Praktische Informatik 3 WS 2002/03

Christoph Lüth <cxl>
Felix Beckwermert <foetus>
Michael Skibbe <mskibbe>
Dennis Walter <dw>
Rafael Trautmann <pirate>
Thomas Meyer <mclee>

3. Übungsblatt

Ausgabe: 18.11.2002
Bearbeitungszeit: Zwei Wochen

In diesem Übungsblatt gehen wir den wahren Kern der Informatik an: Textverarbeitung und Tabellenkalkulationen. Dies ist der erste Schritt in einem strategischen Programm zu einem eigenen Officepaket, das mit dem Aufkauf durch einen international agierenden Computerkonzern in Reichtum und der persönlichen Motoryacht in Saint Tropez enden wird.

5 *Haskell Word* — *10 Punkte*

Die Textverarbeitungskomponente unseres Officepaketes HASKELL XY besteht aus einer Funktion, die Text formatiert— irgendwo muß man ja schließlich anfangen.

Dazu definieren wir eine Funktion

```
format :: Formatter-> Int-> String-> String
```

die einen Text mit einer *Formatierungsvorschrift* (mehr dazu unten) auf die gegebene Breite formatiert.

Die Formatierung geht in drei Schritten vor sich:

1. Wir definieren erst einmal folgende Typen:

   ```
   type Word      = String
   type Paragraph = [Word]
   ```

   Der eingelesene Text wird in Zeilen zerlegt. Absätze werden durch (eine oder mehrere) Leerzeilen getrennt. Die Zeilen werden in Worte getrennt, und danach die Zeilen in einem Absatz wieder zusammengefasst, so dass der Text jetzt durch eine Folge von Absätzen besteht, die aus Folgen von Worten bestehen. All dieses wird durch die folgende zu implementierende Funktion geleistet:

   ```
   paragraphs :: String-> [Paragraph]
   ```

2. In einem zweiten Schritt wird jeder Paragraph in eine Folge von vorformatierten Zeile aufgelöst:

   ```
   type FormatLine = (Bool, [Word])
   type Formatter  = FormatLine-> String
   ```

   Eine vorformatierte Zeile besteht aus einer Liste von Worten, und einem Wahrheitswert, der angibt, ob diese Zeile die letzte in ihrem Absatz ist. Die Funktion `breakLines` bricht einen Absatz in einen vorformatierten Text auf, so dass jede vorformatierte Zeile nicht länger als die angegebene Zeile wird:

```
breakLines :: Int-> Paragraph-> [FormatLine]
```

Hierbei ist zu beachten, dass natürlich der Text länger wird als die Summe der Längen der Worte in der Zeile, da zwischen je zwei Worten mindestens ein Leerzeichen stehen muß.

3. Eine Formatierungsvorschrift ist etwas, was aus einer zu formatierenden Zeile eine Zeichenkette macht. Folgende Formatierungsvorschriften sollen implementiert werden:

```
center  :: Int-> Formatter
left    :: Int-> Formatter
right   :: Int-> Formatter
justify :: Int-> Formatter
```

Der Parameter ist hierbei die Zeilenbreite. Die Formatierungsvorschriften sollen zentrierten, linksbündigen, rechtsbündigen oder Blocksatz implementieren. Beim Blocksatz soll die letzte Zeile eines Absatzes nicht formatiert werden, d.h. linksbündig gesetzt sein.[1]

4. Schließlich wird noch die Hauptfunktion `format` interpretiert. Diese zerlegt den Text mittels `paragraph` und `breakLines` in eine Folge von vorformatierten Zeilen. Diese werden formatiert, und zu einem formatierten Text zusammengefügt.

Hinweise:

Folgende vordefinierten Haskell-Funktion können nützlich sein:

```
words :: String-> [String]
lines :: String-> [String]
unwords :: [String]-> String
unlines :: [String]-> String
replicate :: a-> Int-> [a]
intersperse :: a-> [a]-> [a]
```

Letzteres muß mit `import List (intersperse)` importiert werden. Außerdem kann es nützlich sein, eine Generalisierung von `words` zu implementieren:

```
fields :: (a-> Bool)-> [a]-> [[a]]
```

`fields` zerlegt eine Liste in zusammenhängende *Felder* von Elementen, für die das Prädikat wahr ist; Elemente, für welche das Prädikat falsch ist, werden fallen gelassen. Beispiel:

```
fields isDigit "234  32 xx 23 234"
["234","32","23","234"]
```

[1] Deshalb auch der Wahrheitswert im Typen `FormatLine`!

$\boxed{6}$ *Haskell Excel* *10 Punkte*

Entgegen der landläufigen Meinung wurden Tabellenkalkulationsprogramme nicht von der Firma MicroSoft erfunden, sondern gehen zurück auf das Programm VISICALC, welches im Jahre 1783 von Sir Isaac Newton implementiert wurde, der damit die spezielle Relativitätstheorie entwickelte— oder so ähnlich.

Das Tabellenkalkulationsprogram unseres Office-Paketes verwaltet sogenannte *spread sheets* (i.e. Tabellen). Ein *spread sheet* ist eine Tabelle, die entweder beliebige Zeichenketten, oder Formeln enthalten. Eine Formel kann dabei insbesondere auf andere Werte der Tabelle Bezug nehmen, und aus ihnen neue Werte berechnen.

Die Tabellenkalkulation wird in vier einfachen Schritten erstellt:

1. Eine Tabelle besteht aus Zellen. Eine Zelle enthält entweder
   - nichts (i.e. ist leer), oder
   - ein *label* (eine beliebige Zeichenkette), oder
   - einen Ausdruck, wobei ein Ausdruck wie folgt gebildet werden kann:
     - Addition, Subtraktion, Multiplikation oder Division zweier Ausdrücke;
     - Addition oder Multiplikation über Teile von Zeilen oder Spalten, also zum Beispiel Summe über alle Spalte eins bis drei in Zeile 5, oder Multiplikation über Zeile null bis fünf in Spalte drei;
     - eine konstante Zahl (`Double`);
     - eine Referenz auf eine andere Zelle in Zeile i und Spalte j. Es ist legal, auf eine Zelle zu verweisen, die leer ist, oder einen nichtnumerischen Wert enthält; in dem Fall ist der Wert der Referenz null.

   Definieren Sie zwei algebraische Datentypen `Cell` und `Expr`, die den Inhalt der Zellen der Tabelle, und darin erlaubte Ausdrücke modellieren.

   Definieren Sie danach einen Datentyp `Spreadsheet`, der eine Tabelle modelliert, entweder direkt als Abbilding

   ```
   type Spreadsheet = ((Int, Int)-> Cell, Int, Int)
   ```

   (wobei die beiden ganzen Zahlen die Größe der Tablle darstellen), oder als Liste von Listen.

2. Definieren Sie Funktionen

   ```
   empty    :: Spreadsheet
   set      :: (Int, Int)-> Cell-> Spreadsheet-> Spreadsheet
   get      :: Spreadsheet -> (Int, Int)-> String
   ```

   die eine leere Tabelle erstellen, eine bestimmte Zelle der Tabelle setzen, oder eine bestimmte Stelle auswerten.

3. Damit wird jetzt eine Funktion

```
prnt :: Int-> Spreadsheet -> String
```

implementiert, die eine Tabelle auswertet, und eine gut lesbar formatierte (d.h. alle Zellen auf die gleiche Breite formatierte) textuelle Repräsentation mit der im zweiten Argument angegebenen Breite zurückgibt. Hier eine Beispielausgabe:

```
*Excel> putStrLn (prnt 45 testsheet)
---------------------------------------------
|              |       Übung 1|            16|
---------------------------------------------
|              |       Übung 2|            19|
---------------------------------------------
|              |       Übung 3|             5|
---------------------------------------------
|              |       Übung 4|            10|
---------------------------------------------
|              |    Durchschn.|         12.50|
---------------------------------------------
```

Hier wird in Zelle (5,3) der Durchschnitt aus den Zellen 1 bis 3 der Spalte 3 berechnet.

4. Verkaufen — das Übungsblatt wird gerade rechtzeitig zum Weihnachtsgeschäft fertig.

Hinweise: Für das Verständnis und die Lösung dieser Aufgabe ist der Inhalt der Vorlesung am 25.11.02 nötig, also schlagen wir vor, in der ersten Woche die erste Aufgabe dieses Übungsblattes zu lösen.

Für die Auswertung ist es hilfreich, eine Funktion

```
eval :: Spreadsheet-> (Int, Int)-> Double
```

zu definieren, die den Wert der Tabelle an einer Stelle auswertet. Diese Funktion wird eine Hilfsfunktion haben, die rekursiv über den algebraischen Typen `Expr` definiert ist, und einen Ausdruck auswertet.

Um Listen von Koordinaten zu erzeugen, die in einer Konstante konstant sind, kann die Funktion `zip` zusammen mit der Aufzählung und der Funktion `repeat` verwendet werden:

```
zip :: [a] -> [b] -> [(a,b)]
repeat :: a -> [a]
Prelude> zip (repeat 3) [2..5]
[(3,2),(3,3),(3,4),(3,5)]
```

Dieses ist Version 1.0 von 19. November 2002.