International Journal of Scientific Research in Computer Science and Engineering **Research Paper**
Vol.6, Issue.4, pp.41-55, August (2018) **E-ISSN:** 2320-7639

A Novel Approach to Minimize DFA State Machines Using Linked List

Yogesh Pant

Dept. of CIS, HSET, SRHU, Dehradun, India

**Corresponding Author: yogepant123@gmail.com*

Available online at: www.isroset.org

Accepted: 20/Aug/2018, Online: 31/Aug/ 2018

***Abstract*—** DFA minimization is a significant problem in algorithm design. Minimization of DFA works on the concept of DFA equivalence: Two DFA's are equivalent if and only if both accept the identical strings of same set. Plethora of computational problems can be solved simply by using the encoding method. Combinatorial objects which are to be used as strings over a finite alphabet can be encoded. Standard algorithms from computational linear algebra are subjected to be used efficiently to solve the computational problems if a DFA recognize the set of encoded strings. This proposed method provides better results whereas other methods may face serious problem of time and space complexities.

Keywords— DFA; NFA; Regular expressions; Compiler Design; Linked List

I. INTRODUCTION

A DFA is defined as a mathematical concept, which can be implemented in software and hardware as well to solve many problems due to the deterministic behavior of a DFA. For instance, a DFA can be used for the implementation of a program that decides whether or not entered email address and password is valid [1].

The set of regular languages is recognized by DFA [3] which are helpful in lexical analysis and pattern matching. DFAs can be constructed from nondeterministic finite automata.

A deterministic finite state automaton (DFA) [11] comprises of a finite set of states which are labeled with alphabets and directed edges between two states. For every state, at most one outgoing edge denoted by a given letter from the alphabet is there. Hence, a transition of a state to another labeled by some letter is deterministic. DFA consists of an initial state and a set of the states called accepting or final state. The DFA accepts a string if the letters of the string decide transitions from the initial state to an accepting state or final state. The set of strings which are accepted by a DFA is called a language.

II. DFA MINIMIZATION

DFA minimization is normally a task that converts a deterministic finite automaton (DFA) into another equivalent DFA and that equivalent DFA has minimum number of states. Various algorithms are used to achieve this task [6].

There will be a minimal automaton i.e. a DFA which has a minimum number of states for each and every language that is regular in nature and this DFA is always unique. The minimal DFA makes sure that the computational cost for tasks such as pattern matching is minimal.

There exist two classes of states that we can remove / merge from the original DFA to reduce it without altering the language it accepts .The two states are as follows:

A. UNREACHABLE STATES
For any input string, these are the states which are not accessible or reachable from the initial state of the DFA.

B. NON-DISTINGUISHABLE STATES
For any input string, these are those states that cannot be differentiated from one another.

DFA minimization is basically implemented in three steps, subsequent to the removal/merger of the related states. Since the removal of non-distinguishable states is the most expensive one on the basis of computations, hence it is normally done as the final step. Unreachable states can be simply removed from the DFA without altering the language that it accepts.

III. APPLICATIONS

Deterministic Finite Automata (DFA) and particularly its minimization algorithms possess some applications as listed below:

- Path finding DFA in AI.
- Parser Generator for DSL and other Languages.
- DNA analysis.
- Implementation of Regular Expression and their Search.
- Natural and computer virus scanning.
- Virus Searching.
- Code parser.
- Processing XML Streams[4]
- Optimizing linked list techniques[5]
- Word/code completion

IV. LITERATURE SURVEY

A. Different DFAS AND DFA MINIMIZATION ALGORITHMS AND TECHNIQUES:

A number of algorithms for the construction of minimal acyclic deterministic finite automata (MADFAs) are shown in paper [2]; most of which are originally derived / designed.

These automata work on finite languages and have confirmed helpful in functions such as spell checking, text indexing and virus searching, just because of their acyclic nature. The automata rise up to billions of states in many scenarios which makes their storage complicated without using several compression techniques.

The most significant technique is minimization technique. Previous results confirm that minimization produces a unique automaton (for a given language), subsequent results show that minimization of acyclic automata could be possible in linear time in the number of states. These two results are functional for a rich area of algorithmic research.

The paper [2] depicts both incremental and non-incremental algorithms. The un-minimized acyclic deterministic finite automaton (ADFA) is made first and after that it is minimized in non-incremental techniques,.

The un-minimized ADFA can be quite large indeed; even if it is too large to accommodate within the computer's virtual memory space. As a result, an incremental method for minimization i.e. the ADFA is minimized throughout its construction. Incremental algorithms often have a few overhead, if the un-minimized ADFA effortlessly fits in physical memory, then it may still be faster to implement non-incremental techniques than incremental techniques.

B. The TABLE DRIVEN-DFAs

Paper [1] shows examination of Table Driven DFA (TD DFA) based string processing algorithms from a variety of vantage points. A number of strategies are well-known to execute such algorithms in a cache efficient manner.

It is shown in paper [1] that the connotation semantics of Table Driven algorithms are wrapped in a function. The number of arguments of this function is associated with each implementation approach. These implementation approaches recommend twelve different algorithms, each coming together the implementation techniques in a specific way.

The paper [1] depicts three implementation strategies linked with the table driven algorithm in order to minimize the whole latency of a recognizer. In each case, the revised algorithm performs better than the TD algorithm for an appropriate class of input strings.

The primary strategy, known as the dynamic state allocation (DSA) approach, has already been recommended. This strategy has proven to perform better than TD when a large scale finite automaton is used to process and recognize very large strings expected to repeatedly visit the similar set of states.

The second strategy, known as the State pre-ordering (SPO) strategy, depends on a degree of earlier knowledge about the pattern in which states are likely to be visited at runtime. It is already proven that the linked algorithm performs better than its TD counterpart no matter what type of string is being processed.

The paper [1] also examined several ways to enhance the performance of the traditional TD algorithm by using several implementation strategies. A 6-argument function offers the connotation semantics of several TDFA based string recognizers. Another instantiations of these arguments correspond to new TDFA based algorithms. The algorithms were thereafter implemented and their performance got recorded.

The algorithms were examined on artificially produced data (strings and automata). Hence in accordance to the upcoming work, several experiments will be carried out on real life data such as genetic sequences, microsatellites for network intrusion detection and tandem repeat detection.

C. DFA in REGULAR EXPRESSION MATCHING

Paper [3] depicts that there is a huge requirement in recent network devices to carry out deep packet examination at high speed for security and function specific services. Finite Automata (FAs) are used for executing regular expressions matching in their work; however, they require a large amount of memory. To tackle this issue several recent works has been proposed.

The paper [3] explains a new demonstration for deterministic automata (orthogonal to earlier solutions), called Delta Finite Automata, which extensively reduces states and transitions. It needs a transition per character only in order to allow it for fast matching. Furthermore, a new state encoding method is also proposed and the complete algorithm is then tested for use in the packet classification area. Several imperative services in current networks are based on payload examination along with headers processing. Prevention Systems, Intrusion Detection and traffic monitoring and layer filtering requires an exact study of packet content in search of matching of predefined set of data patterns.

This kind of patterns illustrates particular classes of applications, regularly updated viruses or protocol definitions. Traditionally, the datasets consists of a variety of signatures to be searched along with the assist of string matching algorithms, but nowadays, regular expressions are being used because of their improved expressiveness and ability to explain a broad variety of payload signatures.

They are accepted by familiar tools, like Snort and Bro, also in firewalls and devices utilized by various vendors like Cisco etc.

Normally, finite automata are implemented for regular expression matching. NFAs are illustrations which need more state transitions per character, therefore having a time complexity of O(m) for lookup, where 'm' is the quantity of states in the NFA. Alternatively, NFAs are very space efficient structures.

Deterministic FAs (DFAs) involve only one state transition per character, however, the current regular expression sets involves a great amount of memory. Because of these reasons, this kind of solutions are not appropriate for execution in real deep packet inspection devices, that requires to carry out on line packet processing at a very high speed. Consequently, various works have been just carried out with the objective of memory minimization for DFAs, by utilizing the imperative redundancy in regular expression sets [9],[10].

D. PATTERN MATCHING

Pattern matching [7], [8] seems a very complicated task in numerous network services such as intrusion detection. A DFA is however a simple language recognition device, also it can be viewed as machine that recognizes a given input strings. The major setback of string matching seems to be the area efficiency and memory optimization. Minimized DFA for pattern matching decreases the memory area requirement and also eases in optimization of area.

1) State minimization Algorithm [7]:

Algorithm for reducing Number of States in DFA:

- Eradicate the non-reachable state.
- Create a group of all non-final states as identical.
- Create a group of all final states as identical.
- Reiterate until no more states are differentiable.
- Apply symbol to a group, then split group if states are differentiable.

A state s1∈ Q is said to be inaccessible or unreachable if There exists no string w in Σ* such that δ(s, w) =s1 (s1 ∉ (s2|w ∈ Σ*, δ (s1, w) =s2})

Two states s1 and s2 are indistinguishable if for all w ∈Σ* δ (s2, w) ∈ F) ⇒ δ*(s1,w) ∈ F

δ* (s1,w) ∉ F) ⇒ (s2,w) ∉ F

E. Proposed methodology

In this proposed method, the minimization of states of DFA is achieved by using a tree structure. The concept of partition algorithm will be used in the root node to split the set of all states of given DFA into two partitions. One partition will hold the set of final states and other partition will hold the set of all other states i.e. Non-final states. Subsequently, the child nodes of final and non-final states node will hold the input character of the DFA machine. Now since the terminals will hold multiple states then the linked list will be used for the leave nodes or the terminals of the tree. The transition of one state to another after taking an input character from a string is a major step in DFA; hence the inorder traversal of terminal nodes will determine the transition of one state to another.

Algorithm:

Step 1: The set of all states of DFA will be at the root node of the tree.

Step 2: Split the states into subset of final and non-final states.

Step 3: Create the child nodes of root node as the set of final and non –final state.

Step 4: Create the child node of final and non-final states node as the input characters to the DFA as shown in fig 1.

Step 5: Create the terminals as the multivalued linked list to hold the states taking the character from parent node as an input.

Step 6: The inorder traversal of the terminal nodes will provide the transition of the DFA.

Step 7: States with same input and same output will be group together.

Step 8: Repeat step 6 and step 7 until all the groups or sets have indistinguishable states.

Step 9: Mark the set of indistinguishable states as a new symbol.

Step 10: Modify the terminal nodes with new symbol and a minimized DFA will be formed.

F. MINIMAL DFA

Finite state automata (FSA)[11] are being used ubiquitously in computer science. There are generally two most significant algorithms for FSA processing. One is the translation of a non- deterministic finite automaton (NFA) to a deterministic finite automaton (DFA), and the other one is the generation of the exclusive minimal DFA for the original NFA. There is also a parallel disk-based technique [11] which employs a cluster of 29 commodity workstations to generate a transitional DFA with approximately two billion states and afterward continues by generating the corresponding unique minimal deterministic finite automata (DFA) with less than 800,000 states. This paper [11] depicts the requirement for efficient and scalable algorithms for finite state automata (FSA), by concerning that they are basically the most computationally tractable structure in which to examine the regular languages. This study involves efficient algorithms for both translation of NFA to DFA and minimization of DFA.

TABLE I: PERFORMANCE COMPARISON OF DFA AND MINIMIZED DFA.

Analysis	**DFA**	**Minimized DFA**
Min clock period	15.50ns	9 ns
Max clock frequency	64.15Mz	111.11Mz
Clock to setup	15.50 ns	9 ns
CPU time to completion for same string	4.39sec	3.10 sec
Memory Usage for same string	17236 KB	17186 KB

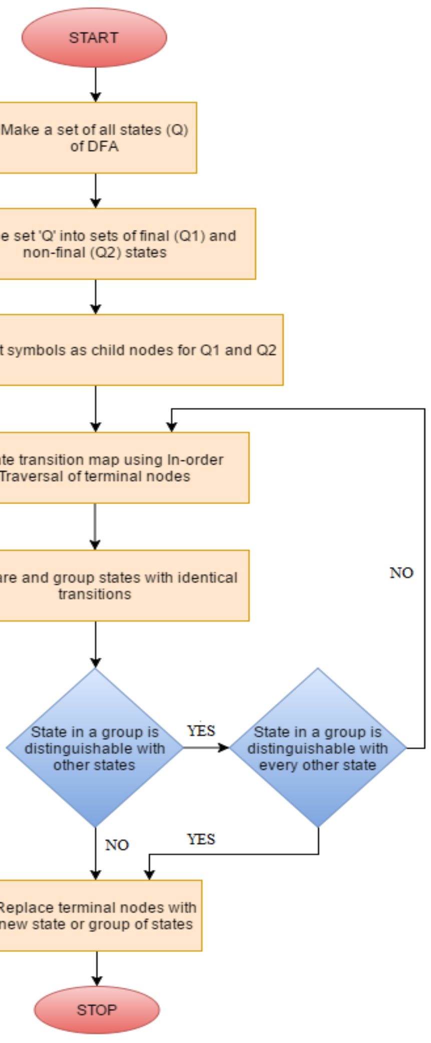
START
Make a set of all states (Q) of DFA
Split the set 'Q' into sets of final (Q1) and non-final (Q2) states
Create input symbols as child nodes for Q1 and Q2
Create transition map using In-order Traversal of terminal nodes
Compare and group states with identical transitions
State in a group is distinguishable with other states
YES
State in a group is distinguishable with every other state
NO
NO
YES
Replace terminal nodes with new state or group of states
STOP

Figure.1 Flowchart of the algorithm described in section E.

G. Performance of DFA-based String Processors

In paper [1], the performances of a few of the several algorithms investigated earlier are presented. To test a string processing algorithm against various data related to definite problem domains such as computer virus scanning, network intrusion detection systems, spell checking, DNA analysis, etc. can be potential strategy to calculate their performance.

The major intention of this work was to examine implementation techniques for DFA based string processing, resulting not just to the creation of a taxonomy graph, but also for the design of a toolkit to implement for string matching purpose.

Establishing these conditions would need that a comprehensive and methodical series of benchmarking tests to be carried out on the several algorithms.

Such an examination comprises a whole and sound research theme on its own and cannot be carried out in the framework of this current work.

This paper enlightens the numerous experiments implemented on artificially produced data and also presents methods used for the measurement of performance, data collection and hardware and software support structure as well on which we depend to conduct the experiments.

H. *OPTIMAL Linked list*

Optimal linked list [5] schemes minimize the total number of memory access which is required to create and access a linked list. With optimal linked list and generating function, we can associate the states of DFA to a linked list. With a multilevel linked list scheme we can accomplish optimal performance and can access huge amount of data at a high speed as well.

V. CONCLUSION

Every finite state machine has to follow a "path", which is basically a string at a given time, in order that adding a sentence or a word gets recognized by the machine automatically. One of the major difficulties is minimizing a NFA to a DFA; Minimization of NFA to DFA can be done using DFA Minimization algorithms. Minimized DFA is implemented for the purpose of pattern matching which enhance the various results like minimized area, enhanced performance, and minimum number of resources. The general DFA may call for up to 2n states, on other hand, the equivalent minimized DFA require only 'n' states. Amount of resources also deducted up to 40%.The minimized DFA is quite efficient than the original DFA. This paper presents algorithms and methods used in transforming NFA to minimal DFA.

VI. FUTURE WORK

Numerous algorithms have been designed over the years for the minimization of DFA. Nearly all of them utilize the arrays for storage of patterns. Although the use of arrays significantly simplifies the implementation process, linear arrays underperform in comparison to further generic data structures for instance, doubly linked list. Relying on the performance in searching the linked list can be used to significantly improve the running time of a DFA.

Since in arrays, searches are very much slower than a linked list, but a linked list is incredibly fast for the same purpose. If we create a DFA using linked list scheme we can accomplish optimal performance and can access huge amount of data comparatively at a very high speed. In imminent future, we would like to design a modified algorithm for the minimization of DFAs.

References

[1] Ernest ketcha am, derrick g. kourie, and brucngasse w. watson,"*On Implementation and Performance of Table-Driven DFA-Based String Processors*". Int. J. Found. Comput. Sci. 19, 53 (2008).
[2] Bruce William Watson, "*Constructing Minimal Acyclic Deterministic Finite Automata*". FASTAR Research Group.
[3] Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci, Gianni Antichi, and Andrea Di Pietro.. "An improved DFA for fast regular expression matching", SIGCOMM Comput. Commun. Rev. 38, September 2008.
[4] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. "*Processing XML streams with*

deterministic automata and stream indexes", ACM Trans. Database Syst. 29, 4 (December 2004)

[5] Isaac Keslassy, David Hay, Yossi Kanizo, Isaac Keslassy, David Hay, Yossi Kanizo. "*Optimal Fast Hashing*", Technical Report Tr08-05, Comnet, Technion, Israel.

[6] Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D. (1974), 4.13 Partitioning, "*The Design and Analysis of Computer Algorithms*", Addison-Wesley, pp. 157–162.

[7] Aakanksha Pandey, Dr. Nilay Khare and Akhtar Rasool" *Efficient Design and Implementation of DFA Based Pattern Matching on Hardware*", IJCSI March 2012.

[8] B. L. Hutchings and R. Franklin and D. Carver "*Scalable Hardware Implementation on Finite Automata*" Department of Electrical and Computer Engineering

[9] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner . " *Algorithms to accelerate multiple regular expressions matching for deep packet inspection*", In Proc. of SIGCOMM '06, pages 339-350. ACM.

[10] R. Smith, C. Estan, and S. Jha. Xfa, "*Faster signature matching with extended automata*", In IEEE Symposium on Security and Privacy, May 2008.

[11] Vlad Slavici, Daniel Kunkle, Gene Cooperman and Stephen Linton, "*Finding the Minimal DFA of Very Large Finite State Automata with an Application to Token Passing Networks*" Northeastern University, 29 March 2011.

Authors Profile

Mr. Yogesh Pant pursed Diploma, B. Tech. and M.Tech, from GPD Dehradun, UTU and GBPEC respectively in 2010, 2013 & 2016. He is currently working as Assistant Professor in Department of Computer information and Sciences from HSET, SRHU since 2017. His main research work focuses on wireless sensor network, Artificial intelligence and quantum computing. He has 3 years of teaching experience.