FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz
Hubert Wagner

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 0

Präsenzaufgaben, keine Abgabe!

Aufgabe 0.1 Welche der folgenden Zeichenreihen stellen syntaktisch korrekte Haskell-Programme dar?

1. `X = [1,2,3]`
2. `f x = x + x * y`
3. `function f x = f x`
4.
```
n = a * b where a = 2
                b = 3
```

Aufgabe 0.2 Definieren Sie ohne Rückgriff auf die Haskell-Funktion prod eine Haskell-Funktion `produkt`, die für eine Liste von Zahlen das Produkt dieser Zahlen berechnet.
Beweisen Sie `produkt [x] = x`.

Aufgabe 0.3 In den Folien zur Vorlesung ist die Sortierfunktion `qsort` enthalten. Wie müsste die Definition von `qsort` modifiziert werden, damit eine Liste in der umgekehrten Reihenfolge sortiert wird?

Aufgabe 0.4 Schreiben Sie auf 2 verschiedene Weisen eine Haskell-Funktion `lastElem`, die angewandt auf eine nichtleere Liste von Elementen das letzte Element dieser Liste liefert. Sie dürfen hierzu nicht die Bibliotheksfunktion `last` verwenden.

Aufgabe 0.5 Welche Funktion wird hierdurch berechnet:

```
f n = sum [1, 3 .. (2*n-1)]
```

Aufgabe 0.6 Erstellen Sie mit einem Editor eine Datei, die eine Liste von Dezimalzahlen (Typ Float) enthält. Schreiben Sie dann eine Haskell-Funktion `einAusFloat`, die bei Eingabe des Dateinamens (als String) und einer weiteren Dezimalzahl die Liste der Dezimalzahlen zunächst einliest, die Dezimalzahl am Anfang der Liste einfügt und anschließend in der Datei mit Namen „erg" speichert.
Machen Sie dann das Entsprechende für eine Liste von ganzen Zahlen (Typ Int oder Integer) bzw. für eine Liste von Strings.

Aufgabe 0.7
Geben Sie den Typ der folgenden Werte an:

1. `['a','b','a']`
2. `(False, True, False)`
3. `([1,2],('a','b'))`
4. `[tail, init, reverse]`

Aufgabe 0.8
Geben Sie den Typ der nachfolgend definierten Funktionen an:

1. `second xs = head (tail xs)`
2. `pair x y = (y,x)`
3. `palindrom xs = xs == reverse xs`
4. `twice f x = f (f x)`

Aufgabe 0.9

1. Eine komplexe Zahl $a + i \cdot b,\ a, b \in \mathbb{R}$, kann als ein Tupel $(a, b) \in \mathbb{R}^2$ repräsentiert werden. In Haskell ist der Datentyp für ein Tupel reeller Zahlen `(Float, Float)`.

   Schreiben Sie für die komplexen Zahlen Haskell-Funktionen für Addition, Subtraktion, Multiplikation, und Division unter Verwendung dieser Repräsentation.

2. Eine quadratische Gleichung $x^2 + ax + b = 0$ mit reellwertigen Koeffizienten a und b kann durch das Tupel (a, b) mit Float-Werten a und b repräsentiert werden.

   Schreiben Sie eine Haskell-Funktion `loesung,` die die Liste der Lösungen des Gleichungssystems $x^2 + ax + b = 0$ mit reellwertigen Koeffizienten a und b berechnet.

Aufgabe 0.10

1. `&&` und `||` sind Haskell-Funktionen in Infix-Schreibweise vom Typ `Bool -> Bool -> Bool`, die 2 Wahrheitswerte unter dem logischen $\wedge$ bzw. dem logischen $\vee$ auswerten.

   Schreiben Sie eine Haskell-Funktion `sortiert :: [Integer] -> Bool`, die überprüft, ob eine Liste von ganzen Zahlen sortiert ist.

2. Schreiben Sie Haskell-Programme, die als Eingabe 2 Strings (das sind Zeichenfolgen vom Typ `[Char]`) xs und ys erhalten und die *True* ausgeben, wenn

   (a) die erste Zeichenfolge xs ein Präfix der zweiten Zeichenfolge ys ist,

   (b) die erste Zeichenfolge xs als Teilstring in der zweiten Zeichenfolge ys enthalten ist.

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 1

Präsenzaufgaben, keine Abgabe!

Aufgabe 1.1

1. Implementieren Sie eine Funktion

   ```
   removeMults :: Eq a => [a] -> [a]
   ```

   die aus einer Liste eventuelle Wiederholungen von Elementen streicht.

2. Schreiben Sie eine Haskell-Funktion

   ```
   splitEO :: [a] -> ([a],[a])
   ```

   die eine Liste L so in 2 Listen L_1 und L_2 zerlegt, dass L_1 alle Elemente von L enthält, die an geraden Postionen in L stehen, und L_2 diejenigen, die an ungeraden Positionen vorkommen. Die Reihenfolge in L soll dabei erhalten bleiben. Z.B. soll also gelten `splitEO [1,3,1,5,7,3,6] =([1,1,7,6],[3,5,3])`.

Aufgabe 1.2

Ein beliebtes Kinderspiel zur Auswahl eines Kindes ist das folgende:

- Eine Gruppe von n Kindern bildet einen geschlossenen Kreis.
- Mit einem ersten Kind beginnend wird jeweils bis zum k-ten Kind weitergezählt. Dieses Kind scheidet aus, der Kreis wird wieder geschlossen.
- Vom Nachbarn (Nachfolger) des ausgeschiedenen Kindes beginnend verfährt man wie zuvor, so lange bis nur noch 1 Kind übrigbleibt.

Schreiben Sie eine Haskell-Funktion, die bei Eingabe einer Liste von Kindern (z.B. mit den Nummern $1, \ldots, n$) und einer positiven ganzen Zahl k errechnet, welches Kind am Ende übrigbleibt. (Das Ergebnis darf auch in Form einer Einerliste vorliegen.)

Aufgabe 1.3

1. Schreiben Sie eine `Haskell`-Funktion `dual2int :: String -> Integer`, die zu einer in Form eines Strings vorliegenden Binärzahl die zugehörige ganze Zahl berechnet. Beachten Sie, dass Binärzahlen keine führende Nullen enthalten.

2. Implementieren Sie die Funktion `int2dual :: Integer -> String`, die eine nichtnegative ganze Zahl in eine Binärzahl in Form eines Strings umwandelt.

Aufgabe 1.4
Implementieren Sie in Haskell den Sortieralgorithmus Insertionsort und sortieren Sie damit einige Listen bestehend aus Zufallszahlen.

Zur Generierung von Zufallszahlen setzen Sie dazu an den Anfang Ihres Programms die Zeile

```
import Random
```

mit der das Modul Random.hs geladen wird. In diesem Modul sind Zufallszahlengeneratoren definiert. Eine unendliche Liste von Zufallszahlen erzeugen Sie dann z.B. mit der folgenden Funktion

```
randomInts :: Int -> Int -> [Int]
randomInts bound seed = tail (randomRs (0,bound) (mkStdGen seed))
```

wobei ganzzahlige Zufallszahlen aus dem Bereich von 0 bis `bound` erzeugt werden. `seed` legt unterschiedliche Startwerte für den Zufallszahlengenerator fest, so dass durch Wahl von unterschiedlichen Werten für `seed` unterschiedliche Listen generiert werden.
Mit `take n (randomInts 15000 50)` wird dann z.B. eine Liste der Länge n von Zufallszahlen aus dem Bereich von 0 bis 15000 erzeugt.

Aufgabe 1.5
In dieser Aufgabe soll die First-Fit Heuristik für das Bin-Packing Problem implementiert werden:
Das Bin-Packing Problem:

Gegeben: n Gegenstände der Größen $w_1, \ldots, w_n$ und beliebig viele Kisten der Größe k.

Gesucht: Die kleinste Anzahl von Kisten, mit der alle Gegenstände aufgenommen werden können.

Die First-Fit Heuristik geht so vor, dass die Gegenstände in einer festen Reihenfolge betrachtet werden und dann jeder Gegenstand in die erstmögliche Kiste gelegt wird, in die er passt.

Testen Sie Ihre Implementierung, indem Sie mit dem Zufallszahlengenerator eine Liste von 1000 Größen zwischen 5 und 100 erzeugen und als Größe der Kisten 150 wählen.
Zur Sortierung der Liste verwenden Sie die Funktion `sort`. Damit diese zur Verfügung steht, muss am Anfang des Programms die Zeile `import Data.List` eingesetzt werden.

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz
Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 2

Abgabe per Email bis 08.11.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

```
FP-Abgabe Gruppe 3
```

Beachten Sie, dass Gruppenabgaben mit einer Gruppengröße von 2 oder 3 Personen gefordert sind. (Ausnahmen nur nach Rücksprache mit dem Betreuer der Übungsgruppe.)

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstaben beginnend>.hs`,

z.B. also `fp02Meier.hs`

Für jede selbstdefinierte Funktion ist der Typ der Funktion anzugeben, ferner ist eine informelle Erläuterung der Argumente sowie der Definition gefordert. In ganz einfachen Fällen darf beides auch schon einmal weggelassen werden.

Aufgabe 2.1 (Präsenzaufgabe)
Eine *ideale Zahl* ist eine Zahl, die genauso groß ist wie die Summe ihrer *echten* Teiler (d.h. aller Teiler, die ungleich der Zahl selbst sind). So ist z.B. 6 ideal, da für die echten Teiler $1, 2$ und 3 gilt: $1 + 2 + 3 = 6$.
Schreiben sie eine Haskell-Funktion `ideal`, die für eine positive ganze Zahl $n > 1$ die Liste aller idealen Zahlen aus $\{1, \ldots, n\}$ liefert.

Aufgabe 2.2 (Präsenzaufgabe)
Das folgende Problem „Türme von Brahma" (bekannt auch als „Türme von Hanoi") geht auf eine hinterindische Sage zurück: In einem im Dschungel verborgenen hinterindischen Tempel sind Mönche seit Beginn der Zeitrechnung damit beschäftigt, einen Stapel von 50 goldenen Scheiben mit nach oben hin abnehmendem Durchmesser, die durch einen goldenen Pfeiler in der Mitte zusammengehalten werden, durch sukzessive Bewegungen jeweils einer einzigen Scheibe auf einen anderen goldenen Pfeiler umzuschichten. Dabei dürfen sie einen dritten Pfeiler als Hilfspfeiler benutzen, müssen aber darauf achten, dass niemals eine Scheibe mit einem größerem Durchmesser auf eine mit kleinerem Durchmesser zu liegen kommt.
Die Sage berichtet, dass das Ende der Welt gekommen ist, wenn die Mönche ihre Aufgabe beendet haben.
Schreiben Sie eine Haskell-Funktion, die bei n Scheiben ausgibt, wie die Scheiben umzuschichten sind. Die einzelnen Schritte sollen dabei durchnummeriert sein.

Aufgabe 2.3 (25 Punkte)

1. Implementieren Sie die Binomialfunktion

$$binom\ n\ p = \binom{n}{p}$$

auf 3 verschiedene Arten: unter Verwendung der Funktion `product`, auf eine direkte, rekursive Weise sowie unter Verwendung des Pascalschen Dreiecks. Welche der Implementierungen ist die schnellste? (15 Punkte)

2. Schreiben Sie eine Haskell-Funktion, die für ganze Zahlen m und n, $(m, n > 1)$, die größte Potenz von m berechnet, die kleiner oder gleich n ist. (10 Punkte)

Aufgabe 2.4 (25 Punkte)

1. Implementieren Sie die Funktion `f :: Integer -> Integer -> Integer` mit

$$f\ p\ k := \sum_{i=1}^{k} i \cdot p \cdot (1-p)^{i-1}$$

einmal rekursiv und einmal unter Verwendung von Listenkomprehension. (je 8 Punkte)

2. Schreiben Sie eine rekursiv definierte Haskell-Funktion, die bei Eingabe einer ganzen Zahl n und einer endlichen Liste $[m_1, \ldots, m_k]$ von ganzen Zahlen die kleinste Zahl m_i in dieser Liste mit $m_i > n$ ausgibt. Existiert keine solche Zahl, so soll 0 ausgegeben werden. Die Funktion soll dabei in linearer Zeit die Ausgabe berechnen. Für die Definition der Funktion darf keine Listenkomprehension verwendet werden. (9 Punkte)

Aufgabe 2.5 (50 Punkte)
Wir wollen ein Polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ in einer Variablen x und mit Koeffizienten, deren Typen in der Klasse *Num* liegen, über eine Liste $[a_n, a_{n-1}, \ldots, a_1, a_0]$ repräsentieren.

1. Schreiben Sie Haskell-Funktionen zur Berechnung von

   (a) Summe (5 Punkte)
   (b) Differenz (5 Punkte)
   (c) Produkt und (10 Punkte)
   (d) Division (15 Punkte)

   von Polynomen in x, die in dieser Weise repräsentiert sind.

2. Implementieren Sie eine Haskell-Funktion, die ein Polynom $p(x) = \sum_{i=0}^{n} a_i \cdot x^i$ an der Stelle b nach dem Horner-Schema auswertet:

   Das Horner-Schema macht dabei Gebrauch von der folgenden Darstellung von $p(x)$:

$$p(x) = (\ldots((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \cdot x + \ldots + a_1) \cdot x + a_0\ .$$

(15 Punkte)

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 3

Abgabe per Email bis 14.11.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

`FP-Abgabe Gruppe 3`

Beachten Sie, dass Gruppenabgaben mit einer Gruppengröße von 2 oder 3 Personen gefordert sind. (Ausnahmen nur nach Rücksprache mit dem Betreuer der Übungsgruppe.)

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`,

z.B. also `fp03Meier.hs`

Für jede selbstdefinierte Funktion ist der Typ der Funktion anzugeben, ferner ist eine informelle Erläuterung der Argumente sowie der Definition gefordert. In ganz einfachen Fällen darf beides auch schon einmal weggelassen werden.

Aufgabe 3.1 (30 Punkte)

1. Implementieren Sie eine Haskell-Funktion

   `nIt :: (a -> a) -> a -> Int -> [a]`,

   für die

   $$nIt\ f\ x\ n = [f^n(x), f^{n-1}(x), \ldots, f^1(x)]$$

   gilt. Hierbei ist $f^n(x) = \underbrace{f(f(\ldots f}_{n-mal}(x)\ldots))$. Die Haskell-Funktion `iterate` darf hierzu nicht verwendet werden. (8 Punkte)

2. Durch Einbinden des Moduls `Ratio` mit dem Befehl `import Ratio` zu Beginn eines Haskell-Programms stehen in diesem Programm die rationalen Zahlen zur Verfügung. Die Haskell-Notation für eine rationale Zahl $\frac{n}{m}$ ist dabei `n % m`.

   Die n-te Harmonische Zahl ist definiert durch $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Implementieren Sie unter Verwendung einer Listenkomprehension die Haskell-Funktion `h`, die bei Eingabe n die n-te Harmonische Zahl H_n berechnet. (6 Punkte)

3. Geben sie eine Haskell-Funktion an, die für eine Liste L und eine natürliche Zahl n die Liste aller Teillisten von L mit Kardinalität n berechnet. (16 Punkte)

Aufgabe 3.2 (35 Punkte)

In dieser Aufgabe verwenden wir als Datentyp für gerichtete und ungerichtete Graphen Adjazenzlisten. Wir führen hierzu als Abkürzung ein:

```
type Graph a = [(a,[a])]
```

Für ungerichtete Graphen setzen wir im Folgenden voraus, dass die Graphen schlingenfrei sind, d.h., dass keine Kante von einem Knoten zu sich selbst führt.

1. Schreiben Sie eine Haskell-Funktion `kompGraph :: Graph a -> Graph a`, die zu einem ungerichteten Graphen G seinen Komplementgraphen $\overline{G}$ berechnet. Dieser hat dieselbe Knotenmenge, und zwischen zwei verschiedenen Knoten ist in $\overline{G}$ genau dann eine Kante vorhanden, wenn sie im Ausgangsgraphen G nicht existiert. (15 Punkte)

2. Implementieren Sie eine Haskell-Funktion, die zu einem gerichteten Graphen G seine reflexive und transitive Hülle G^* berechnet.

   Zur Erinnerung:

   Für einen gerichteten Graphen $G = (V, E)$ ist $G^* = (V, E^*)$ mit

   $$E^* = \{(a, b) \in V \times V \mid a = b \text{ oder es gibt in } G \text{ einen Pfad von } a \text{ nach } b\}$$

   (20 Punkte)

Aufgabe 3.3 (35 Punkte)

1. Gegeben sei eine Folge $x_1, \ldots, x_n$ (mit $n \geq 1$) von paarweise verschiedenen Schlüsseln eines binären Suchbaumes, die aus einer Pre-Order-Traversierung des binären Suchbaumes hervorgegangen ist. Schreiben Sie eine Haskell-Funktion, die den binären Suchbaum rekonstruiert. (15 Punkte)

2. Implementieren Sie eine Haskell-Funktion, die zu einem Binärbaum seinen längsten Pfad berechnet. (Beachten Sie, dass der längste Pfad nicht die Wurzel des Baumes enthalten muss.) (20 Punkte)

Aufgabe 3.4 (Präsenzaufgabe)

Schreiben Sie eine Haskell-Funktion `graph :: Int -> Int -> Graph Int`, so dass `graph n seed` randomisiert einen Graphen mit Knotenmenge $\{0, \ldots, n\}$ erzeugt. Mit unterschiedlichen Werten von `seed` sollen bei gleichem n unterschiedliche Graphen erzeugt werden.

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 4

Abgabe per Email bis 21.11.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

```
FP-Abgabe Gruppe 3
```

Beachten Sie, dass Gruppenabgaben mit einer Gruppengröße von 2 oder 3 Personen gefordert sind. (Ausnahmen nur nach Rücksprache mit dem Betreuer der Übungsgruppe.)

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`,

z.B. also `fp03Meier.hs`

Für jede selbstdefinierte Funktion ist der Typ der Funktion anzugeben, ferner ist eine informelle Erläuterung der Argumente sowie der Definition gefordert. In ganz einfachen Fällen darf beides auch schon einmal weggelassen werden.

Aufgabe 4.1 (20 Punkte)

Neben dem einfachen arithmetischen Mittel existieren eine Reihe weiterer Mittelwert-Funktionen. Für Werte $v_1, \ldots, v_n$ und einen Gewichtsvektor $w = (w_1, \ldots, w_n)$ ist das *gewichtete arithmetische Mittel* definiert als

$$\overline{v} = \frac{\sum_{i=1}^{n} w_i v_i}{\sum_{i=1}^{n} w_i}$$

1. Geben Sie den Typ der Haskell-Funktion an, die den gewichteten arithmetischen Mittelwert berechnet. (5 Punkte)

2. Schreiben sie eine Haskell-Funktion, die für zwei Vektoren gleicher Länge das gewichtete arithmetische Mittel berechnet, wobei der erste Vektor den Gewichtsvektor bezeichnet. (15 Punkte)

Aufgabe 4.2 (40 Punkte)

1. Die Haskell-Funktion `elem :: Eq a => a -> ]a] -> Bool`, die die Elementeigenschaft testet, liefert im Falle von unendlichen Listen nicht immer ein Ergebnis, z.B. erzeugt der Funktionsaufruf `elem 3 [0,2..]` eine Endlosschleife.

   Schreiben Sie eine Haskell-Funktion `member :: Ord a => a -> [a] -> Bool`, die bei Eingabe eines Elements x und einer Liste mit den folgenden Eigenschaften

   - die Listenelemente bilden als Folge betrachtet eine streng monoton wachsende Folge
   - im Falle einer unendlichen Liste wächst die Folge unbeschränkt und jedes Listenelement hat in der Liste nur endlich viele Vorgänger,

   feststellt, also ein Ergebnis liefert, ob x in der Liste xs vorkommt. (12 Punkte)

2. Geben Sie eine Haskell-Funktion `split` vom Typ `Char -> String -> [String]` an, die angewandt auf ein Zeichen c und eine Zeichenkette s die Zeichenkette s an allen Vorkommnissen von c aufsplittet und die Liste der entstehenden Teilstrings zurückliefert. Das Trennzeichen c soll beim Aufsplitten "verschluckt" werden. Z.B. würde

   `split 'a' "Man kann es nicht glauben"`

   das Ergebnis ["M", "n k", "nn es nicht gl", "uben"] als Ergebnis liefern.

   (14 Punkte)

3. Schreiben sie eine Haskell-Funktion `join`, die ein Zeichen c und eine Liste von Zeichenketten $s_1, \ldots, s_n$ als Argumente erhält und den konkatenierten String $s_1 c s_2 \ldots c s_n$ zurückliefert. (14 Punkte)

Aufgabe 4.3 (40 Punkte)
Gegeben sei der folgende Datentyp für Binärbäume:

```
data Bintree a = Leaf a | Node (Bintree a) a (Bintree a) deriving (Read, Show)
```

Der Binärbaum `Node (Leaf 'D') 'B' (Leaf 'C')` hätte damit die Wurzel 'B' und das Blatt 'D' als linken und das Blatt 'C' als rechten Unterbaum.

1. Schreiben Sie eine Haskell-Funktion `countLeafs :: Bintree Int -> Int -> Int`, die bei Eingabe eines Binärbaums und einer ganzen Zahl n in dem Binärbaum die Anzahl der Blätter zählt, die einen Wert größer n haben. (12 Punkte)

2. Implementieren Sie eine Haskell-Funktion `nt :: Bintree a -> Bintree Int`, die in einem Binärbaum die Schlüsseleinträge an den Knoten durch die Nummern ersetzt, die die Knoten bei einem Inorder-Durchlauf erhalten. Z.B. würde sich für den oben genannten Binärbaum der Binärbaum `Node (Leaf 1) 2 (Leaf 3)` ergeben. (14 Punkte)

3. Definieren Sie eine Haskell-Funktion `balance :: Bintree a -> Bool`, die entscheidet, ob der Binärbaum ausgeglichen ist. Dabei heißt ein Binärbaum ausgeglichen, wenn für jeden Knoten die Anzahl der Blätter in seinem linken Unterbaum und die Anzahl der Blätter in seinem rechten Unterbaum sich höchstens um 1 unterscheiden. (14 Punkte)

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 5

Abgabe per Email bis 28.11.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

```
FP-Abgabe Gruppe 3
```

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 5.1 (35 Punkte)

In dieser Aufgabe legen wir den schon in Aufgabe 3.2 eingeführten Datentyp

```
type Graph a = [(a,[a])]
```

für Graphen zu Grunde.

1. Schreiben Sie Haskell-Funktionen für das Einfügen und Löschen von Knoten und Kanten in ungerichteten und gerichteten Graphen. Beachten Sie: Falls ein Knoten gelöscht wird, sind auch die zugehörigen Kanten zu entfernen. (5 Punkte)

2. Implementieren Sie eine Haskell-Funktion `dfs :: Eq a => Graph a -> a -> [a]`, so dass bei Eingabe eines schlingenfreien, ungerichteten Graphen `g` sowie eines Startknotens `s` die Liste `dfs g s` die Knoten in der Reihenfolge enthält, in der sie bei einem in `s` beginnenden Tiefensuchdurchlauf durch `g` zum ersten Mal besucht werden. (12 Punkte)

3. Schreiben Sie eine Haskell-Funktion `circle :: Eq a => Graph a -> [a]`, die bei Eingabe eines zusammenhängenden, schlingenfreien, ungerichteten Graphen `g` einen Kreis ausgibt, sofern einer existiert, und andernfalls die leere Liste. (18 Punkte)

Aufgabe 5.2 (40 Punkte)

Für natürliche Zahlen $n, m \in \mathbb{N}$ ist die arithmetische Differenz von n und m gleich $n - m$, falls $m < n$ und 0 sonst.
Gegeben sei nun der Datentyp für die natürlichen Zahlen

```
data Nat = Zero | S Nat deriving (Eq, Read, Show)
```

Schreiben Sie Haskell-Funktionen für die Addition, die arithmetische Differenz, die Multiplikation, die ganzzahlige Division von natürlichen Zahlen des Datentyps `Nat` sowie für die Kleiner-Relation zwischen zwei natürlichen Zahlen. (jeweils 8 Punkte)

Bemerkung: Eine Verwendung der entsprechenden Funktionen von `Int, Integer` usw. zur Definition der oben genannten Funktionen ist nicht erlaubt.

Aufgabe 5.3 (25 Punkte)

Ausgehend von einer vorgegebenen Menge AV von Aussagenvariablen wird in der Aussagenlogik die Menge AL der aussagenlogischen Formeln in der folgenden Weise als kleinste Menge, die die beiden folgenden Bedingungen erfüllt, definiert:

- Jede Aussagenvariable $A \in AV$ ist eine aussagenlogische Formel.
- Sind F und G aussagenlogische Formeln, so auch $\neg F$, $(F \wedge G)$, $(F \vee G)$ und $(F \rightarrow G)$.

1. Geben Sie einen geeigneten Datentyp `data AL` für die aussagenlogischen Formeln an.(3 Punkte)
2. Schreiben Sie eine Funktion `showForm :: AL -> String`, mit der aussagenlogische Formeln in einer Form als String dargestellt werden, die der üblichen Schreibweise nahekommt. (In der Wahl der Symbole für die Repräsentation von $\neg, \wedge, \vee, \rightarrow$ sind Sie frei.) (4 Punkte)
3. Schreiben Sie eine Funktion `nnf :: AL -> AL`, die eine aussagenlogische Formeln in ihre Negationsnormalform überführt. Hierbei ist eine aussagenlogische Formeln in Negationsnormalform, wenn ein Negationssymbol höchstens vor einer Aussagenvariablen steht. (7 Punkte)
4. Implementieren Sie eine Funktion `cnf :: AL -> AL`, die eine aussagenlogische Formel in eine äquivalente konjunktive Normalform überführt. (11 Punkte)

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 6

Abgabe per Email bis 05.12.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

`FP-Abgabe Gruppe 3`

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 6.1 (30 Punkte)

In dieser Aufgabe sind ungerichtete Graphen ohne Schlingen vorausgesetzt.
Eine *Euler-Tour* (auch *Euler-Kreis* genannt) in einem Graphen ist ein geschlossener Pfad durch den Graphen, der jede Kante des Graphen genau einmal durchläuft (und jeden Knoten mindestens einmal). Ein (ungerichteter) Graph hat genau dann eine Euler-Tour, wenn der Graph zusammenhängend ist und jeder Knoten des Graphen einen geraden Grad hat, d.h. an jedem Knoten eine gerade Anzahl von Kanten anliegt. Ein Graph mit diesen Eigenschaften wird *Euler-Graph* genannt.

1. Schreiben Sie eine Haskell-Funktion `euler :: Graph a -> Bool`, die für einen zusammenhängenden Graphen ermittelt, ob es ein Euler-Graph ist. (8 Punkte)

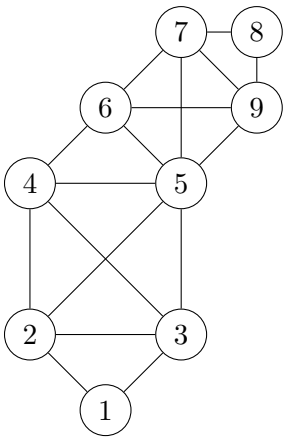
2. Ein elegantes und auch effizientes Verfahren zur Bestimmung einer Euler-Tour basiert auf der folgenden Beobachtung: Entfernt man die Kanten eines Kreises in einem Euler-Graphen, so sind die verbleibenden Zusammenhangskomponenten wiederum Euler-Graphen. Das Verfahren besteht nun darin, einen Kreis im Graphen zu finden, dessen Kanten zu entfernen, um dann rekursiv für jede Zusammenhangskomponte diese Schritte durchzuführen. Algorithmisch formuliert, in freier Formulierung nach `http://www.algorithmist.com/index.php/Euler_tour`, sieht das Verfahren wie folgt aus:

   Es sei `tour` die bisher erstellte Tour, zu Beginn die leere Liste.

```
findTour u:
    für jede Kante e=(u,v) in der aktuellen Kantenmenge E:
        entferne e aus E
        findTour v
    hänge u als Kopfelement an tour an.
```

Implementieren Sie auf der Basis dieses rekursiven Verfahrens eine Haskell-Funktion `eulerTour :: Eq a => Graph a -> [a]`, die zu einem Euler-Graphen eine Euler-Tour berechnet. (22 Punkte)

Testen Sie Ihre implementierte Funktion an dem folgenden Graphen:



Aufgabe 6.2 (Konstruktorbasierter Datentyp) (35 Punkte)

In dieser Aufgabe greifen wir auf Aufgabe 5.3 zurück.

Ein *Literal* ist entweder eine Aussagenvariable $A \in AV$ oder die Negation $\neg A$ einer Aussagenvariablen $A \in AV$. Eine *Klausel* ist eine Menge von Literalen, die wir im Folgenden als Liste von Literalen (ohne Wiederholungen) repräsentieren wollen. Eine Klausel entspricht logisch einer Disjunktion der in ihr enthaltenen Literale. Eine Formel der Aussagenlogik, die in konjunktiver Normalform vorliegt, kann daher als eine Menge (Liste) von Klauseln repräsentiert werden.

1. Schreiben Sie eine Haskell-Funktion, die eine aussagenlogische Formeln in konjunktiver Normalform in eine Liste von Klauseln transformiert. (12 Punkte)
2. Implementieren Sie eine Funktion, die zu einer Liste von Klauseln eine zugehörige konjunktive Normalform liefert. (11 Punkte)
3. Schreiben Sie eine Haskell-Funktion, die zu zwei Klauseln eine Resolvente berechnet, sofern diese existiert. (12 Punkte)

Aufgabe 6.3 (foldr, foldl) (35 Punkte)

Definieren Sie die folgenden Funktionen neu:

1. `reverse :: [a] -> [a]` unter Verwendung von `foldl`. (11 Punkte)
2. `concat :: [[a]] -> [a]` unter Verwendung von `foldr`. (11 Punkte)
3. `takeWhile :: (a -> Bool) -> [a] -> [a]` unter Verwendung von `foldr`. (13 Punkte)

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz
Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 7

Abgabe per Email bis 13.12.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

`FP-Abgabe Gruppe 3`

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 7.1 (Instanzdeklaration) (35 Punkte)

1. In Aufgabe 5.2 war der Datentyp `Nat` für die natürlichen Zahlen definiert worden. Führen Sie für diesen Datentyp eine oder mehrere geeignete Deklarationen als Instanz von Klassen durch, so dass für die in Aufgabe 5.2 zu definierenden Funktionen die üblichen Schreibweisen zur Verfügung stehen. (15 Punkte)

2. Wir betrachten die folgende Klasse von polymorphen Datentypen (dies ist durch den Typparameter a ausgedrückt), die Stacks repräsentieren:

```
class Stack s where
    push :: a -> s a -> s a
    top :: s a -> a
    pop :: s a -> s a
    isEmptyStack :: s a -> Bool
    showStack :: (Show a) => s a -> String
```

   Für die Methode `showStack` wird eine Default-Implementierung angegeben:

```
    showStack st = if isEmptyStack st then "!"
                   else (show (top st)) ++ "−" ++ (showStack (pop st))
```

   Implementieren Sie eine Instanz von `Stack` mittels Listen. (20 Punkte)

Aufgabe 7.2 (25 Punkte)
Definieren Sie eine Typklasse `Queue` mit den für Warteschlangen, die nach dem FIFO-Prinzip arbeiten, üblichen Operationen `emptyQueue`, `enqueue`, `dequeue`, `first` und der Abfrage `isEmptyQueue`. Geben Sie eine Default-Implementierung für `showQueue` an. `emptyQueue` leert dabei die Warteschlange.

Aufgabe 7.3 (40 Punkte)

Schreiben Sie ein Haskell-Programm, das mit Hilfe von binären Suchbäumen einfache Telefonlisten realisiert, in denen nur Name und Telefonnumer gespeichert sind. Menügesteuert oder per Abfrage sollen die folgenden Funktionen unterstützt werden:

- eine neue Telefonliste anlegen
- eine vorhandene Telefonliste öffnen
- einen Eintrag (Name, Telefonnumer) in eine geöffnete Telefonliste einfügen
- einen Eintrag in einer geöffneten Telefonliste löschen
- für eine geöffnete Telefonliste zu einem Namen die Telefonnumer finden
- Ausgabe der Telefonliste

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz
Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 8

Abgabe per Email bis 30.12.2008

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

```
FP-Abgabe Gruppe 3
```

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 8.1 (Klassen und Instanziierung) (40 Punkte)

1. In Haskell ist der Datentyp `Ordering` gegeben durch

   ```
   data Ordering = LE | EQ | GT
   ```

   Die Klasse `Ord` ist als eine von `Eq` abgeleitete Klasse definiert. Geben Sie diese Definition an. (5 Punkte)

2. Definieren Sie in Analogie zur Typklasse `Ord` eine Typklasse `PartOrd` für Halbordnungen mit entsprechenden Default-Implementierungen. Verwenden Sie an Stelle von `Ordering` den Datentyp

   ```
   data POrdering = PLT | PEQ | PGT | PNotComparable deriving (Eq)
   ```

   (15 Punkte)

3. Für Binärbäume wollen wir in dieser Aufgabe den folgenden Datentyp zugrunde legen:

   ```
   data BinTree a = Empty | BT (BinTree a) a (BinTree a).
   ```

   Eine Halbordnung $\prec$ ist auf der Menge der Binärbäume durch die folgende Festlegung gegeben (t_1 und t_2 seien Binärbäume):

   $t_1 \prec t_2$ genau dann wenn t_1 echter Unterbaum von t_2 ist.

   Definieren Sie `BinTree a` als Instanz von `PartOrd`, so dass insbesondere die Methoden von `PartOrd` zur Verfügung stehen. (20 Punkte)

Aufgabe 8.2 (IO-Monade) (60 Punkte)

Wir betrachten eine einfache Variante des Galgenmännchen-Spiels. Ein Wort, dessen Länge bekannt ist, soll mit höchstens n Rateversuchen, bei denen der Spieler Buchstaben des Worts rät, gefunden werden. Dabei wird ein Buchstabe, der richtig geraten wurde, in allen im Wort vorkommenden Positionen aufgezeigt.

Dieses Spiel soll nun als Haskell-Programm realisiert werden. Nach Aufruf des Programms, wobei die Anzahl n der Rateversuche als Argument mit übergeben wird, soll das Programm aus einer bestehenden Datei mit einer Liste von Wörtern (möglichst ohne Umlaute) zufällig eines auswählen und verschlüsselt – für jedes Zeichen des Wortes erscheint ein Bindestrich – auf dem Bildschirm ausgeben. In höchstens n Rateversuchen soll der Benutzer Buchstaben des Wortes raten. Korrekt geratene Buchstaben ersetzen im noch verschlüsselten Wort dann den jeweiligen Bindestrich. Hat der Benutzer nach höchstens n Versuchen das Wort vollständig erraten, so wird er zum Gewinner erklärt. Im anderen Fall ist er der Verlierer und das richtige Wort wird aufgezeigt.

Aufgabe 8.3 (freiwillige Zusatzaufgabe) (50 Punkte)

In dieser Aufgabe sollen Bäume als ungerichtete Graphen repräsentiert werden.

Schreiben Sie eine Haskell-Funktion, die folgendes leistet:

Bei Eingabe eines Baumes mit n Knoten und folglich $n-1$ Kanten ist eine Nummerierung der Knoten von 1 bis n und der Kanten von 1 bis $n-1$ so zu bestimmen, dass für jede Kante (a, b) gilt

$$\text{Kantennummer von } (a,b) = |\text{Knotennummer von } a - \text{Knotennummer von } b|.$$

Im Falle, dass keine solche Nummerierung existiert, soll eine passende Fehlermeldung ausgegeben werden

Aufgabe 8.4 (freiwillige Zusatzaufgabe) (50 Punkte)

Eine Registermaschine (kurz: RM) besteht aus einer zentralen Recheneinheit, einem Speicher und aus einem Programm.

Die zentrale Recheneinheit ihrerseits besteht aus zwei Registern: dem Befehlszähler und dem Akkumulator.

Der Speicher enthält unendlich viele Register: $R_1, R_2, \ldots$ mit den Adressen $1, 2, \ldots$. Der Akkumulator hat die Adresse 0.

Ein RM-Programm besteht aus einer endlichen Folge von RM-Befehlen, die mit 1 beginnend aufsteigend durchnummeriert sind. Wir legen die folgenden RM-Befehle zugrunde:

1. Ein- und Ausgabebefehle:
   - `LOAD` i
   - `STORE` i
2. Arithmetische Befehle mit Konstanten:
   - `CLOAD` i
   - `CADD` i
   - `CSUB` i
   - `CMULT` i
   - `CDIV` i
3. Arithmetische Befehle:
   - `ADD` i
   - `SUB` i
   - `MULT` i
   - `DIV` i
4. Sprungbefehle:
   - `GOTO` j
   - `JZERO` j
   - `END`

Es gibt in diesem Fall also keine indirekte Adressierung.

Geben Sie einen geeigneten Datentyp für Registermaschinen-Programme an. Schreiben Sie dann ein Haskell-Programm, mit dem ein RM-Programm, das als Textdatei gespeichert ist, eingelesen und dann anschließend simuliert wird. Das Programm soll insbesondere auch eine Simulation in Einzelschritten ermöglichen, so dass die Arbeitsweise der RM Schritt für Schritt nachvollzogen werden kann.

Fröhliche Weihnachten und einen guten Rutsch ins Neue Jahr!

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 10

Abgabe per Email bis 24.01.2009

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

`FP-Abgabe Gruppe 3`

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 10.1 (Monadischer Parser) (50 Punkte)

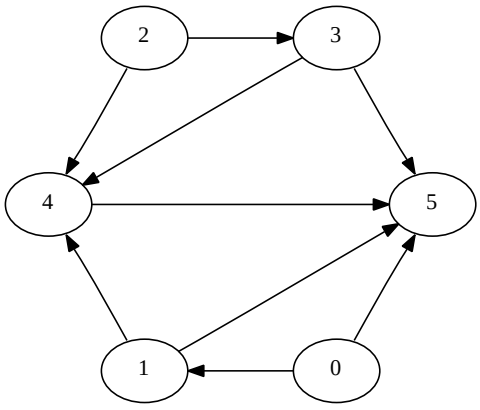
Ein gerichteter Graph wird in seiner einfachsten Form im `.dot`-Format in der folgenden Weise dargestellt:

```
digraph <Name des Graphen> {
<Folge von Kanten>
}
```

wobei die Kanten durch Semikolons voneinander getrennt sind und auch jeweils in eigenen Zeilen stehen dürfen. Z.B. stellt der folgende String

```
digraph a {
 0 -> 1 ; 0 -> 5 ; 1 -> 4 ; 1 -> 5 ; 2 -> 3 ; 2 -> 4 ; 3 -> 4 ; 3 -> 5 ; 4 -> 5
 }
```

die `.dot`-Repräsentation des folgenden gerichteten Graphen dar:



Schreiben Sie einen Parser vom Typ `Parser (Array (Int,Int) Int)`, der die `.dot`-Repräsentation eines gerichteten Graphen mit Knotenmenge $\{0, \ldots, n\}$ in die Adjazenzmatrix-Darstellung des Graphen parst.

Aufgabe 10.2 (Dynamische Programmierung) (50 Punkte)

Die Editierdistanz zweier Strings u und v ist die kleinste Anzahl elementarer Buchstabenoperationen, die erforderlich ist, um den String u in den String v zu überführen, wobei folgende elementare Buchstabenoperationen zur Verfügung stehen:

- einen neuen Buchstaben einfügen,
- einen Buchstaben löschen,
- einen bestehenden Buchstaben umbenennen.

Die maximal mögliche Editierdistanz von zwei Wörtern der Länge m und n beträgt $\max\{m, n\}$. Sind zwei Wörter gleich, ist die Editierdistanz 0.
Es sei nun $d(i, j)$ die Editierdistanz zwischen den Präfixen $u_1 \dots u_i$ von u und $v_1 \dots v_j$ von v. Offensichtlich ist $d(0, j) = j$ und $d(i, 0) = i$. Für $i, j > 0$ ergibt sich die folgende rekursive Beziehung

$$d(i, j) = \min(d(i, j-1) + 1, d(i-1, j) + 1, d(i-1, j-1) + \delta(i, j))$$

wobei

$$\delta(i, j) = \begin{cases} 0 & \text{falls } u_i = v_j \\ 1 & \text{sonst} \end{cases}$$

1. Begründen Sie die rekursive Gleichung von $d(i, j)$. (8 Punkte)
2. Implementieren Sie unter Verwendung von Arrays ein Haskell-Programm, das die Editierdistanz berechnet, und bestimmen Sie zu den beiden Strings „Java kann man gebrauchen" und „Haskell macht Spaß" die Editierdistanz. (42 Punkte)

FB Informatik LS I – Funktionale Programmierung – Prof. Dr. P. Padawitz

Hubert Wagner, Tristan Skudlik

Übungen zur Vorlesung

Funktionale Programmierung

Wintersemester 2008/09

Aufgabenblatt 9

Abgabe per Email bis 17.01.2009

Teilnehmer der Übungsgruppen 1, 2 , 5 und 6 schicken ihre Lösungen an `Hubert.Wagner@udo.edu`, Teilnehmer der Übungsgruppen 3 und 4 an `Tristan.Skudlik@udo.edu`. Die Betreff-Zeile muss dabei für Teilnehmer z.B. der Gruppe 3 folgendermaßen aussehen:

`FP-Abgabe Gruppe 3`

Wichtig:
Ihre Abgaben müssen als Haskell-Programm vorliegen, d.h. Sie müssen eine Haskell-Datei zuschicken, die vom Haskell-Interpreter (hugs oder ghci) ohne Fehlermeldung geladen werden kann. In Kommentarzeilen sollen zu Beginn die Übungsgruppennummer sowie die Autoren der Lösung (**mit Email-Adresse**) aufgeführt werden. Der Dateiname der Abgabe sollte dabei wie folgt aussehen:

`fp<Nr. des Blattes><Name eines Authors, mit Großbuchstben beginnend>.hs`.

Aufgabe 9.1 (Präsenzaufgabe) Die Menge der komplexen Zahlen $\mathbb{C}$, läßt sich auf natürliche Weise als Menge von Tupeln aus $\mathbb{R}^2$ auffassen.

Schreiben Sie einen einfachen Parser in Haskell, der komplexe Zahlen in der aus der Schulmathematik bekannten Standarddarstellung als Summe von Realteil und Imaginärteil (mit den üblichen vereinfachten Darstellungen, z.B. $2-3i$ statt $2+(-3)i$, 2 statt $2+0i$ usw.) parst und jeweils ein Tupel (a, b) zurückliefert.

Aufgabe 9.2 (Präsenzaufgabe) Der Datentyp für aussagenlogische Formeln sei

```
data AL = TRUE | FALSE | Var String | Not AL | And AL AL | Or AL AL
```

Schreiben Sie ohne Verwendung von `read` einen Parser für aussagenlogische Formeln, der in eine zu definierende BooleExpr-Algebra übersetzt.

Aufgabe 9.3 (Monadischer Parser) (50 Punkte)
Schreiben Sie einen Parser, der ein Gleichungssystem bestehend aus 2 Gleichungen in den beiden Variablen x und y parst und die Lösung dieses Gleichungssystems berechnet.
Jede Gleichung soll dabei in einer eigenen Zeile stehen und in der Gestalt $ax + by = c$ mit $a, b, c \in$ `Int` vorliegen.

Aufgabe 9.4 (Arrays) (50 Punkte)

Implementieren Sie die nachfolgend definierten Funktionen effizient unter Verwendung von dynamischer Programmierung.

1. $f : \mathbb{N} \to \mathbb{N}$ mit

$$\begin{aligned} f(0) &= 1 \\ f(k) &= \sum_{i=0}^{k-1} (k-i) \cdot f(i) \text{ für } k > 0 \end{aligned}$$

(20 Punkte)

2. $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ mit

$$\begin{aligned} g(0,n) &= fib\ n \\ g(1,n) &= g(0,n+1) + g(0,n) \\ g(n,m) &= g(n-1,m+1) + n \cdot g(n-2,m) \text{ für } n > 1 \end{aligned}$$

fib ist die schon bekannte Fibonacci-Funktion. (30 Punkte)

Hinweis: In dem Haskell-Modul *Dynamische Programmierung* auf der Veranstaltungswebseite ist die Vorgehensweise beispielhaft ausgeführt.