Universität Bremen

Reaktive Programmierung SoSe 17

Christoph Lüth
Martin Ring

3. Übungsblatt

Ausgabe: 11.05.17
Abgabe: 25.05.17

3.1 *Aktoren in Haskell* *5 Punkte*

In dieser Aufgabe wollen wir Aktoren wie wir sie in Scala kennengelernt haben in Haskell implementieren. Als Vorlage dient die Datei `Actors.hs` aus der Übung (bzw. in `vorlage-03.zip`).

Für dieses Übungsblatt beschränken wir uns auf folgende Eigenschaften von Aktoren:

1. Jeder Aktor ist Teil eines Aktorsystems
2. Aktoren kommunizieren ausschließlich über Nachrichten und agieren nur bei deren Verarbeitung.
3. Aktoren verarbeiten Nachrichten in Reihenfolge des Eingangs (FIFO).
4. Aktoren können während der Verarbeitung einer Nachricht:
   - Neue Aktoren erzeugen,
   - Nachrichten an sich selbst oder andere Aktoren versenden,
   - und bestimmen wie die nächste Nachricht verarbeitet werden soll.
5. Die Nachrichtenübertragung erfolgt asynchron: Der Kontrollfluss wird beim Senden unmittelbar an den sendenden Aktor zurückgegeben. Die Nachricht wird zu einem unbestimmten Zeitpunkt in der Zukunft vom Empfänger vearbeitet.
6. Wenn das Verhalten eines Aktors fehlschlägt wird dies in Form einer Nachricht dem Aktor mitgeteilt, der den fehlerhaften Aktor erzeugt hat (oder dem Aktorsystem bei Top-Level Aktoren).

Untersuchen Sie die Vorlage bezüglich dieser Eigenschaften. Welche Eigenschaften sind bereits erfüllt, welche noch nicht? Implementieren Sie die fehlenden Eigenschaften.

3.2 *Hoogle Sheets* *15 Punkte*

In dieser Aufgabe wollen wir endlich mal etwas Handfestes implementieren. Mit Hilfe der Haskell-Aktoren aus Aufgabe 3.1, oder alternativ mit den Scala-Aktoren in Akka implementieren wir *Hoogle Sheets* — eine aktorbasierte Tabellenkalkulation mit Webinterface.

Das Webinterface sowie einen Haskell- bzw. Scala-Server finden sie in `vorlage-03.zip` (Siehe `README.md`). Der Webclient muss für diese Aufgabe nicht verändert werden.

Im Webclient können Zellen angeklickt werden und Formeln eingetragen werden. Die Formeln werden nach dem Drücken der Enter-Taste an den Server übertragen, und sollen dort verarbeitet werden.

Die Syntax der Formeln ist wie folgt:

```
Expr   := Term | Expr '+' Term | Expr '-' Term
Term   := Factor | Term '*' Factor | Term '/' Factor
Range  := Cell ':' Cell
Factor := Number | '(' Expr ')' | Cell | 'REDUCE' '(' Range ',' BinOp ')'
BinOp  := '+' | '-' | '*' | '/'
Cell   := Column Number
Column := 'A' | .. | 'Z'
Number := Digit+
Digit  := '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Die Semantik sollte sich weitestgehend von selbst erklären. `Range` spannt ein Rechteck auf zwischen zwei Zellen und beinhaltet alle Zellen innerhalb des Rechtecks (inklusive der angegebenen Zellen). Die Funktion `REDUCE` erwartet eine `Range` und kombiniert alle enthaltenen Zellen mit dem angegebenen binären Operator zu einem Wert (wie die gleichnamige Scala-Funktion); der `Range` soll nicht leer sein, sollte hierbei von oben nach unten Zeile für Zeile, und in jeder Zeile von links nach rechts durchlaufen werden.

Der Spreadsheet wird durch eine Aktoren-Infrastruktur implementiert, bei der pro (nicht-leerer) Zelle des Spreadsheets ein Aktor für die Berechung und Propagation des Wertes der Zelle zuständig ist.

Implementieren sie diese Infrastruktur auf dem Server:

1. Ein Aktor `SpreadSheet` soll für jeden Client zunächst ein leeres Spreadsheet repräsentieren.
2. Wenn vom Client Formeln gesendet werden, sollen für die entsprechenden Zellen Aktoren (`Cell`) erzeugt werden.
3. In den `Cell`-Aktoren werden die Formeln parsiert und deren Ergebnisse berechnet, indem für jeden Teilausdruck eine Aktor-Hierarchie erstellt wird (für `Expr`, `Term`, `Range`, etc.). Wenn eine Abhängigkeit zum Wert einer Zelle besteht, muss der Wert dieser Zelle "abonniert" werden. Das Ergebnis des Ausdrucks soll sich anpassen, wenn sich die Werte abhängiger Zellen ändern.
4. Fehler bei der Berechnung (Division durch Null, ungültige Syntax, ungültiger Range oder Referenzen, etc.) sollen in den jeweiligen Zellen angezeigt werden und nicht zum Absturz des Spreadsheets führen.

Viel Spaß beim Rechnen.