IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 9, Issue 5, Oct-Nov 2021
ISSN: 2320 – 8791 (Impact Factor: 2.317)
www.ijreat.org

Designing of Deterministic Finite Automata for a Given Regular Language with Substring

Rashandeep Singh[1], Nishtha Kapoor[2] and Dr. Amit Chhabra[3]
Chandigarh College of Engineering and Technology, Chandigarh,
rashandeepsingh@gmail.com[1], nishthak36@gmail.com[2] and amitchhabra@ccet.ac.in[3]

ABSTRACT
Theory of computation deals with the computation logic in relation to the automata and is an important branch of computer science. There are various formal languages such as regular languages, context-sensitive languages, context-free languages, and so on that can be recognized by different automata. Regular language is recognized by finite automata. Finite automata recognize the symbols as an input and change its state accordingly. A finite automaton can be deterministic or non-deterministic in nature. Deterministic finite automata are used in the first and foremost important phase of compiler design i.e. lexical analysis. Different tokens are recognized by different final states of a DFA. It is a difficult and time consuming task to construct a DFA as there is no fixed approach for creating DFAs and handling string acceptance or rejection validations. The objective of this paper is to propose and implement an algorithm for construction of deterministic finite automata for a Regular Language with the given Substring. The output of this algorithm will be a transition table. The proposed method further aims to simplify the lexical analysis process of compiler design.

Keyword: Regular Language, Deterministic Finite Automata (DFA), Substring, Transition table

1. INTRODUCTION

A language is a collection of strings, each of which is picked from a set of ∑*, where ∑ is a set of alphabet and ∑* is set of all strings over a given alphabet [4]. An example of the alphabet (∑), strings (∑*) and language (L) is given in Figure 1.

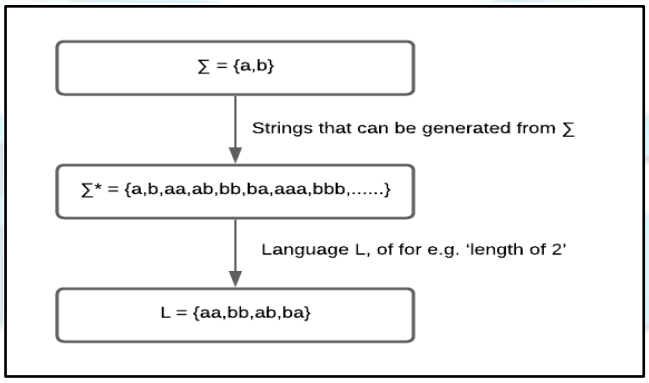

Figure 1: An example of the alphabet (∑), strings (∑*) and language (L)

Therefore we can say that Language is the set of operations over an alphabet (L ⊆ of ∑*). Formal language is defined as a set of specific strings over a given alphabet in accordance with some rules known as grammar. Any formal language can be represented using a finite state machine [9]. At a given instant of time, a finite state machine can be in only one state and the input system results in a transition from current state to next state [17]. Formal Languages are further classified in form of Chomsky hierarchy [1] as given below in Table 1:

Table 1: Chomsky Hierarchy

TYPE	LANGUAGE (GRAMMAR)	ACCEPTING DEVICE
3	Regular Language	Finite Automata
2	Context-Free Language	Pushdown Automata
1	Context-Sensitive Language	Linear-Bounded Automata
0	Recursively Enumerable (Unrestricted Or Phase-Structure)	Turing Machine

Every type of language i is i-1 also. Regular Languages, as shown in the table, are the most restricted sorts of languages [3]. The most efficient approach to describe regular language in the form of mathematical expression is through regular expressions [4]. Further, the regular languages can be recognized using finite automata which can be deterministic or non-deterministic. In deterministic finite automata, for each state and input symbol there is exactly one transition [6] [17]. However, there can be zero, one or more transitions for each state and input symbol in case of non-deterministic finite automata. DFA are also used for string recognition i.e. to check whether a string is accepted by a given DFA or not. 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 [15].
DFA ending with suffix, DFA beginning with prefix, DFA having a substring, DFA containing exactly, at least, or at most number of occurrences of symbol are only a few of the major forms of DFA. The aim of this paper is to implement an algorithm for creating DFAs having a substring.
The paper is further organized as follows. Next section and its subsection describe deterministic finite automata and DFA with substring respectively in detail. Section 3 provides an insight on applications of DFA. Section 4 discusses problem formulation. The detailed algorithm for design of DFA for formal language given as substring is proposed in Section 5. It's corresponding illustration is given in section 6. Finally, conclusion and future scope are presented.

2. DETERMINISTIC FINITE AUTOMATA

The term deterministic refers to the fact that the automaton can only transition to one state from its present state on each input [16]. Deterministic finite automata are defined as those Finite automata that have a transition for every symbol in the input alphabet [2].The null move is not accepted by DFA, which means it cannot change state without any input character.

A DFA can be defined as 5 tuples [8]:
M= (Q, ∑, δ, q_0, F or A)
Where,

Q: A finite set of states

∑: A finite set of the input symbols

q_0: Initial state

F: A set of final or accepting state

δ: Q x ∑→Q Transition function

It's difficult to read a dfa as a five-tuple with a thorough description of the transition function. For describing automata, there are two standard notations.

- A transition table
- A transition diagram

In a transition table transition function is represented in a tabular form. Each row in the transition table will represent a DFA state, and each column will represent an input symbol. Arrow is pointed to the initial state whereas final states are encircled. Cell[i][j] will store the state where transition at state i will take place when input j is provided.

A transition diagram also called "State Transition Diagram" is a directed graph in which the vertices represent states and the edges represent transitions [7]. A double circle denotes nodes that correspond to accepted states and a single circle represents the rest of the states. There is always an arrow into the start state that is not from any other state. Each transition in the transition diagram is represented by an arrow or an edge with the input symbol written over it..

As shown in Figure 2, q_0, q_1, q_2, and q_3 represent states in a graph. The arrow pointed to q_0 represents the start state from where transitions begin, while the encircled q_3 denotes 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.

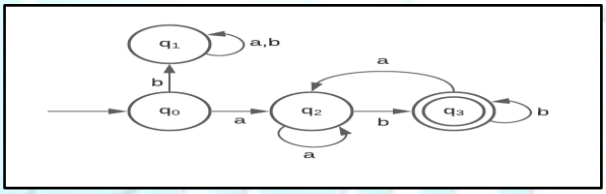

Figure 2: An Example Transition Diagram/Graph

Transition table of the corresponding transition diagram is shown in Table 2. Transition function (δ) requires 2 arguments - current state and input symbols and produces the output of the state [5]. The entry for one row corresponding to state q and the column corresponding to input a is the state δ(q, a).

Table 2: Transition table

States	a	b
→q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_2	q_3
(q_3)	q_2	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, q0, {q_3})

The first column in the table above lists all of the current states. It's transitions on input symbols 'a' and 'b' are displayed in columns a and b.

- When the current state is q_0, its transition on input symbol 'a' is q_2, and on 'b' is q_1, according to the first row of the transition table.

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

- When the current state is q_1, its transition on input symbol 'a' is q_1, and on 'b' is q_1, according to the first row of the transition table.

$$\delta(q_1, a) = q_1 \qquad \delta(q_1, b) = q_1$$

- When the current state is q_2, its transition on input symbol 'a' is q_2, and on 'b' is q_3, according to the third row of the transition table.

$$\delta(q_2, a) = q_2 \qquad \delta(q_2, b) = q_3$$

- When the current state is q_3, its transition on input symbol 'a' is q_2, and on 'b' is q_3, according to the last row of the transition table.

$$\delta(q_3, a) = q_3 \qquad \delta(q_3, b) = q_3$$

2.1 DFA WITH SUBSTRING

A DFA with a given substring is described as a string consisting of substring symbols of a regular language. Example, a DFA over Σ = {a, b} which recognizes strings having substring 'ab' is shown in Figure 3.

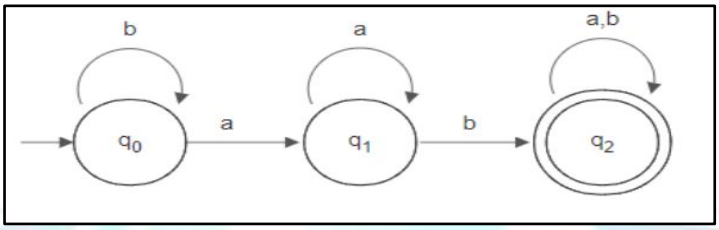

Figure 3: State/Transition table for strings having substring 'ab'

The string 'aabbb' (as it contains substring 'ab') 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 'bbbaa' (as it does not contain substring 'ab') will not be recognized by the DFA as after processing, the string is in state q_1 which is not a final state.

3. APPLICATIONS OF DFA

DFA has extreme importance in many applications such as:

- Token recognition: The lexical analyzer's primary role is to scan the program written one character at a time and generate the corresponding token [10]. Different tokens are recognized by different final states of a DFA.
- Text Parser: Text processors or text filters utilize DFA-like code to scan a text file for strings that match a given pattern [11].
- Vending Machines: DFA can also be used in vending machines, where the value of coins acts as the machine's state, and only a specific combination of coins causes the selected item to be dispensed [13].
- Speech Processing: The DFA approach is frequently used in speech processing and other signal processing systems to convert an input signal [12].
- Pattern Matching: DFA is a machine or a simple language recognition device that recognizes the given input strings. Minimized DFA is more useful because it minimizes the amount of memory space required [14] [15].
- Video games: in games like Pac man there are 4 states like Wander the Maze, Chase Pac-Man, Return to Base, Flee Pac-Man [13]. Thus the DFA approach is used in such video games.

Despite so many applications, learners face difficulty in designing a DFA due to the requirement of a high level of understanding [7]. There is no availability of well defined algorithm for the generation of transition table for the given strings. The need for a well-defined algorithm is discussed in the next section.

4. PROBLEM FORMULATION

DFA is of extreme importance in many applications including lexical analysis of compiler, text parsing, natural language processing, CPU control units and many more. However it is a difficult task to construct a DFA as there is no fixed approach for its construction. Beginners lack high level understanding so they face difficulty in creating a DFA for a given regular language with substring. Therefore there is a need for an algorithm which can help in construction of a DFA and give a transition table as the output. In the next section an algorithm is proposed which will help in designing the DFA with the help of a given substring 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**.** This algorithm has no restrictions on the type of input symbols, as well as the length of the provided substring. Because the approach is not language-specific but rather universal in nature, it can be used to create DFA in any programming language.

5. PROPOSED ALGORITHM FOR THE DESIGN OF DFA FOR A GIVEN REGULAR LANGUAGE WITH SUBSTRING

The algorithm for construction of transition table for deterministic finite automata with given substring is described in Figure 4.

Algorithm 1: Designing DFA for Regular language having given Substring

Input: Set of alphabets and string to be used as substring.

Output: Transition table of the DFA.

Initialization:

Step 1: No. of states = length of string + 1

Step 2: Initialize q_i with acceptance string of length i of given string (i=0, 1, 2…)

Step 3: For each state except final state

3. A. Combine the acceptance string with input symbol
3. B. Look for combined string in acceptance string of each state
3. B. a. If found in acceptance of q_i && length != Λ (0)
i. Resultant state of given state = q_i
ii. If end of input symbol for given state
Go to Step 3 *//Go to next state*
iii. Else
Go to Step 3.A. *//Go to next input symbol*
3. B. b. Else if not found && length != Λ (0)
i. Decrease the length of string
ii. Go to step 3.B.
3. B. c. Else *// length = Λ (0)*
i. Resultant state of given state = q_0
ii. Go to step 3

Step 4: For final state, resultant state is state itself

Stop

Figure 4: Algorithm for Substring

The flowchart for the same is shown in Figure 5.

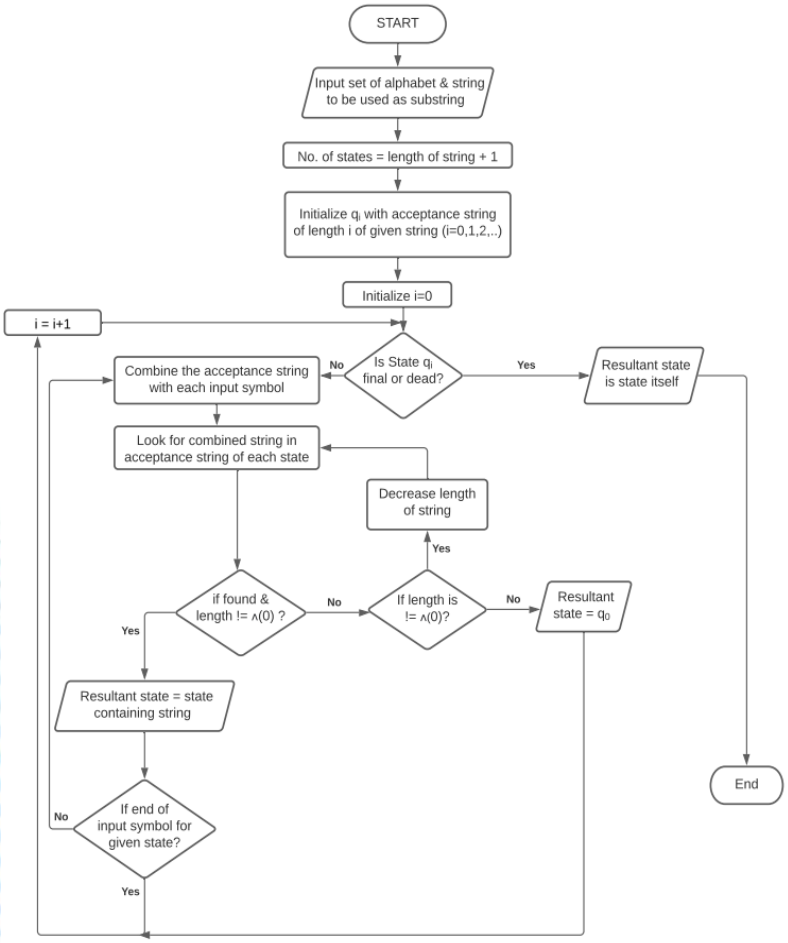

Figure 5: Flowchart for Algorithm for Substring

6. RESULTS AND DISCUSSIONS

An example of designing a transition table for DFA over Σ = {a, b} which recognizes strings having substring 'ab'

No. of states = length of string + 1
q_0 = initial state
q_1 = strings starting with a (a of given string 'a'b)
q_2 = strings starting with ab (final/ given string)
The initial template for the state/transition table for strings having substring 'ab' is shown in Table 3.

Table 3: Initial Template for State/Transition table for strings having Substring 'ab'

Acceptance	States	a	b
∧	q_0		
a	q_1		
ab	q_2		

Steps to fill the table:
Step 1: For state q_0:
$\delta(q_0,a) = q_1$ // q_1 accepts the string a
$\delta(q_0,b) = q_0$ // there is no state accepting b, so q_0

Step 2: For state q_1:
// combining the string of q_1 with input symbol 'a'
$\delta(q_1,aa)$ =not found in acceptance of any state // no state accepts string aa
// $\delta(q_1, \cancel{a}a)$ - now we will go for $\delta(q_1,a)$
$\delta(q_1,a) = q_1$ // q_1 accepts the string a
// combining the string of q_1 with input symbol 'b'
$\delta(q_1,ab) = q_2$ //q_2 accepts the string ab

Step 3: For state q_2:
// for final state, resultant is state itself
$\delta(q_2,a) = q_2$
$\delta(q_2,b) = q_2$

The final transition table as constructed by using steps of the algorithm is shown in Table 4**.**

Table 4: State/Transition table for strings having substring 'ab'

Acceptance	States	a	b
∧	q_0	q_1	q_0
a	q_1	q_1	q_2
ab	q_2	q_2	q_2

From the table, the corresponding transition/state diagram as shown in Figure 3 can be constructed easily.

7. CONCLUSION AND FUTURE SCOPE

Theory of computation helps define infinite languages in finite ways, create algorithms for related problems, and determine if a string is in language or not. Designing an automaton is an important part of automata theory. The present paper proposes and implements an algorithm for designing a DFA with a given regular language as substring, which takes language as input and outputs a transition table, by using this transition table we can create a transition diagram. Therefore with the help of this algorithm new learners can make a DFA easily. An example is used to demonstrate the algorithm. Furthermore, the algorithm only considers DFA with a substring as a unique case. In future a more generalized algorithm could be created so that it can cover all possible cases using which DFA could be constructed.

REFERENCE

[1] Hopcroft, J. E., Motwani R., Ullman J.D. (2008), "Introduction to Automata Theory, Languages and Computation", 3rd Edition, Pearson Education, India.
[2] Introduction-To-The-Theory-Of-Computation-Michael-Sipser 2nd edition 2006
[3] Martin J.C. (2007), "Introduction to languages and theory of computation", Tata McGraw Hill.
[4] Deterministic finite automata [online] available at https://www.geeksforgeeks.org/theory-of-computation-automata-tutorials/ accessed on 2nd June 2021
[5] Zhang, K., Wang, Q., & Giles, C. L. (2020). Adversarial Models for Deterministic Finite Automata. In C. Goutte, & X. Zhu (Eds.), Advances in Artificial Intelligence - 33rd Canadian Conference on Artificial Intelligence, Canadian AI 2020, Proceedings (pp. 540-552). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 12109 LNAI).Springer. https://doi.org/10.1007/978-3-030-47358- 7_55
[6] Ather, D., Singh, R., & Katiyar, V. (2013). 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. International Conference on Information Systems and Computer Networks, IEEE, pages 1–4,DOI: 10.1109/ICISCON.2013.6524162
[7] Shenoy V., Aparanji U., Sripradha K. & Kumar V. (2013). Generating DFA Construction Problems Automatically. International Journal of Computer Trends and Technology, Vol. 4, Issue 4, pp.32-37
[8] K.Senthil Kumar and D.Malathi (March-2015), "A Novel Method To Construct Deterministic Finite Automata From A Given Regular Grammar", International Journal of Scientific & Engineering Research, Volume 6, Issue 3, 106 ISSN 2229-5518.
[9] Liu D, Huang Z, Zhang Y, Guo X, Su S (2016), "Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information", PLoS ONE 11(11): e0165864, https://doi.org/10.1371/journal.pone.0165864
[10] Rajanshu Goyal1 and Gulshan Goyal, "Design and Implementation of Transition Table for Token Recognizer with a Given Suffix", International Journal of Computer Sciences and Engineering, Vol.-7, Issue-5, May 2019, E-ISSN: 2347-2693
[11] Webber A. B., "Formal Language: A Practical Introduction", Franklin, Beedle & Associates Inc., Wilsonville, pp. 35-43, 2008.
[12] Ullman, J. D. (1972), "Applications of language Theory to Compiler Design", Proceedings of the May 16-18, 1972, spring joint computer conference, pp. 235-242, 1972.
[13] Gribko E. "Applications of Deterministic Finite Automata" ECS 120 UC Davis, Spring 2013, pp. 1-9, 2013.
[14] BabuKaruppiah A., Rajaram S., "Deterministic Finite Automata for pattern matching in FPGA for intrusion detection" International Conference on Computer, Communication and Electrical Technology, pp. 167-170, 2011
[15] Ejendibia P., Baridam B. B., "String Searching with DFA-based Algorithm", International Journal of Applied Information Systems, Vol. 9, No. 8, pp. 1-6, 2015
[16]Murugesan N, & Samyukthavarthini B (2013) A Study on Various types of Automata
[17] O'Regan G. (2020) "Automata Theory. In: Mathematics in Computing", Undergraduate Topics in Computer Science, Springer, Cham.