Universität Bremen

Praktische Informatik 3 WS 16/17

Christoph Lüth
Tobias Brandt
Tristan Bruns
Johannes Ganser
Berthold Hoffmann
Alexander Kurth

10. Übungsblatt

Ausgabe: 10.01.17
Abgabe: 20.01.17, 12:00 Uhr

In diesem Aufgabenblatt wollen wir uns den bildenden Künsten widmen, spezifisch der Malerei. Da wir Informatiker sind, wollen wir keine Bilder malen, sonderen Programme schreiben, die Bilder malen.

Dafür wurde die Sprache TSVG erfunden. TSVG ist eine imperative, domänenspezifische Skriptsprache der vierten Generation. Sie ist eine Erweiterung der in der Vorlesung vorgestellten Programiersprache IMP für die Grafik-Domäne. Wir wollen einen Interpreter für TSVG implementieren, der aus TSVG-Programmen SVG-Grafiken[1] erzeugt, die wir mit einem handelsüblichen Browser oder Grafikprogramm anzeigen können.

10.1 *TSVG: The Parser* — *8 Punkte*

Zuerst brauchen wir einen Parser und abstrakte Syntax. Die Grammatik für TSVG erweitert die Ausdrücke von IMP um Punkte, sowie die Kommandos der Sprache um spezifische Grafikkommandos (*drawcmd*):

$$
\begin{array}{rcl}
\mathit{expr} & ::= & \ldots \\
 & | & \texttt{<}\ \mathit{expr}\ \texttt{,}\ \mathit{expr}\ \texttt{>} \\
\mathit{cmd} & ::= & \ldots \\
 & | & \mathit{drawcmd} \\
\mathit{drawcmd} & ::= & \texttt{move}\ \mathit{expr} \\
 & | & \texttt{line}\ \mathit{expr} \\
 & | & \texttt{curve}\ \texttt{(}\ \mathit{expr}\ \texttt{,}\ \mathit{expr}\ \texttt{,}\ \mathit{expr}\ \texttt{)} \\
 & | & \texttt{circle}\ \mathit{expr} \\
 & | & \texttt{slice}\ \texttt{(}\ \mathit{expr}\ \texttt{,}\ \mathit{expr}\ \texttt{,}\ \mathit{expr}\ \texttt{)} \\
 & | & \texttt{colour}\ \mathit{identifier}
\end{array}
$$

Ein TSVG-Programm besteht aus einer Kopfziele, die den Namen der Zieldatei sowie gegebenfalls die gewünschte Größe der Zielgrafik angibt, gefolgt von Deklarationen und Kommandos wie in IMP:

$$\mathit{prog} ::= \texttt{tsvg}\ \mathit{identifier}\ [\texttt{(}\ \mathit{number}\ \texttt{,}\ \mathit{number}\ \texttt{)}]\ \texttt{;}\ \mathit{decls}\ \mathit{cmds}$$

Erweitern Sie die abstrakte Syntax (Module AST) und den Parser (Modul Parser), um die neuen Kommandos zu repräsentieren und TSVG-Programme zu parsieren.

10.2 *TSVG: The Backend* — *2 Punkte*

Wir wollen TSVG-Programme interpretieren und daraus Grafiken im SVG-Format erzeugen. Dazu benutzen wir das Modul TinySVG, welches eine handliche Verkapselung von SVG in Haskell implementiert. Um uns mit TinySVG vertraut zu machen, schreiben wir ein Haskell-Programm, welches die *linke* Grafik in Fig. 1 erzeugt.

Hinweise: Benutzen Sie dabei die Befehle `line`, `moveto`, `curve`, `fillcolor`, `strokecolor`, `circle`, `path`.

TinySVG benötigt das Modul `Text.XML.Light` aus dem Paket `xml`, das mit `cabal install xml` installiert werden kann.

10.3 *TSVG: The Interpreter* — *10 Punkte*

Jetzt wollen wir TSVG-Programme interpretieren (also auswerten), und dabei SVG-Grafiken mit TinySVG erzeugen. Die einzelnen Grafikkommandos sollen dabei wie folgt interpretiert werden:

[1] SVG: Scalable Vector Graphics, `https://en.wikipedia.org/wiki/Scalable_Vector_Graphics`

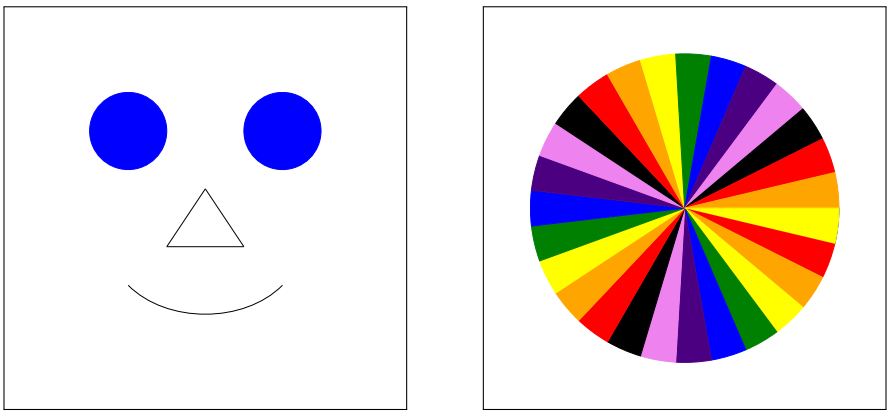

Abbildung 1: Ausgabe der Beispielprogramme.

- `move` bewegt die Startposition, also den Punkt im Bild, von der aus das nächste Kommando ausgeführt wird, auf den angebenen Punkt.
- `line` zieht eine Linie in der aktuellen Farbe von der aktuellen Startposition zum Zielpunkt. Sie verschiebt außerdem die Startposition auf den Zielpunkt des Kommandos.
- `curve` zeichnet eine Bezierkurve, deren Startpunkt die aktuelle Startposition ist, deren Endpunkt das dritte Argument, und deren Kontrollpunkte die ersten beiden Argumente ist. Die Startposition ist danach der Endpunkt der Bezierkurve.
- `circle` zeichnet einem Kreis mit der Startposition als Mittelpunkt und dem angegeben Radius.
- `slice` zeichnet ein gefülltes Kreissegment mit dem Radius als erstes Argument, dem Startwinkel als zweites und dem Endwinkel als drittes Argument.
- `color` ändert die momentane Zeichenfarbe. Es gibt schwarz (`black`), die Farben des Regenbogens (`red`, `orange`, `yellow`, `green`, `blue`, `indigo`, `violet`) und als spezielle Farbe `next`, welche in einer zu definierenden Reihenfolge die nächste der vordefinierten Farben auswählt, so dass wir spektakuläre Farbeffekte erzielen können.

Erweitern Sie den IMP-Interpreter in `Interpreter.hs` zu einem TSVG-Interpreter wie folgt:

1. Erweitern Sie die Auswertung von Ausdrücken, so dass Werte nicht nur Zahlen (Double), sondern auch Punkte (Point) sein können. Beachten Sie, dass die arithmetischen Operationen jetzt auch auf Punkte angewandt werden können.[2] Nicht alle Kombinationen von Operationen und Operanden sind sinnvoll: wir können keine Punkte dividieren, wir können Punkte nicht mir Zahlen (Skalaren) addieren oder vergleichen, und wir können Punkte nicht (sinnvoll) anordnen. Diese Operationen führen zu Laufzeitfehlern.
2. Erweitern Sie den globalen Zustand, so dass er die momentane Startposition, die momentane Zeichenfarbe, und die bisher gezeichneten Grafiken (Graphic aus TinySVG) verwaltet.
3. Die Zeichenkommandos manipulieren dann diese Teile des globalen Zustandes.
4. Die Interpretation eines TSVG-Kommandos startet mit einem globalen Zustand (leere Grafik, Farbe schwarz, Startposition in der Mitte des Bildes), interpretiert das Programm und schreibt danach die resultierende Grafik in die angegebene Zieldatei. Die Größe der Grafik (Breite und Höhe) wird in der Kopfzeile des Programmes angegeben (falls nicht, wird Breite und Höhe 1000 angenommen).

Sie können Ihren Interpreter mit dem Programmen `test1.tsvg` und `test2.tsvg` testen. Diese sollten eine Ausgabe vergleichbar mit Abb. 1 erzeugen.

Änderungen:

- *Version 1.0 Ausgegebene Version*
- *Version 1.1 Syntax korrigiert: NT prog muss mit Semikolon terminiert werden.*

[2] TSVG ist ist, in schlechter Tradition mit anderen Skriptsprachen, damit *dynamisch getypt* — die Typen werden zur Laufzeit durch Auswertung ermittelt.