ schreiben.

a. Schreiben Sie eine Prozedur `curry`, die eine zweistellige kartesische Prozedur als Argument bekommt und geschönfinkelt zurückgibt. Welchen Typ hat `curry`?

b. Schreiben Sie eine Prozedur `uncurry`, die eine zweistellige geschönfinkelte Prozedur als Argument bekommt und kartesisch zurückgibt. Welchen Typ hat `uncurry`?

c. Schreiben Sie eine Prozedur `compose` für Funktionskomposition. Durch den Aufruf `compose (f, g)` erhält man eine Prozedur, die zuerst `g`, dann `f` auf ihr Argument anwendet. Welchen Typ hat `compose`?

d. Betrachten Sie die Deklaration:

```
fun f x = compose (curry, uncurry) x
```

Welchen Typ hat `f`?