Final Project

Formal Languages and Computability

CMPT440L

Project Due: 05/07/2018

Background

For this project, you will develop a version of the grep utility, in a language of your choice, called *grephy*, that searches files for regular expression pattern matches and produces dot graph file output for the automata used in the matching computation.

Name

Grephy

Example call (Java)

```
java grephy.Grep [-n NFA-FILE] [-d DFA-FILE] REGEX FILE
```

Description

Grephy should perform the following high-level steps.

- Learn the alphabet from the input FILE.
- Convert the regular expression REGEX to a NFA.
- Convert the NFA to a DFA.
- Use DFA computation to test each line of the file for accept/reject.
- File lines are delimited by newline characters.
- Accepted lines should be printed to standard output.
- Output the NFA and/or DFA to the specified filenames in DOT language format.
    - http://en.wikipedia.org/wiki/DOT_language
    - http://www.graphviz.org/content/dot-language
- Optionally, for extra credit, visualize your NFA and DFA as a feature of your program. (You may use a library.)

Notes

- Each line should be considered for acceptance in its entirety. This is equivalent to using grep with start (^) and end ($) anchors surrounding the pattern.
- You may compute on the NFA if you prefer, but you still need to generate the DFA for DOT output.
- Your program should work with or without the DOT output files specified.
- Generate a few different test files to test with and include with your submission. (simple match, multiple lines match, negative match)

- Use “good” GitHub etiquette including often commits, useful commit messages, and a well formed README.
    - Getting Git Right - https://www.atlassian.com/git/
    - How to Write a Good Git Commit Message - http://chris.beams.io/posts/git-commit/
    - GitHub For Beginners - http://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1
- Employ a static code analyzer, to ensure syntactic correctness, clean code, and help understand your coding style compliance with existing standards.
    - Sonar - http://www.sonarqube.org
    - Static Code Analysis - https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- It is highly encouraged to use a build mechanism such as Maven or Gradle (depends on your language of choice).
    - Maven - http://maven.apache.org/
    - Gradle - https://gradle.org/

Submission

1. Please submit your final code, including any test data files, via committing and pushing to your GitHub repository.
2. Please submit an archive file (zip) of all code, any build files, test files, and compiled executables via iLearn.

Grading

Grading will be based on content, accuracy, communication, thoroughness, and adherence to instructions for up to 20% of your final grade. The following attributes proved the grading criteria.

- 60% - Syntactic and functional correctness
- 10% - Demonstrates “good” design principles (ease-of-use, accessibility, etc.)
- 10% - “good” GitHub etiquette (Often commits, useful commit messages, well formed README, ect.)
- 10% - Exhibits “good” style (nested indentation, comments, etc.)
- 5% - Uniqueness and/or excellence (discretionary points for the best solutions)
- 5% - Test files (simple match, multiple lines match, negative match)