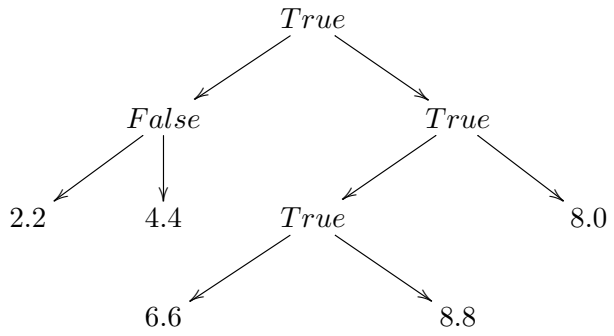
Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
13. Mai 2013

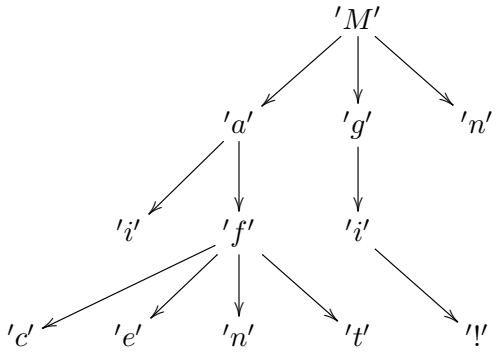
4. Übung zur Vorlesung
Programmierung und Modellierung

A4-1 ***Bäume***

a) Schreiben Sie einen Datentyp, mit dem man folgenden Baum repräsentieren kann:



b) Schreiben Sie erneut einen Datentyp, mit dem man folgenden Baum repräsentieren kann (es darf auch die wieder gleiche Antwort sein, falls passend):



A4-2 ***Ordnung in Wald*** Gegeben ist der folgenden Datentyp:

```
data Tree a b = Leaf a | Node (Tree a b) b (Tree a b)
```

Wir möchten Bäume nach Ihrer Größe ordnen, fügen Sie dazu den Typ `Tree a b` in die Typklasse `Ord` ein, unter der Bedingung, dass die Typen `a` und `b` eine Ordnung in Form einer Instanz von `Ord` vorliegt.

a) Ohne etwas zu implementieren: Was genau müssten wir denn dazu implementieren? Welche Funktion-, Typ-, Klassen- oder Instanzendeklarationen benötigen wir?

b) Implementieren Sie nun das notwendige dazu. Vergleichen Sie zwei Bäume rekursiv: Die Ordnung zwischen zwei Blättern entspricht der Ordnung der Werte die sie tragen. Weiterhin ist jedes Blatt kleiner als jeder Knoten. Die Ordnung zwischen zwei Knoten entspricht zunächst der Ordnung Ihrer enthaltenen Werte. Falls die Werte gleich sind, dann vergleichen Sie zunächst den linken Teilbaum. Sind auch diese gleich, entspricht die Ordnung der beiden Knoten der Ordnung der beiden rechten Teilbäume.

Beispiele:

```
b1,b2,b3,b4,b5 :: Tree Int Char
b1 = Leaf 42
b2 = Node (Leaf 0) 'b' b1
b3 = Node b1 'b' (Leaf 7)
b4 = Node b2 'a' b3
b5 = Node b2 'a' (Node b1 'c' (Leaf 5))
```

```
> b1 > b2
False
> b3 > b2
True
> b4 >= b3
False
> b5 >= b4
True
```

A4-3 ***Abstiegsfunktion III*** Beweisen Sie, dass die Funktion `baz` für nicht-leere Strings, welche nur das Zeichen `'a'` enthalten, wohldefiniert ist; d.h. das die Funktion ohne Fehlermeldung mit einem Ergebnis terminiert.

```
baz :: String -> Double
baz ('a':'a':'a':xs) = 3 * (baz ('a':xs))
baz ('a':c:xs) = (baz ('b':xs)) / 2
baz ('b':d:xs) = 1 + (baz (d:xs))
baz ('c':xs) = undefined
baz [x] | x /= 'd' = 2
        | x == 'e' = 2 * baz ('a':x:'c':[])
```

H4-1 ***Typsignaturen*** **(0 Punkte)** (Abgabeformat: Text oder PDF)
Lösen Sie diese Aufgabe mit Papier und Bleistift! Geben Sie zu jeder der folgenden Deklaration eine möglichst allgemeine Signatur an. Begründen Sie Ihre Antwort informell!

Überlegen Sie sich dazu, welche Argumente jeweils auftreten und wie diese verwendet werden, z.B. welche Funktion auf welches Argument angewendet wird. Falls Sie eine der verwendeten Funktion nicht kennen, schlagen Sie diese in den Vorlesungsfolien oder der Dokumentation der Standardbibliothek nach. Verwenden Sie GHCI höchstens im Nachhinein, um Ihre Antwort zu kontrollieren. Numerische Typklassen müssen Sie zur Vereinfachung nicht angeben, verwenden Sie einfach einen konkreten Typ wie `Int` oder `Double`.

a) `gaa xy z vw = if (read z == xy) then minBound else vw`

Lösungsbeispiel: Wir sehen als erstes, dass es sich um eine Funktion mit drei Argumenten handelt, deren Typ wir bestimmen müssen, sowie den Ergebnistyp der Funktion. Nennen wir diesen Typ vorläufig einmal $a \to b \to c \to d$, also `xy` $:: a$, `z` $:: b$ und `vw` $:: c$.

Das Argument `xy` wird in einem Vergleich verwendet, also muss `Eq` a sein.

`z` ist ein Argument für die Funktion `read` und damit ist $b =$ `String`. Da das Ergebnis aber mit `xy` verglichen wird, muss es den gleichen Typ haben, und wir wissen also, dass auch `Read` a gelten muss, denn es wurde ja ein Wert dieses Typs geparsed.

`vw` wird nur als Ergebnis verwendet, daraus können wir $c = d$ schließen. Da die beiden Zweige des Konditionals den gleichen Typ haben müssen und im `then`-Zweig der Wert `minBound` zurückgegeben wird, muss dieser Typ auch noch in der Klasses `Bounded` sein.

Wir haben also `gaa :: (Eq a, Read a, Bounded c) => a -> String -> c -> c`

b)
```
axx u (v,w) = if minBound || w then succ v else u
```

c)
```
guu _ (h1:h2:t) = [h2]
guu x _         = show x
```

d)
```
foo a b []    e f = show a
foo a b (c:d) e f
  | a==c, read b  = e ++ e
  | a/=c          = show d
  | otherwise     = show e
```

e)
```
bar a b c d
  | b >= c    = bar b a d c
  | otherwise = a==d
```

H4-2 *Fehlerhafte Induktion* (2 Punkte) (Abgabeformat: Text oder PDF)

Durch vollständige Induktion wollen wir zeigen:

Alle Smartphones benutzen das gleiche Betriebssystem.

Um Induktion einsetzen zu können, verallgemeinern die Aussage zu *"In einer Menge von n Smartphones benutzen alle das gleiche Betriebssystem."* Da die Anzahl aller Smartphones in der Welt ist eine natürliche Zahl ist, reicht dies für die ursprüngliche Aussage.

Der Induktionsanfang ist klar: Ein Smartphone benutzt das gleiche Betriebssystem wie es selbst.

Angenommen es gäbe $n + 1$ Smartphones in der Welt. Wählen wir irgendein Smartphone davon aus und bezeichnen es mit s. Die übrigen Smartphones bilden eine Menge von n Smartphones. Nach Induktionsvoraussetzung benutzen diese n Smartphones alle das gleiche Betriebssystem. Nun entfernt man von diesen n Smartphones mit gleichem Betriebssystem eines und fügt s wieder dazu. Die vorliegenden n Smartphones benutzen nach Induktionsvoraussetzung wieder alle das gleiche Betriebssystem. Damit benutzen aber offenbar alle $n + 1$ Smartphones das gleiche Betriebssystem und die Behauptung ist bewiesen.

Wo liegt der Fehler? Begründen Sie Ihre Antwort!

H4-3 *Polynome* (6 Punkte) (`.hs`-Datei als Lösung abgeben)
In einer anderen wichtigen Informatik Vorlesung müssen wir bald mit Polynomen rechnen. Da wir bedauerlicherweise schon vieles in der Schule dazu Gelerntes wieder bereits vergessen haben, wollen wir einen passenden Datentyp für Polynome in Haskell schreiben, mit dem wir später unsere Polynomrechnungen mit Haskell durchführen können – nur zur Kontrolle natürlich!

Wir haben uns entschieden, unsere Polynome als eine Liste von Fließkommazahlen zu implementieren. Beispiele:

$$[] = 0.0 \qquad [1.0] = 1.0 \qquad [3,2,1,0,0] = x^2 + 2x + 3$$
$$[0.0] = 0.0 \qquad [-5,22] = 22x - 5 \qquad [5,0,3,0,1] = x^4 + 3x^2 + 5$$

Allgemein ordnen wird der Liste $[a_1, \ldots, a_n]$ also das Polynom $\sum_{i=0}^{n} a_i \cdot x^i$ zu.

Diese Repräsentation hat den Vorteil, dass Sie leicht nach dem so genannten Horner-Schema berechnet werden kann:

$$x \quad \mapsto \quad (a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot \ldots (a_{n-1} + x \cdot (a_n + x \cdot 0)) \ldots)))$$

Hinweis: Sie dürfen zur Lösung alle Funktionen des Moduls `Prelude` der Standardbibliothek benutzen, wenn Sie möchten, aber keine anderen Haskell Module verwenden.

a) Deklarieren Sie den angegebenen Datentyp `Polynom`

b) Implementieren Sie `berechnePolynom :: Double -> Polynom -> Double`, welche den Wert eines gegebenes Polynoms an einer angegebenen Stelle nach dem oben beschriebenen Horner-Schema berechnet.

c) Machen Sie den Typ `Polynom` zu einer Instanz der Typklasse `Ord`. Wir sortieren Polynome zuerst nach dem höchsten Grad. Bei gleichem Grad vergleichen wir den Koeffizienten des höchstens Gerades, sind auch diese gleich, so prüfen wir kontinuierlich den nächstniederen Koeffizienten.
Beispiele:

$$2x^2 > x^2 + 7x + 3 \qquad x^3 + x^2 > x^3 + 9x + 9 \qquad x^2 + x + 1 > x^2 + x - 2$$

Hinweis: Achten Sie dabei auf alle möglichen Randfälle! Ihre Funktion muss alle Typkorrekten Eingaben richtig verarbeiten.

d) *Empfehlenswerte unbenotete Zusatzaufgabe:* Machen Sie den Typ `Polynom` zu einer Instanz der Typklasse `Show`. Achten Sie dabei auf eine hübsche, lesbare Ausgabe, d.h. die höchste Potenz wird zuerst ausgegeben, Potenzen mit Koeffizient 0 werden ausgelassen, bei negativen Koeffizienten wird anstatt `"1x^4 + -3x^3"` ordentlich `"x^4 - 3x^3"` ausgegeben, usw.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 20.05.2013, 11:00 Uhr mit UniworX abgegeben werden. Die Hausaufgaben müssen von Ihnen alleine gelöst werden; Abschreiben wird als Betrug gewertet und ans Prüfungsamt gemeldet.