International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:6, 2014

The Different Ways to Describe Regular Languages by Using Finite Automata and the Changing Algorithm Implementation

Abdulmajid Mukhtar Afat

***Abstract*—**This paper aims at introducing finite automata theory, the different ways to describe regular languages and create a program to implement the subset construction algorithms to convert nondeterministic finite automata (NFA) to deterministic finite automata (DFA). This program is written in c++ programming language. The program reads FA 5tuples from text file and then classifies it into either DFA or NFA. For DFA, the program will read the string *w* and decide whether it is acceptable or not. If accepted, the program will save the tracking path and point it out. On the other hand, when the automation is NFA, the program will change the Automation to DFA so that it is easy to track and it can decide whether the *w* exists in the regular language or not.

***Keywords*—**Finite Automata, subset construction DFA, NFA.

I. Introduction

THE automata have been resulted through many years by intellective development where it is try to reducing off human interference. In computing field, automata theory represents abstract computing, where it is computing without middle software. Moreover automata exist in most computer science fields.

Therefore, it is very important to invest more time studying automata theories and to build and develop large systems such as programming language compilers.

II. FA Uses

Finite Automata are used as a model for

- Software for designing digital circuits.
- Lexical analyzer of a compiler.
- Searching for keywords in a file or on the web.
- Software for verifying finite state systems, such as communication protocols.
- Designing the states of large systems.
- Natural Language Processing.

III. Paper Background

A. FA Definition

A finite automaton has a set of states, and it is "control" moves from state to state in response to external inputs [1].

Abdulmajid Afat is with the Faculty of Information Technology, Misurata University, Misurata, Libya (Phone: 00218924958946; e-mail: abdoo_84_2004@yahoo.com).

B. FA Types

There are two main types of finite automata and this types is according to the type of movement [3]:

- DFA Deterministic Finite Automat: one input at one time movement will be in just one state.

DFA is easy to track, so it is easy to build the program and perform tracking functions. However, it is hard to design.

DFA quintuples/ definition [4] ($Q,\sum,\delta,q0,F$)

- Q Is finite set of states.
- $\sum$ is finite set of alphabet (input symbols).
- q0 is start state.
- δ is transition function $\delta(q,a)=p$.

a input symbol.
q,p states in Q.

- F subset of Q.

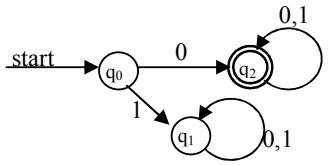

Fig. 1 Automation of DFA for L(A) [2] L(A)={w: w in $\{0,1\}^*$ and w value is even}

- NFA Nondeterministic Finite Automata: one input at one time and the movement may be in several states.

Although NFA is easy to design, it is difficult to track. Therefore, it must be changed to DFA by using subset construction algorithm.

NFA quintuples ($Q,\sum,\delta,q0,F$)

- Q Is finite set of states.
- $\sum$ is finite set of alphabet (input symbols).
- q0 is start state.
- δ is transition function $\delta(Y,a)=X$.

a input symbol.
Y subset of Q.
X subset of Q.

- F subset of Q.

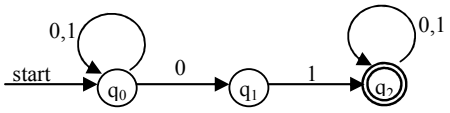

Fig. 2 Automation of NFA for L(A) L(A)={w: w =x01y and w,x,y in $\{0,1\}^*$ }

IV. Formal Languages Concepts

- A formal language is a set of words, i.e. finite strings of letters, symbols, or tokens. The set from which these letters are taken is called the alphabet over which the language is defined [5].
- Alphabet: finite set of symbols

  Examples: {**0**,**1**} (the *binary* alphabet).

$$\{a,b,c,...,z\}$$

- String ***(w):*** sequence of symbols chosen from some alphabet.

  Examples: **01101,** *cin*
- Language***:*** set of strings chosen from some alphabet.

  Examples:
  - the set of string consisting on n 0's followed by n 1's.
    - {ε,01,011,1010,…}
  - The set of compliable C programs.
  - The set of Arabic words.
- Powers of an Alphabet

If **Σ** is an alphabet, define $\mathbf{\Sigma}^{k}$ to be the set of strings of length *k*, consisting of symbols in **Σ**.

The set of *all* strings over **Σ** is denoted $\mathbf{\Sigma}^{*}$. As the following:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$
$$\Sigma^0 = \{\varepsilon\}$$
$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots.$$
$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

V. Regular Languages

Transition Function Extension δ': δ' it is extending δ to be applied on q and string *w*.

Base**:** $\delta'(q_0,\varepsilon) = q_0$

Induction**:** if $w = xa$ where *x* is substring and *a* is an input then

$$\delta'(q_0,xa) = \delta(\delta'(q_0,x),a)$$

Define: language for DFA of $A=(Q,\sum,\delta,q0,F)$

$$L(A)=\{w:\delta'(q_0, w) \text{ in } F\}$$

Define: language for NFA of $A=(Q,\sum,\delta,q0,F)$

$$L(A)=\{w:\delta'(q_0, w) \text{ is subset of } F\}$$

So regular language is language can represent it by automaton FA (DFA or NFA)

VI. Subset Construction Algorithm

Subset construction algorithm is use to change NFA to DFA because NFA is impossible to track. However, if changed to DFA, the automation will be easier to track, and then to build program that represents the DFA. So, for each NFA, there is an equivalent DFA.

Subset construction algorithm tuples:

- $Q_D = \{S:S \text{ subset of } Q_N\}$
- $F_D = \{S:S \text{ subset of } Q_N \text{ and } S \cap F_N \neq \emptyset\}$
- For every S subset of Q_N and *a* in $\sum \delta_D(S,a) = U_{p \text{ in } s}\, \delta_N(p,a)$
- $q_{oD} = q_{oN}$
- NFA and DFA have the same $\sum$.

Example:

The resulted transition table after implement subset construction algorithm on the automation in Fig. 2 is the following.

TABLE I
DFA Transition Diagram

	0	**1**
q_0	q_0, q_1	q_0
q_0, q_1	q_0, q_1	q_0, q_2
q_0, q_2	q_0, q_1,q_2	q_0, q_2
q_0, q_1,q_2	q_0, q_1,q_2	q_1,q_2
q_1,q_2	q_2	q_2
q_2	q_2	q_2

Note: $Q_D = \{S:S \text{ subset of } Q_N\}$, so $|Q_D| = 2^n - 1 = 7$ in the previous example. If assume that $|Q_N| = 10$ then $|Q_D|$ in this case will be $2^{10}-1 = 1023$. It is so hard and expensive (time and memory). Therefore it is better to use rules for accessible States.

Rules for Accessible States:

Base: q_0 is accessible state.

Induction: if S accessible state then $U_{a \in \sum} \delta(S,a)$.

The Resulted diagram is the following:

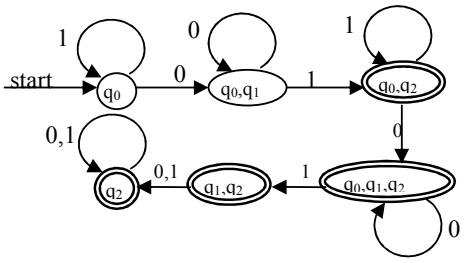

Fig. 3 Automation of DFA for L(A) L(A)={w: w =x01y and w,x,y in $\{0,1\}^*$ }

VII. Problem Definition

In automata theory, the problem is the question of deciding whether a given string *w* is a member of some particular language [1].

If L language over $\sum^*$ and *w* in $\sum^*$, then the problem is deciding whether *w* in L or not.

VIII. Design

The starting point is reading quintuples of FA. The quintuples will be in text file. Thus the program will classify the FA (DFA/NFA) According to quintuples.

- Program inputs:
  - FA quintuples.
  - The string *w*.
- Changing NFA to DFA if NFA
- Program outputs:
  - Deciding if *w* accepted or rejected.
  - Saving the tracking path to be input for other program. Tracking path is all stats were passed by the word *w*.
  - Print out the new quintuples if entered FA was NFA.

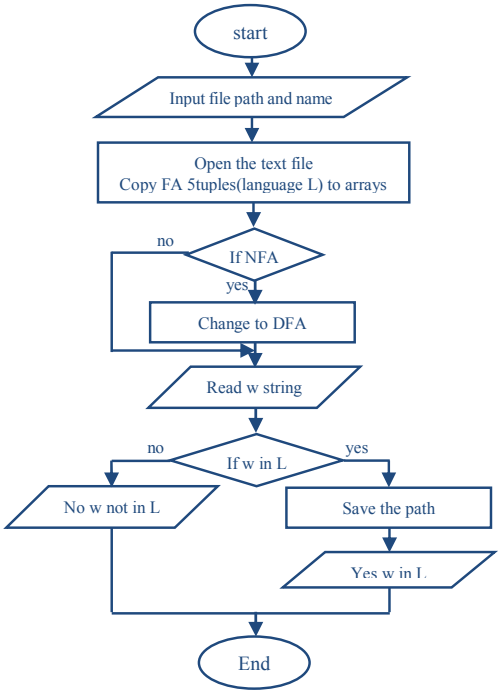

Fig. 4 Program flowchart

IX. Testing

The program has been tested to the following FA's:

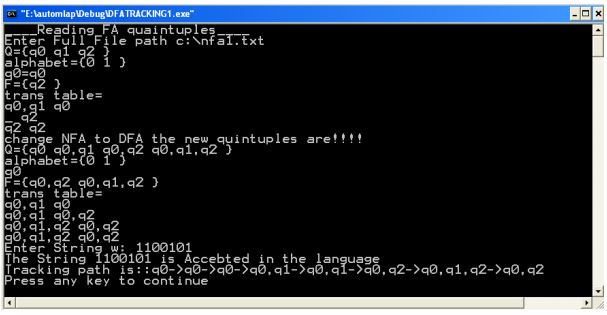

Fig. 5 Testing the program on the example in Fig. 2

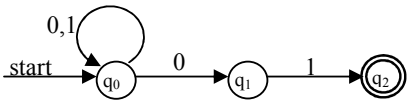

Fig. 6 Automation of NFA for L(A) L(A)={w: w =x01 and w,x in $\{0,1\}^*$ }

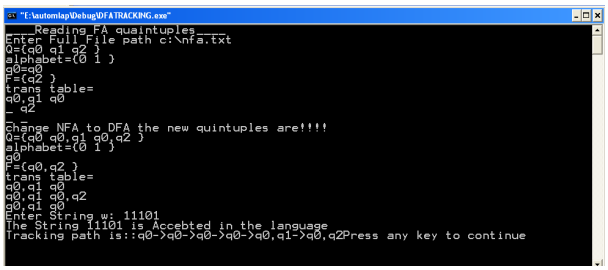

Fig. 7 Testing the program on the example in Fig. 6

X. Program Functions

1. Copyquaitupletoarrays() reading quintuples from text file and save it to arrays.
2. isitNFA() testing the FA(NFA/DFA).
3. NFAtoDFA() convert the NFA to DFA.
4. FinalStates() specify the new final state.
5. Delrep() delete repeting from string.
6. Tracking() return 1 if woard in language/ 0 if not.
7. Other simple function like copy, print arrays.

XI. NFATODFA Source Codes

Source code for subset construction algorithm in c++ language:

```
 void NFAtoDFA(stt seg[50],stt Q[50],stt fs[50],stt
tt[50][50],char q0[],int &Ql,int &segl,int &fsl,int &ttc )
{
stt newQ[50], newtt[50][50];
int newQl;
int i,j,w,f1,tr;
char buf[20];

strcpy(newQ[0].s,q0);
newQl=1;

for(i=0;i<segl;i++)
strcpy(newtt[0][i].s,tt[0][i].s);
int newttR=1;

for(i=0;i<newttR;i++)
for(j=0;j<segl;j++)
{
 f1=0;
for(int m=0;m<newQl;m++)
if (strcmp(newtt[i][j].s,newQ[m].s)==0) f1=1;

if ((f1==0)&&strcmp(tt[i][j].s,"_")!=0)
   {
   strcpy(newQ[newQl].s,newtt[i][j].s);
```

```
   for(w=0;w<=segl;w++)
   newtt[newttR][w].s[0]='\0';
int    ta=0;
   for(int x=0;newQ[newQl].s[x]!='\0';x++)
     {
      if (newQ[newQl].s[x]!=',')
          {
          buf[ta]=newQ[newQl].s[x];
          ta=ta++;
          }
       if ((newQ[newQl].s[x]==',') ||
(newQ[newQl].s[x+1]=='\0'))
          {
          buf[ta]='\0';
          for( w=0;w<Ql;w++)
          if (strcmp(buf,Q[w].s)==0) tr=w;
          for(w=0;w<segl;w++)
          if(strcmp(tt[tr][w].s,"_")!=0)
          {
          strcat(newtt[newttR][w].s,tt[tr][w].s);
          strcat(newtt[newttR][w].s,",");
          }
          strcpy(buf,"");
          ta=0;

          }
     }
     for( w=0;w<segl;w++)
   if (newtt[newttR][w].s[strlen(newtt[newttR][w].s)-1]==',')
   newtt[newttR][w].s[strlen(newtt[newttR][w].s)-1]='\0';
   for(w=0;w<segl;w++)
      delrep(newtt[newttR][w].s);
   for(w=0;w<segl;w++)
   if (strcmp(newtt[newttR][w].s,"")==0)
strcpy(newtt[newttR][w].s,"_");
   newttR++;
   newQl++;
   }
}
```

REFERENCES

[1] Jhone E.Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introduction to automata theory, languages and computation. By Addison- Wesley 2nd Edition, 2001.
[2] Tarek Majid. Theory of Cmputation. Amman- Jordan 1st Edition2005.
[3] S.P. Eugene Xavier. Theory of Automata, Formal Languages and Computation. By New Age International (P) Ltd, 2005.
[4] Finite Automata. Available at https://www.cs.rochester.edu/u/nelson/courses/csc_173/fa/fa.html (Accessed Sep 2014).
[5] Formal language. Available at https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Formal_language.html (Accessed Sep 2014).