A Technique for Conversion of Regular Expression to Finite Automata and Vice-versa

Ashwani, Shabnam Sangwan, Sunita Kumari

Abstract— **The theory of computation is the branch of computer science and mathematics that deals with whether and how efficiently problems can be solved on a model of computation using an algorithm. In theoretical computer science, automata theory is the study of abstract machines and the computational problems that can be solved using these abstract machines. These abstract machines are called automata. Finite automata can be deterministic and non-deterministic. Every regular language that is described by non-deterministic finite automata can also be described by deterministic finite automata. Regular expressions [6] also denote regular languages, which consists of strings of particular type. The patterns of strings described by regular expression are exactly same as what can be described by finite automata. It means every formal language defined by any finite automata is also defined by a regular expression.**

***Keywords*— Finite Automaton, Regular Grammar, Regular Language, Kleen Closure**

I. Introduction

Regular expressions are used to represent certain set of string in algebraic manner. Regular expressions are widely used in the field of compiler design, text editor, search for an email-address, grep filter of unix, train track switches, pattern matching ,context switching and in many areas of computer science. The demand of converting regular expression into finite automata and vice versa motivates research into some alternative so that time taken for above is minimized.

For conversion of deterministic finite automata to regular expression, several techniques like Transitive closure method, Brzozowski Algebraic method and state elimination method have been proposed. None of the above specified technique is able to find smallest regular expression. Our purpose is to find the smallest regular expression equivalent to given deterministic finite automata. State elimination approach is the most widely used and efficient approach for converting deterministic finite automata to regular expression.

The presented paper investigates and compares different techniques used for converting deterministic finite automata to regular expression. Brief comparisons amongst different techniques are presented in this review paper.

Manuscript received June 09, 2015

Ashwani, CSE Deptt, SKITM, Bahadurgarh, India
Shabnam Sangwan, CSE Deptt, SKITM, Bahadurgarh, India
Sunita Kumari, CSE Deptt, PDMCE, Bahadurgarh, India

II. LITERATURE REVIEW

In 1956, S. C. Kleen [21] By using the Transitive Closure method he converted DFAs to regular expressions. He also proved the equivalence of finite automata and regular expressions. In 1964, Brzozowski [19] introduced new concept of derivatives of regular exression by expanding Kleene's method. Then in 1987, G. Berry and R. Sethi[14] brought Brzozowski's paper to the forefront. In 2001, Ding-Shu Du [2] devised the state removal method. The later Linz [8] presents a more straightforward method. In context with the conversion problems, many different approaches and respective algorithms had been introduced. In 1993, Klein [22] enumerated an algorithm, then, in 2001 Wilke [24] translated regular expressions into small ε-free nondeterministic finite automata. Geffert [23] in 2003 converted binary regular expressions into nondeterministic ε-free automa with o(nlogn) transitions. Yamamoto, H. [25] in 2005 introduced a new concept of finite automata corresponding to semi extended regular expressions.

III. Basic Definitions

[A] Deterministic finite automaton (DFA)

Deterministic finite automaton (DFA) is a finite state machine accepting finite strings of symbols. For each state, there is a transition arrow leading out to a next state for each symbol.

Deterministic finite automata (DFA) can be defined by 5-tuples (Q, Σ, δ, q_0, F), where
Q is a finite set of states
Σ is a finite set of symbols
δ is the transition function, that is, δ: Q × Σ → Q.
q_0 is the start state
F is a set of states of Q (i.e. F⊆Q) called accept states.

Transition functions can also be represented by transition table as shown in table 1.

Table 1: Transition Table representing transition function of DFA

State (Q)	Next State δ(q,a)
0	1
1	2
2	2

A finite automata is represented by ({0, 1, 2}, {a}, δ, {0}, {2}) where, δ is shown in the table above.

Transition function can also be represented by transition diagram as shown below in figure 1.

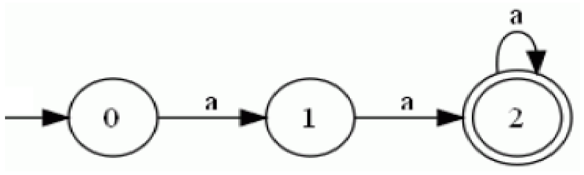

Figure 1: Deterministic finite automata corresponding to table 1.

[B] Non-deterministic Finite Automata

A Non-deterministic finite automata (NFA) is same as DFA except the transition function. Transition function in NFA is defined as: Q × Σ → 2Q. A Non-deterministic finite automaton (NFA) [9] is a finite state machine where for each pair of state and input symbol there may be more than one next state.

Following figure 2 shows non-deterministic finite automata accepting all the strings terminating with 01 and in which state A has two transitions for same input symbol 0.

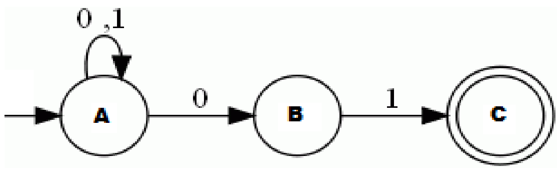

Figure 2: An example of non-deterministic finite automata.

[C] Regular Expression

A regular expression (RE) is a pattern that describes some set of strings. Regular expression over a language can be defined as:

1) Regular expression for each alphabet will be represented by itself. The empty string (ϵ) and null language (ϕ) are regular expression denoting the language {ϵ} and {ϕ} respectively.

2) If E and F are regular expressions denoting the languages L(E) and L(F) respectively, then following rules can be applied recursively.

a. Union of E and F will be denoted by regular expression E+F and representing language L(E) U L(F).

b. Concatenation of E and F denoted by EF and representing language L(E*F) = L(E) * L(F).

c. Kleene closure will be denoted by E* and represent language (L(E))*.

Any regular expression can be formed using 1-2 rules only.

IV. Conversions Between Regular Expression & Automata

This section describes different techniques used for converting deterministic finite automata to regular expression and vice versa.

[A] Conversion of DFA to RE

Kleene proves that every RE has equivalent DFA and vice versa. On the basis of this theoretical result, it is clear that DFA can be converted into RE and vice versa using some algorithms or techniques. For converting RE to DFA, first we convert RE to NFA(Thomson Construction) and then NFA is converted into DFA(Subset construction).For conversion of DFA to regular expression, following methods have been introduced.

- Transitive closure method
- Brzozowski Algebraic method
- State elimination method

[A1] Transitive Closure Method

Kleene's transitive closure method [2, 12] defines regular expressions and proves that there is equivalent RE corresponding to a DFA. Transitive closure is the first mathematical technique, for converting DFAs to regular expressions. It is based on the dynamic programming technique. In this method we use R^k_{ij} which denotes set of all the strings in Σ* that take the DFA from the state q_i to q_j without entering or leaving any state higher than q_k. There are finite sets of R^k_{ij} so that each of them is generated by a simple regular expression that lists out all the strings.

Consider the DFA given in figure 3 and applying transitive closure method on it.

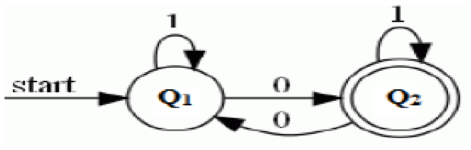

Figure 3: DFA for the language having odd number of 0's

$$r_{11}^{0} = 1+\epsilon \quad r_{22}^{0} = 1+\epsilon \quad r_{12}^{0} = 0 \quad r_{21}^{0} = 0$$

$$r_{12}^{1} = r_{12}^{0} + r_{11}^{0}(r_{11}^{0})^{*} r_{12}^{0} = 0 + (1+\epsilon)^{+} 0$$

$$r_{22}^{1} = r_{22}^{0} + r_{21}^{0}(r_{11}^{0})^{*} r_{12}^{0} = (1+\epsilon) + 0(1+\epsilon)^{*}0$$

$$r_{12}^{2} = r_{12}^{1} + r_{12}^{1}(r_{22}^{1})^{*} r_{22}^{1} = (1+\epsilon)^{*}0(1+\epsilon+01^{*}0)$$

$$r_{12}^{2} = (1)^{*}0(1+01^{*}0)^{*}$$

$$L(M) = L(r_{12}^{2})$$

[A2] Brzozwski Algebraic Method

Brzozowski method [20] is a unique approach for converting deterministic finite automata to regular expressions. In this approach first characteristic equations for each state are created which represent regular expression for that state. Regular expression equivalent to deterministic finite automata is obtained after solving the equation of R_s (regular expression associated with starting state q_s).

Consider the DFA in the following figure 4:

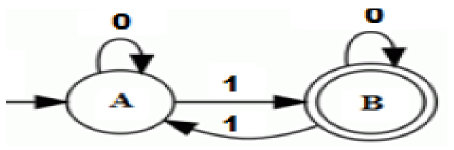

Figure 4: DFA for strings with an odd no of 1's.

Characteristics equations are as follow:

$A = 0A + 1B$
$B = 1A + 0B + \epsilon$

Solving these equations by Arden's theorem

$B = 1A + 0B + \epsilon = 0B + (1A + \epsilon) = 0^{*}(1A + \epsilon)$
$B = 0^{*}(1A) + 0^{*}(\epsilon) = 01^{*}A + 0^{*}$

$A = 0A + 1B = 0A + 1(0^{*}1A + 0^{*}) =$
$0A + 10^{*}1A + 10^{*}$

$A = (0 + 10^{*}1)^{*}(10^{*})$ (Using Arden's rule)

[B] Conversion of RE to FA

It turns out that every Regular Expression has an equivalent NFA and vice versa. There are multiple ways to translate RE into equivalent NFA's but there are two main and most popular approaches. The first approach and the one that will be used during this project is the Thompson algorithm and the other one is McNaughton and Yamada's algorithm.

[B1] Thompson's algorithm

In 1968, Thompson introduced the multiple-state simulation approach. In this approach, the states of the NFA were represented by small machine-code sequences, and the list of possible states was just a sequence of function call instructions. Thompson's algorithm parse the input string (RE) using the bottom-up method, and construct the equivalent NFA. The final NFA is built from partial NFA's, it means that the RE is divided in several subexpressions, in our case every regular expression is shown by a common tree, and every subexpression is a subtree in the main common tree. Based on the operator the subtree is constructed differently which results on a different partial NFA construction. For example the NFA for matching a single character look like:

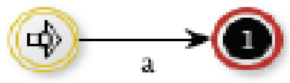

Figure 5: Automaton that represent a single character 'a' (a)

The concatenation is constructed by connecting the final arrow of first expression to the first node of second expression:

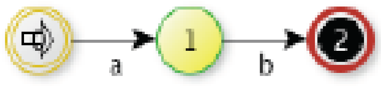

Figure 6: Automaton that represents the concatenation of two characters, 'a' & 'b' (ab)

The alternation of a|b is constructed by adding a new state with a choice of first expression and another choice to second expression:

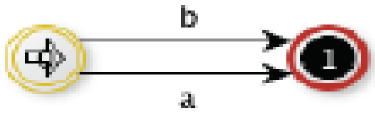

Figure 7: Automaton that represents the union of two characters, 'a' and 'b' (a|b)

The loops as a* or a+ are almost similar, and "a+" can be written as "aa*", so the NFA graph looks like:

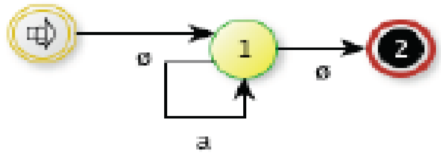

Figure 8: Automata representing a*

[B2] McNaughton and Yamada Algorithm

McNaughton and Yamada in 19 give an alternative approach, to avoid unlabeled arrows instead of allowing NFA states to have multiple outgoing arrows with the same label. It makes diagrams for subexpressions in a recursive way and then puts them together. According to Storer and Chang the McNaughton and Yamada's NFA has a distinct state for every character in RE except the initial state. That is why this is viewed as an alternative approach to Thompson;s algorithm.

The McNaughton and Yamada's algorithm in the initial phase creates disconnected initial and accepted state:

Figure 9: Automaton representing Empty set

CONCLUSION

This paper work provides an insight into the various approaches used for conversion of deterministic finite automata to regular expression and vice versa. Comparisons between different techniques for conversion of DFA to RE are carried out. Researching this project has shown that the conversion of regular expressions to DFA and back again are processes that are well understood and are implementable without any great difficulty. The most time-consuming part of the project was coding the parser for the regular expression. This is because while regular expressions define regular languages, they themselves are not regular and must be described by context-free grammars.

ACKNOWLEDGMENT

I would like to thanks my guide Ms. Shabnam Sangwan, who suggested me to work and research. Her recommendations, innovative ideas and constructive criticism contributed to make the success of this report. Her numerous suggestions, comments, and advice have made this entire paper possible.

REFERENCES

[1] Alfred V. Aho, "Constructing a Regular Expression from a DFA", Lecture notes in Computer Science Theory, September 27, 2010, Available at http://www.cs.columbia.edu/~aho/cs3261/lectures.

[2] Ding-Shu Du and Ker-I Ko, "Problem Solving in Automata, Languages, and Complexity", John Wiley & Sons, New York, NY, 2001.

[3] [Gelade, W., Neven, F., "Succinctness of the complement and intersection of regular expressions", Symposium on Theoretical Aspects of Computer Science. Dagstuhl Seminar Proceedings, vol. 08001, pages 325–336. IBFI (2008).

[4] Janusz A. Brzozowski, "Derivatives of regular expressions", J. ACM,11(4) pages 481-494, 1964.

[5] J. J. Morais, N. Moreira, and R. Reis, "Acyclic automata with easy-to-find short regular expressions", In 10th Conference on Implementation and Application of Automata, volume 3845 of LNCS, pages 349–350, France, June 2005. Springer.

[6] K. Ellul, B. Krawetz, J. Shallit, and M.Wang, "Regular expressions: New results and open problems", Journal of Automata, Languages and Combinatorics, 10(4):pages 407– 437, 2005.

[7] Larkin, H., "Object oriented regular expressions", 8th IEEE International Conference on Computer and Information Technology , vol., no., pages 491-496,8-11 July,2008

[8] Peter Linz, Formal Languages and Automata (Fourth Edition), Jones and Bartlett Publishers, 2006

[9] Michael Sipser, Introduction to the Theory of Computation, Thomson Course Technology, 2006

[10] A. Aho, R. Sethi & J. D. Ullman (1986): Compilers: Principles, Techniques, and Tools. Addison Wesley.

[11] A. V. Aho, J. E. Hopcroft & J. D. Ullman (1974): The Design and Analysis of Computer Algorithms. Addision-Wesley.

[12] A. V. Aho & J. D. Ullman (1972): The Theory of Parsing, Translation and Compiling. I, Prentice-Hall.

[13] V. Antimirov (1996): Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science 155(2), pp. 291–319.

[14] G. Berry & R. Sethi (1986): From Regular Expressions to Deterministic Automata. Theoretical Computer Science 48(3), pp. 117–126.

[15] H. Gruber & M. Holzer (2013): Provably Shorter Regular Expressions From Finite Automata. International Journal of Foundations of Computer Science 24(8), pp. 1255–1279.

[16] S. Lombardy, Y. R´egis-Gianas & J. Sakarovitch (2004): Introducing VAUCANSON. Theoretical Computer Science 328(1–2), pp. 77–96

[17] Z. Manna (1974): Mathematical Theory of Computation. McGraw-Hill

[18] Robert McNaughton & Hisao Yamada (1960): Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers EC-9(1), pp. 39–47

[19] G. Berry and R. Sethi. From regular expressions to deterministic automata. TCS: Theoretical Computer Science, 48:117–126, 1987.

[20] Janusz A. Brzozowski. Derivatives of regular expressions. J. ACM, 11(4):481–494, 1964.

[21] S. C. Kleene. Representation of events in nerve nets and finite automata. In Automata studies, pages 3–40. Ann. of Math. Studies No. 34, Princeton University Press, Princeton, NJ, 1956.

[22] Bruggemann-Klein A. [1993]. "Regular expressions into finite automata". Theoretical Computer Science. vol. 120, no. 2, pp. 197-213.

[23] [23] Geffert, V. [2003]. "Translation of binary regular expressions into nondeterministic ε-free automata with o(nlogn) transitions". Journal of Computer and System Sciences. vol. 66, no. 3, pp. 451-472.

[24] Hromkovic J., S. Seibert, and T. Wilke [2001]. "Translating regular expressions into small ε-free nondeterministic finite automata". Journal of Computer and System Sciences. vol. 62, no. 4, pp. 565-588.

[25] Yamamoto, H. [2005]. "New finite automata corresponding to semi extended regular expressions". Systems and Computers in Japan. vol. 36, no. 10, pp. 54-61.

Author Profile

Ashwani, received the B.Tech degree in Computer Science & Engineering from Tekmanchand College of Engineering, Sonepat affiliated to DCRUST University, Sonepat (Haryana) and pursuing M.Tech (2013 to 2015 batch) from Sat Kabir Institute of Technology and Management (SKITM), Bahadurgarh affiliated to Maharshi Dayanand University, Rohtak (Haryana). Currently I am doing research on RE and their acceptance tool Finite Automata.

Shabnam Sangwan, received the B.Tech degree in Computer Science & Engineering from Maharaja Surajmal Institute of Technology (MSIT), affiliated to Guru Gobind Singh Indraprastha University, New Delhi and M.Tech degree in Computer Science & Engineering from PDM college of Engineering, affiliated to Maharshi Dayanand University, Rohtak (Haryana) in 2011 and 2013 batch respectively. She is presently working in Sat Kabir Institute of Technology and Management (SKITM), Bahadurgarh, Haryana, India and had associated with PDM polytechnic, Bahadurgarh earlier to this. She has 12 publications in international journal and 5 publications in conferences. She is having membership of two international journals.

Sunita Kumari, received the B.Tech and M.Tech degree in Computer Science & Engineering from PDM college of Engineering, affiliated to Maharshi Dayanand University, Rohtak (Haryana) in 2003 and 2008 batch respectively. She is presently working in PDM Group of Institutions Bahadurgarh, Haryana, India. She has a teaching experience of 11 years. She has 20 publications in national and international journal and 12 publications in conferences. She has guided 20 students for M.Tech dissertation. She is having membership of various international journals and professional bodies.