

Einführung in die funktionale Programmierung

Wintersemester 2007/2008

Aufgabenblatt Nr. 4

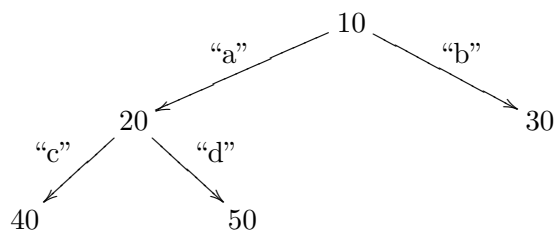
Abgabe: Dienstag 13. November 2007 **vor!** der Vorlesung

Aufgabe 1 (24 Punkte)

Es sei der folgende polymorphe Datentyp in Haskell zur Darstellung von binären Bäumen mit Knoten- und Kantenmarkierungen gegeben:

```
> data Baum a b = Blatt a | Knoten a (b,Baum a b) (b,Baum a b)
>     deriving Show
```

Hierbei ist *a* der Typ der Knotenmarkierungen, und *b* der Typ der Kantenmarkierungen. Ein Beispiel ist der folgende Baum und dessen Repräsentation als `Baum Int String`:



```
> baum = Knoten 10
>         ("a", Knoten 20
>           ("c",Blatt 40)
>           ("d",Blatt 50)
>         )
>         ("b", Blatt 30)
```

a) Definieren Sie in Haskell die folgenden „map“-Funktionen und testen Sie diese mit einigen Bäumen: (je 4 Punkte)

- `mapKnoten :: (a -> c) -> Baum a b -> Baum c b`, die eine Funktion und einen Baum erhält und die Funktion auf alle *Knotenmarkierungen* anwendet.
- `mapKanten :: (b -> c) -> Baum a b -> Baum a c`, die eine Funktion und einen Baum erhält und die Funktion auf alle *Kantenmarkierungen* anwendet.
- `mapKnotenKanten :: (a -> c) -> (b -> d) -> Baum a b -> Baum c d`, die zwei Funktionen und einen Baum erhält und die erste Funktion auf alle *Knotenmarkierungen* und die zweite Funktion auf alle *Kantenmarkierungen* anwendet.

Einige Beispielaufrufe sind:

```
*>mapKnoten (*2) baum
Knoten 20 ("a",Knoten 40 ("c",Blatt 80) ("d",Blatt 100)) ("b",Blatt 60)
*> mapKanten (((++) "M_") baum
Knoten 10 ("M_a",Knoten 20 ("M_c",Blatt 40) ("M_d",Blatt 50)) ("M_b",Blatt 30)
*> mapKnotenKanten (*2) (((++) "M_") baum
Knoten 20 ("M_a",Knoten 40 ("M_c",Blatt 80) ("M_d",Blatt 100)) ("M_b",Blatt 60)
```

- b) Implementieren Sie die folgenden Funktionen in Haskell: (je 4 Punkte)
- Eine Funktion `knotenMarkierungen :: Baum a b -> [a]`, die alle Knotenmarkierungen eines Baumes als Liste berechnet.
 - Eine Funktion `kantenMarkierungen :: Baum a b -> [b]`, die alle Kantenmarkierungen eines Baumes als Liste berechnet.
 - Eine Funktion `blattMarkierungen :: Baum a b -> [a]`, die alle Knotenmarkierungen an Blättern eines Baumes als Liste berechnet.

Für obigen Baum ergibt dies:

```
*> kantenMarkierungen baum
["a","c","d","b"]
*> knotenMarkierungen baum
[40,20,50,10,30]
*> blattMarkierungen baum
[40,50,30]
```

Aufgabe 2 (26 Punkte)

Verwenden Sie für diese Aufgabe den Datentyp `Baum a b` aus der vorherigen Aufgabe.

- a) Implementieren Sie die Tiefensuche auf Bäumen, d.h. implementieren Sie eine Funktion `dfsBaum :: (a -> Bool) -> Baum a b -> Maybe [b]`, die ein Prädikat auf Knotenmarkierungen und einen Baum erwartet und diesen Baum nach einem Knoten, dessen Markierung das Prädikat erfüllt, mithilfe einer Tiefensuche durchsucht. Das Ergebnis soll in diesem Fall `Just xs` sein, wobei `xs` die Liste der Kantenmarkierungen von der Wurzel zum Zielknoten ist. Falls die Suche fehlschlägt, soll `Nothing` als Ergebnis geliefert werden. Welches Ergebnis liefert `dfsBaum (== 9113) beispielBaum`¹? (13 Punkte)
- b) Implementieren Sie eine Funktion `cut :: Integer -> Baum a b -> Baum a b`, die eine Tiefe (als Zahl) und einen (möglicherweise unendlich tiefen Baum) erhält, und diesen in der entsprechenden Tiefe abschneidet, und somit einen endlichen Baum liefert. (7 Punkte)
- c) Verbessern Sie Ihr Verfahren aus Aufgabenteil a), so dass auch unendliche Bäume vollständig durchsucht werden können, indem Sie *iteratives Vertiefen* verwenden. Hierbei wird pro Iteration der Baum nur bis zu einer festen Tiefe k durchsucht. Falls diese Suche erfolgreich ist, wird mit dem Ergebnis beendet, andernfalls wird die nächste Iteration mit Suchtiefe $k + 1$ durchgeführt. Die Funktion `cut` aus Aufgabenteil b) könnte hierbei hilfreich sein.

Testen Sie ihre Funktion, indem Sie den im Quelltext vorgegebenen Baum `beispielBaum` mit dem Prädikat `((==) 407)` durchsuchen. Wie lautet das Ergebnis? (6 Punkte)

¹`beispielBaum` ist im Quellcode, verfügbar von der WWW-Seite zur Veranstaltung, definiert