Universität Bremen

Praktische Informatik 3 WS 18/19

Christoph Lüth
Thomas Barkowsky
Andreas Kästner
Gerrit Marquardt
Tobias Haslop
Matz Habermann
Berthold Hoffmann

9. Übungsblatt

Ausgabe: 18.12.18
Abgabe: 15.01.19 12:00

In diesem Übungsblatt geht es um bunte Bilder. Genauer gesagt erstellen wir erst einen sehr einfachen ADT um Bilder erstellen zu können, und formulieren dann Eigenschaften, welche die Korrektheit des ADT überprüfen.

Um Eigenschaften sinnvoll testen zu können, benutzen Sie Quickcheck (wie in der Vorlesung vorgestellt). Quickcheck ist in das Ihnen bereits bekannte Testframework Tasty integriert; Sie müssen nur das Modul dafür noch mit cabal installieren:

```
cabal install tasty-quickcheck
```

9.1 *Grafiken erstellen* *8 Punkte*

Unser Grafikeditor funktioniert nach dem Prinzip der Turtle-Grafik[1]. Hierbei wird ein Cursor (die Turtle) über eine Zeichenfläche bewegt, und kann (relativ zur momentanen Position) Linien zeichnen. (Der Einfachheit halber zeichnen wir nur rechtwinklige Linien). Die Turtle hat immer eine *Richtung*, in die sie zeigt, und eine *momentane Farbe*, in der sie zeichnen kann.

Wir implementieren den Grafikeditor mit einem abstraken Datentyp T. Dieser repräsentiert den Zustand der Zeichenfläche, mit Komponenten für die Größe, momentane Position, Richtung und Zeichenfarbe der Turtle, sowie den Zustand der eigentlichen Zeichenfläche. Dieser Zustand ist eine Abbildung von Koordinaten (x, y) auf eine Farbe (repräsentiert durch einen Aufzählungsdatentyp Colour), und kann beispielsweise als Array[2] implementiert werden.

Die Operationen der Turtle sind wie folgt:

- Mit der Funktion start wird eine leere (d.h. schwarze) Zeichenfläche der angegeben Größe erzeugt. Die Turtle steht in der Mitte der Zeichenfläche, und ist nach oben ausgerichtet; die momentane Farbe ist weiß.
- Mit turn kann die Turtle sich nach links oder rechts drehen.
- Mit move bewegt sich die Turtle die angegeben Anzahl Felder (ohne zu zeichnen); falls die Bewegung sie außerhalb der Zeichenfläche führen würde, bleibt die Turtle am Rand stehen.
- Mit draw bewegt sich die Turtle die angegeben Anzahl Felder und zeichnet dabei eine Linie in der momentanen Farbe. Die Turtle steht danach *hinter* dem zuletzt gezeichneten Pixel. Falls die Bewegung sie außerhalb der Zeichenfläche führen würde, bleibt die Turtle am Rand stehen.
- Mit colour kann die momentane Farbe inspiziert, und mit setColour gesetzt werden.
- Die Funktion size gibt die Größe des Zeichenfeldes zurück, und die Funktion pixels gibt die *Anzahl* der Pixel in der angegeben Farbe zurück. Ferner gibt die Funktion colourAt den Zustand des Pixels an den angegebenen Koordinate zurück.

In Haskell ist die Signatur der Schnittstelle ist wie folgt:

```
data T

start :: Int→ Int→ T
```

[1] http://de.wikipedia.org/wiki/Turtle_Grafik
[2] Modul Data.Array

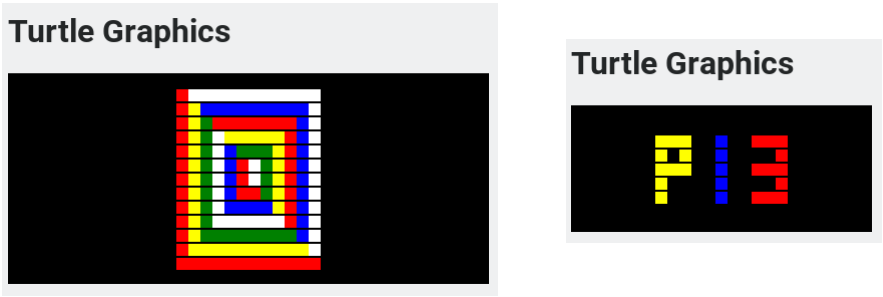

Abbildung 1: Beispielgrafiken spiral und pi3 (aus TurtleTest.hs)

```
data Dir = L | R
turn :: Dir→ T → T

move :: Int → T→ T
draw :: Int → T→ T

data Colour = Black | Red | Green | Blue | Yellow | White
colour :: T→ Colour
setColour :: Colour → T → T

size :: T→ (Int, Int)
pixels :: T → Colour → Int
```

Wir definieren folgende Hilfsfunktion:

```
(|>) :: T→ (T→ T)→ T
(|>) = flip ($)
```

Damit können wir leicht Sequenzen von Zeichenkommandos konstruieren. Dieses zeichnet eine Line:

```
line :: T
line = start 20 10 |> turn L |> setColour Red |> move 5 |> turn R |> turn R |> draw 10
```

In dieser Aufgabe implementieren Sie den Grafikeditor im Modul Turtle. Ihr Turtle-Modul sollte für die Test-Grafiken eine ähnliche Ausgabe wie in Abb. 1 erzeugen.

Hinweise:

1. Die Datei TurtleTest.hs enthält die obigen Tests und noch weitere.
2. Um eine erzeugte Grafik zu visualisieren steht in der Datei Turtle.hs bereits eine Funktion render bereit, welche Sie mit Ihrer Repräsentation des Zeichenfeldes kombinieren können, um den Zustand (als Instanz der Typklasse Show) in einer Zeichenkette zu repräsentieren. Die Ausgabe erfolgt über Kontrollsequenzen[3] auf der Konsole, was unter Windows nicht immer funktioniert.

   Deshalb können Sie alternativ die Funktion

   ```
   displayTo :: String→ T→ IO ()
   ```

   aus dem Modul TurtleDisplay nutzen. Diese Funktion schreibt eine HTML-Repräsentation des Zeichenfläche in die angegebene Datei, die mit jedem handelsüblichen Webbrowser plattformübergreifend gelesen werden kann (das sieht dann aus wie in Abb. 1).[4]

9.2 *Eigenschaften* *12 Punkte*

Jetzt wollen wir zeigen, dass der Grafikeditor auch korrekt uist. Im folgenden charakterisieren wir die Korrektheit durch einige natürlichsprachlich formulierte Bedingungen:

[3] VT100 colour codes

[4] Um TurtleDisplay.hs übersetzen zu können, müssen Sie ggf. die Bücherei blaze–html installieren (`cabal install blaze-html`).

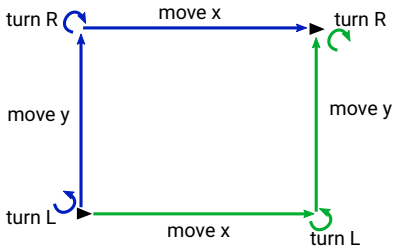

Abbildung 2: Zwei gleiche Bewegungssequenzen (grün und blau).

- Eigenschaften der Drehungen:
  - (1) Erst nach rechts, dann wieder nach links gedreht ist die Identität.
  - (2) Zweimal nach rechts gedreht ist dasselbe wie zweimal nach links.
  - (3) Viermal in die gleiche Richtung gedreht ist die Identität.
- Eigenschaften der Bewegung:
  - (4) Die Bewegung um die Strecke 0 ist die Identität.
  - (5) Wenn x und y größer oder gleich 0 sind, ist sich erst um x und dann um y zu bewegen das gleiche wie sich um $x + y$ zu bewegen.
  - (6) Bewegung im Rechteck: sich erst um x zu bewegen, dann nach links zu drehen, dann um y, und dann nach rechts zu drehen, ist dasselbe wie: erst nach links drehen, dann um y bewegen, dann nach *rechts* drehen, und sich dann um x bewegen. (Man bewegt sich dann im Rechteck, siehe Abb. 2.)
  - (7) Rückwärtsbewegung: sich um $-x$ zu bewegen ist dasselbe wie: sich erst zweimal nach links zu drehen, dann um x zu bewegen, dann zweimal nach rechts zu drehen.
- Eigenschaften des Zeichnens:
  - (8) Wenn ich n Felder zeichne ist die Anzahl der Pixel in der momentanen Farbe danach gleich der Anzahl der Pixel in der momentanen Farbe davor plus n.
  - (9) Wenn ich n Felder zeichne, dann ist für Farben, die nicht die momentane Farbe sind, die Anzahl der Pixel danach gleich der Anzahl der Pixel davor.

(i) Formalisieren Sie die Eigenschaften (1) bis (7) als QuickCheck-Properties (in der Datei `TurtleSpec.hs`), und prüfen Sie, ob Ihre Implementierung die Eigenschaften erfüllt.

(ii) Warum gelten Eigenschaften (8) und (9) nicht? Ersetzen Sie (8) und (9) durch zwei oder mehre *gültige* Aussagen, die Sie erst natürlichsprachlich formulieren, und dann als QuickCheck-Properties formalisieren.

(iii) Formulieren Sie drei weitere nicht-triviale Eigenschaften der Turtle-Grafik erst natürlichsprachlich (wie (1) bis (9) oben), und formalisieren Sie diese dann als QuickCheck-Properties. Die Eigenschaften sollten keine bloße Paraphrasierung der Implementation sein. Sie können dazu Hilfsfunktionen definieren. Die Export-Schnittstelle von `Turtle` sollten sie dabei nicht ändern.

Am Ende sollte Ihre Implementierung die korrekt formulierten Eigenschaften erfüllen — es muss also *beides* korrekt sein, Spezifikation und Implementation.