Christoph Lüth
Marcus Ermler
Christian Maeder
Martin Ring
Diedrich Wolter

8. Übungsblatt

Ausgabe: 12.12.12
Abgabe: 21.12.12

Bremen ist einer der Hauptstandorte der deutschen Luft- und Raumfahrtindustrie. Nun müssen auch auf der internationalen Raumstation Texte (wie Emails oder Protokolle) geschrieben werden, aber aufgrund der rigorosen Anforderungen an Software in der Raumfahrt [1] können wir nicht einfach irgendeinen, wahrscheinlich fehlerhaften, Texteditor dafür hernehmen, sondern müssen die Korrektheit der Software durch umfangreiche Tests sicherstellen. Eine bekannte Bremer Raumfahrtfirma hat hierfür die Universität Bremen um Hilfe gebeten. Wir erledigen den Job mit Haskell und Quickcheck.

8.1 *Anforderungen: Das muss.* *10 Punkte*

Unser Texteditor hat einen internen Zustand, der durch den abstrakten Datentypen Editor repräsentiert wird, und soll folgende Operationen bereitstellen:

- Der Zustand des Editors beinhaltet u.a. den edierten Text, die Cursorposition und das Clipboard. Die folgenden drei Selektorfunktionen geben Zugriff auf diese Komponenten des Zustandes:

  ```
  text      :: Editor→ String
  pos       :: Editor→ Int
  clipboard :: Editor→ String
  ```

  Die Cursorposition ist die Stelle, an der eingefügt oder gelöscht wird. Sie soll sich stets zwischen vor dem ersten und hinter dem letzten Zeichen befinden.

- Der leere Texteditor ist der Ausgangspunkt allen Schaffens:

  ```
  empty :: Editor
  ```

  Am Anfang sind Text und Clipboard leer.

- Mit der Funktion move wird der Cursor bewegt:

  ```
  move :: Int→ Editor→ Editor
  ```

  Der Text soll unverändert bleiben. Ein negatives Argument bewegt den Cursor rückwärts (zum Anfang des Textes hin).

- Mit der Funktion insert wird der gegebene Text an der Cursorposition eingefügt:

  ```
  insert :: String→ Editor→ Editor
  ```

  Der Cursor ist danach hinter dem eingefügten Text positioniert.

- Die Funktion cut entfernt n Zeichen ab der aktuellen Cursorposition, und kopiert sie in das Clipboard; die Funktion copy kopiert n Zeichen ab der aktuellen Cursorposition in das Clipboard.

  ```
  cut  :: Int→ Editor→ Editor
  copy :: Int→ Editor→ Editor
  ```

  Ist der Text ab der Cursorposition kürzer als n Zeichen, wird der gesamte Resttext in das Clipboard kopiert und ggf. gelöscht. Für $n \leq 0$ sind beide Funktionen die Identität. Die Cursorposition bleibt in allen Fällen unverändert. Diese beiden Funktionen sind die einzigen, welche den Inhalt des Clipboards verändern.

- Die Funktion paste fügt das Clipboard an der aktuellen Cursorposition ein:

  ```
  paste :: Editor→ Editor
  ```

  Die Cursorposition ist danach hinter dem eingefügten Text.

- Die Funktion format formatiert den Text im Editor auf die angegebene Zeilenlänge:

  ```
  format :: Int→ Editor→ String
  ```

  Der von format n erzeugte Text soll dieselben Worte wie der Textinhalt des Editors enthalten, und aus durch Zeilenvorschub getrennten Zeilen bestehen, welche nicht länger als n Zeichen sind, es sei denn, sie bestehen nur aus einem einzigen Wort, wobei ein Wort eine zusammenhängende Folge von Nichtleerzeichen ist (wie durch die vordefinierte Funktion words berechnet).

Formalisieren Sie die textuell angeführte Anforderungsspezifikation des Texteditor als Liste von Eigenschaften, die Sie mit Quickcheck überprüfen können.

Hinweise:

1. Auf der Webseite finden Sie die Dateien `Editor.hs`, welche die Deklaration der zu implementierenden Editorfunktionen enthält, und `EditorTest.hs`, welche die erforderliche magische Inkantation enthält. um in Quickcheck einen zufälligen Zustand des Editors zu erzeugen. Mit diesen beiden Dateien können Sie die Tests übersetzen (aber natürlich noch nicht laufen lassen), ohne den Editor implementiert zu haben.
2. Die folgenden vordefinierten Haskell-Funktionen *können* für diese Aufgabe hilfreich sein:

   ```
   lines   :: String→ [String]
   unlines :: [String]→ String
   words   :: String → [String]
   unwords :: [String]→ String
   ```

8.2 *Implementation: Es tut.* *10 Punkte*

Implementieren Sie danach den Texteditor mitsamt den oben angeführten Eigenschaften, und stellen Sie mit Hilfe der Tests aus Aufgabe 8.1 und Quickcheck sicher, dass Ihre Implementation die Anforderungsspezifikation erfüllt.

Welche Situationen werden durch die Anforderungsspezifikation nicht abgedeckt?

? *Verständnisfragen*

1. Wie können wir die Typen und Operationen der Signatur eines abstrakten Datentypen grob klassifizieren, und welche Auswirkungen hat diese Klassifikation auf die zu formulierenden Eigenschaften?
2. Warum „finden Tests Fehler“, aber „zeigen Beweise Korrektheit“, wie in der Vorlesung behauptet? Stimmt das immer?
3. Müssen Axiome immer ausführbar sein? Welche Axiome wären nicht ausführbar?

Literatur

[1] RTCA/DO-178B. *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., Washington, D.C. 20036, 1992.