Compiler
— Blatt 4 —

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Sommersemester 2014

10.05.14 10:22

Aufgabe 1 — RE to DFA (verschoben von Blatt 3 und um c) erweitert)

a) Erzeugen Sie nach Thompson's Konstruktion einen NFA aus dem regulären Ausdruck

   $a^+(b|c^*)b|c$

b) Erzeugen Sie aus dem NFA einen DFA mit dem *subset construction* Algorithmus.

c) Erzeugen Sie mit Hopcroft's Algorithmus einen Minimalen DFA.

Aufgabe 2 — Haskell - Reguläre Ausdrücke

Gehen Sie wieder vom gleichen Scanner wie in Aufgabe 2 auf Blatt 3 aus. Nehmen sie die Anfangsversion von https://www.fpcomplete.com/user/obcode/compiler/blatt-03-aufgabe-3 oder Ihren letzten Stand.

Sie sollen nun den Scanner um Gleitpunktzahlen und Bezeichner erweitern.

Gleitpunktzahlen müssen mit einer Ziffer beginnen (d.h. `.123` muss nicht erkannt werden), haben genau einen Punkt (d.h. `123` ist keine Gleitpunktzahl) und mindestens eine Ziffer nach dem Punkt.

Bezeichner beginnen mit einem Unterstrich oder einem Buchstaben (Sie können sich auf ASCII beschränken), gefolgt von beliebig vielen Unterstrichen, Buchstaben oder Ziffern.

Nutzen Sie zur Implementierung das Haskell-Modul `Text.Regex`.

Sie können eine RE als Zeichenkette angeben und mit der Funktion

```
mkRegex :: String -> Regex
```

daraus eine RE erzeugen.

Mit der Funktion

```
matchRegexAll :: Regex -> String -> Maybe (String, String, String, [String])
```

können Sie die RE auf eine Zeichenkette anwenden. Sie bekommen `Nothing` als Ergebnis wenn es keinen Match gibt. Im anderen Fall bekommen Sie

```
Just ( zeichenketteVorDemMatch
     , match
     , zeichenketteNachDemMatch
     , listeVonUnterausdrücken)
```

Erweitern Sie Ihren `Token`-Datentyp und Ihre `scan`-Funktion entsprechend.

Statt immer wieder verschiedene Eingaben in der Console auszuprobieren, sollten Sie sich sinnvollerweise Tests schreiben. Ein einfaches Test-Modul könnte so aussehen:

```
module Test where

import Main
import Test.HUnit

test1 :: Test
test1 =
    let expr = "(1.2+abc*__2+23"
        expectedValue = Just [ OpeningParan
                             , FloatNum 1.2
                             , Add
                             , Ident "abc"
                             , Mult
                             , Ident "__2"
                             , Add
                             , NatNum 23
                             ]
    in TestCase (assertEqual expr expectedValue $ scan expr )

tests :: Test
tests = TestList [ TestLabel "test1" test1
                 ]
```



```
main :: IO ()
main = do
        runTestTT tests
        return ()
```

Damit der Unit-Test die Gleichheit der beiden Werte berechnen kann, müssen Sie Ihren `Token`-Datentyp noch vergleichbar machen. Ersetzen Sie dazu die Zeile

```
deriving (Show)
```

durch

```
deriving (Show, Eq)
```

im `Main`-Modul.

Damit ist der `Token`-Datentyp in der Typklasse `Eq` und der Compiler leitet automatisch eine Implementierung von `(==)` und `(/=)` her.

Eine mögliche Lösung finden Sie unter https://www.fpcomplete.com/user/obcode/compiler/blatt-04-aufgabe-1.

