Volume 2, No. 3, May-June 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
REVIEW ARTICLE
Available Online at www.ijarcs.info

Comparisons Amongst Different Techniques for Conversion of Deterministic Finite Automata to Regular Expression

Kulwinder Singh*
Student M.E.(CSE),
Computer Science and Engineering Department
Thapar University, TU
Patiala, India
kulwinder.thaparian@gmail.com

Ajay Kumar Loura
Assistant professor,
Computer Science and Engineering Department
Thapar University, TU
Patiala, India
ajaykumar@thapar.edu

Abstract: Regular expression is widely used in the field of computer science. For conversion of deterministic finite automata to regular expression, several techniques like Transitive closure method, Brzozowski Algebraic method and state elimination method are proposed. None of the proposed techniques is able to find the smallest regular expression. This paper investigates and compares different techniques for converting deterministic finite automata to regular expression. Inclusion of bridge state method in state elimination method leads to smaller regular expression as compared to other techniques.

Keywords: Regular expression; computer science; deterministic finite automata; bridge state; techniques

I. INTRODUCTION

Regular expressions denotes regular languages and regular languages are used in Compiler design, Text editor, search for an email- address, grep filter of unix, data validation, fridge thermostats, elevators, train track switches, context switching and in programming languages to search and manipulate text based on patterns [3][4][8] [9][13][15].

Kleene proves that there is equivalent regular expression corresponding to deterministic finite automata [12]. For converting deterministic finite automata to regular expression Kleene provides first technique known as transitive closure approach. Brzozowski extends this transitive closure approach, called Brzozowski algebraic approach [5]. Brzozowski's approach is a recursive approach and also uses Arden's Rule [14] for generating regular expression from deterministic finite automata. In this approach system of equations is created and regular expression is obtained by solving these equations. State elimination method [2][10] is most widely used approach for converting deterministic finite automata to regular expression. In this approach states (except start and final state) of deterministic finite automata are removed one by one and finally regular expression is generated equivalent to given deterministic finite automata. In state elimination method concept of bridge state [16] is also used for obtaining smaller regular expression.

In this paper we analyse the different techniques for converting deterministic finite automata to regular expression and test which is efficient technique for converting deterministic finite automata to regular expression. Paper is organized as follow. In the next section 2 some basic definitions and notations are reviewed. In Section 3, conversion of deterministic finite automata to regular expression is explained using techniques transitive closure approach, Brzozowski algebraic approach, state elimination method and concept of bridge state. In Section 4 comparisons of different techniques is carried out and finally Section 5 include conclusion and future scope.

II. BASIC DEFINITIONS AND NOTATIONS

Alphabet is defined as finite non-empty set of symbols on which the language is defined. Alphabets are denoted by Σ. Language [10] is defined as a subset Σ^*. Empty string and null language are denoted by □ and □ respectively. Various kinds of formal languages can be classified as regular, context free, context sensitive and recursive language. Regular language can be described by regular expression, finite automata (Deterministic or Non-deterministic).

A regular expression (RE) [14] 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 (□) and null language (□) are regular expression denoting the language {□} and {□} 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

- Union of E and F will be denoted by regular expression E+F and representing language L(E) U L(F).
- Concatenation of E and F denoted by EF and representing language L(E*F) = L(E) * L(F).
- Kleene closure will be denoted by E*and represent language (L(E))*

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

Deterministic finite automata (DFA) [14][6] can be defined by 5-tuples $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of symbols, δ is the transition function, that is, $\delta: Q \times \Sigma \rightarrow Q$.

q_0 is the start state and F is a set of states of Q (i.e. $F \subseteq Q$) called accept states.

A Non-deterministic finite automata (NFA) [14][6] is a same as DFA except the transition function. Transition function in NFA is defined as $Q \times \Sigma \rightarrow 2^Q$

III. CONVERSION OF DFA TO RE

First, 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 [1][2][5].

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

A. *Transitive Closure Method*

Kleene's transitive closure method [1][2] defines regular expressions and proves that there is equivalent RE corresponding to a DFA. Transitive closure is the first 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 $\sum^*$ that take the DFA from 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.

Let regular expression R_{ij} represents the set of all strings that transition the automata from node q_i to q_j.

R_{ij} can be constructed by successively constructing R^1_{ij}, R^2_{ij} ,…….. R^m_{ij}

R^k_{ij} is recursively defined as:

$$R^k_{ij} = R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} + R^{k-1}_{ij} \quad (1)$$

Assuming we have initialized R^0_{ij} to be:

$$R^0_{ij} = \begin{cases} r & \text{if } i \neq j \text{ and r transitions from } q_i \text{ to } q_j \\ r + \square & \text{if } i=j \text{ and r transitions from } q_i \text{ to } q_j \\ \square & \text{otherwise} \end{cases}$$

Using "(1)" $R^k_{ij} = R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} + R^{k-1}_{ij}$ [1] , we obtain Regular expression R_{ij}.

Consider the DFA given in "fig. 1" and applying transitive closure method on it.

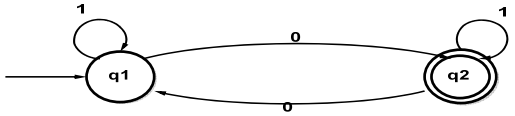

Fig.ure 1. DFA for the language having odd number of 0's

$r^0_{11} = 1 + \square$ $\quad r^0_{22} = 1 + \square$ $\quad r^0_{12} = 0$ $\quad r^0_{21} = 0$

$r^1_{12} = r^0_{12} + r^0_{11} (r^0_{11})^* r^0_{12} = 0 + (1+\square)^+ 0$

$r^1_{22} = r^0_{22} + r^0_{21}(r^0_{11})^* r^0_{12} = (1+\square) + 0(1+\square)^* 0$

$r^2_{12} = r^1_{12} + r^1_{12}(r^1_{22})^* r^1_{2} = (1+\square)^* 0(1+\square+01^* 0)$

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

$L(M) = L(r^2_{12})$

B. *Brzozowski Algebraic Method*

Brzozowski method [1][5] is a unique approach for converting deterministic finite automata to regular expressions. In this approach we create characteristics equations for each state which represent regular expression for that state. After solving the equation of R_s (regular expression associated with starting state q_s).

If R_i is regular expression for state q_i and there is transition a from state q_i to q_j , then term aR_j is added in the equation for R_i . If q_i is final node then term $\square$ (null index) is added in the equation. This leads to a system of equations in the form:

$R_1 = a_1R_1 + a_2R_2 + \ldots\ldots$

$R_2 = a_1 R_1 + a_2R_2 + a_3R_3 + \ldots\ldots$

$R_m = a_1 R_1 + a_2R_2 + \ldots + \ldots \square$ $\quad \square$ is added if R_m is final node

Where $a_x = \emptyset$ if there is no transition from Ri to Rj.

Using Arden's Theorem [14] we solve the obtained equations. It states that if an equation is of the form

$X = AX + B$, its solution is $X = A^* B$

For example DFA shown in the "fig. 2" is converted to its corresponding regular expression using Brzozowski algebraic method.

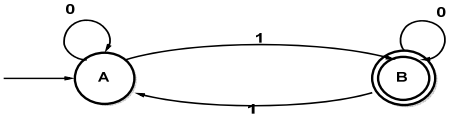

Figure 2. DFA for strings with an odd no of 1's

Characteristics equations are as follow:

$A = 0A + 1B$ (2)

$B = 1A + 0B + \square$ (3)

Solving these equations by Arden's theorem

$B = 1A + 0B + \square = 0B + (1A + \square) = 0^*(1A + \square)$

$B = 0^*(1A) + 0^*(\square) = 0^*1A + 0^*$ (4)

Using "(4)" we obtain

$A = 0A + 1B = 0A + 1(0^*1A + 0^*)$

$A = 0A + 10^*1A + 10^*$

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

C. *State Removal Method*

The state removal approach [1][14] is widely used approach for converting DFA to regular expression. In this approach states of DFA are removed one by one until we left with only starting and final state, for each removed state regular expression is generated. This newly generated regular expression act as input for a state which is next to removed state. The advantage of this technique over the transitive closure method is that it is easier to visualize. This technique is described by Du and Ko [2], but a much simpler approach is given by Linz [10]. At first, if there are multiple edges from one node to other node, then these are unified into a single edge that contains the union of inputs. Suppose from q_1 to q_2 there is an edge weighted 'a' and an edge weighted 'b', those would be unified into one edge from q_1 to q_2 that has the weight a + b. If there are n accepting states, take union of different regular expressions.

For example DFA shown in "fig. 3" is converted to its corresponding regular expression using state elimination method

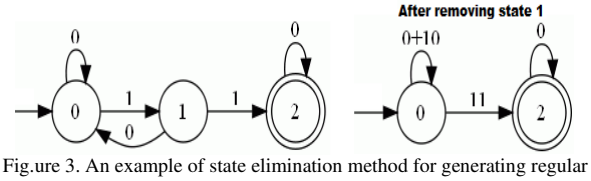

Fig.ure 3. An example of state elimination method for generating regular expression corresponding to given DFA.

Equivalent regular expression corresponding to given DFA is (0+10)*11(0)*

Han and Wood have worked on the concept of bridge state in the state elimination method. A state q_b of automata A is said to be bridge state [16], if it satisfies the following conditions:

1) State q_b is neither a start nor a final state.
2) For each string w □ L(A), its path in A must pass through q_b at least once.
3) Once string w's path passes through q_b for the first time, the path can never pass through any states that have been visited before apart from state q_b.

By using different removal sequences of states, we obtain different regular expressions for the same language. We require a removal sequence which gives smaller regular expression. Elimination of non-bridge state before bridge state gives us smaller regular expression. [16].

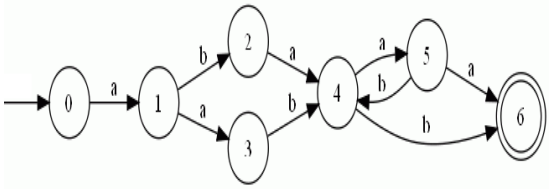

Figure 4. An example of different RE by different removal sequences for a given DFA .

State 1 and state 4 are bridge states of DFA shown in "fig. 4".

R_1=a(ba+ab)(ab)*(aa+b) using state elimination sequence 2-3-5-1-4

R_2=(ab(ab+aa(ba)*(a+bb))+aa(bb+ba(ba)*(a+bb))) using state elimination sequence 1-4-5-2-3

R_3=[(aba+aab)b+(aba+aab)a(ba)*(a+bb)] using state elimination sequence 2-1-3-4-5

Removing the bridge states 1 and 4 at last gives smaller regular expression.

IV. COMPARISON OF APPROACHES USED FOR CONVERTING DFA TO RE

We will apply these three methods on DFA which accepts the string having even number of 0's and 1's.

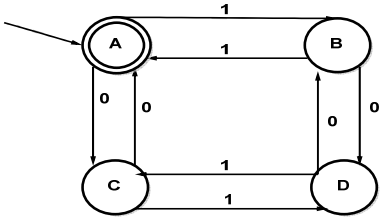

Figure 5. DFA for string having even no of 1's and 0's

A. *Transitive Closure Approach*

Applying transitive closure approach on DFA shown in "fig. 5".

$$R = R^c_{aa} = R^d_{aa} + R^d_{ac} (R^d_{cc})^* R^d_{ca}. \quad (5)$$

$$R^d_{aa} = R^b_{aa} + R^b_{ad} (R^b_{dd})^* R^b_{da}$$

$$R^b_{aa} = R^a_{aa} + R^a_{ab} (R^a_{bb})^* R^a_{ba}$$

$$R^a_{aa} = R^0_{aa} + R^0_{aa} (R^0_{aa})^* R^0_{aa} = □ + □ (□)^* □ = □$$

$$R^a_{ab} = R^0_{ab} + R^0_{aa} (R^0_{aa})^* R^0_{ab} = 1 + □ (□)^* 1 = 1$$

Similarly we will get

$$R^a_{bb} = 1 \quad , R^a_{ba} = 1 , R^b_{aa} = 1 (1)^* 1 \quad , R^b_{ad} = 1 (1)^* 0$$

$$R^d_{aa} = [1 (1)^* 1] + [1 (1)^* 0] [0 (1)^* 0] [0 (1)^*]$$

$$R^d_{ac} = [0+1(1)^*0]+[1(1)^*0][0 (1)^* 0] [1 + 0 (1)^* 1]$$

$$R^d_{cc} = [00+01(1)^*10]+(1) [0 (1)^* 0] [0 + 0 (1)^* 10]$$

$$R^d_{ca} = 0 [(0 + 00) + 111] + 1 [0 (1)^* 0] [0 (1)^* 1]$$

By Putting these values in "(5)"

R= [1(1)* 1] + [1(1)*0] [0(1)*0] [0(1)*] + [0 + 1(1)*0] + [1(1)*0] [0 (1)*0] [1+0(1)* 1] ([00 + 01(1)*10] + (1) [0(1)*0] [0 + 0(1)*10])*0 [(0+00) + 111] + 1[0 (1)* 0] [0 (1)*1]

B. *Brzozowski Algebraic method*

Set of equations for generating regular expression for automata shown in "fig..5".

$$A = 0C + 1B + □ \quad (6)$$

$$B = 0D + 1A \quad (7)$$

$$C = 0A + 1D \quad (8)$$

$$D = 0B + 1C \quad (9)$$

Using "(7)"

A = 0C + 10D + 11A + □ D = 00D + 01A + 1C

Using "(8)".

A = 00A + 01D + 10D + 11A + □

A = (00 + 11)A + (01 + 10)D + □

D = (00 + 11)D + (01 + 10)A

D= (00 + 11)*(01 + 10)A (10)

(using Arden's rule)

Using "(10)"

A = (00 + 11)A + (01 + 10)(00 + 11)*(01 + 10)A + □

A= [00 + 11 + (01 + 10)(00 + 11)*(01 + 10)]A + □

A = [(00 + 11) + (01 + 10)(00 + 11)*(01 + 10)]* (using Arden's rule)

C. *State Removal Method*

Applying state elimination method on DFA shown in "fig. 5"

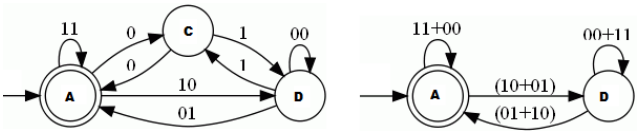

Figure 6. DFA after eliminating state B and state C respectively

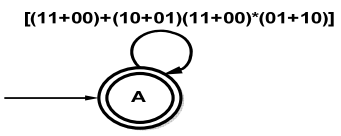

Figure 7. Final DFA after eliminating state D

Regualr expression corresponding to DFA shown in "fig. 5" is R= [(11 + 00) + (10 +01) (00 + 11)*(01 + 10)]*
Due to repeated union of concatenated terms transitive closure method is very complex and gives very long regular expression as compared to Brzozowski algebraic method and state elimination method. Brzozowski algebraic method is recursive approach and generates reasonably compact regular expressions. For recursion oriented languages, like functional languages, transitive closure method is difficult to implement, in that case Brzozowski algebraic method is used.
Brzozowski method takes more time as compared to state elimination method, in which regular expression is obtained by manual inspection. State elimination method using bridge state concept gives shorter regular expression as compared to Brzozowski algebraic method.

V. CONCLUSION AND FUTURE SCOPE

State elimination method takes less time, and generally gives smaller regular expression as compared to Transitive closure and Brzozowski algebraic method. In state elimination method the removal of sequences also give us different regular expression. By choosing a good removal sequence, we can obtain a shorter regular expression.

There is need to develop a software that convert a given deterministic finite automata into regular expression using bridge state. New heuristics can be designed to find smaller regular expression from DFA using state elimination method. Although algorithm exist for finding bridge state, but there is a need of an efficient algorithm for finding bridge state.

VI. REFERENCES

[1] Christoph Neumann ,Converting Deterministic Finite Automata to Regular Expressions, Mar 16, 2005

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

[3] H. Hosoya, Regular expression pattern matching – a simpler design. Technical Report 1397, RIMS, Kyoto University, 2003.

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

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

[6] Mishra K.L.P.& N. Chandrasekaran, Theory of Computer Science (Automata Language and. Computation), PHI, Second edition, 1998

[7] M. Delgado, J. Morais, Approximation to the smallest regular expression for a given regular language, in: Proceedings of CIAA'04, in: Lecture Notes in Computer Science, vol. 3317, 2004, pp. 312–314.

[8] M. Lesk and E. Schmidt. Lex - A Lexical Analyzer Generator. Computing Science Technical Report No. 39, Bell Laboratories, USA, 1975.

[9] Mulder,M.;Nezlek,G.S., Creating protein sequence patterns using efficient regular expressions in bioinformatics research , 28th International Conference 2006 ,Page(s):207 – 212

[10] Peter Linz. An introduction to Formal Languages and Automata. Jones and Bartlett Publishers, Sudbury, MA, third edition, 2001.

[11] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IEEE Transactions on Electronic Computers 9 (1960) 39–47.

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

[13] Ulman, J., A. V. Aho and R. Sethi. Compiler Design: Principles, Tools, and Techniques. Reading, Pearson Education Inc, ISBN 0-201-10088-6,1986.

[14] Ullman, J., J. E. Hopcroft and R. Motwani. Introduction to Automata Theory, Languages, and Computation. Pearson Education Inc, ISBN 0-201-44124-1. Reading, MA: Addison Wesley, 2001.

[15] Will Drewry and Tavis Ormandy, Google, Inc., Insecure Context Switching: Inoculating regular expressions for survivability

[16] Yo-Sub Han, Derick Wood, Obtaining shorter regular expressions from finite-state automata, Theoretical Computer Science, Volume 370, Issues 1-3, 12 February 2007, Pages 110-120, ISSN 0304-3975, DOI: 10.1016/j.tcs.2006.09.025.

[17] Y.-S. Han, D. Wood, The generalization of generalized automata: Expression automata, International Journal of Foundations of Computer Science 16 (3) (2005) 499–510.

