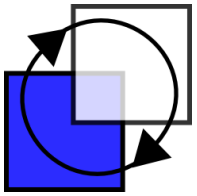
REACTIVE SYSTEMS GROUP

Universität des Saarlandes

Prof. Bernd Finkbeiner, Ph.D.
Markus Rabe, M.Sc.



Programmierung 1 (SS 2010) - 5. Übungsblatt

http://react.cs.uni-saarland.de/prog1/

Lesen Sie im Buch bis Kapitel 5.3

Bei Problemen gehen Sie in die Office Hours: Mo-Mi von 13 bis 14 Uhr.

Aufgabe 5.1

Betrachten Sie den Ausdruck $1::2::nil \,@\, 3::4::nil$.

a) Geben Sie die Baumdarstellung des Ausdrucks an.

b) Geben Sie die Baumdarstellung der beschriebenen Liste an.

c) Geben Sie die beschriebene Liste in der linearen Darstellung an („$[\ldots]$“).

Aufgabe 5.2

Macht es für die dargestellten Listen einen Unterschied, wie die folgenden Ausdrücke geklammert sind? Falls ja, geben Sie ein Gegenbeispiel an.

a) $(e_1 :: e_2)@e_3$ oder $e_1 :: (e_2@e_3)$,

b) $(e_1@e_2)@e_3$ oder $e_1@(e_2@e_3)$,

c) $(e_1 :: e_2) :: e_3$ oder $e_1 :: (e_2 :: e_3)$.

Aufgabe 5.3

Schreiben Sie mithilfe der Prozedur *iter* eine Prozedur $tab : int \to (int \to \alpha) \to \alpha\ list$, die für Argumente n und f dasselbe Ergebnis liefert wie *tabulate* für (n,f).

Aufgabe 5.4

Schreiben Sie mithilfe der Prozedur *iterdn* aus Kapitel 3 eine Prozedur $enum : int \to int \to int\ list$, die zu zwei Zahlen $m \leq n$ die Liste $[m, \ldots, n]$ liefert. Beispielsweise soll $enum\ 3\ 6 = [3, 4, 5, 6]$ gelten. Für $m > n$ soll *enum* die leere Liste liefern.

Aufgabe 5.5

Schreiben Sie eine polymorphe Prozedur $member : {''a} \to {''a\ list} \to bool$ die testet, ob ein Wert als Element in einer Liste vorkommt. Lösen Sie dies auf drei Arten:

a) durch eine regelbasierte Prozedurdeklaration (formulieren Sie zunächst passende Rekursionsgleichungen),

b) mithilfe der vordeklarierten Prozedur $List.exists$,

c) mithilfe der Prozedur $foldl$.

Aufgabe 5.6

Schreiben Sie mit $foldl$ eine Prozedur $prod : int\ list \to int$, die das Produkt der Elemente einer Liste liefert. Für die leere Liste soll 1 geliefert werden.

Aufgabe 5.7

Schreiben Sie mithilfe der Prozedur $foldl$ eine polymorphe Prozedur $count : {''a} \to {''a\ list} \to int$, die zählt, wie oft ein Wert in einer Liste als Element vorkommt. Beispielsweise soll $count\ 5\ [2, 5, 3, 5] = 2$ gelten.

Aufgabe 5.8

Sei eine Prozedur $gcd : int * int \to int$ gegeben, die den größten gemeinsamen Teiler zweier positiven Zahlen bestimmt (siehe Aufgabe 1.20 im Skript). Schreiben Sie mit $foldl$ eine Prozedur $gcdL : int\ list \to int$, die den größten gemeinsamen Teiler der Elemente einer nichtleeren Liste von positiven Zahlen bestimmt. Beispielsweise soll $gcdL[15, 75, 20]$ das Ergebnis 5 liefern.

Aufgabe 5.9

Deklarieren Sie die Faltungsprozedur $foldr$ mithilfe der Faltungsprozedur $foldl$. Verwenden Sie dabei keine weitere rekursive Hilfsprozedur.
Tipp: Reversieren Sie die Liste zunächst (mit foldl). Es geht aber auch ohne.

Aufgabe 5.10

Die Faltungsprozedur $foldl$ kann mithilfe der Faltungsprozedur $foldr$ deklariert werden, ohne dass dabei eine weitere rekursive Hilfsprozedur verwendet wird. Das können Sie sehen, wenn Sie wie folgt vorgehen:

a) Deklarieren Sie $append$ mithilfe von $foldr$.

b) Deklarieren Sie rev mithilfe von $foldr$ und $append$.

c) Deklarieren Sie $foldl$ mithilfe von $foldr$ und rev.

d) Deklarieren Sie $foldl$ nur mithilfe von $foldr$.

Aufgabe 5.11

Die Rückführung von *foldl* auf *foldr* gelingt auf besonders elegante Weise, wenn man *foldr* auf einen prozeduralen Startwert anwendet.

a) Finden sie eine Abstraktion e, welche uns foldl mit folgender Deklaration foldl definieren lässt:

```
fun foldl f s xs = (foldr e (fn g => g) xs) s
```

Die Abstraktion soll keine Hilfsprozeduren verwenden.

b) Überzeugen Sie sich davon, dass die obige Deklaration auch umgekehrt funktioniert. Wir können einfach die Bezeichner *foldl* und *foldr* vertauschen.

Aufgabe 5.12

a) Deklarieren Sie eine Prozedur $last : \alpha\ list \to \alpha$, die das Element an der letzten Position einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.

b) Schreiben Sie mit *foldl* eine Prozedur $max :\ int\ list \to int$, die das größte Element einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.

Aufgabe 5.13

Entscheiden Sie für jeden der folgenden Werte, ob er das Muster $(x,\ y :: \_ :: z,\ (u, 3))$ trifft. Geben Sie bei einem Treffer die Bindungen für die Variablen des Musters an.

a) $(7,\ [1],\ (3, 3))$

b) $([1, 2],\ [3, 4, 5],\ (11, 3))$

Aufgabe 5.14

Welche der folgenden Muster der 4 Regeln der Prozedur *test* treffen den Wert $[(2, 5),\ (7, 3)]$? An welche Werte werden die Variablen der Muster dabei gebunden?

```
fun test [] = 0
 | test [(x,y)] = x + y
 | test [(x,5),(7,y)] = x * y
 | test (_::_::ps) = test ps
```

Aufgabe 5.15

Deklarieren Sie eine Prozedur $primelist : int \to int\ list$, die zu $n \geq 0$ die Liste der ersten n Primzahlen in aufsteigender Ordnung liefert. Schreiben Sie dazu zunächst eine Prozedur $isPrime : int \to bool$, welche testet, ob eine Zahl eine prim ist. Danach verwenden Sie *first* um eine Prozedur $nextPrime : int \to int$ zu deklarieren, welche zu einer Zahl n die kleinste Primzahl über n zurück gibt. Verwenden Sie schließlich *iter* um mit *nextPrime* die ersten n Primzahlen zu finden und als Liste zurückzugeben.

Aufgabe 5.16

Schreiben Sie mithilfe der Prozeduren *explode* und *ord* eine Prozedur *less* : *string* $\rightarrow$ *string* $\rightarrow$ *bool*, die zwei Strings s, s' dasselbe Ergebnis liefert wie $s < s'$.

Aufgabe 5.17

Schreiben Sie eine Prozedur *isPalindrome* : *string* $\rightarrow$ *bool*, die über einen String sagt, ob dieser ein Palindrom ist. Ein String $s_0 s_1 \ldots s_n$ ist genau dann ein Palindrom, wenn $s_i = s_{n-i}$ für alle $0 \leq i \leq n$ gilt. Der Einfachheit halber werden nur Kleinbuchstaben und Sätze ohne Leerzeichen betrachtet. Zum Beispiel liefert *isPalindrome* "*erikafeuertnuruntreuefakire*" den wert *true*.

Aufgabe 5.18

a) Schreiben Sie eine Prozedur *sorted* : *int list* $\rightarrow$ *bool*, die testet, ob eine Liste aufsteigend sortiert ist. Verwenden Sie dabei keine Hilfsprozedur.

b) Schreiben Sie *sorted* jetzt mit Hilfe der Prozedur *foldl*.

Aufgabe 5.19

Schreiben Sie eine Prozedur *perm* : *int list* $\rightarrow$ *int list* $\rightarrow$ *bool*, die testet, ob zwei Listen bis auf die Anordnung ihrer Elemente gleich sind. Verwenden Sie dabei die Prozedur *isort* aus dem Skript, die Sortieren durch Einfügen realisiert, und die Tatsache, dass *int list* ein Typ mit Gleichheit ist.

Aufgabe 5.20

Schreiben Sie eine Prozedur *issort* : *int list* $\rightarrow$ *int list*, die eine Liste sortiert und dabei Mehrfachauftreten von Elementen eliminiert. Beispielsweise soll für $[3, 1, 3, 1, 0]$ die Liste $[0, 1, 3]$ geliefert werden.

Aufgabe 5.21

Deklarieren Sie eine Prozedur *lex* : $(\alpha * \alpha \rightarrow order) \rightarrow (\beta * \beta \rightarrow order) \rightarrow (\alpha * \beta) * (\alpha * \beta) \rightarrow order$ die zu zwei Ordnungen für die Typen α und β die lexikalische Ordnung für den Produkttyp $\alpha * \beta$ liefert. Deklarieren Sie mit Ihrer Prozedur *lex* eine Prozedur, die die lexikalische Ordnung für Paare des Typs *int* $*$ *real* darstellt, die in der ersten Position absteigend und in der zweiten Position aufsteigend sortiert. Zum Beispiel soll die Liste $[(3, 4.0), (2, 2.0), (2, 3.0)]$ gemäß dieser Ordnung sortiert sein.