Universität Bremen

Praktische Informatik 3 WS 22/23

Christoph Lüth
Raphael Baass
Thomas Barkowsky
Tede von Knorre
Alexander Krug
Tarek Soliman

4. Übungsblatt

Ausgabe: 14.11.22
Abgabe: 20.11.22

Ihr Bekannter der angehende Eismagnat (er hat für sein Eiscafé schon eine *holding company* auf den British Virgin Islands inkorporiert, und sich Visitenkarten aus Aluminium mit dem Aufdruck *The Ice Age is Coming* machen lassen) ist mit der Auswertungsfunktionalität des von Ihnen implementierten Business Intelligence Paketes nicht zufrieden. "Ich brauche einen Spreadsheet", beschwert er sich. Da wir weder Bedürfnis noch Notwendigkeit verspüren uns mit der idiosynkratischen Syntax der Software des Branchenführers zu beschäftigen implementieren wir kurzerhand unseren eigenen Spreadsheet. Und billiger ist es auch!

4.1 *Tabellenkalkulation leichtgemacht.* *4 Punkte*

Ein Spreadsheet ist im wesentlichen eine zweidimensionale Tabelle von Zellen; was genau eine Zelle ist klären wir gleich. Im Internet[1] finden wir ein Modul, welches zweidimensionale Tabellen als Datentyp `Table a` mit folgenden Funktionen modelliert:

```
data Table a

new  :: a→ Int→ Int→ Table a
rows :: Table a→ Int
cols :: Table a→ Int

set  :: (Int, Int)→ a→ Table a→ Maybe (Table a)
get  :: (Int, Int)→ Table a→ Maybe a
```

Mit `new` können wir eine neue Tabelle der spezifizierten Größe (Breite oder Anzahl der Spalten, Höhe oder Anzahl der Zeilen) erzeugen, die mit dem angegeben Element gefüllt ist. `set` und `get` ändern einzelne Zellen (indiziert durch Spalte, Zeile) bzw. geben den Wert zurück. Indizes werden ab 1 gezählt; für illegale Indizes — kleiner gleich 0 oder größer als die Anzahl Zeilen/Spalten — wird `Nothing` zurückgegeben. `rows` und `cols` gibt die Anzahl Zeilen bzw. Spalten zurück.

Die *Zellen* unseres Spreadsheets sind entweder leer, oder sie enthalten einen Ausdruck, oder eine Zeichenkette. Mit Ausdrücken können wir rechnen (wobei wir immer in doppeltgenauen Fließkommazahlen nach IEEE 754[2] rechnen), Zeichenketten werden einfach nur dargestellt:

```
data Cell  = Empty | Expr Expr | Label String
type  Spreadsheet = Table Cells
```

Als Ausdrücke sind zulässig:

- Konstante Zahlen,
- Referenzen auf ein andere Zelle,
- die binären Operatoren ($+$, $-$, $\cdot$, $/$, min, max) angewandt auf zwei Argumente, oder
- binäre Operatoren angewandt auf eine Liste.

Hierbei werden Listen durch den Datentyp `Range` dargestellt; es ist entweder eine Liste innerhalb einer Spalte (d.h. ein Teil einer Spalte) oder eine Liste in einer Zeile. Beispielsweise ist `ColRange 1 (3,5)` die dritte bis fünfte Zelle der 1. Reihe, d.h. die Zellen `[(1,3), (1,4), (1,5)]`:

```
data Expr  = Num Double
           | Ref (Int, Int)
```

[1] Genauer gesagt in der Vorlage für dieses Übungsblatt
[2] Was insbesondere bedeutet, dass die Division durch 0 die Konstanten `infinity` (oder `-infinity`) ergibt.

```
          | BinOp BinaryOp Expr Expr
          | RangeOp BinaryOp Range

data BinaryOp = Plus | Minus | Times | Div | Max | Min

data Range = ColRange   Int (Int, Int)
           | RowRange (Int, Int) Int
```

Wir implementieren zuerst die Funktionen, die ein Spreadsheet erzeugen, manipulieren und inspizieren.

1. `empty w h` erzeugt einen leeren Spreadsheet der genannten Größe (Breite `w` und Höhe `h`), dessen Zellen mit `Empty` gefüllt werden:

   ```
   empty :: Int→ Int→ Spreadsheet
   ```

2. Folgende Funktionen verändern einzelne Zellen eines Spreadsheets (für illegale Indizes soll ein Laufzeitfehler (`error`) geliefert werden:

   ```
   setExpr  :: (Int, Int)→ Expr→ Spreadsheet→ Spreadsheet
   setLabel :: (Int, Int)→ String→ Spreadsheet→ Spreadsheet
   setNum   :: (Int, Int)→ Double→ Spreadsheet→ Spreadsheet
   ```

3. Folgende zwei Funktionen erlauben den Zugriff auf den Spreadsheet, wobei `get` den uninterpretierten Inhalt der Zelle zurückgibt, und `eval` diesen auswertet. Für illegale Indizes soll `Empty` bzw. 0 zurückgegeben werden:

   ```
   getCell :: Spreadsheet→ (Int, Int)→ Cell
   eval    :: Spreadsheet→ (Int, Int)→ Double
   ```

   Es ist legal, Zellen auszuwerten (oder darauf zu verweisen), die leer sind oder einen nichtnumerischen Wert enthalten; auch illegale Referenzen sind erlaubt. In all diesen Fällen ergibt die Auswertung der der Referenz 0. Zyklische Verweise sind auch legal; in diesem Fall terminiert die Auswertung nicht.[3]

Hier wird eine kleine Beispieltabelle aufgebaut, und exemplarisch ausgewertet (`|>` ist die umgedrehte Funktionsanwendung, `x |> f = f x`):

```
test1 =
  empty 4 2
  |> setLabel (1, 1) "Montag"
  |> setLabel (2, 1) "Dienstag"
  |> setLabel (3, 1) "Mittwoch"
  |> setLabel (4, 1) "Maximum"
  |> setVal (1, 2) 16
  |> setVal (2, 2) 19
  |> setVal (3, 2) 6
  |> setExpr (4, 2) (RangeOp Max (RowRange (1, 3) 2))

getCell test1 (2, 1)  ⇝ Label "Dienstag"
eval test1 (4, 1)  ⇝ 0
eval test1 (4, 2)  ⇝ 19
```

4.2 *Ergebnisausgabe* *4 Punkte*

Natürlich will niemand seine Tabelle Zelle für Zelle mühsam inspizieren, deshalb implementieren wir zwei Funktionen, die eine Zelle und eine ganze Tabelle darstellen:

```
printCell :: Spreadsheet→ Int→ (Int, Int)→ String
prnt :: Int→ Spreadsheet→ String
```

[3] Das ist ein Feature, kein Bug: Der Benutzer wollte es eben so.

`printCell` stellt den (ausgewerteten) Inhalt der Zelle in der angebenen Breite (dritter Parameter) rechtsbündig dar. Zahlen werden mit zwei Nachkommastellen ausgegeben; ist die Zahl zu groß für die angegebene Breite, wird die Zelle nur mit # gefüllt, Zeichenketten werden abgeschnitten wenn sie zu lang sind.

In unserer Beispieltabelle von oben liefert `printCell` f<olgende Ausgaben:

```
printCell test1 6 (3,2)  ⇝  "␣␣7.50"
printCell test1 3 (2,2)  ⇝  "###"
```

`prnt` gibt die gesamte Tabelle in der angegeben Breite aus. Hier ist die Beispielausgabe für den Test von oben:

```
putStr (prnt 8 test1)  ⇝

+--------+--------+--------+--------+
|  Montag|Dienstag|Mittwoch| Maximum|
+--------+--------+--------+--------+
|    6.50|    7.90|    7.50|    7.90|
+--------+--------+--------+--------+
```

4.3 *Business Intelligence* *2 Punkte*

Ihr Bekannter ist von Ihrer Implementierung wenig begeistert. “Ich brauche ein *dashboard* für meine *KPIs*, damit ich immer *actionable* bin.” sagt er. Unklar was er damit genau meint, aber wahrscheinlich will er kein Haskell programmieren, um seine Daten einzugeben. Deshalb lassen Sie sich überreden, eine Funktion

```
importSales :: [(String, Double)]→ Spreadsheet
```

zu implementieren, die aus den Tagesumsätzen in der Form

```
[("Montag", 280.35), ("Dienstag", 210.25), ("Mittwoch", 220.30),
 ("Donnerstag", 280.50), ("Freitag", 323.45), ("Sonnabend", 525.75), ("Sonntag", 518.25)]
```

einen Spreadsheet generiert, der ausgegeben in diesem Beispiel wie folgt aussieht (ausgegeben mit `prnt 10`):

```
+----------+----------+----------+
|       Tag|    Umsatz|Abweichung|
+----------+----------+----------+
|    Montag|    280.35|    -56.62|
+----------+----------+----------+
|  Dienstag|    210.25|   -126.72|
+----------+----------+----------+
|  Mittwoch|    220.30|   -116.67|
+----------+----------+----------+
|Donnerstag|    280.50|    -56.47|
+----------+----------+----------+
|   Freitag|    323.44|    -13.52|
+----------+----------+----------+
| Sonnabend|    525.75|    188.77|
+----------+----------+----------+
|   Sonntag|    518.25|    181.27|
+----------+----------+----------+
|     Summe|   2358.84|          |
+----------+----------+----------+
|Durchschn.|    336.97|          |
+----------+----------+----------+
```

Hierbei ist `Summe` die Summe der Umsätze, Durchschnitt der durschnittliche Umsatz, und die Spalte `Abweichung` die Differenz zwischen Tagesumsatz und durschnittlichem Umsatz. Die Tage und Umsätze (Spalte 1 bzw. 2, Zeile 2-8) sollen aus der Argumentliste gefüllt werden, die restlichen Zellen sollen mit entsprechenden konstanten Ausdrücken gefüllt werden. Sie können nicht davon ausgehen, dass die Liste genau sieben Tage umfasst (es könnten beispielsweise Feiertage dabei sein).

Änderungen:

- *Version 1.0: Ausgegebene Version.*
- *Version 1.1: Inkonsistenzen zur Vorlage behoben.*
- *Version 1.2: Zeichenketten sind wie Zahlen rechtsbündig, Beispiel korrigiert.*