Final Project: Final Report

Thom Dickson, Alex Cozzie

Fall 2022

1 Description

This project is a Regular Expression Search and Replace runs from the command line.

It takes in 3 arguments; a Filename, a Regular Expression to match, and a Regular Expression to replace.

It reads all lines of the file, then, for each line, replaces all strings that match the "match" regular expression with the string generated by the "replace" regular expression. The match expression utilizes all regular expression syntax. This includes And, Or "|", Zero or more "*", 1 or more "+", Term priority "{}", Ranges "[]", and Capture Groups "()".

The output is a copy of the read in file, with all terms matching the match expression replaced to the string generated by the replace expression. This is printed to the terminal (`stdout`). The output can be piped to any given filename by the shell, if saving a copy is the goal.

2 Program Organization

2.1 Main function

The main function interprets three program arguments then reads the file by line. Once file has read in the information, the `replaceLine` function is run on each line of the file through maps. The result is printed to the terminal.

```
main :: IO ()
main = do
  args@[search, replace, filename] <- getArgs
  let search_term = parseMatch $ lexer search
  let replace_term = parseReplace $ lexReplace replace
  content <- readFile filename
  let text = lines content
  let replaced = map (replaceLine search_term replace_term) text
  putStr $ unlines replaced
```

2.2 Lexing

Reads a Line of a file in the form of a String into tokens.

```
lexer :: String -> [Token]
```

2.3 Parsing

2.3.1 Shift Reduce Parser

Shift reducer. Converts the list of Tokens into a simplified "Parsed Expression" Token

```
sr :: [Token] -> [Token] -> [Token]
```

2.3.2 Match Term Parsing

Parser. Converts a list of Parsed Expressions into a Match Regular Expression

```
parseMatch :: [Token] -> RegEx
```

2.3.3 Replace Term Parsing

The `parseReplace` converts a list of Parsed Expressions into a Replacement Regular Expression.

```
parseReplace :: [Token] -> ReplaceEx
```

2.4 Concatenate Tokens

This function will take a list of token `LiteralT` and concatenate them into a single string.

```
concatTok :: [Token] -> String lexReplace :: String -> [Token]
```

2.5 Evaluate Capture Group Replacement

2.5.1 In Replace Term

Given a list of Strings as an environment, and a replacement regular expression, this function returns the generated String for replacement.

```
replace :: [String] -> ReplaceEx -> String
```

2.5.2 In Search Term

This function enables Capture Group functionality in search term by recurring through the `RegEx` and replacing `CapIdRE` with the regex found in the `[String]` environment.

```
replaceRE :: [String] -> RegEx -> RegEx
```

2.6 Matching Strings

Match the beginning of the string to the `RegEx` and if they match, return the environment, matched string, and remaining string.

```
match :: RegEx -> ReplaceEx -> String -> Maybe ([String], String, String)
```

2.7 Searching a Line

This function iterates through the characters of a line executing `match` to replace any potential matches in the string.

```
replaceLine :: RegEx -> ReplaceEx -> String -> String
```

3 Research

No external research was performed. All ideas and code were self derived.

4 Discussion

We implemented capture groups and ranges. The capture groups function both in the match term, and in the replace term, which was another extension of our implementation. The ranges only work in matching, but that is to be expected, because variable strings in the replace expression would need to be represented by a capture group, not a range.

The section that caused us the most difficulty was actually the Concatenation. Everything else was fairly well designed to function on top of Concatenation; this also allowed us to continually add back in other features, however, the base “ab” regex actually gave us the most trouble.

5 Future Extensions

Currently, our `Or` operation runs into some issues with order of precedence. As best we can tell, this is an issue with Shift Reducers; the `Or` has lower priority than `Concat`, but `Concat` happens first due to it’s requirement to simplify expressions for other operations. This has led us to implement `{}` as term coherence. This, unfortunately, means that `{}` is not available for custom repetition counts.

After finding a solution for `Or` precedence, implementation of custom repetition should be easy.

Another issue we have found is that sometimes, `Star (Wildcard)` will, due to its greedy implementation, consume too many characters, not allowing the rest of the string to match. We have been unable to find a solution for this; in a non-functional language, the solution would be to step backwards and try a different depth first path.These last parts of the code that we would like to add would reach full regex search and replace functionality.

6 Summary

Our project runs from terminal, similar to a normal regex search and replace tool. It currently does not edit the file, but output can be piped to whichever file you would like to save as via terminal. It supports all normal functions of a regex search and replace tool, and most exotic forms, excluding repetition count.

We learned that Shift Reducers don’t function terrifically with many levels of order precedence, and if we repeated the project, likely we would find a different method of parsing.