3. Übungsblatt zu Programmierung WS 99/00

Andreas Rossberg, Prof. Gert Smolka

Abgabe: 12. November 1999 vor der Vorlesung

Allgemeiner Hinweis: Die Lösungen zu allen Aufgaben schicken Sie bitte zusätzlich per Email an Ihren Übungsleiter.

Aufgabe 3.1: Listen ganzer Zahlen (6 Punkte)

(a) Schreiben Sie eine Prozedur `maximum: int list -> int`, die das maximale Element einer Liste von Zahlen liefert. Im Falle einer leeren Liste soll die Ausnahme `Empty` geworfen werden.

(b) Schreiben Sie eine Prozedur `count: int list * int -> int`, die die Anzahl der Vorkommen einer gegebenen Zahl in einer Liste zählt. So soll beispielsweise `count([3,2,0,3,8], 3)` den Wert `2` liefern.

Aufgabe 3.2: Klammerung (6 Punkte)

Geben Sie die folgenden Ausdrücke mit möglichst wenig Klammern an.

(a) `( f((3,4)), g(6 :: ([4,5])) )`

(b) `((all) p) (List.tl(4::(5::([]))))`

(c) `(List.map (List.rev) (h((f x)+(g y), 4-(2-1))))`

(d) `p((if (~1)>x then (q (0.1) y) else (x-y)), (3+x))`

(e) `((f(3,4)) :: (g(5,6))) :: (h(3))`

(f) `if (if a then p(x) else q(x)) then (if b then x else (~x)) else y`

Aufgabe 3.3: Listenfunktionale (12 Punkte)

(a) Schreiben Sie kartesische Varianten zu den aus dem Skript bekannten Prozeduren `map`, `all` und `exists`:

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

(ii) `all': ('a -> bool) * 'a list -> bool`

(iii) `exists': ('a -> bool) * 'a list -> bool`

(b) Formulieren Sie die Prozeduren aus dem Skript neu, ohne Regeln zu benutzen. Verwenden Sie stattdessen `hd`, `tl` und `null`.

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

(ii) `all: ('a -> bool) -> ('a list -> bool)`

(iii) `exists: ('a -> bool) -> ('a list -> bool)`

(c) Schreiben Sie dieselben Prozeduren wie in (b) noch einmal, aber unter Verwendung expliziter Abstraktion mittels `fn` (siehe die zweite Variante von `map` im Skript, aber ohne Regeln).

Aufgabe 3.4: Anwendung von Listenfunktionalen (6 Punkte)

Schreiben Sie die folgenden Prozeduren in nicht-rekursiver Form, indem Sie die Prozeduren `map`, `all` und `exists` verwenden.

(a) Eine Prozedur `f: real list -> bool list` die zu einer Liste von Fließkommazahlen eine Liste von booleschen Werten berechnet, die angeben, ob der Sinus der korrespondierenden Zahl negativ ist (`true`) oder nicht (`false`). Zum Beispiel sei

```
f [0.0, Math.pi, 4.0, 2.2*Math.pi] = [false, false, true, false]
```

(b) Eine Prozedur `elem: ''a -> ''a list -> bool`, so dass `elem x xs` den Wert `true` liefert, genau dann wenn `x` Element der Liste `xs` ist.

(b) Eine Prozedur `allNeg: int list list -> bool`, die auf eine Liste von Listen von ganzen Zahlen angewendet wird und `true` genau dann liefert, wenn mindestens eine der inneren Listen nur negative Zahlen enthält. Zum Beispiel

```
allNeg [[4,~3], [~2], [~3,~1]] = true
allNeg [[3,4], [~0,~14], [12]] = false
```

Aufgabe 3.5: Listenreduktion (7 Punkte)

(a) Schreiben Sie eine Prozedur `reduce: ('a * 'a -> 'a) -> ('a list -> 'a)`, die alle Elemente einer Liste 'zusammenfasst', indem sie sie mit der Argumentprozedur verknüpft. So soll zum Beispiel der Ausdruck

```
reduce f [1,2,3,4,5]
```

als Ergebnis `f(1, f(2, f(3, f(4, 5))))` liefern. Bei einer einelementigen Liste soll das Listenelement geliefert werden, im Falle einer leeren Liste die Ausnahme `Empty` geworfen werden.

(b) Welche Typen haben die folgenden Prozeduren und welche Berechnungen führen sie aus? Schreiben Sie zwei Prozeduren, die dieselben Funktionen berechnen, ohne `reduce` zu verwenden.

(i) `val f = reduce (op+)`

(ii) `fun g xs = reduce (op@) xs`

Aufgabe 3.6: Anwendung der Listenreduktion (4 Punkte)

Schreiben Sie die folgenden Prozeduren in nicht-rekursiver Form, indem Sie die Prozedur `reduce` aus Aufgabe 3.5 benutzen.

(a) Eine nicht-rekursive Version der Prozedur `maximum` aus Aufgabe 3.1.

(b) Eine Prozedur `sum: int list list -> int`, die auf eine Liste von Listen von ganzen Zahlen angewendet wird, und die Summe aller dieser Zahlen berechnet. So soll zum Beispiel

```
sum [[3], [4,5], [~3,0,7]]
```

den Wert `16` liefern. Ist die äußere oder eine der inneren Listen leer, so wird die Ausnahme `Empty` geworfen.