Volume 66, Issue 2, 2022

Journal of Scientific Research

of

The Banaras Hindu University

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

Rashandeep Singh[1] and Dr. Gulshan Goyal[2]

[1, 2]Chandigarh College of Engineering and Technology, Chandigarh, India
[1]rashandeepsingh@gmail.com, [2]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 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 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 ($\sum$), is a set of non-empty and finite input symbols. e.g. $\sum = \{0, 1\}$ represents binary alphabet consisting of 0 and 1.
2. Strings (w), is a finite arrangement of input symbols selected from the alphabet $\sum$, normally denoted by w, x, y, z. e.g. if $\sum = \{0, 1\}$ then 1011 and 111 are example strings.
3. $\sum^*$ is set of all strings over an alphabet $\sum$. e. g. if $\sum = \{0, 1\}$ then $\sum^* = \{\wedge, 0, 1, 00, 01, 10, 11, \ldots\}$
4. Languages (L), is a set of specific strings selected from $\sum^*$. Mathematically L is a subset of $\sum^*$.

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].

DOI: 10.37398/JSR.2022.660203

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],

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

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.

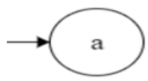
Initial State		*a, is an initial state. DFA has only one initial state*

Fig. 1. Representational Description of Initial State

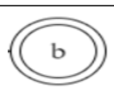
Final State		*b, is a final state. DFA has one or more final states*

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 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, $\delta: Q \times \sum \rightarrow Q$

$\delta(a, 0) \rightarrow b$,where $a, b \in Q$ and $0 \in \sum$

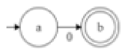
Transition		*Represented using arrow with transition symbol on it*

Fig. 3. Representational Description of Transition Function

Transition Graph is also named "State Transition Diagram" in which nodes represent states and the edges represent transitions from one state to another [16]. The labels on the vertices are used to show the names of the states, while the labels on the edges show the present 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 another.

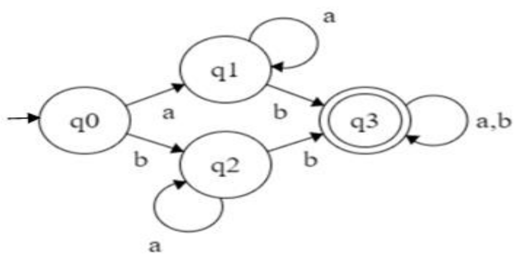


Fig. 4. An Example Transition Diagram/Graph

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

States	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_3
q_3	q_3	q_3

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\})$$
where $Q = \{q_0, q_1, q_2, q_3\}$, $\sum = \{a, b\}$ and $F = \{q_3\}$

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. Figure 5 represents a DFA over ∑= {a, b} which recognizes strings with the prefix ‘ab’, and Table 4 represents the transition table for the same.

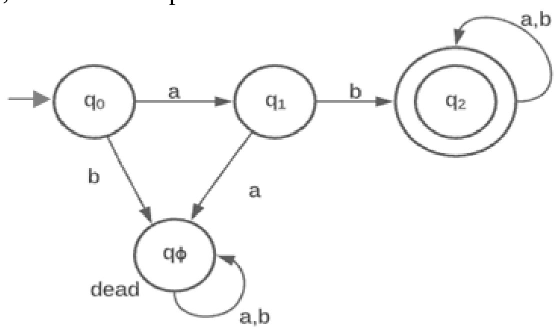


Fig. 5. State/Transition Diagram for Strings Having Prefix ‘ab’

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

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 ‘aabba’ will not be recognized by the DFA as after processing, the string is in the state q_ϕ 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 Prefi

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.

5 Results and Discussions

An example of designing a transition table for DFA over $\sum$ = {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. An initial template for the 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 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 staring 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, 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 dead state, transition is the state itself. So, the transition of state q_ϕ on input symbol ‘a’ and ‘b’ goes to q_ϕ i.e.

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

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’

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.

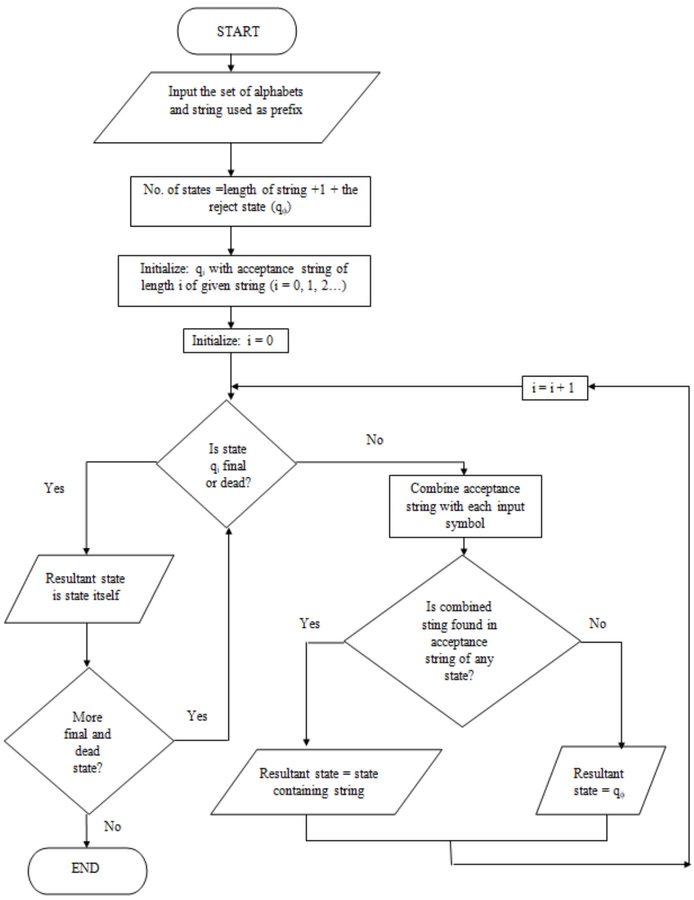


Fig. 7. Flowchart of Algorithm for Prefix

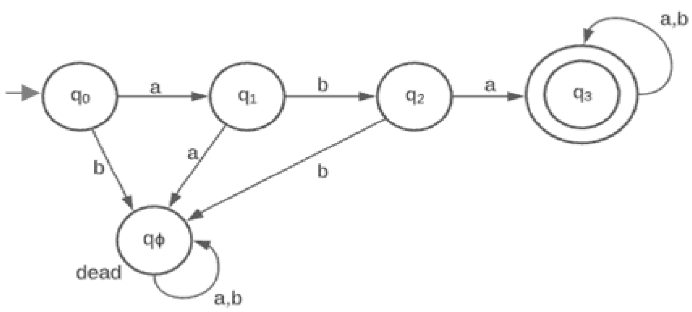


Fig. 8. State/Transition Diagram for Strings having Prefix 'aba'

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 including regular languages are described in Chomsky classification of formal languages. The regular languages find an important application in compiler design for the recognition of tokens. This paper proposes an algorithm for designing automata for regular languages that accept prefix strings as input. 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. A learner with a better understanding of the DFA construction algorithm will be able to undertake applications 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

[1] Singh Kamaplreet & Goyal Gulshan: Algorithm Design and String Recognition for Suffix Strings Using Deterministic Finite Automata. IJSRR, Vol.8 No.2, ISSN: 4406-4413 (2019).
[2] Hopcroft, J. E., Motwani R. & Ullman J.D.: Introduction to Automata Theory, Languages and Computation", 2nd Edition, Pearson Education, India, pp. 37-55 (2008).
[3] Martin J.C.: Introduction to languages and theory of computation, Tata McGraw Hill pp. 277-283 (2007).
[4] Mishra K.L.P. & Chandrasekaran N.: Theory of Computer Science Automata, Languages and Computation, 3rd Edition, Prentice Hall of India pp. 120-129 (2004).
[5] Ullman, J. D.: Applications of language Theory to Compiler Design. Proceedings of the spring joint computer conference (ACM), spring, pp. 235-242 (1972).
[6] Hopcroft J.E & Ullman J.D: Introduction to Automata Theory, Languages and Computation, Addison – Wesley, pp. 37-53 (1979).
[7] Liu, A. X., & Torng, E.: Overlay automata and algorithms for fast and scalable regular expression matching. IEEE/ACM Transactions on Networking, Volume 24, Issue 4, pp. 2400-2415 (2016).
[8] O'Regan G.: Automata Theory. In: Mathematics in Computing. Undergraduate Topics in Computer Science. Springer, Cham, https://doi.org/10.1007/978-3-030-34209-8_23 (2015).
[9] Ather, D., Singh, R., & Katiyar, V.: 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 (2013).
[10] Parekh, R. G. & Honavar, V. G.: Learning DFA from Simple Examples. Journal of Machine Learning, Vol. 44, Issue 1-2, pp. 9-35 (2001).
[11] Ejendibia P. & Baridam B. B.: String Searching with DFA-based Algorithm. International Journal of Applied Information Systems, Volume 9, No. 8, pp. 1-6 (2015).
[12] Murugesan N, and Samyukthavarthini B.: A Study on Various types of Automata. M.Phil., Dissertation, Bharathiar University (2013).
[13] Babu Karuppiah A. & Rajaram S.: Deterministic Finite Automata for pattern matching in FGPA for intrusion detection. International Conference on Computer, Communication and Electrical Technology, pp. 167-170 (2011).
[14] K.Senthil Kumar & D.Malathi: 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 (2015).
[15] Murugesan NG.: Principles of Automata theory and Computation, Sahithi Publications (2004).
[16] Shenoy V., Aparanji U., Sripradha K. & Kumar V.: Generating DFA Construction Problems Automatically. International Journal of Computer Trends and Technology, Volume 4, Issue 4, pp.32-37 (2013).
[17] Sergeyev, Yaroslav D. & Garro, Alfredo: Observability of Turing Machines: A Refinement of the Theory of Computation. Informatica, Volume 21, No. 3, pp. 425-454 (2010).

[18] McCulloch, W.S., & Pitts, W: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, Volume 5, No.4, pp. 115-133 (1943).
[19] Liu D, Huang Z, Zhang Y, Guo X, Su S.: Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information. PLoS ONE 11(11): e0165864. https://doi.org/10.1371/journal.pone.0165864 (2016).
[20] Webber A. B.: Formal Language: A Practical Introduction. Franklin, Beedle & Associates Inc., Wilsonville, pp. 35-43 (2008).
[21] Gribko E.: Applications of Deterministic Finite Automata. ECS 120 UC Davis, Spring, pp. 1-9 (2013).
[22] Raj N. & Dubey R.: Snakes and Stairs Game Design using Automata Theory. International Journal of Computer Sciences and Engineering, Volume 5, Issue 5, pp.58-62 (2017).
[23] Shakhmetova, G., Saukhanova, Z., Udzir, N. I., Sharipbay, A., & Saukhanov, N.: Application of Pseudo-Memory Finite Automata for Information Encryption. In IntelITSIS, pp. 330-339 (2021).
[24] Zhang K., Wang Q., Giles C.L. (2020) Adversarial Models for Deterministic Finite Automata. In: Goutte C., Zhu X. (eds) Advances in Artificial Intelligence. Canadian AI 2020. Lecture Notes in Computer Science, Volume 12109. Springer, Cham. https://doi.org/10.1007/978-3-030-47358-7_55 (2020).
[25] Grachev, P., Lobanov, I., Smetannikov, I., & Filchenkov, A.: Neural network for synthesizing deterministic finite automata. Procedia computer science, 119, 73-82, (2017).
[26] Krishnaprasath, V. T., and J. Preethi: Finite automata model for leaf disease classification. Agricultural Economics (2021).
[27] Lee, J., Lee, J., Moon, S., Nam, D., & Yoo, W.: Basketball event recognition technique using Deterministic Finite Automata (DFA). In 2018 20th International Conference on Advanced Communication Technology (ICACT), pp. 675-678, IEEE (2018).
[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).

***