A REVIEW of DFA MINIMIZING STATE MACHINES USING HASH-TABLES

Vishal Garg¹, Anu²

¹Department of Computer Science & Engineering, JMI University, Kurukshetra University, Yamuna Nagar, Haryana, India.
²Department of Computer Science & Engineering, JMI University, Kurukshetra University, Yamuna Nagar, Haryana, India

Abstract—In algorithm design, DFA minimization is an important problem. DFA minimization is based on the notion of DFA equivalence: Two DFA's are called equivalent DFA's if and only if they accept the same strings of same set. A large number of computational problems can be solved easily by the encoding method. We can encode the combinatorial objects which we want to use as strings over a finite alphabet. If a DFA accept the collection of encoded strings, then standard algorithms from computational linear algebra can be efficiently used to solve the computational problems.

Keywords—Deterministic Finite Automata, NFA, Regular expressions, Compiler Design, Generic Programming, Performance.

I. INTRODUCTION

A DFA can be defined as an abstract mathematical concept, but due to the deterministic nature of a DFA, can be implemented in hardware and software to solve a number of specific problems. For example, a DFA is used for the implementation of software that decides whether or not online user-input such as email addresses are valid [1].

DFAs recognize exactly the set of regular languages [3] which are, among other things, useful for doing lexical analysis and pattern matching. DFAs can be built from nondeterministic finite automata through the power-set construction.

A deterministic finite state automata(DFA) [11] consists of a finite set of states and these states are labeled with alphabets and also having directed edges between pairs of states. For each state, there is at most one outgoing edge labelled by a given letter from the alphabet. So, a transition from a state to another state dictated by a given letter is deterministic. There is an initial state and also certain of the states are called accepting or final state. The DFA accepts a word if the letters of the word determine transitions from the initial state to an accepting state or final state. The set of words which are accepted by a DFA is called a language.

II. DFA MINIMIZATION

DFA minimization is basically a task which converts a given deterministic finite automaton (DFA) into an equivalent DFA. That equivalent DFA has minimum number of states. Two DFAs are called equivalent if and only if they recognize the same regular languages. Several different algorithms are used to accomplish this task [6].

For each and every regular language that can be accepted by a DFA, there will be a minimal automaton i.e. a DFA which has a minimum number of states and this DFA is unique. The minimal DFA ensures minimal computational cost for tasks such as pattern matching.

There are two classes of states that can be removed / merged from the original DFA to minimize it without affecting the language it accepts. These two states are:

A. UNREACHABLE STATES
These are those states that are not reachable from the initial state of the DFA, for any input string.

B. NON-DISTINGUISHABLE STATES
These are those states that cannot be distinguished from one another for any input string.

DFA minimization is usually done in three steps, corresponding to the removal/merger of the relevant states. Since the elimination of non-distinguishable states is computationally the most expensive one. So it is usually done as the last step. Unreachable states can be easily removed from the DFA without affecting the language that it accepts.

III. APPLICATIONS

Deterministic Finite Automata (DFA) and especially its minimization algorithms have various applications as listed below:

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

IV. LITERATURE SURVEY

A. VARIOUS DFAS AND DFA MINIMIZATION ALGORITHMS AND TECHNIQUES:

The paper [2] shows a number of algorithms to construct the minimal acyclic deterministic finite automata (MADFAs), most of these algorithms are originally derived/designer or co-discovered.

Because of acyclic nature, these automata show finite languages and have proven helpful in applications such as spell checking, text indexing and virus searching. The automata raise up to billions of states in many of these applications which makes their storage difficult without using various compression techniques.

The most important technique is minimization technique. Previous Results show that minimization yields a unique automaton (for a given language), afterward results show that minimization of acyclic automata is possible in time linear in the number of states. These two results are useful for a rich area of algorithmic research.

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

The un-minimized ADFA can be very large indeed, even it is too large to fit within the computer’s virtual memory space. As a consequence, incremental techniques for minimization i.e. the ADFA is minimized during its construction become interesting. Incremental algorithms often have some overhead, if the un-minimized ADFA easily fits in physical memory, it may still be faster to use non-incremental techniques than incremental techniques.

B. THE TABLE DRIVEN-DFA

In the paper [1] Table driven DFA (TD DFA) based string processing algorithms are examined from a number of vantage points. A variety of strategies to implement such algorithms in a cache efficient manner are well-known.

The paper [1] shows that the denotation semantics of Table Driven algorithms are encapsulated in a function. The various arguments of this function are linked with each implementation strategy. These implementation strategies suggest twelve different algorithms, each blending together the implementation strategies in a particular way.

The paper [1] presents three implementation strategies associated with the table driven algorithm to minimize the overall latency of a recognizer. In every case, the revised algorithm out performs the TD algorithm for a suitable class of input strings.

The first strategy, called as the dynamic state allocation (DSA) strategy, has already been suggested. This strategy has proven to outperform TD when a large scale finite automaton is used to recognize very large strings likely to repeatedly visit the same set of states.

The second strategy, called as the State pre-ordering (SPO) strategy, relies on a degree of former knowledge about the order in which states are expected to be visited at runtime. It is proven that the associated algorithm out performs its TD counterpart no matter which type of string being processed.

The paper [1] also investigated various ways to improve the performance of the conventional TD algorithm using various implementation strategies. A 6-argument function provides the denotation semantics of various TDFA based string recognizers. Alternative instantiations of these arguments represented new TDFA based algorithms. The algorithms were then implemented and their performance got recorded.

The paper [1] represents that, based on the strings made of long repeated sequences, some of the algorithms outperformed the traditional TD algorithm, but the others were not of interest due to the correctness of the kind of string considered and the policy that is used for replacement.

The algorithms were tested on artificially generated data (strings and automata). Thus in accordance to the future work, various experiments will be conducted on real life data like genetic sequences, microsatellites for tandem repeat detection and network intrusion detection.

C. DFA IN REGULAR EXPRESSION MATCHING

Paper [3] shows that there is a high requirement in modern network devices to perform deep packet inspection at high speed for security and application specific services. Finite Automata (FAs) are used for implementing regular expressions matching in their work, but they need a huge amount of memory. To address this issue various recent works have proposed improvements.

The paper [3] describes a new representation for deterministic automata (orthogonal to previous solutions), called Delta Finite Automata, which significantly reduces states and transitions. It requires a transition per character only, so it allows fast matching. Moreover, a new state encoding scheme is also proposed and the complete algorithm is tested for use in the packet classification area.

Numerous important services in present networks are based on payload inspection, in addition to headers processing. Intrusion Detection, Prevention Systems as well as traffic monitoring and layer filtering need an exact analysis of packet content in search of matching with a predefined set of data patterns.

This type of patterns characterizes specific classes of applications, viruses or protocol definitions, and constantly updated. Conventionally, the datasets were composed of a number of signatures to be searched with the help of string matching algorithms, but these days’ regular expressions are
used because of their increased expressiveness and their ability to explain an extensive variety of payload signatures. They are adopted by well known tools, like Snort and Bro, and in firewalls and devices used by different vendors such as Cisco.

Usually, finite automata are employed to implement regular expression matching. Nondeterministic FAs (NFAs) are representations which require more state transitions per character, thus having a time complexity for lookup of $O(m)$ where $m$ is the number of states in the NFA. On the other hand, NFAs are very space efficient structures.

Instead of Deterministic FAs (DFAs) require only one state transitions per character, but for the present regular expression sets they requires an extreme amount of memory. For these reasons, this type of solutions do not seems to be appropriate for implementation in real deep packet inspection devices, which require to perform on line packet processing at high speeds. Therefore, several works have been just presented with the goal of memory reduction for DFAs, by exploiting the essential redundancy in regular expression sets [9],[10].

D. PATTERN MATCHING

Pattern matching [7], [8] is a very difficult task in various network services like intrusion detection. A DFA is a simple language recognition device or it can be seen as a machine which recognizes the given input strings. The main problem of string matching is area efficiency and memory optimization. Minimized DFA for pattern matching reduces the memory area requirement and also helps in optimization of area.

1) State minimization Algorithm [7]:

Algorithm for Minimizing Number of States in DFA:

- Remove the not reachable state.
- Make a group of all non-final states as indistinguishable.
- Make a group all final states as indistinguishable.
- Repeat till no more states are distinguishable.
- Apply symbol to a group and split group if states are distinguishable.

A state $s \in Q$ is said to be inaccessible or unreachable if there exists no string $w$ in $\Sigma^*$ such that $\delta(s, w) =s1$ ($s1 \notin (s2 \in \Sigma^*, \delta(s1, w)=s2)$). Two states $s1$ and $s2$ are indistinguishable if for all $w \in \Sigma^* \delta(s2, w) \in F \implies \delta^*(s1, w) \in F$ $\delta^*(s1, w) \notin F \implies (s2, w) \notin F$

E. MINIMAL DFA

Finite state automata (FSA)[11] are used everywhere in computer science. There are two most important algorithms for FSA processing. First one is the conversion of a non-deterministic finite automaton (NFA) to a deterministic finite automaton (DFA), and second one is the production of the unique minimal DFA for the original NFA. There is a parallel disk-based algorithm [11] that uses a cluster of 29 commodity computers to produce an intermediate DFA with almost two billion states and then continues by producing the corresponding unique minimal DFA with less than 800,000 states. This paper [11] presents the need for efficient, scalable algorithms for finite state automata (FSA), by noting that they are usually the most computationally tractable form in which to analyze the regular languages. This study requires efficient algorithms both for conversion of NFA to DFA and minimization of DFA.

F. PERFORMANCE OF DFA-BASED STRING PROCESSORS

Paper [11] represents the performances of some of the various algorithms investigated previously. A promising approach to evaluate the performance of a string processing algorithm would be to test it against several data pertaining to well defined problem domains such as network intrusion detection systems, DNA analysis, natural and computer virus scanning, Spell checking, etc.

The main purpose of this work was to investigate implementation strategies for DFA based string processing, leading not only to the building of a taxonomy graph, but also

### TABLE I: PERFORMANCE COMPARISON OF DFA AND MINIMIZED DFA

<table>
<thead>
<tr>
<th>Analysis</th>
<th>DFA</th>
<th>Minimized DFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min clock period</td>
<td>15.50 ns</td>
<td>9 ns</td>
</tr>
<tr>
<td>Max clock frequency</td>
<td>64.15Mz</td>
<td>111.11Mz</td>
</tr>
<tr>
<td>Clock to setup</td>
<td>15.50 ns</td>
<td>9 ns</td>
</tr>
<tr>
<td>CPU time to completion for same string</td>
<td>4.39 sec</td>
<td>3.10 sec</td>
</tr>
<tr>
<td>Memory Usage for same string</td>
<td>17236 KB</td>
<td>17186 KB</td>
</tr>
</tbody>
</table>
to the design of a toolkit that would be shortly implemented for string matching purpose.

This approach laid down the basis for constructing a generalized taxonomy, and a toolkit. However, while this work has established the basis for having a reservoir of potentially useful algorithms, it has not paid very much attention to the conditions under which a toolkit user should opt to make use of a particular type of string processing algorithm.

To establish these conditions would require that a thorough and methodical sequence of benchmarking tests be carried out on the various algorithms. Such an investigation constitutes a complete and sound research theme on its own right and cannot be carried out in the context of this present work.

This work, discuss the various experiments conducted on artificially generated data. And presents approach used for performance measurement and data collection as well as the hardware and software support structure on which we relied to conduct the experiments.

G. **OPTIMAL FAST HASHING**

Optimal hashing [5] schemes minimize the total number of memory access which are required to construct and access an hash table. With a multilevel hash table scheme we can achieve optimal performance and also can access large amount of data at a very high speed.

V. CONCLUSION

Any State machine has to take a "path", which is usually a string, at a time, so that adding a sentence or a word automatically gets accepted in the machine. One of the most difficult things to do is minimizing a NFA to a DFA; Minimization of NFA to DFA can be done using DFA Minimization algorithms. Minimized DFA is implemented for pattern matching which improves the several results like reduced area, better Performance, less number of resources. The general DFA may require up to 2n states but the equivalent minimized DFA require n states. Number of resources also reduced up to 40%. The minimized DFA is very much efficient than the original DFA.

This review work presents algorithms and techniques used in minimizing NFA to DFA. Some of the techniques are old as (1950’s) but many are still being realized recently in the research community.

VI. FUTURE WORK

Various algorithms have been developed over the years for the minimization of a NFA to DFA. Almost all use the simple arrays for storage of patterns. The Use for arrays although Greatly Simplifies the implementation process, However, Linear Arrays fade in performance comparison to more generic data structures for instance, Hash-Tables. Because of their Performance in Searching the Hash-Tables can be used to greatly improve the running time of a DFA.

There exists an implementation possibility of DFA State machine using Hash-tables. As it is well known that in Arrays Searches are many times slower than a Hash-Table, but a Hash-Table is incredibly fast for same purpose. If we construct a DFA using hash table scheme we can achieve optimal performance and also can access large amount of data at a very high speed. In near future, we would like to implement such an algorithm and access the Space/time tradeoffs of the same.

REFERENCES


[8] B. L. Hutchings and R. Franklin and D. Carver “Scalable Hardware Implementation usonf Finite Automata’Departmen of Electrical and Computer Engineering

