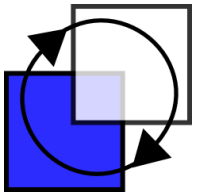
# REACTIVE SYSTEMS GROUP

Universität des Saarlandes

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

## Programmierung 1 (SS 2010) - 6. Übungsblatt

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

Lesen Sie im Buch bis zum Ende von Kapitel 6

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

### Aufgabe 6.1 (Wiederholung)

a) Welche Schlüsselwörter haben wir bisher kennengelernt um Deklarationen einzuleiten? Es sind insgesamt 5.

b) Leiten Sie das Typschema des folgenden Ausdrucks her. Bestimmen Sie dazu zunächst die Typen aller vorkommenden Variablen und Teilausdrücke.

```
fn y => fn x => (fn f => f (x=y)) (fn b => (b,()))
```

### Aufgabe 6.2

Deklarieren Sie eine Prozedur $partition : int \to int\ list \to int\ list * int\ list$, die zu einer Zahl $x$ und einer Liste $xs$ ein Paar aus zwei Listen $us$ und $vs$ wie folgt liefert:

(i) $sort\,(us \mathbin{@} vs) = sort\ xs$.

(ii) Alle Elemente von $us$ sind echt kleiner als $x$ und alle Elemente von $vs$ sind größer oder gleich $x$.

Schreiben Sie *partition* mit *foldl*. Orientieren Sie sich dabei an der Prozedur *split*.

### Aufgabe 6.3 (Quicksort)

Quicksort ist ein klassischer Sortieralgorithmus, der die zu sortierende Liste gemäß der Prozedur *partition* aus der vorigen Aufgabe in zwei Teillisten zerlegt und ihre durch Rekursion berechneten Sortierungen mit Konkatenation wieder zusammenfügt. Die Essenz von Quicksort wird durch die bedingte Gleichung

$$sort\,(x :: xr) = (sort\ us) \mathbin{@} [x] \mathbin{@} (sort\ vs) \quad \text{falls} \quad (us, vs) = partition\ x\ xr$$

beschrieben. Deklarieren Sie eine Prozedur *qsort*, die Listen über *int* gemäß Quicksort sortiert.

### Aufgabe 6.4

Vergleichen Sie die Laufzeiten der Prozeduren *isort*, *msort* und *qsort* für die Listen

```
val xs = List.tabulate(5000, fn x => x)
val ys = rev xs
val zs = List.tabulate(30000, fn x => ((x div 2)*(x+11001)+876) mod 5393 )
val zs = zs@zs@zs@zs;
```

### Aufgabe 6.5

a) Schreiben Sie eine polymorphe Prozedur *smerge*, die zwei strikt sortierte Listen zu einer strikt sortierten Liste kombiniert.
Beispielsweise soll gelten: *smergeInt.compare* [1,3] [2,3] = [1,2,3].

b) Schreiben Sie eine polymorphe Prozedur *ssort*, die Listen strikt sortiert. Beispielsweise soll gelten: *ssortInt.compare* [5,3,2,5] = [2,3,5].

c) Machen Sie sich klar, dass es sich bei *ssortInt.compare* um eine Prozedur handelt, die zu einer Liste *xs* die eindeutig bestimmte strikt sortierte Liste liefert, die dieselbe Menge wie xs darstellt.

### Aufgabe 6.6

Wir können endliche Teilmengen der natürlichen Zahlen eindeutig durch strikt sortierte Listen darstellen, so dass die Listendarstellung einer Menge $M$ die strikt sortierte Liste ist, deren Elemente genau die Elemente der Menge $M$ sind. Schreiben Sie die Prozeduren

a) *member* : *int* $\rightarrow$ *int list* $\rightarrow$ *bool*, welche testet, ob eine Zahl Element einer Menge ist.

b) *union* : *int list* $\rightarrow$ *int list* $\rightarrow$ *int list*, welche die Vereinigung zweier Mengen liefert.

c) *intersection* : *int list* $\rightarrow$ *int list* $\rightarrow$ *int list*, welche den Schnitt zweier Mengen liefert.

d) *difference* : *int list* $\rightarrow$ *int list* $\rightarrow$ *int list*, welche die Differenz zweier Mengen liefert.

e) *eqset* : *int list* $\rightarrow$ *int list* $\rightarrow$ *bool*, welche testet ob zwei Mengen gleich sind.

f) *subset* : *int list* $\rightarrow$ *int list* $\rightarrow$ *bool*, welche für zwei Mengen $X$ und $Y$ testet ob $X \subseteq Y$. Realisieren sie *subset* mit *member* und alternativ mit *eqset* und *intersection* gemäß der Äquivalenz $X \subseteq Y \iff X = X \cap Y$.

Sie dürfen annehmen, dass die Argumentlisten bereits strikt sortiert sind, müssen Sie dies aber für die erzeugten Ergebnislisten sicherstellen. Überlegen Sie sich Typschemata für polymorphe Varianten der obigen Prozeduren (nicht eingeschränkt auf Gleichheitstypen).

### Aufgabe 6.7

Seien endliche Mengen durch Listen ohne Mehrfachauftreten dargestellt. Deklarieren Sie eine Prozedur *power* : $\alpha$ *list* $\rightarrow$ $\alpha$ *list list*, die zu einer Menge $X$ die Menge liefert, die genau die Teilmengen von $X$ enthält. Dabei ist die Reihenfolge beliebig, aber es soll keine Mehrfachauftreten geben. Beispielsweise wäre es korrekt, wenn *power* für $[1, 2]$ die Liste $[[1, 2], [1], [2], []]$ liefert.
Testen Sie bis zu welcher Größe der Mengen die Prozedur *power* funktioniert.

### Aufgabe 6.8

Wir erweitern den Datentyp *shape* um weitere Formen. Dazu deklarieren wir einen neuen Datentyp, welcher auch 3-dimensionale geometrische Objekte enthält:

```
datatype shape = Circle of real
               | Square of real
               | Triangle of real*real*real
datatype shape3d = S of shape
               | Ball or real
               | Box of real*real*real
               | Tetrahedron of real
```

Schreiben Sie sie eine Prozedur $volume : shape3d \to real$ welche das Volumen berechnet.

### Aufgabe 6.9

Deklarieren Sie einen Datentyp *complex*, welcher komplexe Zahlen mithilfe von zwei reellen Zahlen darstellt. Falls nötig, schlagen Sie auf Wikipedia nach was komplexe Zahlen sind. Schreiben Sie dann Prozeduren $re : complex \to real$ und $im : complex \to real$, welche auf den reellwertigen Anteil, beziehungsweise den imaginären Anteil, projezieren, sowie Prozeduren $add : complex \to complex$ und $mul : complex \to complex$, welche die Addition und die Multiplikation komplexer Zahlen berechnen.

### Aufgabe 6.10 (Vars)

Deklarieren Sie eine Prozedur $vars : exp \to var\ list$, die zu einem Ausdruck eine Liste liefert, die die in dem Ausdruck vorkommenden Variablen enthält. Orientieren Sie sich an der Prozedur *subexps*.

### Aufgabe 6.11 (Count)

Deklarieren Sie eine Prozedur $count : var \to exp \to var$, die zählt, wie oft eine Variable in einem Ausdruck auftritt. Beispielsweise tritt $x$ in $x + x$ zweimal auf.

### Aufgabe 6.12 (Check)

Deklarieren Sie eine Prozedur $check : exp \to exp \to bool$, die für zwei Ausdrücke $e$ und $e'$ testet, ob $e$ ein Teilausdruck von $e'$ ist.

### Aufgabe 6.13 (Instantiate)

Schreiben Sie eine Prozedur $instantiate : env \to exp \to exp$, welche alle Variablen in einem Ausdruck gemäß einer Umgebung instantiiert. Zu einer Umgebung *env* und einem Ausdruck $e$, liefert $instantiate\ env\ e$ den Ausdruck, den man aus $e$ erhält, indem man alle Variablen gemäß *env* durch Konstanten ersetzt. Orientieren Sie sich an der Prozedur *eval*.

**Aufgabe 6.14 (Symbolisches Differenzieren)**

Schreiben Sie eine Prozedur, die Ausdrücke nach einer Variable ableitet. Ein Beispiel:

$$(x^3 + 3x^2 + x + 2)' = 3x^2 + 6x + 1$$

a) Schreiben Sie eine Prozedur *derive* : *exp* → *exp*, die die Ableitung eines Ausdrucks gemäß den folgenden Regeln berechnet:

| | | | |
|---|---|---|---|
| $c'$ | = | $0$ | für Konstanten |
| $x'$ | = | $1$ | für die Variable x |
| $y'$ | = | $0$ | für andere Variablen |
| $(u+v)'$ | = | $u' + v'$ | für die Addition zweier Teilausdrücke |
| $(u \cdot v)'$ | = | $u' \cdot v + u \cdot v'$ | Multiplikationen |
| $(u^n)'$ | = | $n \cdot u^{n-1} \cdot u'$ | Potenzen |

Die Ableitung darf vereinfachbare Teilausdrücke enthalten (z.B. $0 \cdot u$).

b) Schreiben Sie eine Prozedur *simplifyTop* : *exp* → *exp*, die versucht einen Ausdruck auf oberster Ebene durch die Anwendung einer der folgenden Regeln zu vereinfachen:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $0+u$ | → | $u$ | | $u+0$ | → | $u$ |
| $0 \cdot u$ | → | $0$ | | $u \cdot 0$ | → | $0$ |
| $1 \cdot u$ | → | $u$ | | $u \cdot 1$ | → | $u$ |
| $u^0$ | → | $1$ | | $u^1$ | → | $u$ |

Wenn keine der Regeln auf oberster Ebene anwendbar ist, soll der Ausdruck unverändert zurückgeliefert werden.

c) Schreiben Sie eine Prozedur *simplify* : *exp* → *exp*, die einen Ausdruck gemäß der obigen Regeln solange vereinfacht, bis keine Regel mehr anwendbar ist. Gehen Sie bei zusammengesetzten Ausdrücken wie folgt vor:

1. Vereinfachen Sie zuerst die Komponenten.
2. Vereinfachen Sie dann den Ausdruck mit den vereinfachten Komponenten mit Hilfe von *simplifyTop*.

**Aufgabe 6.15**

In § 6.6 wurde die Typdeklaration *mylist* eingeführt.

a) Schreiben Sie eine Prozedur *append* : $\alpha$ *mylist* → $\alpha$ *mylist* → $\alpha$ *mylist*, die zwei gemäß des Typkonstruktors *mylist* dargestellten Listen konkateniert.

b) Schreiben Sie eine Prozedur, die eine Liste gemäß des Typkonstruktors *mylist* reversiert.

**Aufgabe 6.16 (Last)**

Schreiben Sie eine Prozedur *last* : $\alpha$ *list* → $\alpha$ *option*, die das letzte Element einer Liste liefert.