A Novel Method To Construct Deterministic Finite Automata From A Given Regular Grammar

K. Senthil Kumar¹ D. Malathi²

Abstract—Membership checking is an important task in language formalism. In regular Language, Deterministic Finite Automata plays membership checking in a convenient way. Regular language is normally represented by regular expression from which people are finding Deterministic Finite Automata through Non Deterministic Finite Automata. In this paper we have proposed a second degree polynomial algorithm to find the Deterministic Finite Automata for a given Regular grammar. This paper describes a novel method which finds Deterministic Finite Automata directly from a given regular grammar without going through NFA. We extend the idea of GOTO and CLOSURE functions in LR parsing model to Regular grammars and use the same to find the states of the required Deterministic Finite Automata. Also we have proposed a new algorithm which minimizes the number of states of the obtained Deterministic Finite Automata.

1 INTRODUCTION

Regular language and finite automata plays a crucial role in pattern matching. Regular expression is used to specify certain pattern of interest and Non deterministic automata and Deterministic automata are the models to recognize the pattern. Deterministic Finite Automata plays a vital role in lexical analysis phase of compiler design, Control Flow graph in software testing, Machine learning[2] etc.

In the literature various methods are available for constructing Deterministic Finite Automata (DFA) like subset construction method which finds DFA from Non Deterministic Finite Automata (NFA). Using Thomson method we can find DFA from given regular expression through an ε-NFA. In this paper we have proposed a novel method to find Deterministic Finite Automata directly from a given Regular grammar. This paper is organized as follows. Section 2 deals with notations and preliminaries, Section 3 discusses related works, Section 4 and its subsection deals with our proposed algorithm and Section 5 deals with minimization of the obtained Deterministic Finite Automata and finally conclusion is given.

2. NOTATIONS AND PRELIMINARIES

We assume the basics in automata theory as contained in [8]. Throughout this paper, \( \Sigma \) denotes input alphabet (finite set of symbols), \( G \) represents words and \( \text{mod}(w) \) represents the length of the word. A special word \( \epsilon \) is used to denote empty word, whose length is assumed to be 0. Also a language \( L \) over \( \Sigma \) is defined as \( L \subseteq \Sigma^* \). A Deterministic Finite Automata is a five tuple which is defined [8] as a quintuple \( A = (Q, \Sigma, \delta, q_0, F) \), where

- \( \Sigma \) — Finite input alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) the transition function
- \( q_0 \) — Initial state
- \( F \subseteq Q \) — Final states

A Non Deterministic Finite Automata is also a five tuple except in transition function \( \delta \) which is defined as \( \delta : Q \times \Sigma \rightarrow 2^Q \) Where \( 2^Q \) represents the power set of \( Q \).

Regular Language:
1. A language \( L \) is regular if there exists an Finite automata \( M \) such that \( L(M) = L \).
2. A language \( L \) is regular if there exists a regular expression \( r \) such that \( L(r) = L \).

A regular language is a language for which a Regular grammar or Regular Expression or NFA or DFA can be constructed.

Regular Grammar:
A Grammar \( G \) is said to be left linear if its rules are of the form

\[ A \rightarrow B \alpha \text{ (or) } A \rightarrow \alpha \]

A Grammar \( G \) is said to be Right linear if its rules are of the form

\[ A \rightarrow \alpha B \text{ (or) } A \rightarrow \alpha \]

A Regular grammar is one which is either left linear or right linear. (Also \( A \rightarrow \epsilon \) is to be added if it contains empty word).

¹K. Senthil Kumar, Assistant Professor, is with the Department of Computer Science and Engineering, S.R.M University, Tamil Nadu, Pin : 603203, India. (phone: +919840860221; E-mail: senthilkumar.k@ktr.srmuniv.ac.in)

²Corresponding Author: D. Malathi, Professor, is with the Department of Computer Science and Engineering, S.R.M University, Tamil Nadu, Pin: 603203, India. (phone: +919442554055; E-mail: malathi.d@ktr.srmuniv.ac.in, malaa_kam@yahoo.com)
3. RELATED WORKS IN DFA CONSTRUCTION AND MINIMIZATION

In the Regular language formalism we know that regular expression, NFA(with ε), NFA (without ε), DFA are equivalent. Minimization of DFA is also an important as it prevents some unnecessary computations. Jielan Zhang and Zhongsheng Qian [4] have discussed the equivalence between Regular grammar and DFA. R. McNaughton and Yamada [12] have proposed algorithms which construct DFA for the given regular expression in \(O(n^2)\) time and space. Anne Bruggemann-Klein [9] showed that the Glushkov automaton can be constructed in a time quadratic in the size of the expression. Rajesh Parekh, Codrin Nichituiru and Vasanth [2] have described an efficient incremental algorithm for learning regular grammars from labeled examples and membership queries. Chia-Hsiang Chang [3] presented an algorithm which computes the same NFA in the same asymptotic time \(\theta(m)\) where \(m\) denotes number of edges as in Berry and Sethi [10] but it improves the auxiliary space to \(\theta(s)\) where \(s\) denotes number of states of NFA. Myhill-Nerode [14] famous theorem minimizes DFA. Moore’s algorithm [15] maintains a partition that starts off separating the accepting states from rejecting states, and repeatedly refines the partition until no more refinements can be made. In minimization also we have the famous J.E. Hopcroft [11] \(n \log n\) minimization algorithm which uses partition refinement technique. Jean Berstel, Luc and Oliver Carton [13] have proved that the bound in J.E. Hopcroft algorithm is actually tight by providing an automata of family of size \(n=2^k\) for which their algorithm runs in \(k2^k\). Also recently S. Bhargava and G. N. Purohit [5] used graph grammar rules to construct a minimal DFA from the given regular expression with a time complexity of \(O(n \log n)\).

4. ALGORITHM TO FIND DFA FOR A GIVEN REGULAR GRAMMAR

In this paper we have considered right linear regular grammar for which an algorithm is proposed. This algorithm finds all states and finally the required DFA. GOTO( ) and CLOSURE( ) functions for context free grammar are defined as in [1] by A. V. Aho et al. We extend this idea to regular language. We define GOTO( ) and CLOSURE( ) functions for regular language as follows:

**Definition:** GOTO(Qi, X) where Qi is a state (set of items) and X is a grammar and symbol is \([A \rightarrow aX.]\) such that \([A \rightarrow a.X]\) is in Qi.

**Definition:** CLOSURE(Qi) of an item for a regular grammar G is constructed by the following two rules.

1. **Every item Qi is added to** CLOSURE(Qi) **in**
2. **If there is an item** A \(\rightarrow a.B\) **in** CLOSURE(Qi) **and** B \(\rightarrow x\) (a is a terminal; x may be a terminal or x=cM for terminal c and variable M).

**Assumptions:** We assume the following:

i. The given regular grammar is in right linear form.
ii. The required DFA has only one final state.
iii. It has no epsilon productions (If it has some epsilon productions, then we have more than one final state)
iv. The input alphabet contains two elements only.

The following algorithm finds all states of the required DFA.

**4.1 Algorithm:**

**Input:** A Regular Grammar G describing language L(Containing k rules)
- Which we convert into an Augmented Grammar G' by introducing \(S' \rightarrow S\)
- Each rule will have almost 4 characters and G' will have k+1 rules.

**Output:** The Deterministic Finite Automata for the corresponding language L.

**Find_statesDFA(G')** // this will find all states of DFA corresponding to the given grammar

Initially \(Q_0 = \text{CLOSURE}(S' \rightarrow S)\) is the only state in required DFA states and it is unmarked;

While (there is an unmarked state Qi in DFA_states)
do
Mark Qi;
For each a \(\in \sum\) do begin
\(Q' = \text{GOTO}(Q, a)\)
If \(Q'\) is not in DFA_states then
Add \(Q'\) as an unmarked state to DFA_states;
\(\text{GOTO}(Q, a) = Q'\);
End

Also we define a procedure for CLOSURE function for a state Qi which contains some set of items

**Compute_CLOSURE(Qi)**

For each item \([A \rightarrow a.B]\) belongs to Qi
for each production \(B \rightarrow x\) or \(B \rightarrow xC\) \(\epsilon G'\)
If \(B \rightarrow x\) or \(B \rightarrow xC\) is not in Qi add
\(B \rightarrow x\) or \(B \rightarrow xC\) to Qi
Return(Q);

**Function GOTO(Qi, a terminal)**

Let I be the set of items \([A \rightarrow a.B]\)
Such that $A \rightarrow xBcQ$

Return $\text{CLOSEURE}(I)$;

4.2 Construction of the DFA

After finding all the states (items) by the proposed algorithm, we construct the corresponding DFA as follows.

1. $Q_0$ is the initial state and if $\text{GOTO}(Q_0,a)=Q_j$ construct an edge from $Q_0$ to $Q_j$ with $a$ where $a$ is a terminal

2. Last state which we obtain from the above algorithm is the final state.

3. If $\text{GOTO}(Q_i,a)$ is not available for an input alphabet $a$ then make an edge from $Q_i$ to a state $Q_d$, which is known as dead state with edge value $a$.

4. $Q_d$ will have self loop for all $a \in \Sigma$

4.3 Time Complexity Computations

In the algorithm mainly we use three functions.

1. $\text{Compute}_{-}\text{CLOSEURE}(Q)$

   In this computation, we identify all the variables and we include all those rules in subsequent computation. Hence it can be done in a second degree polynomial time.

2. $\text{GOTO}(Q,\text{terminal})$

   This function also scans all the items of $Q$ to find the corresponding terminal and then it applies $\text{CLOSEURE}$ function. This can also be done in a linear polynomial time of $n$ (where $n<4k+1$ where $k$ denotes number of production rules)

4.4 An illustration

Suppose that the given regular grammar is

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow aA$

$A \rightarrow bB$

$B \rightarrow b$

The augmented grammar becomes

$S' \rightarrow S$

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow aA$

$A \rightarrow bB$

$B \rightarrow b$

We calculate initial state $Q_0$ as follows.

$Q_0=\text{CLOSEURE}(S' \rightarrow S)$

Therefore

\[Q_0:\]

$S \rightarrow S$

$S \rightarrow aS$

$S \rightarrow bS$

$S \rightarrow aA$

\[\text{GOTO}(Q_0,a)= \text{CLOSEURE}([S \rightarrow a.S](S \rightarrow a.A))]\]

The required DFA is shown in Fig1.
5. MINIMIZATION OF DFA

In this section we have explained the proposed algorithm which minimizes the Deterministic Finite Automata. (We take two alphabets only and the result can be generalized to n alphabets).

Definition: Two states P and Q are set to be “equivalent” if $\text{GOTO}(P,a)=\text{GOTO}(Q,a)$ for all $a \in \Sigma$.

5.1 Theorem

‘Equivalent’ relation is an equivalence relation on $Q$-the set of states of the DFA.

Proof:

Given $Q=\{Q_1, Q_2, \ldots, Q_n\}$ the relation ‘Equivalent’ is defined as $Q_i$ is equivalent to $Q_j$ if and only if $\text{GOTO}(Q_i,a)=\text{GOTO}(Q_j,a)$ for all $a \in \Sigma$. Clearly the relation satisfies

1) Reflexive property as $\text{GOTO}(Q_i,a)=\text{GOTO}(Q_i,a)$; Hence $Q_i$ is equivalent to $Q_i$.

2) Symmetric property as $Q_i$ is equivalent to $Q_j$ if $\text{GOTO}(Q_i,a)=\text{GOTO}(Q_j,a)$ for all $a \in \Sigma$. Clearly it can be written as $\text{GOTO}(Q_j,a)=\text{GOTO}(Q_i,a)$ for all $a \in \Sigma$. This implies $Q_j$ is equivalent to $Q_i$.

3) Transitivity property is satisfied as $Q_i$ is equivalent to $Q_j$, $Q_j$ is equivalent to $Q_k$, to prove $Q_i$ is equivalent to $Q_k$ $Q_i$ is equivalent to $Q_j$ if $\text{GOTO}(Q_i,a)=\text{GOTO}(Q_j,a)$ for all $a \in \Sigma$, $Q_j$ is equivalent to $Q_k$ if $\text{GOTO}(Q_j,a)=\text{GOTO}(Q_k,a)$ for all $a \in \Sigma$.

Therefore, we infer that $Q_i$ is equivalent to $Q_k$. Therefore, if there are $C_n$ classes then the obtained DFA is already in Minimized form.

5.2 Minimization Algorithm

We partition the states of DFA as follows. Initially start with first three states (since in each state with given two input symbols we may get at most two new states). If there is no equivalent states in the initial partition then set $C_1=Q_1$ consider next partition as $\{Q_2, Q_3, Q_4\}$. If two states are equivalent they should be merged. Suppose that $Q_{a}, Q_{a+1}, Q_{a+2}$ is the partition at a particular point of time suppose that $Q_{a}, Q_{a+1}$ are equivalent they should be merged and the next partition is $\{Q_{a+2}, Q_{a+3}, Q_{a+4}\}$. Suppose $Q_{a}$ and $Q_{a+2}$ are equivalent then the partition starts from $Q_{a+3}$.

Minimization algorithm:

Input: Given a DFA with $n$ states $Q_1, Q_2, \ldots, Q_n$

Output: A Minimized DFA which accepts the same language

Method:

Minimize_DFA($Q[1..n]$) // This contains $n$ states

$i \leftarrow 1$;

II: while ($i < n$)

$j \leftarrow i + 1$;

while ($j \leq i + 2$)

if ($\text{GOTO}(Q_i,a)=\text{GOTO}(Q_j,a)$) and ($\text{GOTO}(Q_i,b)=\text{GOTO}(Q_j,b)$) {

merge($Q_i$ & $Q_j$);

$i \leftarrow j + 1$;

} goto II;


If we consider the same example, it can easily be proved that $Q_0$ and $Q_2$ can be merged and the resulting minimized DFA is shown below in Fig 2.
5.3 Time Complexity Computation

Since each time we consider three states and check the equivalent states among the partition; if exists then we merge those states. Then we consider next partition. So if we have \( n \) states we may have \( n-2 \) possible partitions. Also merging can be done almost by a linear polynomial time. Hence total minimization can be done by a linear polynomial time of \( n \).

6. CONCLUSION

We have proposed a novel algorithm which finds Deterministic Finite Automata directly from a given regular grammar without going through Non Deterministic Finite Automata. Also we can extend this idea to regular expression which can be converted to regular grammar by a polynomial time of input size \( n \) (where \( n \) denotes the regular expression length). Also we have proposed a new algorithm which minimizes the number of states of the obtained Deterministic Finite Automata, in which minimization is done with a linear polynomial time which is superior to the existing algorithms. We have proved that a Regular grammar can be converted very easily from a given regular expression which involves two terminals only and it can also be extended for \( k \) alphabets (terminals).

REFERENCES

School Of Computer Science and Engineering, Faculty of Engineering and Technology, S.R.M University, Tamil Nadu, India. Her research interests include Artificial Neural Networks, Image Processing, Pattern Recognition and Signal Processing. She has guided many B.Tech and M.Tech Projects, Guiding Ph.D Scholars and published papers in national and international conferences and journals and attended and organized symposiums and workshops. Email: malathi.d@ktr.srmuniv.ac.in, mala_kam@yahoo.com