Assignment 2: A SubScript Parser

Nicolas Ringsmose Larsen (vgn209)
Philip Rajani Lassen (vgh804)

September 2018

1 ReadP

For this assignment, we decided to use *ReadP* rather than *Parsec*. Most of the examples we could find for *Parsec* used the "forbidden" functions, but mainly *ReadP* seemed simpler, and we had enough trouble grasping the concepts already, so we went with that. Furthermore the readP was very similar to the literature on monadic parsing, which made following the documentation far simpler.

2 Initial implementation

Even after extensive reading, the concepts of parsers were unintuitive to us. This resulted in our initial implementation being brute forced and "hacky". Since we knew that we would not pass this assignment the first time around, we preferred to discard the somewhat working code, and try to reimplement our parser in a way that use monads properly instead. This was done in the hopes that we would get a chance to resubmit, and perhaps give us a chance to receive feedback so that we know if we are heading in the right direction. However, this also broke some of the working functions, leaving our parser with very few working features.

Our initial implementation handled the empty string poorly using if then else statements. This created some big problems in the reausability of our code. We could no longer use the features of the monad which make for elegant parsers. Not only did this poor implementation lead to bad code it also made developing the code take far longer.

3 Re-Implementation

In our re-implementation we made sure to take out the left recursive Grammars as we ran into infinite loops when we tackled them naively. Thus we started by factoring out the grammars on paper before we wrote any additional code.

3.1 Working Parts

```
Expr Expr1
::= Expr ‘,’ Expr | Expr1
::= Number
| ‘true’
| ‘false’
| ‘undefined’
| Expr1 ‘+’ Expr1
| Expr1 ‘-’ Expr1
| Expr1 ‘*’ Expr1
| Expr1 ‘%’ Expr1
| Expr1 ‘<’ Expr1
| Expr1 ‘===’ Expr1
```

We were able to get the Rules above working. In order to do this we followed page 27 from the parser notes quite closely. We defined the following helper functions to take care of white space and grab symbols.

```
token p = skipSpaces >> p
symbol = token . string
```

4 Testing

Since we ended up with so little working code, we have not put much time into testing. We used the REPL during the impelementation to check for simple cases, but other than that, we did not really have an opportunity for setting up a proper testing environment.