International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:13, No:3, 2019

Using ε Value in Describe Regular Languages by Using Finite Automata, Operation on Languages and the Changing Algorithm Implementation

Abdulmajid Mukhtar Afat

***Abstract*—**This paper aims at introducing nondeterministic finite automata with ε value which is used to perform some operations on languages. a program is created to implement the algorithm that converts nondeterministic finite automata with ε value (ε-NFA) to deterministic finite automata (DFA).The program is written in c++ programming language. The program inputs are FA 5-tuples from text file and then classifies it into either DFA/NFA or ε -NFA. For DFA, the program will get the string *w* and decide whether it is accepted or rejected. The tracking path for an accepted string is saved by the program. In case of NFA or ε-NFA automation, the program changes the automation to DFA to enable tracking and to decide if the string *w* exists in the regular language or not.

***Keywords*—**Finite automata, DFA, NFA, ε-NFA, Eclose, operations on languages.

I. INTRODUCTION

A regular language is a language that can be presented by FA (DFA/ NFA). Some automation may needs to move in ε value. ε value is also crucial to implement operations on languages. Such operations combine a group of languages into one language. For these reasons there is more addition blue print of NFA which **(ε-NFA)**. so we can imagine the result will be complicated and more useful.

Therefore, it is very important to invest more time reviwing automata theories and formal languages operations to desgin complicated or composet language that help to build and develop many kinds of software or applcations.

II. EPSILON VALUE

Epsilon value (ε) mean the empty string, where some times the automation need to move between states in ε.

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

A finite automata has two classification DFA and NFA.

E-NFA it is another extension of FA, the new "feature" is that we allow a transition on ε, the empty string. In effect, an NFA allowed to make transition spontaneously, without receiving an input symbol [1], [2].

There are two uses of ε value:

- **Design ε-NFA.**

To design FA to the decimal numbers, it is so hard to design DFA directly, and it is impossible to design NFA without ε value, because we need ε to the start stat, where any decimal number maybe stat with +/- or start with no sign(start with ε value). As the transition diagram:

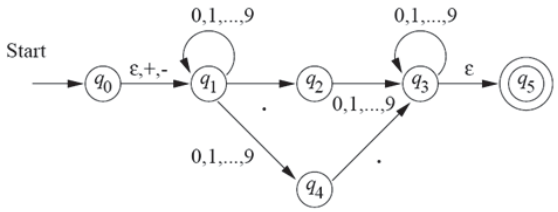

Fig. 1 An ε-NFA accepting decimal numbers L(A) [1] L(A)={w: w is decimal number}

- converting regular expression REGEX (Regular expression) to DFA.
- **Making some operation on languages.**

There are some concept of languages:

- ***Alphabet***: finite set of symbols
- Examples: {**0**,**1**} – {on,off}-{a-z}.
- ***String (w):*** text string from an alphabet.
- Examples: **010101, science.**
- ***Language:*** set of strings or words chosen from some alphabet.
- ***A set*** is a well-defined collection of objects [4].

A regular language over has an explicit formula. A regular expression for the language is a slightly more user friendly formula and it is recognized by FA [5].

Operations on languages:

If L and M are languages then:

Union :

$L \cup M=\{w : w \in L \text{ or } w \in \}$

Concatenation:

$L . M=\{w : w=xy \text{ and } w \in L \text{ and } w \in \}$

Power:

$L^0=\{\varepsilon\}$, $L^1=L$, $L^{k+1}=L.L^k$

Kleen closure:

$L^*= U^4_{i=0} L^i$

$L^*= L^0 \cup L^1 \cup L^2 \cup L^3 \ldots$

The end result of all previous operations is new language, so the operation provide help to produce a new and a complicated languages. Therefore the problem can be divided to small

Abdulmajid Afat is with the Faculty of Information Technology, Misurata University, Misurata, Libya (phone: 00218924958946; e-mail: majid.afat@it.misuratau.edu.ly).

pieces and the dealing with each piece (design language for each piece) as the following:

The union operation.

L(A)={w:w contain01 or contain on 11, w in{0,1}*}

Problem dividing into to pieces:

1. L(A)={*w*:*w* contain on 01, *w* in{0,1}*}

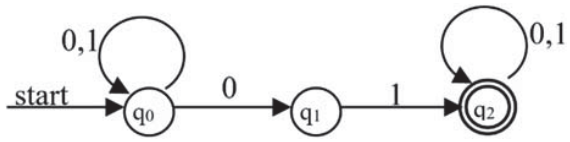

Fig. 2 An NFA for L(A) ={x01y: x,y in {0,1}*}

2. L(A)={*w*:*w* contain on 11 , *w* in{0,1}*}

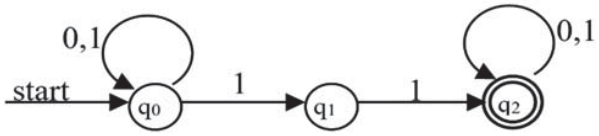

Fig. 3 An NFA for L(A) ={x11y: x,y in {0,1}*}

Using ε value to composed the pieces:

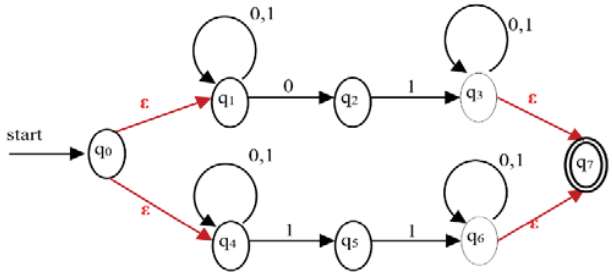

Fig. 4 An ε-NFA for L(A) ={w: w in {0,1}* And w contain on 01 Or 11}

The Concatenation operation:

L(A)={w:w contain01 and contain on 11 , w in{0,1}*}

After dividing the language to pieces, connect the pieces by ε value as the following:

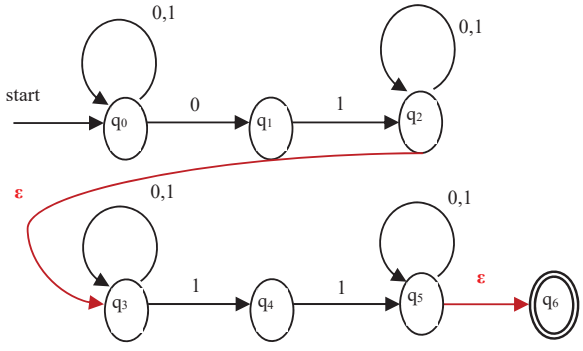

Fig. 5 An ε-NFA for L(A) ={w: w in {0,1}* And w contain on 01 And 11}

III. Regular Expressions

Regular expressions are another type of language-defining notation [1].

The regular languages are those languages that can be constructed from the "big three" set operations viz., (a) Union (b) Concatenation (c) Kleene star [3].

By using regular expression the accepted strings expressed to one expression, where it is consider as declarative way to describe a language. Sometimes regular expressions can do what automata do not and vice versa. The regular expression are used as input to some systems such that:

Search command.

Lexical analyzer generator.

The languages operation can be implemented to the regular expression.

Inductive definition of regular expression:

Basis:

ε is a regex and Ø is a regex. L(ε)={ε}, and L(Ø)= Ø.

If a in ∑ then a is a regex. L(a)={a}.

Induction:

If E is a regex's, then (E) is a regex. L((E))=L(E).

If E and F are regex's, then E+F is a regex. L(E+F)=L(E) U L(F).

If E and F are regex's, then E.F is a regex. L(E.F)=L(E) . L(F).

If E is a regex's, then E* is a regex. L(E*)= (L(E))*.

Every byte or letter in regular expression can be considered as one regex.

Examples: regular expression to the following languages.

- L(A)={w:w contain01, w in{0,1}*}
  $(0+1)^*01(0+1)^*$
- L(A)={w in{0,1}*: 0 and 1 are alternate in w}
  $(01)^*+ (10)^*+1(01)^*+0(10)^*$
  Or, equivalently
  $(\varepsilon+1) (01)^* (\varepsilon+0)$
- L(A)={w : w is a decimal numbers}
  $(\varepsilon+'+'+'-')(0123456789)^*('.') (0123456789)^*$

Equivalence of FA's and regex's:

For each DFA's, NFA's, ε-NFA's and regex's are equivalent. where every language can be presented by one of them.

To change regex to DFA. First set changing the regex to ε-NFA, and then changing the resulted ε-NFA to DFA.

From regex's to ε-NFA:

Starting by simple regex and getting the automata for ε, Ø and *a*.

Fig. 6 The automata for ε, Ø and *a*

Induction: Automata for R+S, RS and R*.

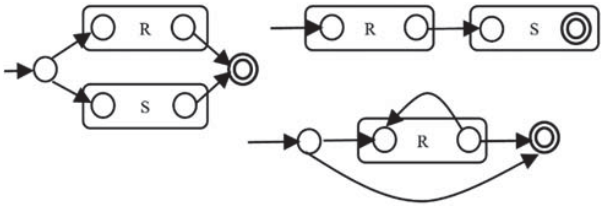

Fig. 7 The automata for ε, Ø and *a*

Finally by connected diagrams by ε value. The resulted diagram is ε-NFA.

Example: L(A)={w:w end by 01, w in{0,1}*}

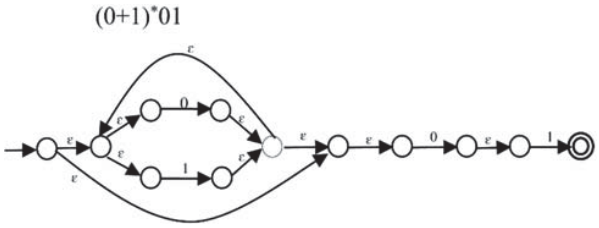

Fig. 8 Equivalence ε-NFA to the regex (0+1)*01

IV. Changing E-NFA to DFA

To change ε-NFA to DFA we have to define the ECLOSE first.

ECLOSE to the state P: it is set content on all states can be reachable by sequence of ε starting with state q.

Basis:

p **0** ECLOSE(p)

Induction:

q **0** ECLOSE(p) and $r \in \delta(q,\varepsilon) \mapsto r \in$ ECLOSE(p)

as an example in fig.1 ECLOSE(q_0) ={q0,q1}

the detail of construction algorithm:

- Q_D={$S : S \phi Q_E$ and S = ECLOSE(S)}
- q_D = ECLOSE(q_0)
- F_D={$S : S \in Q_D$ and S 1 $F_E \neq \emptyset$}
- $\delta_D(S,a)$ = U$_{\{ECLOSE(P) : P \in \delta(t,a) \text{ for some } t \in S\}}$

Note: for Q_D the accessible state only will be included, where the first accessible stat is q_D = ECLOSE(q_0).

The transition diagram in Example in fig 1 is.

TABLE I
E-DFA Transition Diagram

	ε	+,-	.	0,…,9
q0	q1	q1	Ø	Ø
q1	Ø	Ø	q2	q1,q4
q2	Ø	Ø	Ø	q3
q3	q5	Ø	Ø	q3
q4	Ø	Ø	q3	Ø
***q5**	Ø	Ø	Ø	Ø

After implement the construction algorithm the new transition table:

TABLE II
DFA Transition Diagram

	+,-	.	0,…,9
q0,q1	q1	q2	q1,q4
q1	Ø	q2	q1,q4
q2	Ø	Ø	q3,q5
q1,q4	Ø	q2,q3,q5	q1,q4
***q3,q5**	Ø	Ø	q3,q5
***q2, q3,q5**	Ø	Ø	q3,q5

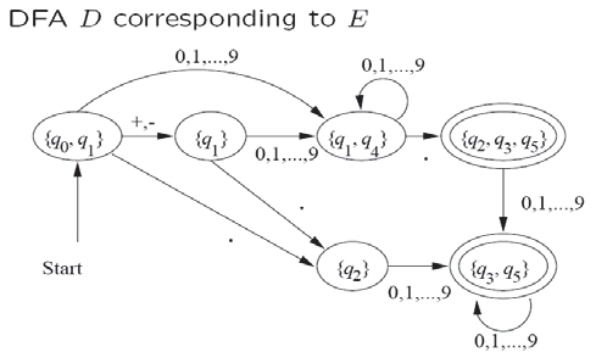

Fig. 9 DFA accepting decimal numbers [1] L(A)={w: w is decimal number}

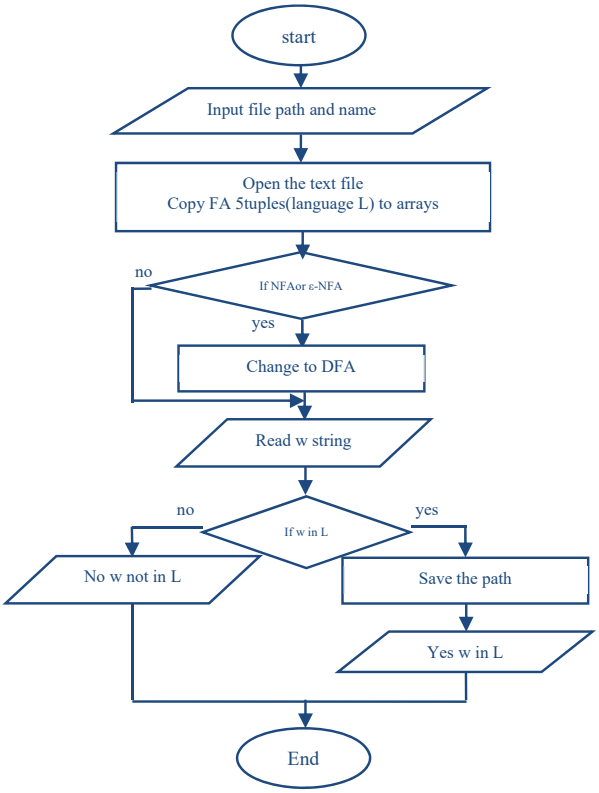

Fig. 10 Program flowchart

V. Design

In design phase, the starting point is reading 5-tuples of FA from text file. Thus the program will classify the FA (DFA/NFA or ε-NFA) According to quintuples.

- Program inputs:
  - FA quintuples.
    - Q is finite set of states.
    - ∑ is finite set of alphabets (input symbols).
    - q_0 is start state.
    - δ is transition function δ(q,*a*)=p.
    - F subset of Q.
  - The string *w*.
- Changing FA to DFA if NFA or ε-NFA
- Program outputs:
  - Deciding if w accepted or rejected.
  - Saving the tracking path.
  - Printing the new quintuples if entered FA was NFA or ε-NFA.

VI. Testing

The program has been tested the automation in Fig. 1:

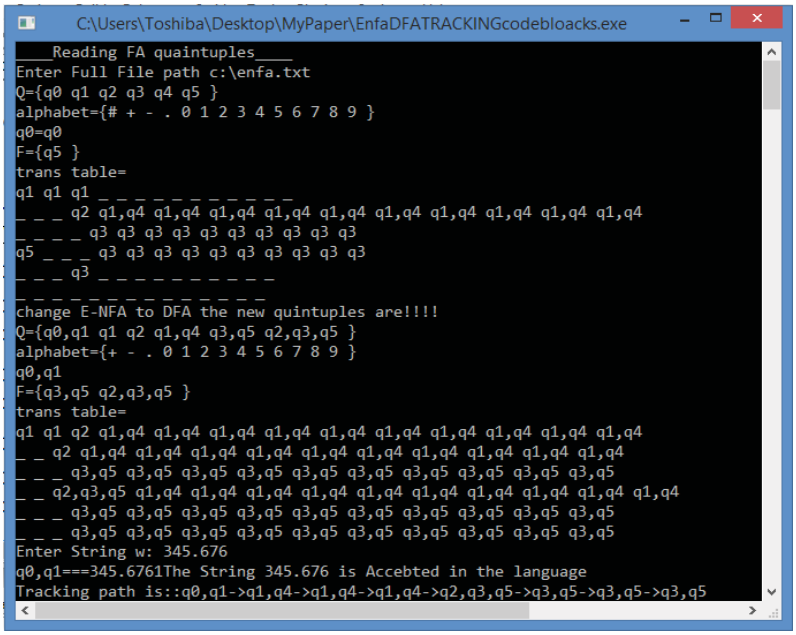

Fig. 11 Program result

VII. Program Functions

1. Copyquaitupletoarrays() reading text file consist of FA tuples and save them to matrices.
2. isitNFA() testing the FA(NFA/DFA or ε-NFA).
3. NFAtoDFA() convert the NFA to DFA.
4. ENFAtoDFA() convert the ε-NFA to DFA.
5. FinalStates() specify the new final state.
6. Eclose() finding single state e-close.
7. EcloseState()finding composite state e-close
8. Delrep() delete repeting from string.
9. Tracking() return 1 if woard in language/ 0 if not.
10. Other simple function like copy, print arrays.

VIII. NFA to DFA Source Code

source code for ε-NFA to DFA algorithm in c++ language:

```
void ENFAtoDFA(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],a[50];
int newQl,n;
int i,j,w,f1,tr;
char buf[20],prclose[50];
Eclose(prclose,Q[0].s,tt,Q,Ql);
strcpy(newQ[0].s,prclose);
strcpy(q0,prclose);
newQl=1;
int newttR=0;
split (newQ[0].s,a,n);
for(i=0;i<n;i++)
{
for( w=0;w<n;w++)
        if (strcmp(a[i].s,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);
        if (i<n-1)strcat(newtt[newttR][w].s,",");
        else newtt [newttR][w].s[2*n+(n-1)]='\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++)
   if (strcmp(newtt[newttR][w].s,"")==0)
strcpy(newtt[newttR][w].s,"_");
for(w=0;w<segl;w++)
```

```
{
EcloseState(prclose,newtt[newttR][w].s,tt,Q,Ql);
strcpy(newtt[newttR][w].s,prclose);
}
newttR=1;
int r,c,pp;
newtt[0][0].s[2]='\0';
for(r=0;r<newttR;r++)
for(c=0;c<segl;c++)
{
   f1=0;
   for(int m=0;m<newQl;m++)
   if (strcmp(newtt[r][c].s,newQ[m].s)==0) f1=1;
   if ((f1==0)&&strcmp(newtt[r][c].s,"_")!=0&&c!=0)
      {
        strcpy(newQ[newQl].s,newtt[r][c].s);
        split (newQ[newQl].s,a,n);
         for(i=0;i<n;i++)
         {
                  for(w=0;w<Ql;w++)
                     if (strcmp(a[i].s,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);
                  if (i<n-1)strcat(newtt[newttR][w].s,",");
                  else strcat(newtt[newttR][w].s,"\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++)
            if (strcmp(newtt[newttR][w].s,"")==0)
strcpy(newtt[newttR][w].s,"_");

         for(w=0;w<segl;w++)
         {
            if(strcmp(newtt[newttR][w].s,"_")!=0)
            {
            EcloseState(prclose,newtt[newttR][w].s,tt,Q,Ql);
            strcpy(newtt[newttR][w].s,prclose);
            }
         }
         newttR+=1;
         newQl+=1;
      }

}
for(i=0;i<newttR;i++)
for(j=0;j<segl;j++)
     newtt[i][j]=newtt[i][j+1];
 newttR=newttR-1;
for(j=0;j<segl;j++)
seg[j]=seg[j+1];
segl=segl-1;
FinalStates(newQ, newQl, fs,fsl);
CopyArray(Q,Ql,newQ,newQl);
for(i=0;i<newQl;i++)
for(int j=0;j<segl;j++)
tt[i][j]=newtt[i][j];
Ql=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] K. l. P. Mishra, N. Chandrasekaran. Theory of Computer Science Automata, languages and Computation. third Edition, 2008.
[5] John C. Martin. Introduction to languages, and the theory of computation. By McGraw-Hill 4th, 2011.