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

K.Senthil Kumar[1] D.Malathi[2]

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,$\sum$ denotes input alphabet (finite set of symbols).*Let* $\sum^* = \bigcup_{i=0}^{\infty} \sum^i$ *a,β,γ* represent words and *mod(w)* represent the length of the word. A special word *ε* is used to denote empty word, whose length is assumed to be *0*.Also a language *L* over $\sum$is defined as $L \subseteq \sum^*$.A Deterministic Finite Automata is a five tuple which is defined [8] as a quintuple$A=(Q, \sum, \delta, q_0, F)$, where

- Q - Finite set of states
- $\sum$ - Finite input alphabet
- $\delta : QX\sum \rightarrow Q$ *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 δ which is defined as
$\delta : QX\sum \rightarrow 2^Q$ 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

$A \rightarrow Ba (or)\ A \rightarrow a$

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

$A \rightarrow aB (or)\ A \rightarrow a$

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

[1]*K.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)*

[2]***Corresponding Author :****D.Malathi, Professor, is with the Departmentof Computer Science and Engineering,S.R.M University, TamilNadu, Pin:603203, India. (phone: +919442554055; E-mail:malathi.d@ktr.srmuniv.ac.in, mala_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 ZhongshengQian[4] have discussed the equivalence between Regular grammar and DFA.R.McNaughton andYamada [12] have proposed algorithms which construct DFA for the given regular expression in $O(n^2)$ time and space. AnneBruggemann-Klein [9] showed that the Glushkov automaton can be constructed in a time quadratic in the size of the expression. Rajesh Parekh,CodrinNichitiu andVasanth [2] have described an efficient incremental algorithm for learning regular grammars from labeled examples and membership queries. Chia-Hsiang Chang [3] presented analgorithm 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] *nlogn* 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 $O(nlogn)$.

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 etal.We extend this idea to regular language. We define GOTO( ) and CLOSURE( ) functions for regular language as follows:

Definition: GOTO(Q_i,X) where Q_i 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 Q_i.

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

1. *Every item Qi is added to CLOSURE(Qi)*
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.

4.1Algorithm:

Input : A Regular Grammar *G* describing language *L(Containing k rules)*

//Which we convert it in to 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=*CLOSURE($S' \rightarrow .S$)* is the only state in required *DFA_states* and it is unmarked;

While (there is an unmarked state *Q*in *DFA* _states)

do

begin

Mark Q;

For each a $\varepsilon \sum$ do begin

Q′=GOTO (Q,a)

If Q′ is not in DFA _states then

Add Q′ as an unmarked state to DFA _states;

GOTO(Q,a)=Q′;

End

End

}

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

Compute_CLOSURE(Q)

{

For each item[$A \rightarrow a.B$] belongs to Q

for each production $B \rightarrow x$ or $B \rightarrow xC \varepsilon G'$

If $B \rightarrow .x$ or $B \rightarrow .xC$ is not in Q add

$B \rightarrow .x$ or $B \rightarrow .xC$ to Q

Return(Q);

}

Function GOTO(Q,a terminal)

{

Let I be the set of items [$A \rightarrow x.B$]

Such that A →.xBεQ

Return CLOSURE(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 $GOTO(Q_0,a)=Q_j$ 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 GOTO*(Qi,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* $a\varepsilon\sum$

4.3Time Complexity Computations

In the algorithm mainly we use three functions.

1. *Compute_CLOSURE(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. *GOTO(Q,terminal)*

   This function also scans all the items of *Q* to find the corresponding terminal and then it applies CLOSURE 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=CLOSURE(S'→.S)

Therefore

Q_0:
S'→.S
S→.aS
S→.bS
S→.aA

$GOTO(Q_0,a)= CLOSURE(\{(S\rightarrow a.S)(S\rightarrow a.A)\})$

Q_1:
$S\rightarrow a.S$
$S\rightarrow a.A$
$S\rightarrow .aS$
$S\rightarrow .bS$
$S\rightarrow .aA$
$A\rightarrow .bB$

$GOTO(Q_0,b)= CLOSURE(S\rightarrow b.S)$

Q_2:
$S\rightarrow b.S$
$S\rightarrow .aS$
$S\rightarrow .bS$
$S\rightarrow .aA$

$GOTO(Q_1,a)=CLOSURE(\{(S\rightarrow a.S)(S\rightarrow a.)\})$
$= Q_1$

$GOTO(Q_1,b)=CLOSURE(\{(S\rightarrow b.S)$
$(A\rightarrow b.B)\})$

Q_3: $S\rightarrow b.S$
$A\rightarrow b.B$
$S\rightarrow .aS$
$S\rightarrow .bS$
$S\rightarrow .aA$
$B\rightarrow .b$

$GOTO(Q_2,a)=CLOSURE(\{(S\rightarrow a.S)$
$(S\rightarrow a.A)\})$
$=Q_1$

$GOTO(Q_2,b)=CLOSURE(\{S\rightarrow b.S\})$
$=Q_2$

$GOTO(Q_3,a)= CLOSURE(\{(S\rightarrow a.S)$
$(S\rightarrow a.A)\})$
$= Q_1$

$GOTO(Q_3,b)= CLOSURE(\{(S\rightarrow b.S)(B\rightarrow b.)\})$

Q_4:
$S\rightarrow b.S$
$B\rightarrow b.$
$S\rightarrow .aS$
$S\rightarrow .bS$
$S\rightarrow .aA$

$GOTO(Q_4,a)= CLOSURE(\{(S\rightarrow a.S)$
$(S\rightarrow a.A)\})$
$= Q_1$

$GOTO(Q_4,b)= CLOSURE(S\rightarrow b.S)$
$=Q_2$

The required DFA is shown in Fig1.

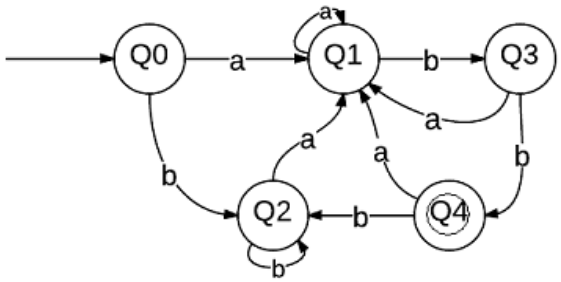

Fig 1: Resultant DFA obtained from proposed algorithm

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*GOTO(P,a)=GOTO(R,a) for all a* $\varepsilon\sum$.

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 $GOTO(Q_i,a)=GOTO(Q_j,a)$ $\forall a \in \sum$. Clearly the relation satisfies

1) *Reflexive property as*
$GOTO(Q_i,a)= GOTO(Q_i,a)$;Hence
$Q_i$is equivalent to Q_i.

2) *Symmetric property as*
$Q_i$is equivalent to $Q_j$iff
$GOTO(Q_i,a)=GOTO(Q_j,a)$ $\forall a \in \sum$
Clearly itcan be written as
$GOTO(Q_j,a)= GOTO(Q_i,a) \forall a \in \sum$
This implies Q_j is equivalent to Q_i

3) *Transitivity property is satisfied as*

Given: $Q_i$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 $Q_j$iff$GOTO(Q_i,a)=GOTO(Q_j,a)$ $\forall a \in \sum$
$Q_j$is equivalent to $Q_k$iff$GOTO(Q_j,a)=GOTO(Q_k,a)$ $\forall a \in \sum$

Therefore, we infer that Q_i is equivalent to Q_k. Therefore,if Q denotes the set of states of the given DFA i.e.$Q=\{Q_1,Q_2,\ldots Q_n\}$// we represent states in an ascending order, then we can write$Q=\bigcup_{i=0}^{k} Ci$ where $C_i \cap C_j=\emptyset$where each C_i must contain at least one state and at most two states.If there are C_n classes then the obtained DFA is already in Minimized formC_i.

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_i,Q_{i+1},Q_{i+2} is the partition at a particular point of time suppose that $Q_i,Q_{i+1}$are equivalent they should be merged and the next partition is$\{Q_{i+2},Q_{i+3},Q_{i+4}\}$.Suppose $Q_i$and$Q_{i+2}$are equivalent then the partition starts from Q_{i+3}

Minimization algorithm:

Input : Given a DFA with '*n*' states $Q_1, Q2,\ldots Q_n$
Output: A Minimized DFA which accepts the same language
Method:

```
Minimize_DFA(Q[1..n])                    //  This
  contains  n states
{
i←1;
      l1:  while(i<n)
  {
      J←i+1;
      while(j<=i+2)
      {
                  if(GOTO(Q[i],a)==GOTO(Q[j],a))&&
                                          (  GO-
TO(Q[i],b)== GOTO(Q[j],b))
                  {
                        merge(Q[i]&Q[j]);
                  i←j+1;
                        goto l1; }
   else
   j←j+1;          }}
   i←i+1;      }
```

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 Fig2.

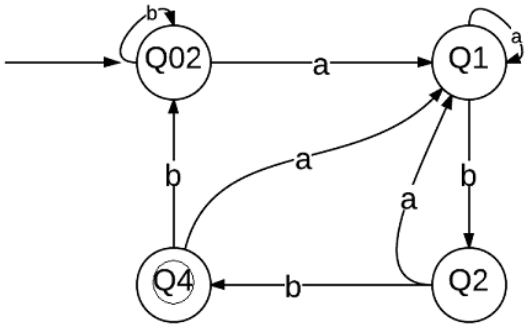

Fig2.Minimum DFA

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 constructed very easily from a given regular expression which involves two terminals only and it can also be extended for *k* alphabets (terminals).

REFERENCES

[1] Aho A. V., Lam M. S., SethiR. and Ullman J. D., "Compilers: Principles, Techniques, and Tools," 2nd Edition, Addison-Wesley, New York, 2007.

[2] Rajesh Parekh,CodrinNichitiu and VasanthHonovar, "A polynomial Time incremental Algorithm for Regular Grammar Inference", in Proceedings of the Third International Colloquium on Grammar Inference (ICGI-96). Berlin: Springer-Verlag. pp. 238-249,1996.

[3] Chia-Hsiang Chang, "From Regular Expressions to DFA's Using Compressed NFA's", A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy Computer Science Department –October1992, Courant Institute of Mathematical Sciences.

[4] Jielan Zhang, ZhongshengQian, "The Equivalent Conversion between Regular Grammar and Finite Automata", Journal of Software Engineering and Applications,Vol. 6 No.1,pp. 33-37,January 2013.

[5] BhargavaS. and Purohit G. N., "Construction of a Minimal Deterministic Finite Automaton from a Regular Expression," International Journal of Computer Applications, Vol. 15, No. 4, pp. 16-27,2011.

[6] Stearns R. E. and Hunt H. B., "On the Equivalence and Containment Problems for Unambiguous Regular Expressions, Regular Grammars and Finite Automata," SIAM Journal on Computing, Vol. 14, No. 3, pp. 598-611,1985.

[7] HopcroftJ. E., MotwaniR. and UllmanJ. D., "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, New York, 2007.

[8] LaurikariV., "NFAs with Tagged Transitions, Their Conversion to Deterministic Automata and Application to Regular Expressions", inProceedings of the 7th International Symposium on String Processing Information Retrieval, IEEE CS Press, New York, pp. 181-187, 2000.

[9] BruggemannKleinA., "Regular Expressions into Finite Automata," Springer link Lecture notes in Computer Science, Vol.583 pp. 87-98, 1992.

[10] Berry G. and Sethi R., "From Regular Expressions to DeterministicAutomata", Theoretical Computer Science, Vol. 48 pp. 117-126, 1986.

[11] Hopcroft and John, "An *nlogn* algorithm for minimizing states in a finite automaton", Theory of Machines and Computations (Proc. Internat.Sympos.,Technion, Haifa, 1971), New York: Academic Press, pp. 189–196, 1971.

[12] McNaughtonR. and Yamada H., "Regular expressions and state graphs for automata", IRE Transactions EC 9, pp.39-47, 1960.

[13] Jean Berstel and Lucand Oliver Carton, "On the complexity of Hopcroft's state minimization algorithm", Implementation and application of automata Lecture notes in Computer science Springer Link Volume 3317,2005,pp 35-44,Fifth Colloquium on Mathematics and Computer Science DMTCS Proc. AI, 2008, 351–362, 2005.

[14] Nerode, "Linear automaton transformations", in Proc. of theAmerican Mathematical Society 9,pp. 541–544, 1958.

[15] Moore Edward F, "Gedanken-experiments on sequential machines", Automata studies, Annals of mathematics studies, no. 34, Princeton, N. J. Princeton University Press, pp. 129–153, 1956.

Senthil Kumar.Kreceived M.Tech in Computer Science and Data Processing fromIIT Kharagpur, India, in 2002, M.Sc Mathematics from IIT Chennai, India in 1993 and B.Sc Mathematics from Madras University, Chennai, India in 1990. Since 2008, He has been workingas Assistant Professor with the Computer Science and Engineering Department, SRM University, Chennai.He is currently pursuing the Ph.D. degree in Computer Science and EngineeringDepartment, SRM University. His research interests include Theoretical Computer science, Network Security, Artificial Intelligence. He has participated in various national and international conferences, symposiums and workshops. Email: senthilkumar.k@ktr.srmuniv.ac.in.

Malathi. D, Ph.D. received A.M.I.E in Electronics and Communication Engineering degree from The Institution of Engineers, India, and Calcutta in 1991,and M.E (CSE) from MadrasUniversity, Chennai, 1998, and Ph.D. Information & Communicationfrom AnnaUniversity, Chennai in 2010. She is a Professor,

School Of Computer Science and Engineering, Faculty of Engineering and Technology, S.R.MUniversity, 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