**Eric Huang**

**Ho Yin Pun**

**Jung Gi Lee**

**Hyangjin Lee**

**Report**

**Implementation:**

The implementation of this project was divided into a tokenizer, a character class generator, an NFA generator, a DFA generator and a table walker. The tokenizer is used to scan the regular expression and create distinct tokens based on the input grammar specifications for the language. The character generator will create character classes based on these tokens and store the named classes into a table. The regular expression is converted into an NFA using the NFA generator and the character classes. The result will go into the DFA generator to create the corresponding DFA. This DFA can then be minimized for the final result used by the table walker.

The tokenizer is responsible for scanning through the regular expression and only retrieves valid tokens specified in the grammar. It accepts a regular expression string as an input to operate on. It keeps track of the current token position in the string and only increment the position after a token was consumed. It also provides a peek method to look ahead by one or more tokens without actually consuming the tokens which provides a way to go backward on regular expression. The peek method serves as a future detection system for the range and exclude functions implementation. A token object contains an opcode and a value field. The opcode field determines that type of the token it is and the value field holds the character that is contained in the token.

The character class generator is responsible for creating a set of classes that contains valid characters for the regular expression to operate on. It uses the functions provided by the tokenizer to retrieve valid token objects and use them to determine the values in the character classes that the regular expression is allowed to use. A series of accepted characters, a range of characters and also excluding characters from another class are ways to populate the class. The union and intersection functions can also be performed between two classes when needed to combine elements or exclude elements. Character classes with a valid name are stored in a hash map for quick lookup.

The NFA generator is responsible for creating an NFA represented by a graph data structure from the regular expression and the character classes. Each state is an NFA node that contains fields that indicates the type of node it is (nonaccepting or accepting) and all transitions from the current state outward stored in a list structure. The transition triggers include valid characters and epsilon. Operations supported between different NFA’s are union and concatenation which adds edges to the appropriate nodes to connect them in the correct order. An NFA may also perform a kleene star operation on itself for to represent zero or more repetitions. The final output from the generator is a single NFA representing the given regular expression.

The DFA generator class makes a DFA from an NFA. A graph structure is used to represent the DFA. Each DFA node is mapped to a set of NFA nodes which is connected together by epsilon closure. Each DFA node contains information on the transitions to another DFA node, and whether it is final accepted node or not. The resulting DFA may not be minimal at this point. In order to reduce the number of DFA states, the minimization function can be performed to produce a minimal DFA. This function will check each state with one another to determine if they can be distinguished or not. Indistinguishable states that have the same transitions for the same set of character triggers can be combined into a single state.

The table walker is responsible for scanning the program and create tokens from the DFA of the program. When the table walker is called, it takes in a string input and a DFA list that contains multiple DFA’s from the DFA generator. Then on each distinct DFA, it walks on its nodes until it hits the accepting node and checks if each node has a valid transition by calling isTriggered() method. Only when the accepting node of each DFA is hit then it saves the collection of the input characters, or a string as a token. By this point, there must be one or more tokens, and the table walker compare each token to one another and finds the longest token that was obtained from walking each DFA’s. When the longest token is found, the table walker saves it in the token list. After the entire input file is processed, the table walker prints out the tokens from the token list.

**Assumptions:**

* All the inputs are valid.
* The NFA must have a correct match or else the NFA simulator may loop forever.
* The NFA generator assumes that the input is formatted as $[token-name] [regex]
* No ambiguous match for DFA’s for the table walker.
* Longest match for the length of tokens for the table walker.
* The exclusion (^) symbol follows immediately after the left open bracket ([) for character class generator in the case of character exclusion. There cannot be any character immediately before (^) other than ([).
* All character class value, range or exclusion must exist inside a pair of open and close brackets. Ex. [abc], [a-c], [^abc]
* No regular expressions that matches epsilon.
* No extraneous information in the input specification.

**Problems:**

* Decide on an organized and structured design to implement the program.

**Test Cases:**

* Some components have their own tests in the Test package.

**\*\*The bonus part of the project was implemented.**