Weekly Assignment 2

Advanced Programming 2014 @ DIKU

Martin Jørgensen
University of Copenhagen
Department of Computer Science
tzk173@alumni.ku.dk

Casper B. Hansen
University of Copenhagen
Department of Computer Science
fvx507@alumni.ku.dk

September 21, 2014

Abstract

A parser should be implemented for a domain specific language, descriping curves and operations on them.

Tasks

Parser

Expr

This section is about the buttom most part of the grammar, the expressions. This part cannot contain any other types of nodes except for numbers. The grammar for expressions looks like so:

```
Expr  ::= Expr '+' Expr
        |  Expr '*' Expr
        |  'width' Curve
        |  'height' Curve
        |  Number
        |  '(' Expr ')'
```

Starting with the simplest we first created the only real terminal in the language, the "number", the code can ben see in Figure 1.

```
72 t = do n <- number
73        return $ Const n
```

Figure 1: Implementaiton of the terminal "number". (../CurvySyntax.hs)

The next part of the expressions we implemented was addition and multiplication, which like the number terminal is easy to implement as can be seen in Figure 2 and Figure 3.

```
76 op0 = (do string "+"
77           return Add)
```

Figure 2: Implementation of the + operator. (../CurvySyntax.hs)

```
80 op1 = (do string "*"
81           return Mult)
```

Figure 3: Implementation of the * operator. (../CurvySyntax.hs)

The above functions are bound together in the implementation of the parser `expr` which is shown in 4.

```
59 expr = width <|> height <|> e0 <|> e1
60     where width     = (do string "width"
61                           spaces
62                           c <- curve
63                           return $ Width c)
64           height    = (do string "height"
65                           spaces
66                           c <- curve
67                           return $ Height c)
68           e0 = chainl1 e1 op0
69           e1 = chainl1 t op1
```

Figure 4: The final expression parser. (../CurvySyntax.hs)

We first try to match expressions with `width`, the `height`, then `*` and lastly `+`.

We did not have time to add the option for using paranthesis around expressions yet. Which means the entire grammar for expressions is not complete, but should still work.

Curve

`parseString`

`parseFile`

`parseFile` was implemented with the suggestion from the assignment text and can be seen in Figure 5.

```
93 -- Parses a file with a Curvy program into a proper Curve.
94 parseFile :: FilePath -> IO (Either Error Program)
95 parseFile filename = fmap parseString $ readFile filename
```

Figure 5: The implementation of the `parseFile` method. (../CurvySyntax.hs)

Testing

Positive tests

Negative tests