Algorithm Design for Deterministic Finite Automata for a Given Regular Language with Prefix Strings

Rashandeep Singh\(^1\) and Dr. Gulshan Goyal\(^2\)

\(^1\)Chandigarh College of Engineering and Technology, Chandigarh, India
\(^2\)rashandeepsingh@gmail.com, gulshangoyal@ccet.ac.in

Abstract. Computer Science and Engineering have given us the field of automata theory, one of the largest areas that is concerned with the efficiency of an algorithm in solving a problem on a computational model. Various classes of formal languages are represented using Chomsky hierarchy. These languages are described as a set of specific strings over a given alphabet and can be described using state or transition diagrams. The state/transition diagram for regular languages is called a finite automaton which is used in compiler design for recognition of tokens. Other applications of finite automata include pattern matching, speech and text processing, CPU machine operations, etc. The construction of finite automata is a complicated and challenging process as there is no fixed mathematical approach that exists for designing Deterministic Finite Automata (DFA) and handling the validations for acceptance or rejection of strings. Consequently, it is difficult to represent the DFA’s transition table and graph. Novel learners in the field of theoretical computer science often feel difficulty in designing of DFA. The present paper proposes an algorithm for designing of deterministic finite automata (DFA) for a regular language with a given prefix. The proposed method further aims to simplify the lexical analysis process of compiler design.

Keywords: Automata, Deterministic Finite Automata, Formal language, Prefix strings, Regular language.

1 Introduction

Languages are means of communication and can be natural or formal. Examples of natural languages include English, Hindi, and Punjabi, etc. which have a predefined fixed set of alphabets. However, a formal language can be described as a set of strings over a given alphabet [1]. For example, binary language helps in communicating with computers. The alphabet of binary language includes two input symbols, namely 0 and 1.

According to the Chomsky hierarchy [2], [3], [4], formal languages are further divided into the following types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Language (Grammar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Regular Language</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free Language</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive Language</td>
</tr>
<tr>
<td>0</td>
<td>Recursive and Recursively Enumerable Language</td>
</tr>
</tbody>
</table>

Out of these formal languages, regular languages form an important part of the lexical analysis phase of compiler design. The lexical analysis phase of compiler design deals with scanning a source program and separating the program units into logical categories called tokens. The tokens are described using regular expressions. Further, the tokens can be recognized using finite automata [5]. Therefore, it becomes important to study about regular languages. The regular language can be represented using a finite state machine also called finite automata [6], [7]. The automata can be in only one state at a time and the input system results in transition from current to next state [8].

Some basic terms used in automata theory are:
1. Alphabet ($\Sigma$), is a set of non-empty and finite input symbols. e.g. $\Sigma = \{0, 1\}$ represents binary alphabet consisting of 0 and 1.
2. Strings (w), is a finite arrangement of input symbols selected from the alphabet $\Sigma$, normally denoted by w, x, y, z. e.g. if $\Sigma = \{0, 1\}$ then 1011 and 111 are example strings.
3. $\Sigma^*$ is set of all strings over an alphabet $\Sigma$. e.g. if $\Sigma = \{0, 1\}$ then $\Sigma^*\{\lambda, 0, 1, 00, 10, 11, \ldots\}$
4. Languages (L), is a set of specific strings selected from $\Sigma^*$. Mathematically L is a subset of $\Sigma^*$.

It is possible to represent a formal language using a finite state machine [9]. For instance, a regular language that is described using mathematical expressions called regular expressions can be recognized by finite automata [7]. Finite automata can be either deterministic or non-deterministic in nature. In deterministic finite automata, each state and input symbol undergoes exactly one transition [8]. In non-deterministic finite automata, however, each state and input symbol may undergo zero, one or more transitions [10]. State/Transition diagram or table is used to represent a DFA. Finite automata help in string recognition or rejection, i.e. if by the end of the input string, if the current position is the final state then the string is accepted otherwise it gets rejected [11].
There is the extreme importance of DFA in numerous applications including pattern matching, video games, text processing, speech processing, real-life mechanisms such as elevators, traffic lights, and token recognition in compiler design. Also, a given input in 0 and 1 format is processed in CPU to generate the output in the same format which is further converted in user understandable format. Therefore, the CPU performs machine operations internally and automata is used to design such machines. However, the difficulty is faced by novel learners in this field to design DFA due to its complex nature.

The JFLAP tool provides open-source free software for design of machines such as finite state machines, PDA, and Turing machines. However, there must be a mechanism in order to define and present the transitions. DFA is used to solve numerous problems because it is implementable in both software and hardware due to its deterministic nature. There is no well-defined algorithm for its design. Therefore, an algorithm is required to help in the automatic generation of DFA. This paper focuses on designing an algorithm for a regular language with a prefix string. A description of DFA is presented in the next section.

2 Deterministic Finite Automata

Deterministic finite automata can be defined as a finite state machine which allows exactly one transition for each state and input symbol [12]. Finite Automata (M) is mathematically stated as a 5-tuple set as described in Table 2 [13], [14].

\[
M = (Q, \Sigma, \delta, q_0, F \text{ or } A)
\]

Where,

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Finite set of states</td>
</tr>
<tr>
<td>(\Sigma)</td>
<td>Finite set of input symbols</td>
</tr>
<tr>
<td>(\delta)</td>
<td>Transition function. Mathematically, (Q \times \Sigma \rightarrow Q)</td>
</tr>
<tr>
<td>(q_0)</td>
<td>Initial or start state and (q_0 \in Q)</td>
</tr>
<tr>
<td>F or A</td>
<td>Set of accepting/final states F</td>
</tr>
</tbody>
</table>

Representational descriptions of an initial and final state are shown in Figure 1 and Figure 2 respectively.

![Fig. 1. Representational Description of Initial State](image)

![Fig. 2. Representational Description of Final State](image)

DFA accepts a string \(w\) if starting at state \(q_0\), the DFA ends at an accept state (F) on reading the string \(w\) [15]. The DFA accepts a language \(L\) if every string in \(L\) is recognized by \(M\) and is denoted as \(L(M)\), which is pronounced as “\(M\) recognizes \(L\)”.

Transitions occurring from one state to another can be defined by the transition function as shown in Figure 3, \(\delta: Q \times \Sigma \rightarrow Q\)

\[
\delta(a, 0) \rightarrow b \; \text{where } a, b \in Q \text{ and } 0 \in \Sigma
\]

![Fig. 3. Representational Description of Transition Function](image)

Transitions occurring from one state to another can be defined by the transition function as shown in Figure 3, \(\delta: Q \times \Sigma \rightarrow Q\)

As shown in Figure 4, \(q_0, q_1, q_2, \text{ and } q_3\) represent states in a graph. \(q_0\) is the initial state from where transitions begin and \(q_3\) is the final state. If after string processing, we are at the final state then the string is accepted, otherwise rejected. \(\{a, b\}\) represents the transition symbols through which we can go from one state to another.

![Fig. 4. An Example Transition Diagram/Graph](image)

The Transition table as shown in Table 3 is a tabular representation of transition function which requires two arguments – current state and input symbol. The output of the transition function is a state. The state marked with an arrow is the start state and the state marked with concentric circles is the final state.

![Table 3. Transition table](image)
2.1 DFA with Prefix

A DFA with a given prefix is described as a string consisting of leading symbols of a regular language. Figure 5 represents a DFA over \( \Sigma = \{a, b\} \) which recognizes strings with the prefix ‘ab’, and Table 4 represents the transition table for the same.

![State/Transition Diagram for Strings Having Prefix ‘ab’](image)

**Table 4. State/Transition Table for Strings Having Prefix ‘ab’**

<table>
<thead>
<tr>
<th>States</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_1)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_2)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_2)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_0)</td>
<td>(q_0)</td>
</tr>
</tbody>
</table>

So, after processing, the string 'ababb' is in the final state \(q_2\), which is recognized by the DFA because starting from the initial state \(q_0\), it is in the final state \(q_2\). Whereas the string 'aabbb' will not be recognized by the DFA as after processing, the string is in the state \(q_0\) which is not a final state.

3 Literature Survey

Theoretical computer science has been a challenging area in research for the past few decades. Alan Turing started the initial research of this field in 1936 when he came up with the idea of the Turing machine, which is a theoretical tool in computer science used for modelling mechanical computation [17]. The concept of finite automata came forward in 1943 during modelling of the human thought process by Warren McCulloch and Walter Pitts [18]. A finite automaton in earlier models consisted of set of transitions and states without any input. In the 1950s, the researchers in [2] proposed the powerful machines namely Mealy and Moore machines in which output was also considered. The Mealy machine assigns the output with every transition, while the Moore machine assigns the output to every DFA state.

The minimization of DFA can be achieved using a novel method defined in [19], having two phases where first phase includes partitioning the state set into many blocks and the second phase is used to refine the state set using a hash table and DFA can also be applied on acyclic or cyclic automata.

DFA has extreme importance in many applications such as it is used in token recognition in compiler design [5]. It is also used in text processing [20] and speech processing [5]. Other DFA applications include pattern matching [11, 13], vending machines [21], and path-finding DFA in AI-driven video games [22].

Also, the field of cryptography has made use of finite automata. finite automata also have their application in the field of cryptography. For encrypting and decrypting processes, various types of finite automata are used, including simple, structural, automata with I/O memory, and automata with pseudo-memory [23].

A Neural Network-based adversarial model is proposed to disclose the sensitive transitions of DFA. Additionally, a substring is used to find the critical patterns of DFA which can be used in cyber-physical systems [24]. Additionally, a neural network-based approach is proposed in [25], which only works efficiently for automata up to four characters and six states.

Recently, applications of finite automata are widely increasing in various fields. A model is designed for leaf detection classification to predict the leaf disease and further proliferating the yield [26]. DFA is also used in abstract protocols like TCP and mechanical procedures such as traffic lights and elevators. A method to automatically track and analyse the activities of players’ movements in basketball games is proposed in [27] using deterministic finite automata. A DFA for referee and player is made which consists of a large number of states.

Researchers have examined the characteristics of DFA and proposed an algorithm to design a DFA over alphabet consisting of ‘a’ and ‘b’ having x number of a’s and y number of b’s is discussed in [1]. Furthermore, an algorithm is defined in [28] that only considers the designing of DFA that accepts ‘N’ base number so that the remainder is X, when ‘N’ is divided by ‘M’.

Based on the literature, it can be concluded that despite the fact that DFA has so many applications, learners have difficulty designing it because a high level of understanding is required [16]. For the given string, no well-defined algorithm exists for generating a transition table. Identified gaps and challenges are as given below:

1. There is no standard approach for the construction of DFA due to which novel learners face difficulty in the field of theoretical computer science.
2. Various tools such as JFLAP are used for designing deterministic finite automata from the available transitions. There is a need for some mechanism to define and present transitions for applying in JFLAP.
3. DFA is useful in conceptualizing and visualizing many emerging applications in real life.

Therefore, there is a need for a simple method for designing DFA. The present paper focuses on the design and implementation of an algorithm that is used to generate a DFA for prefix strings in an easy and timely manner. The proposed algorithm will provide a transition table and transition graph which together as a whole provides a great insight to understand and implement computation models easily. It can work on any number of characters in the prefix string and any number of states.
4 Proposed Algorithm for the Design of DFA for Regular Language given as Prefix

Present section focus on proposing an algorithm for designing of DFA for the language which accepts strings given as prefix. The steps of the proposed approach/method are as shown in Figure 6:

**Algorithm 1: Designing DFA for Regular Language having given Prefix**

*Input:* Input for the algorithm will be set of alphabets and string to be used as prefix.

*Output:* Output for the following algorithm will be transition table of the DFA.

**Initialization:**
1. No. of states = length of string + 1 + the reject state (qΦ)
2. Initialize q0 with acceptance string of length i of given string (i=0, 1, 2...)
3. For each state except final state and dead state
   3.1. Combine the acceptance string with input symbol
   3.2. Look for combined string in acceptance string of each state
      3.2.1. If found in acceptance state of any q
         a. Resultant state of given state = q
      3.2.2. Else
         a. Resultant state of given state = qΦ
   3.3. If end of input symbol for given state
      3.3.1. Go to Step 3
      3.4. Else
         3.4.1. Go to Step 3
   4. For final state and dead state
   4.1. Resultant state of given state = state itself

**Fig. 6. Algorithm for Designing of Deterministic Finite Automata with Given Prefix**

The flowchart for the same is shown in Figure 7.

5 Results and Discussions

An example of designing a transition table for DFA over $\Sigma = \{a, b\}$ which recognizes strings having prefix (starting) 'aba'.

No. of states = length of string + 1 + the dead state (qΦ)
$q_0$ = initial state
$q_1$ = strings starting with a (a of given string ‘a’ba)
$q_2$ = strings starting with ab (ab of given string ‘ab’ba)
$q_3$ = final state and string starting with aba (final/given string)
$q_0$ = dead state

**Table 5.** An initial template for the State/Transition table for strings having prefix ‘aba’

<table>
<thead>
<tr>
<th>Acceptance</th>
<th>States</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda$</td>
<td>$q_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$q_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>$q_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aba</td>
<td>$q_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dead</td>
<td>$q_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Steps to fill the table:

**Step 1:** For state $q_0$:
--- Transition for state $q_0$ on input symbol ‘a’ goes to state $q_1$ i.e.

$\delta(q_0, a) = q_1$

Here, the state $q_1$ accepts the string starting with ‘a’

**Step 2:** for state $q_1$:
--- Combining the string of $q_1$ with input symbol ‘a’; transition for state $q_1$ on input symbol ‘aa’ goes to state $q_6$ i.e.

$\delta(q_1, aa) = q_6$

It is because there is no state accepts string starting with ‘aa’

--- Combining the string of $q_1$ with input symbol ‘b’; transition for state $q_1$ on input symbol ‘ab’ goes to state $q_2$ i.e.

$\delta(q_1, ab) = q_2$

Here, $q_2$ accepts the string starting with ‘ab’

**Step 3:** for state $q_2$:
--- Combining the string of $q_2$ with input symbol ‘a’; transition for state $q_2$ on input symbol ‘aba’ goes to state $q_3$ i.e.

$\delta(q_2, aba) = q_3$

Here, $q_3$ accepts the string starting with ‘aba’

--- Combining the string of $q_2$ with input symbol ‘b’; transition for state $q_2$ on input symbol ‘abb’ goes to state $q_6$ i.e.

$\delta(q_2, abb) = q_6$

It is because no state accepts string starting with ‘abb’

**Step 4:** for state $q_3$:
--- For the final state, transition is the state itself. So, the transition of state $q_3$ on input symbol ‘a’ and ‘b’ goes to $q_3$ i.e.

$\delta(q_3, a) = q_3$

$\delta(q_3, b) = q_3$

**Step 5:** for state $q_4$:
--- For dead state, transition is the state itself. So, the transition of state $q_4$ on input symbol ‘a’ and ‘b’ goes to $q_4$ i.e.

$\delta(q_4, a) = q_4$

$\delta(q_4, b) = q_4$

The final transition table as constructed by using steps of algorithm is shown in table 6.

**Table 6.** State/Transition Table for Strings having Prefix ‘aba’

<table>
<thead>
<tr>
<th>Acceptance</th>
<th>States</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda$</td>
<td>$q_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$q_1$</td>
<td>$q_6$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>ab</td>
<td>$q_2$</td>
<td>$q_1$</td>
<td>$q_6$</td>
</tr>
<tr>
<td>aba</td>
<td>$q_3$</td>
<td>$q_3$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>dead</td>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_0$</td>
</tr>
</tbody>
</table>

From the table, the corresponding transition/state diagram can be constructed easily as shown in Figure 8.
domains like text and speech processing, pattern matching, vending machines, traffic lights systems, etc. In the future, the algorithm can be further optimized and simplified and can be extended to other regular languages.

**References**


[28] Ather, Danish, Raghuraj Singh, and VinodaniKatiyar: To Develop an Efficient Algorithm that Generalize the Method of Design of Finite Automata that Accept “N” base Number such that when “N” is Divided by” M” Leaves Reminder” X”. International Journal of Computer Applications 60.10 (2012).