**Tutorial Problem:**

**1, Create NFA for each of the following**

**( a | b)                  ID\_1;**

**( a (b\*) a )              ID\_2;**

**( a b a )+           ID\_3;**

**2. Construct a single DFA for the above token specifications. using subset construction**

**3. Use the DFA to get tokens from the following input strings based on the longest match convention:--**

**abaabaabbabaaaaabaaaa**

**Nondeterministic Finite Automata**

A nondeterministic finite automaton (NFA) consists of:

1. A finite set of states S.

2. A set of input symbols �, the input alphabet. We assume that E, which stands for the empty string, is never a member of �.

3. A transition function that gives, for each state, and for each symbol in � U {E} a set of next states.

4. A state 80 from S that is distinguished as the start state (or initial state) .

5. A set of states F, a subset of S, that is distinguished as the accepting states (or final states) .

We can represent either an NFA or DFA by a transition graph, where the nodes are states and the labeled edges represent the transition function. There is an edge labeled a from state 8 to state t if and only if t is one of the next states for state 8 and input a. This graph is very much like a transition diagram, except:

**Algorithm 3.22 :**

**Simulating an NFA.**

**INPUT**: An input string x terminated by an end-of-file character eof. An NFA N with start state So, accepting states F, and transition function move.

OUTPUT: Answer "yes" if M accepts x; "no" otherwise.

**METHOD:**

The algorithm keeps a set of current states S, those that are reached from So following a path labeled by the inputs read so far. If c is the next input character, read by the function nextCharO, then we first compute move(S, c) and then close that set using E-closureO. The algorithm is sketched in Fig. 3.37. o

**1) S = E-closure(so);**

**2) c = nextCharO;**

**3) while ( c != eof) {**

**4) S = E-closure( move(S, c));**

**5) c = nextCharO;**

**6) }**

**7) if ( S n F != 0 ) return "yes ";**

**8) else return "no" ;**

**1.**

a **ID1**

b

**ID2**

a a

**ID3**

a b a

**Conversion of an NFA to a DFA**

*E-closure( s ) :Set of NFA states reachable from NFA state s on E-transitions alone. E-closure( T): Set of NFA states reachable from some NFA state s in set T on E-transitions alone; = Us in T E-closure(s) .*

*move(T, a) :Set of NFA states to which there is a transition on input symbol a from some state s in T.*

t-closure(so) is the only state in Dstates, and it is unmarked;

while ( there is an unmarked state T in Dstates )

{

mark T;

for ( each input symbol a ) {

U = t-closure( move(T, a));

if ( U is not in Dstates )

add U as an unmarked state to Dstates;

Dtran[T, a] = U;

}

}

**Deterministic Finite Automata**

A deterministic finite automaton (DFA) is a special case of an NFA where a E st� b Figure 3.26: NFA accepting aa\* lbb\*

1. There are no moves on input E, and

2. For each state S and input symbol a, there is exactly one edge out of s labeled a.

If we are using a transition table to represent a DFA, then each entry is a single state. we may therefore represent this state without the curly braces that we use to form sets

**s = so;**

**c = nextCharO;**

**while ( c != eof)**

**{**

**s = move(s, c) ;**

**c = nextCharO;**

**}**

**if ( s is in F ) return "yes ";**

**else ret urn "no"** ;

**string:str=”abbba”**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **intial** | **Iter-2** | **Iter-3** | **Iter-4** | **Iter-5** | **Final** |
| **S=A** | **S=B** | **S=E** | **S=E** | **S=E** | **S=D** |
| **C=a** | **C=b** | **C=b** | **C=b** | **C=a** | **C=eof** |
| **C!=eof** | **C!=eof** | **C!=eof** | **C!=eof** | **C!=eof** |  |
| **S=>B** | **S=>E** | **S=>G** | **S=>G** | **S=>D** |  |
| **C=b** | **C=b** | **C=b** | **C=a** | **C=eof** |  |

**D is final state which is ID\_2 so this string accepted by ID\_2**

b

a

a

a

a

a

b

a

b

b

a

b

**2.Construct a single DFA using subset construction**

The start state A of the equivalent DFA is t-closure(O), or

**A = {0,1,2,4,7,150,16}**, slnce these are exactly the states reachable from state 0 via a path all of whose edges have label t. Note that a path can have zero edges, so state 0 is reachable from itself by an t-Iabeled path.

The input alphabet is {a, b} . Thus, our first step is to mark A and compute Dtran[A, a] = -closure(rriove(A, a))

and Dtran[A, b] = -closure(move(A, b)) . Among the states 0, 1, 2, 4, 7,15 and 16, only 2 , 7 and 16 have transitions on a, to 3, 8 and 17 , respectively.

Thus, move(A, a) = {3, 8,17}.

Also, -closure( {3, 8,17}) = {6,9,10,12,18}=B

so we conclude

mov(A,b)={5}

-closure(mov(A,b)) =-closure({5}) ={6}=C

***For state B:***

-closure(mov(B,a)) =-closure(mov({6,9,10,12,18},a)) =-closure({14}) ==ID\_2=D

-closure(mov(B,b)) ==-closure(mov({6,9,10,12,18},b)) =-closure({11,19}) ={10,12,13,20}=E

***For state E***

-closure(mov(E,a)) =-closure(mov({10,12,13,20},a)) =-closure({14,21}) ={15,16,22}=F=(ID\_2 and ID\_3)

-closure(mov(E,b)) ==-closure(mov({10,12,13,20},b)) =-closure({11}) ={10,12,13}=G

***For state F***

-closure(mov(F,a)) =-closure(mov({15,16,22},a)) =-closure({17}) ={18}=H

-closure(mov(F,b)) ==-closure(mov({15,16,22},b)) =-closure({}) =

***For state G*:**

-closure(mov(G,a)) =-closure(mov({10,12,13},a)) =-closure({14}) =

-closure(mov(G,b)) ==-closure(mov({10,12,13},b)) =-closure({11}) ={10,12,13}=G

***For state H*:**

-closure(mov(H,a)) =-closure(mov({18},a)) =-closure({}) =

-closure(mov(H,b)) ==-closure(mov({18},b)) =-closure({19}) ={20}=I

***For state I*:**

-closure(mov(I,a)) =-closure(mov({20},a)) =-closure({21}) ==J

-closure(mov(I,b)) ==-closure(mov({20},b)) =-closure({}) =

***For state J*:**

-closure(mov(J,a)) =-closure(mov({16,22},a)) =-closure({17}) ==H

-closure(mov(J,b)) ==-closure(mov({16,22},b)) =-closure({}) =

**3. Use the DFA to get tokens from the following input strings based on the longest match convention:--**

**abaabaabbabaaaaabaaaa**

**string=”abaabaabbabaaaaabaaaa”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C=a** | **C=b** | **C=a** | **C=a** | **C=b** | **C=a** | **C=a** | **C=b** | **C=b** | **C=a** |
| S=A | S=B | **S=E** | **S=F** | **S=H** | **S=I** | **S=J** | **S=H** | **S=I** |  |
| **S->B** | **S->E** | **S->F** | **S->H** | **S->I** | **S->J** | **S->H** | **S->I** | **S->problem** |  |

Longest match is **abaabaab.**