IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

Design and Implementation of Transition Table for Token Recognizer with a Given Suffix

Rajanshu Goyal[1*], Gulshan Goyal[2]

[1,2]Department of CSE, Chandigarh College of Engineering & Technology (Degree), Chandigarh, India

[*]*Corresponding Author: rajanshugoyal@gmail.com*

DOI: https://doi.org/10.26438/ijcse/v7i5.15381542 | Available online at: www.ijcseonline.org

Accepted: 12/May/2019, Published: 31/May/2019

***Abstract*—** Token description and recognition are two important functions of a lexical analyser in compiler design. A token can be described using mathematical expression like notation called regular expression. The process of token recognition is carried out with the help of finite automata. A finite state automaton is a type of machine in which for each state and input symbol, a transition takes place. A finite automaton can be deterministic or non-deterministic depending on number of possible transitions for each state and input symbol. The design of automata for recognition of tokens is an important and challenging task. From the available transitions, a token recognizer can be designed using JFLAP tool which further help in token recognition. However, there is no standard algorithm or procedure for construction of transition table for token recognizer. Present paper proposes an algorithm for construction of a language recognizer using deterministic finite automata for all tokens having a given suffix. The algorithm is implemented and tested for various token strings and results are used and compared with JFLAP results.

***Keywords*—** Lexical Analysis, Token, Language Recognizer, Deterministic Finite Automata, Suffix.

I. INTRODUCTION

Lexical analysis is first and most important phase in compiler design. The basic function of lexical analyser is to scan the source program one character at a time and generate the corresponding token. In lexical analysis, lexemes are smallest possible addressable units in a program while tokens are logical categories to which lexeme belongs [1]. For example, consider the following statement:

$$a = b + c; \quad (1)$$

The statement consists of lexemes and tokens as shown in Table 1. There are two main tasks related to tokens:

1. Token Description
2. Token recognition

The tokens of a source language can be described using the mathematical notations called regular expressions while the tokens can be recognized using finite state machines called automaton [1]. As an example, a token described using the regular expression (a/b)*ab can be recognized using automaton as shown in Figure 1.

Table 1. Lexemes and tokens

Sr. No.	Lexeme	Token
1.	a	Identifier
2.	=	Operator
3.	B	Identifier
4.	+	Operator
5.	C	Identifier

In general, a token recognizer can be deterministic finite automata (DFA), non-deterministic finite automata (NFA) or non-deterministic finite automata with null transitions (NFA with null). Present paper describes an algorithm for design of token recognizer with a given suffix. As an example the recognizer shown in Figure 1 can be represented using a table called transition table as shown in Figure 2.

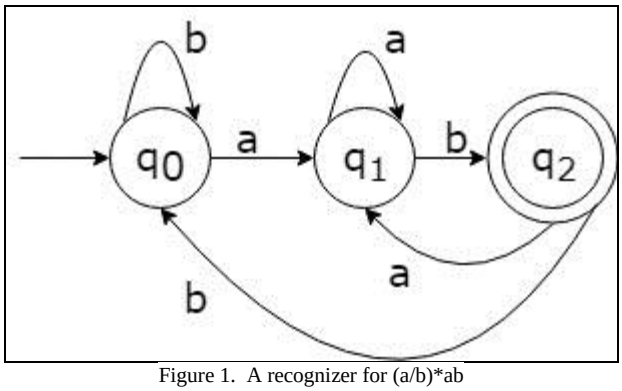

Figure 1. A recognizer for (a/b)*ab

	a	b
->q_0	q_1	q_0
q_1	q_1	q_2
(q_2)	q_1	q_0

Figure 2. Transition Table for (a/b)*ab

II. DETERMINISTIC FINITE AUTOMATA AS TOKEN RECOGNIZER

In Lexical analysis phase of compiler, source code is broken into small tokens and validity of tokens is checked according to the rules specified in any particular programming language. These rules are specified in the mathematical expressions called regular expressions [2]. These expressions are used to describe the tokens. The program that simulates the given regular expressions into DFA can be used to check whether the given token belongs to the specified language or not. If DFA accepts the given token, then it is a valid token otherwise it is invalid [3].

Deterministic finite automata commonly called as DFA is defined as a machine in which for each state and input symbol, a transition takes place. Mathematically, it can be represented by 5 tuples:

$$M= (Q, \sum, \delta, q_{0,} F) \quad (2)$$

Where

Q = Finite set of states,
$\sum$ = Input alphabet
δ = Transition function for any input symbol at particular state and is defined as:

$$\delta: Q \times \sum \rightarrow Q \quad (3)$$

q_0 =Initial state of a machine,
F= Set of final states of a machine

A state $d_o \in Q$ such that $\delta (d_o, a) = d_o \forall a \in \Sigma$ is called a dead state [4]. An example of DFA for all strings over $\sum = \{a, b\}$ having suffix 'ab' is shown in Figure 1 and corresponding transition table is shown in Figure 2. Here,

$$Q = \{q0, q1, q2\}, \quad (4)$$

$$\sum = \{a, b\}, \quad (5)$$

$$\delta (q0, a) =q1, \delta (q0, b) =q0, \quad (6)$$

$$\delta (q1, a) =q1, \delta (q1, b) =q2, \quad (7)$$

$$\delta (q2, a) =q1, \delta (q2, b) =q0, \quad (8)$$

q0 is an initial state, and (9)

$$F = \{q2\} \quad (10)$$

A token string is said to be recognized by a DFA if a final or accepting state can be reached starting from an initial state and processing the string one symbol at a time. For example, consider the token string w = 'baab' and process it in above DFA. The sequences of steps showing the recognition process are:

q0 --b--> q0 --a--> q1 --a--> q1 --b--> q2

As q2 is a final state therefore the token string 'baab' is recognized by the recognizer of Figure 1.

A. OTHER APPLICATIONS OF DFA

Apart from the importance of DFA in token recognition, there are many other applications based on use of automata. Some of these applications are:

a) Text processor: To search a text file for strings that matches with a given pattern, text processors or text filters also uses DFA like code [5].

b) Speech Processing: Speech processing and other signal processing systems often uses a DFA like technique to transform an incoming signal [6].

c) Pattern Matching: DFA is a simple language recognition device or a machine which recognizes the given input strings. Minimized DFA can be more helpful as it reduces the memory area required [7, 8].

d) Vending Machines: DFA can also be used in Vending machine in which value of coins can act as a state of a machine and only some combination of coins will lead to dispense the selected item [9].

e) Path finding DFA in Video Games having AI [9,10].

III. PROBLEM FORMULATION

As mentioned in previous section that DFA is of extreme importance in many applications including token recognition in compiler design. However, most of the computer science learners in this field face difficulty in designing Deterministic Finite Automata (DFA) due to requirement of high level of understanding [11]. JFLAP (Java Formal Languages and Automata Package) is a free and open source software tool used for designing finite automata, pushdown automata, and turing machines from the available transitions. Any kind of DFA can be designed in JFLAP manually. However, there has to be some mechanism for defining and presenting transitions for applying in JFLAP, which is helpful for understanding the design of DFA.

Due to the deterministic nature of a DFA, it is implementable in hardware and software for solving various specific problems [12]. There is no availability of well formatted algorithm for the automatic generation of transition table for recognizers which can recognize token. Therefore, there is a need of such algorithm which can facilitate automatic generation of transition table for language string recognizer. In the scope of present paper, an algorithm for automatic generation of transition table for a language strings having some given suffix is discussed. In this algorithm there is no kind of limitations on number and type of input symbols and also the length of given suffix.

IV. PROPOSED METHODOLOGY

Present section describes a simple and easily understandable approach to generate transition table for token recognizer with a given suffix. To generate transition table of token recognizer for a given suffix the following steps are applied:

1. Total number of states in the recognizer will be equal to length of given suffix plus one.
2. A transition for initial state (state 0) is defined to go to the next state (state 1) for the first symbol (at index 0) of suffix. For all remaining input symbols at initial state it will create self-loop and will stay at initial state.
3. Transitions for all remaining states i: At every state i some part of suffix is already traversed till index i-1. For state i and input symbol at index i in suffix, transition to state i+1 will take place.
4. For all remaining input symbol j at state i the algorithm will look for two longest matching substrings (with length L) in suffix: First from starting index to some index less than i and Second from some index greater than index zero to index i-1 in which input symbol j will be concatenated.
5. This matched substring of length L will tell that with input symbol j at state i the part of suffix recognized till this state will be of length L and the transition to state L will take place for input symbol j at state i.

The detailed algorithm for the generation of transition table of token recognizer for a given suffix is shown in Figure 3.

Algorithm: Generating a Transition Table for Token Recognizer with a given Suffix

Input: a) Alphabet Count (*inputs*), b) Alphabet symobols (*in_s[inputs]*), c) Enter desired suffix (*dfa_str[j]*)

Output: Transition Table for Token Recognizer with a given Suffix

Begin
1.initialization
  Set number of states *nostate* = length of *dfa_str* + 1,
  i=0, pcount=0, ptemp=0, j=0
2.transitions for inital state *i=0*
  for each input symbol *j* ∈ *in_s[]*
    (a[i][j] is transition for i^{th} state and j^{th} input symbol)
    if *in_s[j] = dfa_str[0]*
      a[i][j] = 1
    else
      a[i][j] = 0
3.i=1
4.for all the states except state *0* (i.e. upto nostate)
    set *ptemp* and *ptemp2* to *pcount*
    for each input symbol *j* ∈ *in_s[]*
      set *ptemp* to *ptemp2*
      if alphabet at i^{th} index in *dfa_str[]* = j^{th} symbol in *in_s[]*
        set *a[i][j] = i+1*
        set *max_s* to alphabet at i^{th} index in *dfa_str[]*
        set *max_e* to alphabet at $pcount^{th}$ index in *dfa_str[]*
        if max_s = max_e
          increment *pcount* by *1* (for remaining states)
        else
          loop until *pcount* becomes *zero*
            set *start* to substring of *dfa_str* from index *[0, pcount-1]*
            set *last* to subsstring of *dfa_str* from index *[i-pcount+1,i]*
            if *start = last*
              break the loop
            else
              pcount = pcount-1
      else if alphabet at $ptemp^{th}$ index in *dfa_str[]* = j^{th} symbol in *in_s[]*
        a[i][j] = ptemp+1
      else
        loop until *ptemp* becomes zero
          set *start* to substring of *dfa_str* from index *[0,ptemp-1]*
          set *last* to substring of *dfa_str* from index *[i-ptemp+1,i-1]*
          insert input symbol *j* at the end of '*last*'
          if *start=last*
            break the loop
          else
            ptemp=ptemp-1
        a[i][j]=ptemp
5.All transitions are completed
End

Figure 3. Proposed Algorithm for Token Recognizer for Given Suffix.

V. RESULTS AND DISCUSSION

The proposed algorithm is implemented using C++ Language using the concept of strings and vectors. The result is tested by generating transition table of token recognizer for various suffixes. Snapshot for one of the suffix 'abb' over Σ ={a,b} is shown in Figure 4. Transition table in Figure 4 is shown as output of token recognizer in which '->' represents initial state and '*' represents final state.
The string 'ababb' is checked for recognition using transition table constructed in Figure 4 and output is shown in Figure 5.

By using transition table generated by proposed algorithm in Figure 4, its corresponding token recognizer is constructed in JFLAP as shown in Figure 6. The same string 'ababb' is checked for recognition by a recognizer designed in Figure 6 in JFLAP also and its output is shown in Figure 7.

The results of string recognition by both transition table and JFLAP are same as both are showing the same state transitions and string 'ababb' as accepted (recognized). The string is shown as recognized when the last transition leads to the final state. In this way transition table of token recognizer for different suffixes are generated using algorithm in present paper and its corresponding token recognizer is constructed in JFLAP. Many strings are traversed in JFLAP for designed recognizer and all are giving the same results as of results given by algorithm in present paper.

```
Enter No. of input symbol
2
Enter input symbols:
a b
Enter Suffix:
abb
        |       a       b
________|_______________________
->q0    |       q1      q0
  q1    |       q1      q2
  q2    |       q1      q3
 *q3    |       q1      q0
```

Figure 4. Transition Table for suffix 'abb'

```
Enter string to be recognized
ababb

q0---a-->q1---b-->q2---a-->q1---b-->q2---b-->q3
 String Accepted
```

Figure 5. Token String Recognition Using Proposed Algorithm

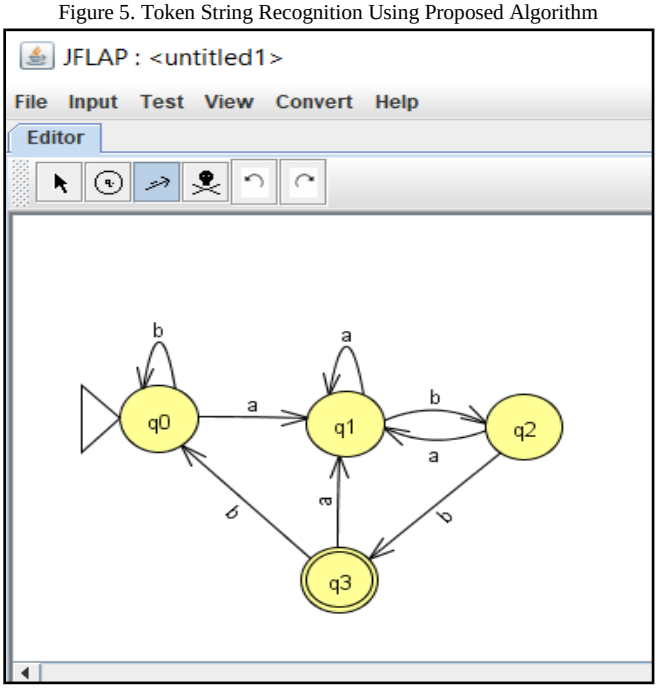

Figure 6. Token Recognizer Design using JFLAP

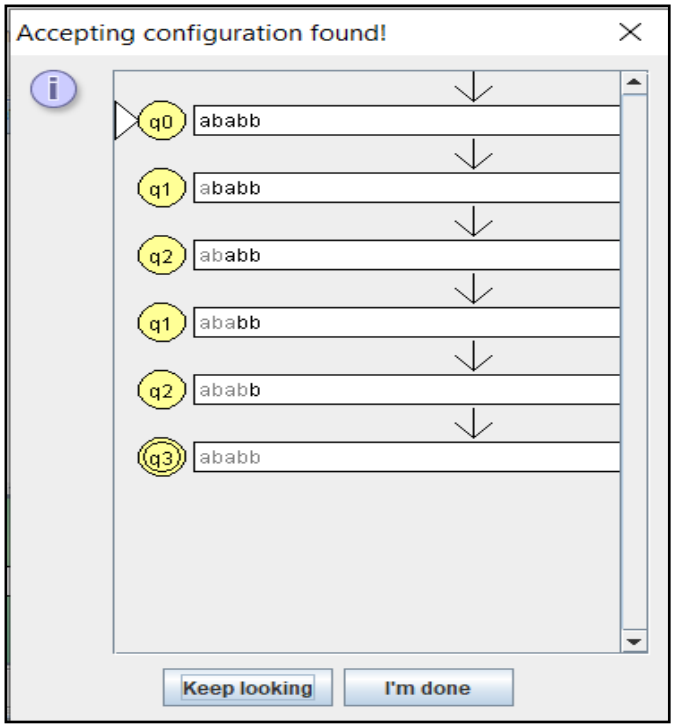

Figure 7. Token String Recognition Using JFLAP

VI. CONCLUSION AND FUTURE SCOPE

Token description and recognition is first and most important activity in compiler design. Constructing a transition table for a recognizer has been challenging task over years. Using JFLAP, transitions can be used for designing a recognizer but cannot be constructed, which is helpful for understanding the design of DFA. In JFLAP one has to give predefined transitions manually for desired token recognizer, which is a time consuming process. Present paper proposes an approach to implement the transition table for token recognizer with a given suffix. The results of proposed algorithm are used to design the recognizer using JFLAP. In present paper, the algorithm designed for implementing token recognizer using DFA with a given suffix, transition table can be constructed automatically and stores the DFA in a well formatted transition table. In future the algorithm can be made simpler and its time and space complexity can also be taken into consideration to make the algorithm computationally more efficient. Further, the algorithm can be extended to other types of token strings also.

REFERENCES

[1] Aho A. V., Ullman J. D., *"Principles of Compiler Design"*, Narosa Publishing House, 2002

[2] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman, *"Automata Theory, Language, and Computation"*, Delhi: Pearson, 2008.

[3] Aparna, Goyal G., *"Application Review of Automata Theory"* International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Vol. 3, Issue 1, pp. 947-955, 2018.

[4] Parekh, R. G., Honavar, V. G., *"Learning DFA from Simple Examples"* Journal of Machine Learning, Vol. 44, Issue 1-2, pp. 9-35, 2001.

[5] Webber A. B., *"Formal Language: A Practical Introduction",* Franklin, Beedle & Associates Inc., Wilsonville, pp. 35-43, 2008.

[6] 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.

[7] 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

[8] 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

[9] Gribko E. *"Applications of Deterministic Finite Automata"* ECS 120 UC Davis, Spring 2013, pp. 1-9, 2013.

[10] Raj N., Dubey R., *"Snakes and Stairs Game Design using Automata Theory"*, International Journal of Computer Sciences and Engineering, Vol. 5, Issue 5, pp.58-62, 2017

[11] Shenoy V., Aparanji U., Sripradha K., Kumar V., *"Generating DFA Construction Problems Automatically"* International Journal of Computer Trends and Technology, Vol. 4, Issue 4, pp.32-37 ,2013

[12] Pant Y., *"A Novel Approach to Minimize DFA State Machines Using Linked List",* International Journal of Scientific Research in Computer Science and Engineering, Vol. 6, Issue 4, pp. 41-45, 2018

Authors Profile

Mr. Rajanshu Goyal is pursuing Bachelor of Engineering in Computer Science & Engineering from Chandigarh College of Engineering & Technology (Degree Wing), Chandigarh (India). He is a member of ACM since 2017. His main research work focuses on Theoretical Computer Science and Data Structures & Algorithm.

Dr Gulshan Goyal has persued Bachelor of Technology, Master of Technology and Ph.D. in Computer Science & Engineering. He is currently working as Assistant Professor in Department of Computer Science & Engineering at Chandigarh College of Engineering & Technology (Degree Wing), Chandigarh (India). He has published many papers in reputed Journals and Conferences. His main research work focuses on Digital Image Processing, Theoretical Computer Science and Algorithm Design. He has more than 16 years of teaching experience.

