**Technische Universität München**
**Institut für Informatik**
**Prof. Tobias Nipkow, Ph.D.**
**L. Noschinski, L. Hupel, M. Eberl**

**WS 2014/15**
**16.01.2015**
**Abgabe: 23.01.2015**

# Einführung in die Informatik 2
## 13. Übung

Am 23.01.2015 findet von 13-15 Uhr eine Sprechstunde im Raum MI 01.11.018 statt. Sie können dort den Tutoren allgemeine Fragen zum Unterrichtsstoff, zu den Übungsaufgaben (auch vergangene Blätter) und zu Altklausuren stellen. Die Altklausuren finden Sie online auf unseren Lehrstuhlseiten, sowie gedruckt zum Abholen in der Sprechstunde.

**Aufgabe G13.1** Rätselraten ums richtige Reduzieren

Identifizieren Sie die Redexe in den folgenden `Integer`-Ausdrücken. Welche davon sind innermost, outermost, beides oder keins von beiden?

1. `1 + (2 * 3)`
2. `(1 + 2) * (2 + 3)`
3. `fst (1 + 2, 2 + 3)`
4. `fst (snd (1, 2 + 3), 4)`
5. `(\x -> 1 + x) (2 * 3)`

**Aufgabe G13.2** Arbeitsscheue Auswertung von Argumenten

Sei das folgende Haskell-Programm gegeben:

```
inf :: a -> [a]
inf x = x : inf x

f :: [Int] -> [Int] -> [Int] -> Int
f (x:xs) (y:ys) zs | x > y = y
f (x1:x2:xs) ys (z:zs) = x1
```

Betrachten Sie die Aufrufe:

```
f (inf (1+0)) (inf (1+1)) (inf (1+2))
f (inf (1+2)) (inf (1+1)) (inf (1+0))
f (inf (1+0)) [] (inf (1+1))
```

Wie weit muss Haskell jeweils die Argumente auswerten, um Ihnen das Ergebnis dieser Funktionsaufrufe anzeigen zu können?

**Aufgabe G13.3** Faule Fibonacci-Funktionen

In dieser Aufgabe wollen wir Fibonacci-Zahlen als unendliche Liste darstellen.

1. Definieren Sie eine Variable `fib1 :: [Integer]`, die die Liste aller Fibonacci-Zahlen enthält. Zur Erinnerung: Die Fibonacci-Zahlen sind die Folge $(f_n)_{n\in\mathbb{N}}$ mit $f_0 = 0$, $f_1 = 1$ und $f_n = f_{n-1} + f_{n-2}$ für alle $n \geq 2$.

   *Hinweis:* Nutzen Sie die Funktionen `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]` und `tail :: [a] -> [a]` um `fib1` ohne Hilfsfunktion zu implementieren. Beachten Sie, dass auch die Definition einer Variable rekursiv sein kann (vergleiche `inf` aus Aufgabe G13.2). Die Lösung finden Sie auf Folie 345.

2. Geben Sie eine Liste von Schritten an, um die ersten 5 Elemente von `fib1` auszuwerten.

3. Wenn wir eine Fibonacci-ähnliche Liste mit beliebigen Startwerten haben wollen, bietet sich die Funktion

   ```
   fib2 :: Integer -> Integer -> [Integer]
   fib2 m n = m : fib2 n (m + n)
   ```

   an. Geben Sie die Auswertungsschritte an, die notwendig sind, um die ersten 4 Elemente von `fib2 0 1` anzuzeigen.

4. *(optional)* Warum wird sowohl `fib1 !! n` als auch `fib2 0 1 !! n` mit weniger Schritten ausgewertet als `fib n`? `fib :: Integer -> Integer` sei dabei wie folgt definiert:

   ```
   fib 0 = 0
   fib 1 = 1
   fib n = fib (n - 1) + fib (n - 2)
   ```

**Aufgabe G13.4** Viel verwürfeln (optional)

Definieren Sie eine Funktion `mix :: [[a]] -> [a]`, die eine (unendliche) Liste von (unendlichen) Listen nimmt und sicherstellt, dass jedes Eingabeelement in der (unendlichen) Ausgabeliste enthalten ist. Die Reihenfolge spielt dabei keine Rolle.

Formal: Mit einer Eingabeliste $xs = [[x_{1,1}, x_{1,2}, \ldots], [x_{2,1}, x_{2,2}, \ldots], \ldots]$ muss für alle $m, n$ gelten: `elem` $x_{m,n}$ (`mix` $xs$).

**Aufgabe H13.1** Alle Bäume

Gegeben sei der folgende Datentyp, der einen Binärbaum beschreibt, wie Sie ihn bereits kennen, nur ohne Werte an den Knoten. (Relevant ist also nur die „Form“ des Baums)

```
data SkeleTree = Tip | Node SkeleTree SkeleTree
```

Schreiben Sie eine Funktion `trees`, die die unendliche Liste aller `SkeleTree`s zurückgibt. Die Bäume müssen dabei aufsteigend nach Anzahl Knoten sortiert sein. Die Reihenfolge von Bäumen

der gleichen Größe ist beliebig. Die Liste darf keine Duplikate enthalten. Beachten sie, dass `Tip` der leere Baum ist, also 0 Knoten hat; es zählen nur die `Node`s. Der Baum `Node Tip Tip` hingegen hat einen Knoten.

*Beispiel:*

```
trees = [Tip, Node Tip Tip, Node (Node Tip Tip) Tip,
         Node Tip (Node Tip Tip),
         Node (Node Tip Tip) (Node Tip Tip), ...]
```

oder

```
trees = [Tip, Node Tip Tip, Node Tip (Node Tip Tip),
         Node (Node Tip Tip) Tip,
         Node (Node Tip Tip) (Node Tip Tip)...]
```

*Hinweis:* Eine Hilfsfunktion `treesOfSize :: Integer -> [Tree]`, die eine Liste aller Bäume mit genau $n$ *Node*s zurückgibt, kann hilfreich sein. Bewertet wird jedoch nur die `trees`-Funktion.

### Aufgabe H13.2 Mehr als nur ein paar Paare

Schreiben Sie eine Funktion `incPairs :: Ord a => [a] -> [(a,a)]`. Für eine (möglicherweise unendliche Liste) $xs = [x_1, x_2, \ldots]$ soll `incPairs` $xs$ eine Liste zurückgeben, die alle Paare $(x_i, x_j)$ enthält, für welche gilt $i < j$ und $x_i < x_j$.

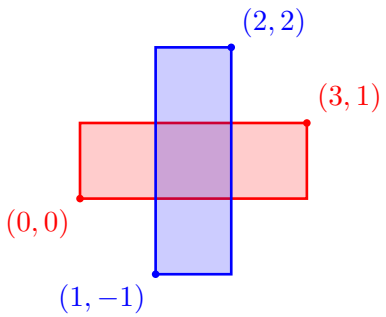
Zusätzlich soll ihre Funktion folgende Eigenschaft erfüllen: Ist $xs$ ein Präfix von $ys$, so ist `incPairs` $xs$ ein Präfix von `incPairs` $ys$.

### Aufgabe H13.3 Wer hat die längste Subsequenz?

Gesucht ist eine Funktion `longestSubsequence :: ([a] -> Bool) -> [a] -> Maybe [a]`, die in einer Liste die längste Teilsequenz findet, die ein gewisses Prädikat erfüllt. Eine Liste $[x_1, x_2, \ldots, x_n]$ gilt dabei als *Teilsequenz* einer Liste $ys$, wenn $ys$ sich so zerlegen lässt, dass alle $x_i$ in der gleichen Reihenfolge vorkommen, dazwischen aber beliebige andere Elemente auftreten können. In andere Worten, $ys = [x_1, \ldots, x_2, \ldots, x_3, \ldots, x_{n-1}, \ldots, x_n]$. Gibt es mehrere gleich lange Teilsequenzen die das Prädikat erfüllen, so kann ein beliebiges zurückgegeben werden. Gibt es keine Teilsequenz die das Prädikat erfüllt, so muss `Nothing` zurückgegeben werden.

### Aufgabe H13.4 Eingekästelt (Wettbewerbsaufgabe)

Der MC ist großer Fan von TV-Spielshows. Sein Lieblingsspiel geht wie folgt: Man zeichnet in ein zweidimensionales Koordinatensystem einige Rechtecke. Dabei entstehen z.B. durch den Schnitt zweier Rechtecke neue Rechtecke. Nun sind alle Rechtecke zu zählen, die zu sehen sind.



In der Abbildung sind z.B. zwei Rechtecke mit den angegebenen Koordinatenpaaren gezeichnet worden. Insgesamt sind allerdings 11 Rechtecke zu sehen.

Um nun auch eine Chance auf den Gewinn zu haben, möchte der MC gerne ein Programm haben, was die Gesamtzahl der Rechtecke berechnet. Der Einfachheit halber definieren wir Rechtecke per Typsynonym:

```
type Point = (Integer, Integer)
type Rectangle = (Point, Point)
```

Sie dürfen davon ausgehen, dass der erste angegebene Wert den unteren linken und der zweite angegebene Wert den oberen rechten Punkt des Rechtecks repräsentiert. Ferner sind die Seiten eines Rechtecks immer mindestens eine Einheit lang. Außerdem ist immer mindestens ein Rechteck vorhanden. Gesucht ist nun eine Funktion

```
allRects :: [Rectangle] -> [Rectangle]
```

welche sämtliche auffindbaren Rechtecke, insbesondere also auch die Eingabeliste, in einer beliebigen Reihenfolge zurückgibt. Die zurückgegebene Liste darf keine Duplikate enthalten, sofern auch die Eingabeliste keine Duplikate aufweist.

Das Wettbewerbskriterium ist Effizienz. Ihre Implementation sollte mit hunderttausenden von entstehenden Rechtecken zurechtkommen.

**Wichtig:** Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {`-WETT-`} . . . {`-TTEW-`} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.