Development of a Non-Deterministic Finite Automaton with Epsilon Moves ($\epsilon - NFA$) Generator Using Thompson’s Construction Algorithm

Gilbert M. Arces  
Eugene Val C. Mangaoang  
Michael Anthony Jay B. Regis  
Jomari Joseph A. Barrera  
Visayas State University  
Visca, Baybay City, Leyte, Philippines

Abstract

The study designed and developed a finite state machine generator which converts a regular expression to its equivalent non-deterministic finite automaton with $\epsilon$-moves ($\epsilon - NFA$). It implements Thompson’s construction algorithm, an algorithm that works recursively by splitting an expression into its constituent subexpressions, from which the $\epsilon - NFA$ will be constructed by a set of rules. The system accepts a valid regular expression and rejects an invalid one. The regular expression is then parsed and evaluated by a parse tree to be converted to its equivalent states. The states is then connected and generated as a whole to show the finite state machine. The finite state machine generator was implemented using the Java programming language with the JavaFX package to produce better quality and easy to use graphical interface. An online acceptability survey obtained a 3.96 (exceeded beyond expectation) acceptability rating.

Keywords: Regular Expression; Finite State Machine; Parse Tree

Introduction

A string composed of any combination of normal characters and special meta-characters or meta-sequences are known as regular expression. These represent constructs for location, character types, quantity, or instruction syntax. Most programming languages including scripting languages, database query language, and command-line tools used regular expression for syntactic specification (Stubblebine, 2007). Although regular expressions provide representation for regular language, finite automata is ideal for internal computing structure (Bruggemann-Klein, 1993). Finite-State Machine (FSM) is a computational model that describes possible histories that would affect a finite number of future behavior. The machine would accept some external inputs, then transition to exactly one possible state from a finite number of states. So, an FSM is characterized by an initial state, the list of possible states and condition for transition (Kohavi & Jha, 2010). A FSM can be a Deterministic Finite Automaton (DFA) and a Non-Deterministic Finite Automaton (NFA). In a DFA, for each input only one state would transition from the current to the next state. In contrast, an NFA has the capacity to be on several state at once, implying that it could “guess” its input (Hopcroft, Motwani, & Ullman, 2001). Thompson’s construction algorithm transform a regular expression
into its equivalent NFA (Thompson, 1968). The algorithm would recursively split an expression into a subexpression, then a set of rules from a regular expression would construct the NFA. This transformation is crucial since NFA format is better suited for machine execution. To put it into perspective numerous pattern matching task such as network packet inspection (Becchi & Crowley, 2007), Field Programmable Gate Array based (FPGA) regular expression circuit (Nakahara, Sasao, & Matsuura, 2010), General Purpose Graphics Processing Unit (GPGPU) optimized pattern matching (Cascarano, Rolando, Risso, & Sisto, 2010), Perl compatible regular expression matching relies on regular expression to NFA transformation (Becchi & Crowley, 2008). Several tools exist that would covert regular expression to FSM. A C++ class library named FIRE engine (Watson, 1994) implements regular expression and finite FSM algorithms. Furthermore, JFLAP (Rodger & Finley, 2006) is an experimenting software that includes topic finite automata, multi-tape turning machines, parsers, non-deterministic pushdown automata, language systems and types of grammars. Finally, RegExpert tool (Budiselic, Srbljic, & Popovic, 2007) converts regular expression to NFA epsilon using a modified construction algorithm with configuration panel for regular expression generator. RegExpert tool was created as an upgrade to Softlab, a software which simulates automata but extends its capability to be a fully distributed e-learning tool. This study goes back to basic by generating $\epsilon - NFA$ using the unmodified Thompson’s construction algorithm. In a way aiding students in Automata theory in visualizing how certain rules in regular expression is implemented and expanding the list of available tools on this domain.

Materials and Methods

Application Analysis

To understand on how the regular expression is converted, a tree is created, as shown in Figure 1 to determine how the regular expression is parsed using the Thompson’s construction algorithm. As example, the tree represents the $aa^*+(b+a)$ regular expression. From the root node, which is the UNION node, it will traverse the tree until it reaches the leaf node, passing the SEQUENCE node, KLEENE STAR node (*), and the SYMBOL node as the leaf node. As the tree reaches the leaf node, it will evaluate if the string is a symbol or another union. Since the first string is just a symbol, which is the ‘a’ symbol, it assigns the symbol to the SYMBOL node as shown in Figure 2.

After the evaluation, the SYMBOL node has already a symbol then returns to its upper node which is a KLEENE STAR node. In this node, it evaluates if the next string in the regular expression is a Kleene star. If the next string is not a Kleene star, the node returns the SYMBOL node to the upper node which is the SEQUENCE node as shown in Figure 3. Since the SEQUENCE node still has symbol or character, it will then branch out and traverse until reaches the leaf node. And again, as the tree reaches the leaf node, it evaluates if the string is a symbol or another union, since this is just a simple symbol, which is another ‘a’ symbol, the tree assigns the symbol to the SYMBOL node and return to its upper node, which is the KLEENE STAR node. The KLEENE STAR node then evaluates if the next string in the regular expression is a Kleene star, and since it has a Kleene star, the node will wrap the symbol with a Kleene star symbol and return to the upper node. The upper node which is the SEQUENCE node concatenate or add the additional node as shown in Figure 4.

After the KLEENE STAR node evaluates and return to its upper node, the SEQUENCE node then evaluates if there is still more
character or symbol that needs to be checked, and since there are no more characters, it returns to the upper node, the UNION node. The UNION node then evaluates if the next string is a ‘+’ symbol, and if it is, it branches out and traverse the tree until it reaches a leaf node. Note that, if the UNION node evaluates and the next string is not a ‘+’ symbol, it stops branching out and return the final tree. Since regular expression sample has a ‘+’ symbol that is evaluated in the UNION node, the node branches out and traverse until reaches the leaf node. The leaf node which is the SYMBOL node, evaluates if the character is just a symbol or another union, and since this is another union, the SYMBOL node recursively traverses the inner tree. As the inner tree traverse and reaches the leaf node, another SYMBOL node. The node then evaluates if character is just a symbol node or another union. Since it is just a symbol node, the ‘b’ symbol, the tree assigns the
symbol to the SYMBOL node, and returns to the upper node. The upper node which is the KLEENE STAR node evaluates the next string in the regular expression, and since the string is not a Kleene star, it returns to its upper node. The SEQUENCE node which evaluates if there is still another character in the sequence, and since there are no more character in the sequence, it then returns the node to UNION node as shown in Figure 5.

The UNION node evaluates if it can branch out by checking the next string in the regular expression which is the ‘+’ symbol. Since the next string is a ‘+’ symbol, it branches out and traverse the tree until it reaches the leaf node. The leaf node which is the SYMBOL node evaluates again the symbol, and since this is just a symbol, the ‘a’ symbol, the tree assigns the symbol to the SYMBOL node, and returns to its upper node. The upper node which is
the KLEENE STAR node evaluates again if the next string is a Kleene star (*) symbol, and since it is not a Kleene star symbol, it just returns to its upper node. The SEQUENCE node then evaluates if there are more symbol or characters in the sequence. Since there are no more symbols, it then returns the node to the upper node, and add the symbol to the UNION node as shown in Figure 6.

Since the inner tree is already complete, the node returns to its upper node which is the KLEENE STAR node. The KLEENE STAR node then evaluates for the next string, and since the next string in the regular expression is not a Kleene star (*) symbol, it automatically returns the node to its upper node which is the SEQUENCE node. The SEQUENCE node again evaluates if there are more symbol or character in the sequence, since there is no more character, it returns the node to the upper node which is the UNION node. The UNION node then evaluates if it can still branch out, and since there are no more string in the regular expression that should be evaluated, the tree stops branching out and return its final state as shown in Figure 7.

System Architecture Design

The finite state machine generator in this study can be used by anyone who has a certain knowledge with automata theory. In this architecture, the user inputs a regular expression, which is then validated. Once the string has been validated, it will then convert the regular expression to its respective $\epsilon - NFA$ based on Thompson’s construction algorithm. The system will then display the $\epsilon - NFA$ and will ask the user if another regular expression shall be converted. If not, then the system closes down. Figure 8 shows the logical flow diagram of the finite state machine simulator.

Thompson’s Construction Algorithm

The following rules are depicted according to Thompson’s construction algorithm. Alphabets $a$ and $b$ will be used as an example for showing these rules:

- The empty expression which is depicted using $\$$ symbol as shown in Figure 9.
- The symbol $a$ as an input alphabet as shown in Figure 10.
- The union expression $a + b$ as shown in Figure 11.
- The concatenation expression $ab$ as shown in Figure 12.
- The Kleene star expression $a^*$ as shown in Figure 13.
- The parenthesized expression $(a)$ is converted to a itself as show in Figure 14

Based on the figures shown, the empty expression is used as the initial transition from the initial state to the next state. Furthermore, the empty expression is present before the transition of the input alphabet to the next state. There is also only one final state in all machines unlike in a nondeterministic finite automaton which can have multiple final states.

User Interface Design

The user interface of the system was graphically designed using JavaFX GUI implementation. The system opens a small startup window. At the startup window, it will ask the user to enter a regular expression to be validated. As the user enters a regular expression, the system will also validate it as it goes. The system will open another window once the conversion has been initiated to show the finite state machine. The second window will adjust its size depending on the size of the finite state machine.
The system was tested and evaluated by ten people selected randomly who understands the concept of automata theory. A laptop or desktop was used for testing the game. Each
Figure 13. The Kleene Star Expression

Figure 14. The Parenthesized Expression

Figure 15. Regular Expression $aabb$ accepted

Figure 16. Operators of the System
user used the system to evaluate the validity of the conversion of the inputted regular expression to its respective nondeterministic finite automaton. The participant evaluated the system using a survey questionnaire. The total number of responses was obtained and tabulated.

Results and Discussion

User Interface of the System

The system starts off by accepting a valid regular expression. The system’s text field will turn green once it recognizes a valid regular. For example in Fig. 15, the string aabbb is accepted as a valid input.

The system can also accept special
characters except for the default special characters used as operators. The default operators used are “+” for union shown in Fig. 16a, (“and”)” for grouping characters shown in Fig. 16b and “*” for Kleene star shown in Fig. 16c.

The system will also reject a regular expression. The system's text field will turn red if the following scenarios are met: There is a missing character from either left or right part when using the union operator as shown in Fig. 17a and Fig. 17b. There is a missing parenthesis when grouping characters as shown in Fig. 17c and Fig. 17d respectively. Multiple Kleene star is used for a single character as shown in Fig. 17e. Default operators like union and Kleene star are used as initial input as show in Fig. 17f and Fig. 17g respectively.

Once a valid expression has been accepted, it will then generate to the respective finite state machine based on Thompson's constructions algorithm when the Generate e-NFA button (Fig. 18) is clicked. Then it
will open up another window which shows the finite state machine equivalent to the regular expression (Fig. 19).

**System Evaluation**

The development of the finite state machine generator was evaluated using the rating scale in Table 1 by 10 people selected randomly online that has knowledge with automata theory. The evaluation was divided into two categories: system analysis and graphics. Each category was evaluated through a survey after the respondents have tested the system.

The weighted average is calculated in the survey to check each category of the game as shown in Table 2. Each category has a corresponding number of indicator, and each indicator was tabulated and calculated using its weighted average.

The result of the evaluation for the system and graphics evaluation is 4.1 and 3.8 respectively, so a grand mean of 3.96 is obtained. This means that the overall system performance exceeds the expectation.

The overall weighted average for the two categories is also calculated which results to 3.96. This means that the result of the evaluation for the whole application is acceptable since the performance meets expectation.

**Summary, Conclusion and Recommendation**

Regular expression and finite state machine are among the useful topics in computer science to understand pattern matching and validating strings. This study designed and developed a system which allows users to input a regular expression and convert it to its respective non-deterministic finite automaton (NFA) with $\epsilon$-moves ($\epsilon - \textit{NFA}$) using Thompson’s construction algorithm. After evaluating the system an overall average of 3.96 was obtained, meaning the system operates beyond expectation. The graphical representation aids in better understanding how a regular expression is converted NFA and is very useful for students in computer science to better understand this topic in Automata theory. However, it is recommended to have an equivalent transition table generated along with the finite state machine. Moreover, the transition lines
should not overlap with the states as the transition is set to static. On the other hand, a button to convert the $\epsilon - NFA$ to NFA and DFA and vice versa is also recommended in order to differentiate on how these three different machines behave. Lastly, the system should generate a finite state machine using a three-way union regular expression.

References Cited


