CS 3240 Project Report
Phase I

Team:
Surenkumar Nihalani
Robert Harrison
Lin Dong

Description:
Regular expression parser:
We started by writing a useless space advancing function. Then we wrote a function for each production rule. All the character set rules returned a character or set of characters. We passed cloned copies of input and we advanced pointer every time the each production rule reported success.
After moving from character production rules, we wrote junit tests to maintain the strength of our codebase.
We wrote regular expression production rule parsing functions. Each function returned a NFA and every level connected NFAs using concatenation, star or union of the returned NFAs.
Now we had all these individual NFAs and we took union of them with the token type metadata.
After taking their union, we wrote the algorithm to convert NFA to a DFA. We take the epsilon closure of the start state. We convert every combination of set of states reachable to an individual DFA state. In the end, we convert this DFA to a table using a breath first traversal.

Assumptions made:
- When walking the table, we end up at a stage with no transition for a empty space, we discard the state and restart walking the table. This was necessary to parse the sample specification
- The dollar sign needs to escaped in RE_CHAR classes.
- The DFA matching is greedy. Only last known accepting state is printed out on failure.
- All defined classes names start with a dollar
- All token classes names start with a dollar

Problems faced:
- Maintaining the state of how much of the input is consumed and restoring it on failure was a challenge. We cloned objected and set it when the functions returned true.
- Debugging the production rule was a challenge. We wrote junit tests to make sure that rules kept working on each commit
- Most of the algorithms were error prone. We code reviewed most of our commits.
- Test cases:
    - We have junit tests in place for verifications.

- We used the given specification and tried different classes and verified DFAs manually using eclipse's debugging interface..

Organization of the code:

| - .git/ - version control folder
| - build.xml – Ant build file.
| - spec – specification test file
| - input – input file to parse
| - table – the DFA table outputted.
| - output – expected output
| - output.debug – output files containing the token
| - RegexParserTest.java – contains all the junit tests.
| - Phase1UML.png – UML diagram of our code
| - src/
| ----- Driver.java - Reads the arguments, sets up objects, files for reading and parses.
| ----- TableWalker.java – Walks the DFA and writes out the tokens
| -----DFA package/
| ---------- DFA.java – models a DFA.
| ---------- DFAState.java – models a DFA state
| -----NFA package/
| ----------NFA.java – models a NFA
| ----------NFAState.java – models a NFA state.
| ------Parser package/
| ----------RegexParser.java – Contains all the production parsing rules
| ----------RegexParserInput.java – Wrapper around the input string
| ----------RegexParserOutput.java – Communicates the metadata parsed

Instructions to run the code:

Unzip the directory
Install ant
Modify build.xml to change the input output and specs file name.
You need to modify the following arguments:

--specs-file lexical specifications file
--input-file input file to tokenize
--output-file name of the output file for tokens

Run "ant" on terminal