Ludwig-Maximilians-Universität München
Department "Institut für Informatik"
Prof. Dr. Christian Böhm
Elke Achtert, Arthur Zimek

München, 22.12.2006

Informatik 1

WS 2006/07

Übungsblatt 10: Funktionen höherer Ordnung, (rekursive) Datentypen

Besprechung: 15.01.–19.01.2007

Abgabe aller mit **Hausaufgabe** markierten Aufgaben bis Freitag, 12.01.2007, 18:00 Uhr

Aufgabe 10-1 *Funktionen höherer Ordnung* **(Hausaufgabe)**

Die Datei `10-1.sml` enthält eine Definition

```
type student = {name:string, vorname:string, fach:string, matnr:int,
                punkte:int};
```

sowie eine Liste von Beispiel-Werten dieses Typs. Geben Sie eine Kopie der Datei ab, in der zusätzlich folgende Funktionen höherer Ordnung definiert sind.

(a) Funktion `auswahl selektor vergleich wert studenten` vom Typ
`(student->'a) -> ('a*'b->bool) -> 'b -> student list -> student list`
die die Liste derjenigen Elemente von `studenten` berechnet, deren selektierter Wert in der gegebenen Vergleichsrelation zum gegebenen Wert steht, und die wie folgt verwendet werden kann:

```
auswahl #fach    (op =)  "Informatik" info1hoerer;
auswahl #punkte  (op >)  70           info1hoerer;
```

(b) Funktion `mittelwert selektor studenten` vom Typ
`(student -> int) -> student list -> real`
die den Mittelwert der selektierten Werte liefert, und die wie folgt verwendet werden kann:

```
mittelwert #punkte info1hoerer;
mittelwert #matnr  info1hoerer;
```

(c) Funktion `top selektor studenten` vom Typ
`(student -> int) -> student list -> student list`
die die Liste derjenigen Elemente von `studenten` berechnet, deren selektierter Wert größer ist als der entsprechende Mittelwert aller Elemente, und die wie folgt verwendet werden kann:

```
top #punkte info1hoerer;
```

Hinweise:
1. Die Datei enthält einen Kommentar mit den Ergebnissen der jeweiligen Beispiele.
2. Funktionen wie `List.map` und `List.filter` und `List.foldl` sind nützlich für diese Aufgabe.
3. Es ist ratsam, den formalen Parameter `studenten` jeweils mit Typ-Constraint anzugeben:
`(studenten : student list)`.

Aufgabe 10-2 *Funktionen höherer Ordnung* **(Hausaufgabe)**

Das Polynom mit Koeffizienten $k_n, k_{n-1}, \ldots, k_2, k_1, k_0$ ist die Funktion

$$x \;\mapsto\; k_n \cdot x^n + k_{n-1} \cdot x^{n-1} + \ldots + k_2 \cdot x^2 + k_1 \cdot x + k_0$$

die auch nach dem so genannten Horner-Schema berechnet werden kann:

$$x \;\mapsto\; (k_0 + x \cdot (k_1 + x \cdot (k_2 + x \cdot \ldots (k_{n-1} + x \cdot (k_n + x \cdot 0)) \ldots)))$$

Der Einfachheit halber seien alle Koeffizienten ganzzahlig.

(a) Definieren Sie **mit Hilfe der Faltungsfunktionen** eine Funktion `polynom` vom Typ
`int list -> int -> int`, die zu einer Liste $[k_n, k_{n-1}, \ldots, k_2, k_1, k_0]$ von Koeffizienten die Funktion zur Berechnung des Polynoms liefert. Das Polynom mit der leeren Liste von Koeffizienten sei die Funktion $x \mapsto 0$.

(b) Eine Liste von ganzen Zahlen $[z_n, \ldots, z_1, z_0]$ mit $0 \leq z_i < b$ kann als Darstellung zur Basis b einer natürlichen Zahl aufgefasst werden, wie in folgenden Beispielen:

Liste `[1,1,0,1]`	Basis b	dargestellte natürliche Zahl
	2	13
	3	37
	8	577
	10	1101

Die dargestellte Zahl errechnet sich also aus der Basis b und den Zahlen z_i durch

$$\sum_{i=0}^{n} z_i b^i$$

Definieren Sie in möglichst kompakter Weise eine Funktion `vonBasis` vom Typ
`int -> int list -> int`, die zu einer Basis b und einer Liste die dargestellte natürliche Zahl liefert.

Geben Sie Ihre Definitionen in einer Datei `10-2.sml` ab.

Aufgabe 10-3 *Neue Datentypen: Weihnachtsdekoration und Weihnachtsbäume* **(Hausaufgabe)**

Beim Einkauf der Dekoration Ihres Weihnachtsbaumes haben Sie folgende Auswahl:

```
datatype Dekor = Kerze of bool | Kugel of Farbe | Stern of int
```

Die Farbe ist hierbei spezifiziert durch

```
datatype Farbe = Gold | Silber | Blau
```

Beide Typen sind in der Datei `10-3.sml` definiert.

(a) Eine Kerze kostet 1 Euro, eine blaue Kugel 2 Euro, eine silberne oder goldene Kugel 4 Euro. Ein Stern wird umso teurer, je mehr Strahlen er hat (die Anzahl der Strahlen ist durch den Integer-Wert im Konstruktor des Sterns angegeben). Das Basis-Model mit höchstens 4 Strahlen kostet 1 Euro. Für bis zu 4 Strahlen mehr wird ein Stern jeweils um 1 Euro teurer.

Definieren Sie in einer Kopie der Datei `10-3.sml` eine Funktion
`preis : Dekor -> int`,
die für einen Wert vom Typ Dekor den Preis nach diesem Schema angibt.

(b) Ihr Einkauf ist in Datei `10-3.sml` aufgelistet wie hier:

```
val einkauf = [Kerze(false), Kugel(Gold), Kugel(Blau), Stern(4),
               Kerze(false), Stern(8), Stern(6), Stern(12),
               Kerze(false), Kugel(Gold), Kugel(Blau),
               Kerze(false), Kugel(Silber), Kerze(false), Stern(16)];
```

Definieren Sie nun weiterhin in derselben Datei eine Funktion
`kosten : Dekor list -> int`,
die die Kosten für eine solche Einkaufsliste berechnet.

(c) Besorgen Sie sich nun einen Weihnachtsbaum. Der Datentyp Weihnachtsbaum ist ebenfalls in Datei `10-3.sml` definiert, und zwar wie folgt:

```
datatype Schmuck = Leer | schmuck of Dekor;

datatype Weihnachtsbaum = Blt of Schmuck
                        | Knt of Weihnachtsbaum * Schmuck * Weihnachtsbaum;
```

Jeder Knoten und jedes Blatt kann also entweder leer sein, oder ist mit Dekor geschmückt.

Um sich einen Weihnachtsbaum zu besorgen, der ideal in Ihr Wohnzimmer paßt, definieren Sie eine Funktion
`edeltanne : int -> Weihnachtsbaum`,
die einen Weihnachtsbaum einer angegebenen Höhe erzeugt. Die Höhe eines Baumes ist die Länge des längsten Astes. Natürlich soll Ihr Weihnachtsbaum schön gleichmäßig aussehen, d.h. jeder Ast soll gleichlang sein.

Bei diesem Weihnachtsbaum soll zunächst jeder Knoten mit `Leer` markiert sein.

(d) Definieren Sie in derselben Datei eine Funktion
`schmuecken : Dekor list * Weihnachtsbaum -> Dekor list * Weihnachtsbaum`,
die einen Weihnachtsbaum mit dem Dekor aus einer Liste schmückt, bis entweder die Liste leer ist oder der Weihnachtsbaum vollständig geschmückt, also keinen leeren Knoten mehr hat. Die Funktion soll ein Tupel zurückgeben, bestehend aus der Liste, die das noch nicht verbrauchte (übrige) Dekor enthält, und dem geschmückten Weihnachtsbaum.

Hinweis: Die linke Seite Ihres Weihnachtsbaumes ist die sichtbare, also schmuecken Sie diese zuerst, und die andere nur dann, wenn Sie noch etwas vom Dekor übrig haben.

(e) Nach vollbrachtem Werk möchten Sie überprüfen, wieviele Stellen noch leer sind. Dazu schreiben Sie eine Funktion
`knotenLeer : Weihnachtsbaum -> int`,
die zählt, wieviele Knoten (inklusive Blätter) in einem Weihnachtsbaum noch den Wert `Leer` haben.

(f) Wenn eine Kerze brennt, hat sie den Wert `true`, sonst `false`. Sie haben die Kerzen natürlich nicht brennend gekauft. An dem geschmückten Baum möchten Sie nun alle Kerzen anzünden. Definieren Sie dazu eine Funktion
`anzuenden : Weihnachtsbaum -> Weihnachtsbaum`,
die einen neuen Weihnachtsbaum zurückgibt, bei dem alle Kerzen brennen. Wenn Sie gerade dabei sind, solch nützliche Helferlein zu programmieren, definieren Sie auch gleich eine Funktion
`loeschen : Weihnachtsbaum -> Weihnachtsbaum`,
die alle Kerzen löscht.

(g) Für alle Fälle sollten Sie nun noch eine Funktion
`geloescht : Weihnachtsbaum -> bool`,
definieren, die überprüft, ob alle Kerzen gelöscht sind. Sie sollten diese Funktion stets vor dem Schlafengehen aufrufen.

(h) Schließlich brauchen Sie noch eine Funktion
`knut : Weihnachtsbaum * Dekor list -> Weihnachtsbaum * Dekor list`,
die das Dekor von Ihrem Weihnachtsbaum wieder entfernt und in eine Liste aufsammelt. Die Funktion gibt ein Tupel zurück: den geleerten Weihnachtsbaum und die Liste, die alles Dekor wieder enthält, mit dem der Weihnachtsbaum geschmückt war.

Aufgabe 10-4 *Neue Datentypen: Poker-Spiel* **(Hausaufgabe)**

Zu Silvester möchten Sie ins Casino Royal zum Pokerspielen gehen. Da Sie gegen gefürchtete Gegner antreten müssen, bereiten Sie sich durch ein SML-Programm vor:

(a) Definieren Sie zunächst die Datentypen `farbe`, `wert` und `poker_karte`. Alle gültigen Werte für `farbe` und `wert` können Sie dann auflisten wie folgt:

```
val farben = [Herz, Karo, Pik, Kreuz];
val werte  = [k2, k3, k4, k5, k6, k7, k8, k9, k10, Bube, Dame, Koenig, Ass];
```

Der Datentyp `poker_karte` ist die Menge `wert` $\times$ `farbe` und soll genau einen (zweistelligen) Konstruktor haben mit der Signatur
`PokerKarte : wert * farbe -> poker_karte`.

Bilden Sie eine Liste
`spiel : poker_karte list`,
die alle Pokerkarten eines Spieles enthält.

(b) Die einfachsten Kombinationen, die Sie auf der Hand haben können, sind ein Paar, ein Drilling und ein Vierling (Poker) von Karten mit gleichen Werten.

Definieren Sie die entsprechenden Funktionen
`paar : poker_karte list -> bool`,
`drilling : poker_karte list -> bool`,
`poker : poker_karte list -> bool`,
die für eine Hand (eine Liste von Pokerkarten) überprüfen, ob sie eine entsprechende Kombination enthält.

(c) Etwas komplexere Kombinationen sind das Doppelpaar und das Full House, also zwei Paare mit jeweils gleichen Werten bzw. ein Paar und ein Drilling mit jeweils gleichen Werten.

Definieren Sie die entsprechenden Funktionen
`doppelpaar : poker_karte list -> bool`,
`fullhouse : poker_karte list -> bool`,
die für eine Liste von Pokerkarten überprüfen, ob sie eine entsprechende Kombination enthält.

(d) Beim Flush sind die Werte der Karten gleichgültig, sie müssen nur alle die gleiche Farbe haben.

Definieren Sie eine Funktion `flush : poker_karte list -> bool`,
die für eine Liste von Pokerkarten überprüft, ob sie ein Flush ist.

(e) Sehr gute Kombinationen sind schließlich der Straight, der Straight Flush und der Royal Flush. Ein Straight ist eine Folge von Karten mit mit aufsteigenden, direkt benachbarten Werten beliebiger Farbe. Das `Ass` kann hierbei unter `k2` oder oberhalb des `Koenig` eingeordnet werden. Zwei Beispiele für einen Straight sind also

```
val lowStraight  = [PokerKarte (Ass,Herz),
                    PokerKarte (k2,Kreuz),
                    PokerKarte (k3,Karo),
                    PokerKarte (k4,Karo),
                    PokerKarte (k5,Herz)]

val highStraight = [PokerKarte (k10,Herz),
                    PokerKarte (Bube,Karo),
                    PokerKarte (Dame,Pik),
                    PokerKarte (Koenig,Herz),
                    PokerKarte (Ass,Pik)]
```

Ein Straight Flush ist ein Straight, bei dem auch alle Karten die gleiche Farbe haben. Ein Royal Flush ist ein Straight Flush von `k10` bis `Ass`. Das `Ass` ist also hierbei oberhalb des `Koenig` eingeordnet.

Definieren Sie die entsprechenden Funktionen
`straight : poker_karte list -> bool`,
`straightFlush : poker_karte list -> bool`,
`royalFlush : poker_karte list -> bool`,
die für eine Liste von Pokerkarten überprüfen, ob sie eine entsprechende Kombination enthält.

Hinweis: Die Reihenfolge der Kombinationen sind in dieser Aufgabe so gewählt, dass die Lösungen z.T. aufeinander aufbauen können oder ähnliche Typen von Kombinationen den gleichen Lösungsweg verwenden können. Die Reihenfolge der Wertigkeit der Kombinationen ist eine andere: Paar $<$ Doppelpaar $<$ Drilling $<$ Straight $<$ Flush $<$ Full House $<$ Poker $<$ Straight Flush $<$ Royal Flush.

Wir wünschen viel Spaß im Casino!