© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 05 | July - September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

An Efficient Technique for Conversion of Regular Expression to and from Finite Automata

Neha Sharma, Department of Computer Science, *Neha_sh0815@yahoo.com*
South Point Institute of Technology and Management,Sonepat

Abstract**—** 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. In this paper, for Conversion of regular expression to NFA we study the Thomson Algorithm; to convert NFA to DFA we use Subset Construction method, to minimized DFA constructed from previous step we use partition method and finally to convert DFA back to RE we use Universal Technique.

Keywords**— Regular Expression, *DFA*, *NFA*, ε-closure**

I. INTRODUCTION

In formal language theory regular expressions consist of strings of symbols from a finite alphabet Σ combined by various operators. In computing in general they can be used to match and replace strings, but formally they define regular languages. Regular languages can be roughly defined, somewhat recursively, as any language consisting of a potentially infinite set of sequences of finite symbols from a finite alphabet that can be described by a regular expression or deterministic or nondeterministic finite automaton.

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 and several heuristics are explored for obtaining smaller regular expression using state elimination approach. Here we define and implement algorithms to convert regular expressions to NFA, to convert these NFA to DFA, minimization of these DFA and finally conversion of these minimized DFA back into a regular expressions. The algorithms addressed include Thompson's Algorithm and the Rabin and Scott's Subset Construction. To minimized DFA constructed from previous step we use partition method and finally to convert DFA back to RE we use Universal Technique.

II. LITERATURE REVIEW

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.[2, 12, 10]

- 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.

INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 05 | July - September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

[A2] Brzozwski Algebraic Method

Brzozowski method [10] 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).

[A3] State Elimination Method

The state removal approach [12, 13] 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 [13]. 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 q1 to q2 there is an edge weighted 'a' and an edge weighted 'b', those would be unified into one edge from q1 to q2 that has the weight (a + b). If there are n accepting states, take union of n different regular expressions.

[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

Thompson algorithm was first described by Thompson in his CACM paper in 1968. 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.

[B2] McNaughton and Yamada Algorithm

The idea of the McNaughton and Yamada algorithm is that it makes diagrams for subexpressions in a recursive way and then puts them together. According to Storer [10] and Chang [9] the McNaughton and Yamada's NFA has a distinct state for every character in RE except the initial state. We can say that McNaughton and Yamada's automaton can also be viewed as a NFA transformed from Thompson's NFA.

III. Overview of Work

[A] Conversion of Regular Expression to NFA - Thompson's construction

The simplest method to convert a regular expression to a NFA is Thompson's Construction, also known as Thompson's Algorithm. Roughly speaking this works by reducing the regular expression to its smallest constituent regular expressions, converting these to NFA (shown here as state diagrams) and then joining these NFA together.

Ex: We will construct the NFA for the regular expression (a | b)* | c using Thompson's Construction.

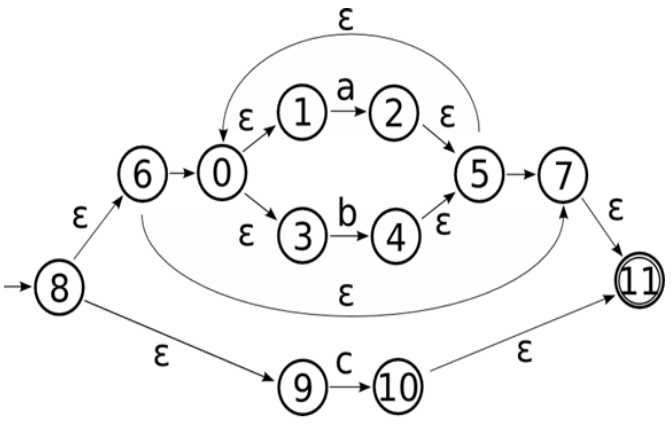


Figure 1: Thomson Construction: (a | b)* | c

[B] Conversion of NFA to DFA: The Subset Construction

To convert the NFA to DFA we will use Rabin and Scott's Subset Construction. Central to this is the concept of the *closure*. One of the major steps of the subset construction is:

Find the ε-closure for each state.

This is defined as:

The ε-closure of state 0 is the set of all states reachable from 0 by zero or more ε-transitions.
Construct DFA from the NFA of Regular Expression (a | b) * c.

INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 05 | July - September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

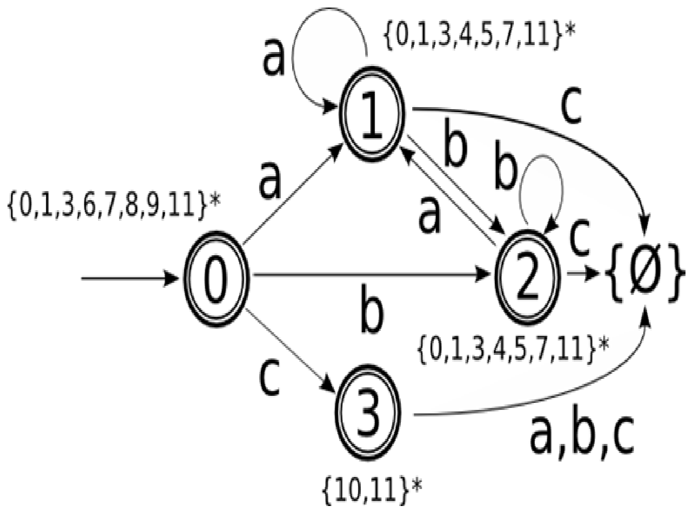


Figure 2: **Subset Construction: (a | b) * c**

[C] Minimisation of DFA

For any DFA there is a DFA with equal or fewer states that matches exactly the same language. In other words, there can be several DFAs that represent a language, some will be bigger than others and only one will be minimal.

To minimise a DFA three steps must be performed:

▪ *Removal of dead states.*
▪ *Removal of inaccessible states.*
▪ *Merging of identical states.*

IV. PROPOSED WORK

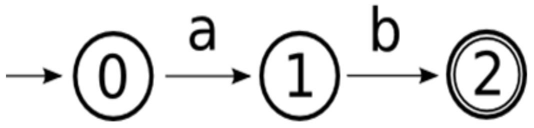
To convert our DFA back to a regular expression we will use universal technique.

A universal regular expression is a non-specific regular expression which is lengthy then the specific regular expression but it is cleaner then specific regular expression. We can create a universal regular expression that contains the original regular expression by performing three steps on our minimised DFA:

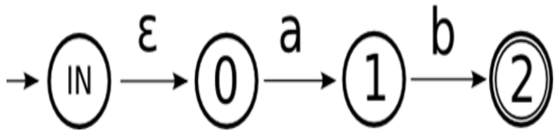
▪ *Add one start state that steps to all other start states via an ε.*
▪ *Add one final state that final states step to via an ε.*
▪ *Remove each state between these in turn until we are left with the regular expression.*

The first two steps are best represented using state diagrams and the last is best demonstrated using a matrix. Indeed the implementation of the third step involves successive 2-dimensional arrays.

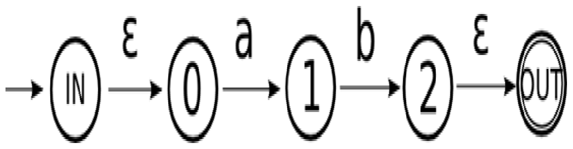
First we consider a simple DFA that we could have generated from the regular expression ab. Here is our initial DFA (with transitions to null states removed for clarity):



First we add a new start state IN:



Next we add a new final state OUT:



Thirdly we create a matrix (transition table) from which we will remove states:

Table 1: **Transition table 1 (for the string *ab*)**

	IN	**0**	**1**	**2**	**OUT**
IN	-	ε	-	-	-
0	-	-	a	-	-
1	-	-	-	b	-
2	-	-	-	-	ε
OUT	-	-	-	-	-

The rows represent the state being stepped from and the columns represent the state being stepped to. For example, the symbol *a* in the table represents the step from state 0 to state 1 using the symbol *a*.

Next we will remove one state from the matrix to create a new matrix. For this example we will consistently remove the zero state. In order to do this the steps between other nodes may have to be altered to compensate for the missing state. We calculate this with the following pseudo-code:

```
rip = 0
for i = IN to 2 inclusive excluding 0 do
for j = 1 to OUT inclusive do
```

INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 05 | July - September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

```
if (i == IN)
cell[i, j - 1] = cell[i,j] | ( cell[i, rip] (cell[rip, rip])* cell[rip, j] )
else
cell[i – 1, j - 1] = cell[i,j] | ( cell[i, rip] (cell[rip, rip])* cell[rip, j] )
end if
end for
end for
```

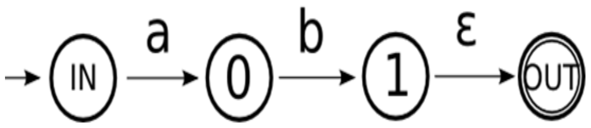
We exclude state 0 from both i and j as we are removing it. i doesn't address the OUT state as nothing can step from it and j doesn't address the IN state as there is no way it can be stepped to. The *if and else* are present as the new grid will have one fewer row and column than the original. We need to reduce the indices otherwise some of the new values won't fit.

Applying the above procedure to our original matrix leaves us with the following new matrix:

Table 2: **Transition table 2 (for the string *ab*)**

	IN	**0**	**1**	**OUT**
IN	-	*a*	-	-
0	-	-	b	-
1	-	-	-	ε
OUT	-	-	-	-

Which looks like:

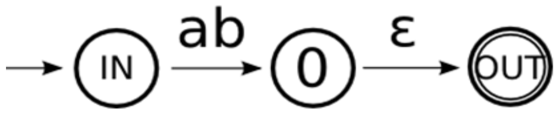


Next we will perform the same procedure on the new grid. This gives us:

Table 3: **Transition table 3 (for the string *ab*)**

	IN	**0**	**OUT**
IN	-	*ab*	-
0	-	-	ε
OUT	-	-	-

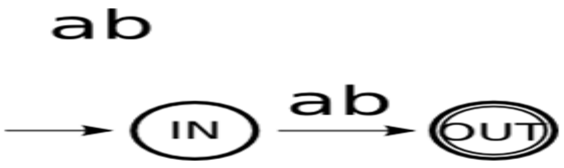
This can be represented as:



Here we can see the distinctive feature of universal regular expression over DFA and NFA, that there can be transitions consisting of regular expressions. We can already see that we have our original regular expression back again, but this is not always the case for all conversions. For completeness we will show the remaining step:

Table 4: **Transition table 4 (for the string *ab*)**

	IN	**OUT**
IN	-	ab
OUT	-	-



The top right hand cell of the final two-by-two matrix shows the regular expression. In this instance the regular expression generated is the same as the original. This is often not the case, the regular expressions produced by this method can be much longer than the original and have many redundant terms.

The regular expressions (a|b)* when turned into DFA and back again by our code produces the regular expression (epsilon|(a(a)*))|((b)|(a(a)*b)((b)|(a(a)*b))* epsilon|(a(a)*)))

After removing excess brackets we are left with (epsilon|(aa*))|(b|(aa*b)(b|(aa*b))*epsilon|(aa*)))

If examined closely it can be seen that this is indeed the same as (a|b)* although it's far from obvious.

IV. Results & Discussions

[A] Results

In order to obtain the desired results from the conversion software we require following steps:

1. Type the regular expression term in the concern text field.

2. Press the generate FA tables button.

The software will perform the following actions and the result of these steps is obtained on the screen.

1. The NFA table text area is filled with the transition table of NFA states.

2. The DFA table text area filled with the transition table of corresponding DFA states.

3. The minimized DFA table text area is filled with the transition table of minimized DFA states.

4. The final term text field is filled with enhanced form of regular expression constructed from minimized DFA states.

5. The label at the bottom will display success message "Regex conversion completed".

[B] Discussions

Thompson's Algorithm

Thompson's algorithm can be shown to have a running time linear to the length of the regular expression. This is obvious from the structure of the parser in the class RegexAutomaton – it passes through the regular expression once and constructs the NFA from simple components. The Thomson algorithm requires storage one more than the length of the regular expression.

Subset Construction

Rabin and Scott's subset construction as outlined in this document has a worst case of n^2 where n is the number of NFA nodes, although in most cases it is considerably less. It is the main method of converting an NFA to a DFA. It requires storage same as the number of NFA nodes.

State Minimisation

The state minimisation algorithm outlined here is one of the commonest and has a worst-case running time of n^2. In the first step it reduces one state, in the 2nd another state. Hence its complexity is n^2. It require storage equal to the number of nodes in minimize DFA.

DFA to Regex via Universal Technique

This algorithm has a running time that has a worst case running time of n^2. This can readily be deduced from its use of a matrix, which clearly hints at the running time being at least proportional to the square of the number of terms. The terms themselves can also be very lengthy and the matrix can consume a lot of storage. It can consume storage space equal to width of final regular expression constructed.

V. CONCLUSION

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. In this paper, for Conversion of regular expression to NFA we study the Thomson Algorithm; to convert NFA to DFA we use Subset Construction method, to minimized DFA constructed from previous step we use partition method and finally to convert DFA back to RE we use Universal Technique.

.

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] Gruber H. and Gulan, S. (2009), "Simplifying regular expressions: A quantitative perspective", IFIG Research Report 0904.

[5] Gruber H. and Holzer, M., "Provably shorter regular expressions from deterministic finite automata", LNCS, vol. 5257, pages 383–395. Springer, Heidelberg (2008).

[6] Gulan, S. and Fernau H., "Local elimination-strategies in automata for shorter regular expressions", In Proceedings of SOFSEM 2008, pages 46–57 (2008).

[7] H. Gruber and M. Holzer, "Finite automata, digraph connectivity, and regular expression size", In Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Iceland, July 2008. Springer.

[8] H. Gruber and J. Johannsen, "Optimal lower bounds on regular expression size using communication complexity", In Proceedings of the 11th International Conference Foundations of

Software Science and Computation Structures, volume 4962 of LNCS, pages 273–286, Budapest, Hungary, March–April 2008. Springer.

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

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

[11] 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.

[12] 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.

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