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

Rashandeep Singh
Chandigarh College of Engineering and Technology, Chandigarh, India

Gulshan Goyal
Chandigarh College of Engineering and Technology, Chandigarh, India
Corresponding author: Rashandeep Singh, Email: rashandeepsingh@gmail.com

The field of automata theory is one of the most important areas in the field of Computer Science and Engineering that deals with how efficiently a problem can be solved on a model of computation using an algorithm. There exists a variety of formal languages like regular, context free, context sensitive, etc. These formal languages are described as 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 the lexical analysis phase of compiler design for recognition of tokens. The construction of finite automata is a complicated and challenging process as there is no fixed mathematical approach that exists for designing of DFA and handling the validations for acceptance or rejection of strings. Consequently, it is difficult to represent the DFA's transition table and graph. 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.

2021. In Raju Pal & Praveen K. Shukla (eds.), *SCRS Conference Proceedings on Intelligent Systems*, 133–142. Computing & Intelligent Systems, SCRS, India.
https://doi.org/10.52458/978-93-91842-08-6-11

1 Introduction

Languages are means of communication and can be natural or formal. Examples of natural languages include English, Hindi, and Punjabi, etc. which has 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. Formal Languages are further classified in the form of Chomsky hierarchy [2], [3], [4] as given below:

Table 1. Chomsky Hierarchy

Type	Language (Grammar)
3	Regular Language
2	Context-Free Language
1	Context-Sensitive Language
0	Recursive and Recursively Enumerable Language

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 of 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 regular languages. The regular language can be represented using a finite state machine also called finite automata [6], [7]. These automata can be in only one state at a time and the input system causes a transition from the current state to the next state [8].

Some basic terms used in automata theory are:
1. Alphabet (∑), it is a set of finite and non-empty input symbols. e.g. ∑ = {0, 1} represents binary alphabet consisting of 0 and 1.
2. Strings (w), a finite sequence of input symbols chosen from some alphabet ∑, normally denoted by w, x, y, z. e.g. if ∑ = {0, 1} then 1011 and 111 are example strings.
3. ∑* is set of all strings over an alphabet ∑. e. g. if ∑ = {0, 1} then ∑* = {ʌ, 0, 1, 00, 01, 10, 11, ...}
4. Languages (L), set of strings all of which are chosen from some ∑*, where ∑ is the alphabet. Mathematically L is a subset of ∑*.

A formal language can be represented using a finite state machine [9]. For example, a regular language that is described using mathematical expressions called regular expressions can be recognized by finite automata [7]. Finite Automata can be deterministic or non- deterministic in nature. In deterministic finite automata, there is exactly one transition for each state and input symbol [8]. However, in the case of non-deterministic finite automata, there can be zero, one or more transitions for each state and input symbol [10]. A DFA can be represented using a State/Transition diagram or table. 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]. The next section provides detailed insights into deterministic finite automata (DFA).

2 Deterministic Finite Automata

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

$$M= (Q, \sum, \delta, q_0, F \text{ or } A)$$

where,

Table 2. 5-tuple Set of Finite Automata (M)

Tuple	Description
Q	Finite set of states
$\sum$	Finite set of input symbols
δ	Transition function. Mathematically, $Q \times \sum \rightarrow Q$
q_0	Initial or start state and $q_0 \in Q$
F or A	Set of accepting/final states F

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

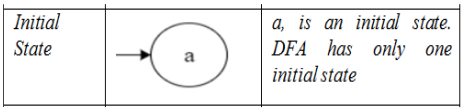

Fig. 1. Representational Description of Initial State

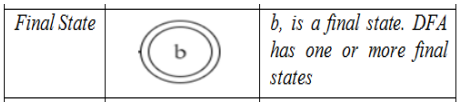

Fig. 2. Representational Description of Final State

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 $L \subseteq \sum^*$ if every string in L is accepted by M and no more which 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, δ: $Q \times \sum \rightarrow Q$

$\delta(a, o) \rightarrow b$, where $a, b \in Q$ and $o \in \sum$

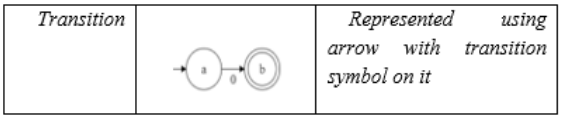

Fig. 3. Representational Description of Transition Function

Transition Graph also called "State Transition Diagram" in which the vertices represent states and the edges represent transitions from one state to another [16]. The labels on the vertices represent the names of the states while the labels on the edges represent the current values of the input symbol.

As shown in Figure 4, q_0, q_1, q_2, 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 other.

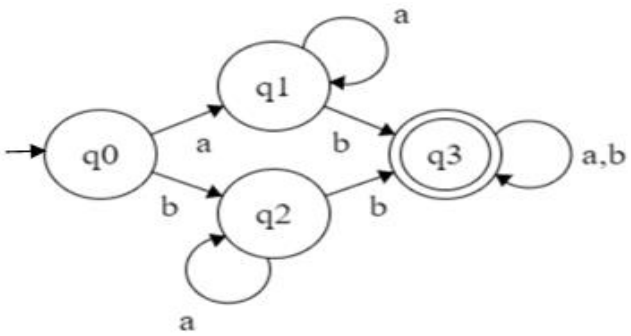

Fig. 4. An Example Transition Diagram/Graph

The Transition table as shown in Table 3 is a tabular representation of the transition function that requires two arguments – current state and input symbol. The output of the transition function is a state. The column corresponds to the state in which the automaton will be on the input represented by that particular column. The row corresponds to the current state. The entry for one row corresponding to state q and the column corresponds to input b is the state δ (q, b). 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

States	a	b
→q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_3
(q_3)	q_3	q_3

Here Q = {q_0, q_1, q_2, q_3}, ∑ = {a, b} and F = {q_3}

M = ({q_0, q_1, q_2, q_3}, {a, b}, δ, q_0, {q_3})

Transitions:

$\delta(q_0, a) = q_1$	$\delta(q_0, b) = q_2$
$\delta(q_1, a) = q_1$	$\delta(q_1, b) = q_3$
$\delta(q_2, a) = q_2$	$\delta(q_2, b) = q_3$
$\delta(q_3, a) = q_3$	$\delta(q_3, b) = q_3$

2.1 DFA with Prefix

A DFA with a given prefix is described as a string consisting of leading symbols of a regular language. For example, a DFA over ∑= {a, b} which recognizes strings having prefix 'ab' is shown in Figure 5, and the transition table is shown in Table 4.

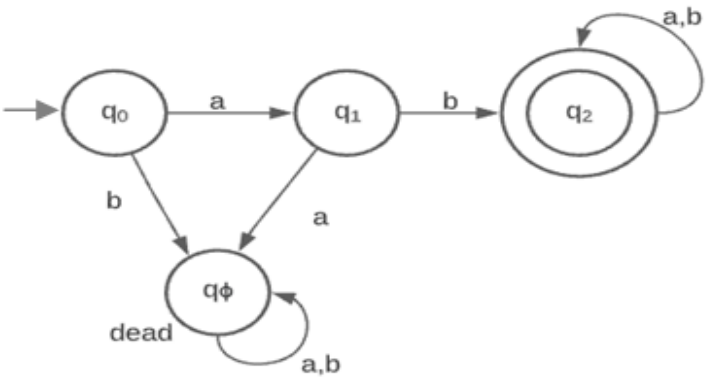

Fig. 5. State/Transition Diagram for Strings Having Prefix 'ab'

Table 4. State/Transition Table for Strings Having Prefix 'ab'

States	a	b
→q_0	q_1	q_ϕ
q_1	q_ϕ	q_2
(q_2)	q_2	q_2
q_ϕ	q_ϕ	q_ϕ

The string 'ababb' will be recognized by the DFA because starting from the initial state q_0 and after processing; the string is in final state q_2. Whereas the string 'aabba' will not be recognized by the DFA as after processing, the string is in state q_ϕ which is not a final state.

3 Literature Survey

In the past few decades, the theory of computation has been one of the challenging fields in research. 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 the modelling of the human thought process by Warren McCulloch and Walter Pitts [18]. A finite automaton in earlier models was a set of transitions and states with no output. In the 1950s, G.H. Mealy and E.F. Moore proposed the powerful machines namely Mealy and Moore machines in which output was also considered. In the Mealy machine, the output is assigned with every transition and in Moore machine output is assigned to every state in DFA [2].

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 [19] and speech processing [5]. Other DFA applications include pattern matching [11], 13], vending machines [20], and path finding DFA in video games having AI [21].

Despite so many applications, learners face difficulty in designing DFA due to the requirement of a high level of understanding [16].

Researchers have examined a finer-grained understanding of the characteristics of DFA. An algorithm is proposed to identify important transitions and critical patterns of a DFA [22]. An algorithm to

design a DFA over alphabet consisting of a and b having exactly x number of a's and y number of b's is discussed in [9].

From the literature, it is inferred that there is no availability of a well-defined algorithm for the generation of transition table for the given strings. So, the present paper focuses on the design and implementation of an algorithm that can be 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.

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 (q_ϕ)
2. Initialize q_i 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_i
            - a. Resultant state of given state = q_i
        - 3.2.2. Else
            - a. Resultant state of given state = q_ϕ
    - 3.3. If end of input symbol for given state
        - 3.3.1. Go to Step 3 *//Go to next state*
    - 3.4 Else
        - 3.4.1. Go to Step 3.1 *//Go to next input symbol*
4. for final state and dead state
    - 4.1. Resultant state of given state = state itself

End

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

The flowchart for the same is shown in figure 7.

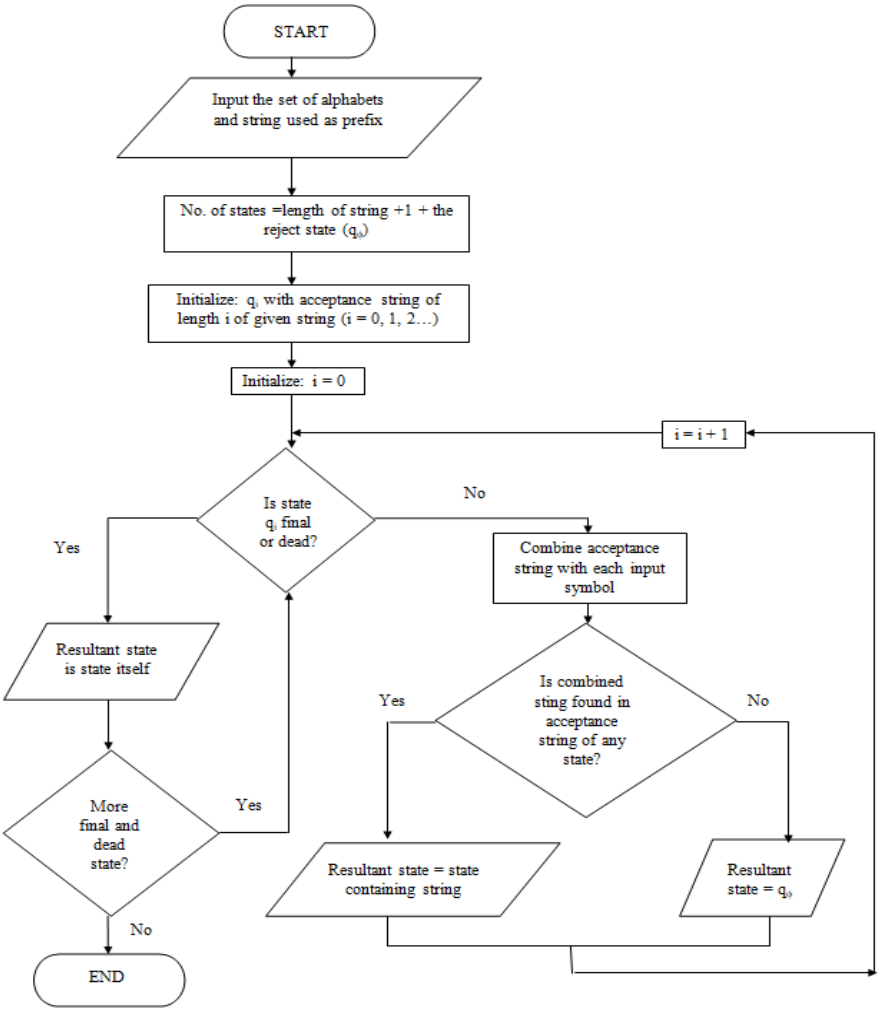

Fig. 7. Flowchart of Algorithm for Prefix

5 Results and Discussions

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

No. of states = length of string + 1 + the dead state (q_ϕ)
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'a)

q_3 = final state and string starting with aba (final/given string)
q_ϕ = dead state

Table 5. Initial template for State/Transition table for strings having prefix 'aba'

Acceptance	States	a	b
Λ	→q_0		
a	q_1		
ab	q_2		
aba	(q_3)		
dead	q_ϕ		

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'

--- Transition for state q_0 on input symbol 'b' goes to state q_ϕ i.e.
$\delta(q_0,b) = q_\phi$
It is because there is no state accepting 'b'

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_ϕ i.e.
$\delta(q_1,aa) = q_\phi$
It is because there is no state that 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.
$\delta(q2,aba) = q_3$
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_ϕ i.e.
$\delta(q_2,abb) = q_\phi$
It is because no state accepts string starting with 'abb'

Step 4: for state q_3:
--- For the final state, the transition is the state itself. So, the transition of state q_3 on input symbol 'a' and 'b' goes to q_3 i.e.
$\delta(q_3,a) = q_3$
$\delta(q_3,b) = q3$

Step 5: for state q_ϕ:

--- For the dead state, transition is state itself. So, the transition of state $q_\phi$on input symbol 'a' and 'b' goes to q_ϕi.e.

$\delta(q_\phi,a) = q_\phi$

$\delta(q_\phi,b) = q_\phi$

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

Table 6. State/Transition Table for Strings having Prefix 'aba'

Acceptance	States	a	b
Λ	→q_0	q_1	q_ϕ
a	q_1	q_ϕ	q_2
ab	q_2	q_3	q_ϕ
aba	q_3	q_3	q_3
dead	q_ϕ	q_ϕ	q_ϕ

From the table, the corresponding transition/state diagram can be constructed easily as shown in figure 8.

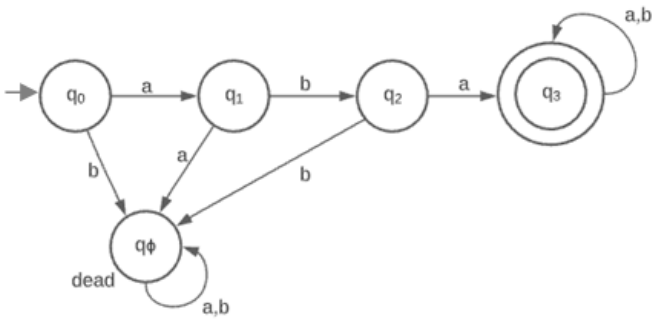

Fig 8. State/Transition Diagram for Strings having Prefix 'aba'

6 Conclusion

Theoretical computer science is an important area of computer science that focuses on mathematical aspects of the theory of computation. The major categories of formal languages in the theory of computation include regular, context-free, context sensitive, recursive, and recursive enumerable. The regular languages find an important application in the lexical analysis phase of compiler design for the recognition of tokens. In the scope of the present paper, an algorithm for the design of automata for a regular language given as a prefix string is proposed. The algorithm is discussed with the help of an example case study. The algorithm can benefit the novel learners in the field of automata theory and compiler design. In the future, the algorithm can be further optimized and simplified. Further, the algorithm can be extended to other regular languages.

References

[1] K. Singh and G. Goyal, "Algorithm Design and String Recognition for Suffix Strings Using Deterministic Finite Automata", *IJSRR*, vol.8, no.2, pp. 4406-4413, 2019.

[2] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation,* 2nd ed. India: Pearson Education, 2008.

[3] J. C. Martin, *Introduction to languages and theory of computation*, 4th ed. Tata McGraw Hill, 2007.

[4] K. L. P. Mishra and N. Chandrasekaran, *Theory of Computer Science Automata, Languages and Computation,* 3rd ed. Prentice Hall of India, 2004.

[5] J. D. Ullman, "Applications of language Theory to Compiler Design", in *the spring joint computer conference (ACM), spring,* 1972.

[6] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 2nd ed. Addison – Wesley, 1979.

[7] A. X. Liu and E. Torng, "Overlay automata and algorithms for fast and scalable regular expression matching", *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2400-2415, 2016.

[8] G. O'Regan. (2020). *Automata Theory. In: Mathematics in Computing. Undergraduate Topics in Computer Science*. Springer, Cham. https://doi.org/10.1007/978-3-030-34209-8_23.

[9] D. Ather, R. Singh and V. Katiyar, "An algorithm to design finite automata that accept strings over input symbol a and b having exactly x number of a & y number of b", in *Int. Conf. on Information Systems and Computer Networks*, 2013.

[10] R. G. Parekh and V. G. Honavar, "Learning DFA from Simple Examples", *J. Machine Learning*, vol. 44, no. 1-2, pp. 9-35, 2001.

[11] P. Ejendibia and B. B. Baridam, "String Searching with DFA-based Algorithm", *Int. J. Appl. Inform. Systems*, vol. 9, no. 8, pp. 1-6, 2015.

[12] N. Murugesan and B. Samyukthavarthini, *A Study on Various types of Automata*, M.Phil., Dissertation, Bharathiar University, 2013.

[13] A. B. Karuppiah and S. Rajara, "Deterministic Finite Automata for pattern matching in FGPA for intrusion detection", in *Int. Conf. on Computer. Communication and Electrical Technology*, pp. 167-170, 2011.

[14] K.S. Kumar and D. A. Malathi, "Novel Method to Construct Deterministic Finite Automata from A Given Regular Grammar", *Int. J. Scientific & Engineering Research*, vol. 6, no. 3, pp. 106-111, 2015.

[15] N. G. Murugesan, *Principles of Automata theory and Computation*, Sahithi Publications, 2004.

[16] V. Shenoy, U. Aparanji, K. Sripradha and V. Kumar, "Generating DFA Construction Problems Automatically", *Int. J. Computer Trends and Technology*, vol. 4, no. 4, pp.32-37, 2013.

[17] Y. D. Sergeyev and A. Garro, "Observability of Turing Machines: A Refinement of the Theory of Computation", *Informatica,* vol. 21, no. 3, pp. 425-454, 2010.

[18] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol.5, no.4, pp. 115-133, 1943.

[19] A. B. Webber, *Formal Language: A Practical Introduction*. Franklin, Beedle & Associates Inc., Wilsonville, 2008.

[20] E. Gribko, *Applications of Deterministic Finite Automata*. ECS 120 UC Davis, Spring, 2013.

[21] N. Raj and R. Dubey, "Snakes and Stairs Game Design using Automata Theory", *Int. J. Computer Sciences and Engineering*, vol. 5, no. 5, pp.58-62, 2017.

[22] K. Zhang, Q. Wang and C. L. Giles, *Adversarial Models for Deterministic Finite Automata*. In: Goutte C., Zhu X. (eds) Advances in Artificial Intelligence. Canadian AI 2020. Lecture Notes in Computer Science, vol 12109. Springer, Cham. https://doi.org/10.1007/978-3-030-47358-7_55.