CS 361: Project 3 (Regular Expressions)

Due date: December 6 , 2020

Fall 2020

1 Project Overview

In this project, you will write a code that constructs an NFA for a given regular expression (RegEx). You will have to parse a RegEx in some way. We suggest you to use the recursive descent parsing algorithm. Refer to http://matt.might.net/articles/parsing-regex-with-recursive-descent/ for an excellent explanation and some code snippets.

2 Objectives

- Using Java Archive files, i.e., jar files, to use in your implementation and working with Javadocs of the classes inside the jar file.
- Continue to practice implementing interfaces: writing `RE` class that extends `REInterface`.
- Practicing recursive algorithms to parse a regular expression and build an NFA object.

3 Specification

- You are given one package `re`, one jar file `CS361FA.jar` with its Javadocs in `CS361FAdocs.zip` and test cases. Below is the directory structure of the provided files:

```
|-- re
| |-REDriver.java
| |-REInterface.java
|-- tests
| |-- p3tc1.txt
| |-- p3tc2.txt
| |-- p3tc3.txt
|-- CS361FA.jar
|-- CS361FAdocs.zip
```

`CS361FA.jar` contains binaries, i.e., class files, of NFAs and DFAs implementation. To compile `re.REDriver` from the top directory of these files you will have to include the jar file into the Java's `classpath` as follows :

```
[you@onyx]$ javac -cp ".:./CS361FA.jar" re/REDriver.java
```

Where "-cp" option tells Java's Virtual Machine where to look for classes. In our case we say look from the current directory, i.e., "." and inside "./CS361FA.jar", ":" is the separator of different locations in Unix environment. Similarly, to run `re.REDriver` use the same additions to the classpath:

```
[you@onyx]$ java -cp ".:./CS361FA.jar" re.REDriver ./tests/p3tc1.txt
```

In order to use this jar file in Eclipse you need to add it as an external jar to your build path. We also provide you with Javadocs for classes and methods inside the jar file. We encourage you to explore it and familiarize yourself with the interfaces of the provided classes (basically what we had in the first two projects).

3.1 Input file

The input file has the following format:

- The first line of a file is a regular expression, which consists of the following seven characters only: **a**, **b**, **e**, |, (,) and *

  The character | represents the union operator, the character * is the star operator, and **e** denotes the empty string. The alphabet contains only symbols **a** and **b**. (,) are used to set the precedence of the operations. Regular expressions are always valid.
- After the first line, the following lines contain some input strings, one per line. Your program should determine whether each of them is a member of the language described by the regular expression. An input string contains only **a**, **b** and **e** characters. All input strings are valid strings over the alphabet.

Here is an example of the input file:

```
(a*|b)b
e
aaa
bb
b
aaaab
bbbbba
```

We give you three test files and the expected outputs, but we encourage you to create several of your own.

3.2 `re.REDriver` (provided class, the driver class)

Note: You don't need to modify the class. You will use it to test your RE implementation.

In `re` package you are given a class named `REDriver` that reads the input file, instantiates a corresponding RE object, obtains an equivalent NFA, then the equivalent DFA and simulates that DFA instance on each input string. `REDriver` is adequately documented. Please read the comments. This class takes the test case as the input and produces the following output: "yes" if the string in the language and "no" if the string is not in the language. Refer to Sample Input/Output section for examples.

3.3 `re.RE` (class you need to implement)

The class must implement `re.REInterface` interface and have a constructor that takes a RegEx string as its only parameter. `re.REInterface` has only one `getNFA()` method. It returns the equivalent NFA for the RegEx used to instantiate `RE` object. `getNFA()` is where you will spend the majority of your time. This method should parse a RegEx and build an NFA from the leaves up of the corresponding parse tree.

Note: Read on the recursive descent parser http://matt.might.net/articles/parsing-regex-with-recursive-descent/.

Your code should not build an explicit parse tree as it is shown in the recursive descent parser explanations, but instead build an NFA.

For example, instead of

```
private RegEx factor(){
RegEx base = base();
...
return base;
}
```

your code should be

```
private NFA factor(){
NFA baseNFA = base();
...
return baseNFA;
}
```

and so on. Also, you should not have inner classes - they are only useful when we create an explicit parse tree. Please do not copy and paste the code from the link, but rather use it as a guide to write your own recursive decent parser.
You will be using the NFA library provided in CS361FA.jar file. Please read the documentation to understand how NFA class works and how NFAState works. These APIs are a bit different from the NFA requirements of **p2**.

4 Sample Input/Output

Below is the sample input/output for the first test cases provided to you.

```
[you@onyx p3]$ cat ./tests/p3tc1.txt
(a|b)*a
aaaa
aba
bba
a
b
bbb

[you@onyx p3]$ java -cp ".: ./CS361FA.jar" re.REDriver ./tests/p3tc1.txt
yes
yes
yes
yes
no
no

[you@onyx p3]$ cat ./tests/p3tc2.txt
((a|b)(a|b))*
abbabb
e
```

```
aa
ab
aaaaa
bba
[you@onyx p3]$ java -cp ".: ./CS361FA.jar" re.REDriver ./tests/p3tc2.txt
yes
yes
yes
yes
no
no

[you@onyx p3]$ cat ./tests/p3tc3.txt
aaa*b*a*a
aabaa
aaa
aabba
abbaa
abbbbbbbbbbbbbbbbbba
bbaa
[you@onyx p3]$ java -cp ".: ./CS361FA.jar" re.REDriver ./tests/p3tc3txt
yes
yes
yes
no
no
no
```

5 Grading Rubrics

1. 5 points – the properly commented (Javadocs and inline comments) code and Javadocs generated without errors.
2. 5 points – the properly formatted and detailed README.
3. 3 points – program submitted correctly, compiles and runs on `onyx`.
4. 10 points – the code quality, i.e., easy to read, proper data structures used and proper variable naming.
5. 5 points – no intermediate parse tree is generated and no inner classes created.
6. 72 points for program running correctly. Programs running longer than 30 second will be killed) There will be 12 test cases (3 of them are available to you), each with 6 input strings. If your program gives the correct output for an input string, you get 1 point. Thus, you can get a maximum of 72 points for all $12 \times 6 = 72$ inputs strings in 72 test cases.

6 Submitting Project 3

If you haven't done it already, add a **Javadoc comment** to your program. It should be located immediately before the class header and before each method that was not inherited

- Have a class javadoc comment before the class.
- Your class comment must include the *@author* tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Use *@param* and *@return* tags. Use inline comments to describe how you've implemented methods and to describe all your instance variables.
- Check the correctness of generating documentation for your program with the following command

  ```
  javadoc -author -d doc re/RE.java
  ```

  This will put the document output in a folder named "doc". Check to see if it did using the "ls" command.
- Open the generated documentation page using Firefox or Google Chrome

  ```
  google-chrome doc/index.html
  ```

Include a plain-text file called **README** that describes your program and how to use it. Expected formatting and content are described in README_TEMPLATE. An example is available in README_EXAMPLE.
You will follow the same process for submitting each project.

1. Open a console and navigate to the project directory containing your source files,
2. Remove all the `.class` files using the command:

   ```
   rm *.class
   ```
3. In the same directory, execute the submit command : Section 1:

   ```
   submit cs361 cs361 p3_1
   ```

   Section 2:

   ```
   submit cs361 cs361 p3_2
   ```
4. Look for the success message and timestamp. If you don't see a success message and timestamp, make sure the submit command you used is EXACTLY as shown

Required Source Files
Make sure the names match what is here **exactly**

- `RE.java` in `re` package.
- `README`

After submitting, you may check your submission using the "check" command as in the example below for Section 1

```
submit -check cs361 cs361 p3_1
```