Compiler 1

Lecture 4: Finite Automata

BY: Assist Prof. Dr. Ielaf O Abdul Majjed

Finite Automata

Finite Automata (FA) is the simplest machine to recognize patterns. The finite automata or finite state machine is an abstract machine that has five elements or tuples. It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol. Basically, it is an abstract model of a digital computer. A finite automaton (FA) is an abstract, mathematical machine, also known as a finite state machine, These are essentially graphs, like transition diagrams, with a few differences:

1. Finite automata are recognizers; they simply say "yes" or "no" about each possible input string.

2. Finite automata come in two flavors:

(a) ***Nondeterministic Finite Automata (NFA)*** have no restrictions on the labels of their edges. A symbol can label several edges out of the same state, and E, the empty string, is a possible label.

(b) ***Deterministic Finite Automata (DFA)*** have, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state.

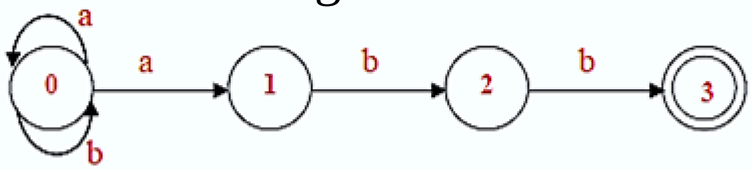
Both **Deterministic** and **Nondeterministic Finite Automata** are capable of recognizing the same languages.

Nondeterministic Finite Automata (NFA)

1. A finite set of states **S** .
2. A set of input symbols **C**, the ***input alphabet***. We assume that Ɛ, which stands for the empty string, is never a member of **C**.
3. A ***transition function*** that gives, for each state, and for each symbol in **Σ** U (Ɛ) a set of ***next states***.
4. A state **S0** 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***).

Example:

The transition graph for an **NFA** recognizing the language of regular expression **(a|b)*abb.** This abstract example, describing all strings of ***a***'s and ***b***'s ending in the particular string ***abb***, will be used throughout this section.



Transition Tables:

We can also represent an **NFA** by a transition table, whose rows correspond to states, and whose columns correspond to the input symbols and Ɛ. The entry for a given state and input is the value of the transition function applied to those arguments.

If the transition functions have no information about that state-input pair, we put **Ø** in the table for the pair

The transition table Advantage and Disadvantage

Advantage**:** that we can easily find the transitions on a given state and input.

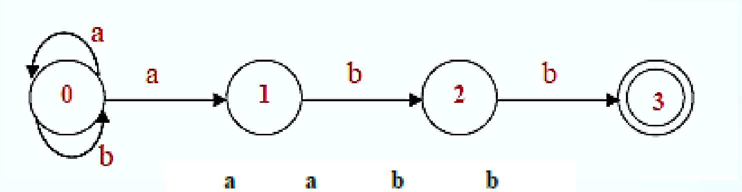
Disadvantage : is that it takes a lot of space, when the input alphabet is large, yet most states do not have any moves on most of the input symbols

The transition table

State	a	b	ε
0	{0,1}	{0}	Ø
1	Ø	{2}	Ø
2	Ø	{3}	Ø
3	Ø	Ø	Ø

The string ***aabb*** is accepted by the **NFA** of

The path labeled by ***aabb*** from state 0 to state 3 demonstrating this fact is:

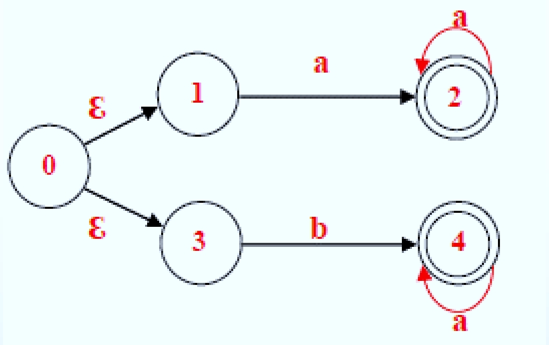


$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

Example:

The transition graph for an **NFA** recognizing the language of regular expression L(**aa*|bb***)

String ***aaa*** is accepted because of the path



$$0 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{a} 2 \xrightarrow{a} 2$$

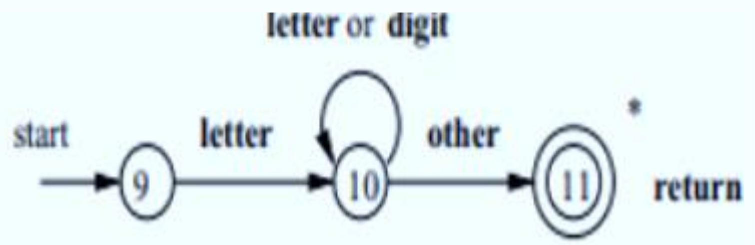
Deterministic Finite Automata (DFA)

A Deterministic Finite Automaton (DFA) is a special case of an NFA where:

1. There are no moves on input Ɛ.
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. While the NFA is an abstract representation of an algorithm to recognize the strings of a certain language, the DFA is a simple, concrete algorithm for recognizing strings.

DFA for identifiers and keywords



Simulating a DFA Algorithm

Input: An input string **X** terminated by an **end-of-file character eof**. A **DFA D** with start state S_0, accepting states **F** , and **transition function** move.

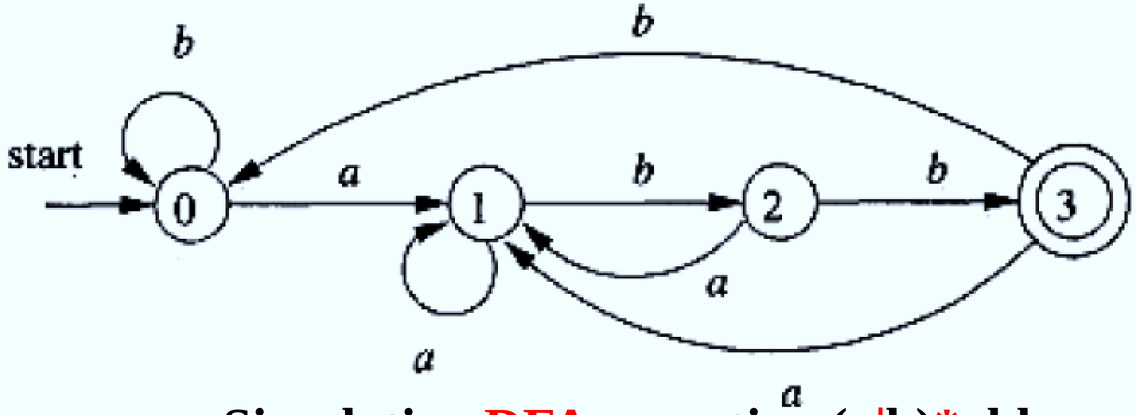
Output: Answer "**Yes**" if **D** accepts **X**; "**No**" otherwise.

Method: Apply the algorithm on the input string **X**. The function ***move***(**S**, **C**) gives the state to which there is an edge from state **S** on input **C**. The function ***next Char*** returns the next character of the input string **X**.

Example:

the transition graph of a **DFA** accepting the language **(a|b)*abb**. Given the input string ***ababb***, This **DFA** enters the sequence of states 0,1,2,1,2,3 and returns "yes.“

```
S = S0;
C = nextchar();
While (C! = eof)
{S = move(S,C);   C = nextchar();}
if (S is in F ) return "Yes";
else return "No";
```

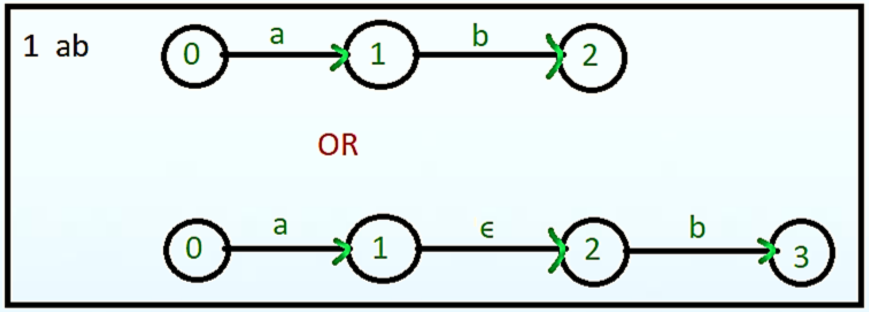


Simulating DFA accepting (a|b)*abb

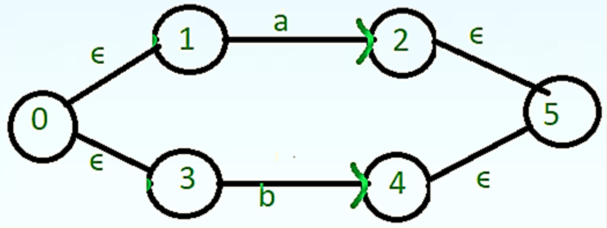
Regular expression to ∈ - NFA

∈-NFA is similar to the NFA but have minor difference by epsilon move. This automaton replaces the transition function with the one that allows the empty string ∈ as a possible input. The transitions without consuming an input symbol are called ∈-transitions.

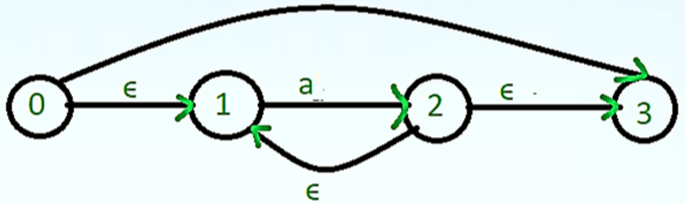
Common regular expression used in make ∈-NFA:



2. a+b

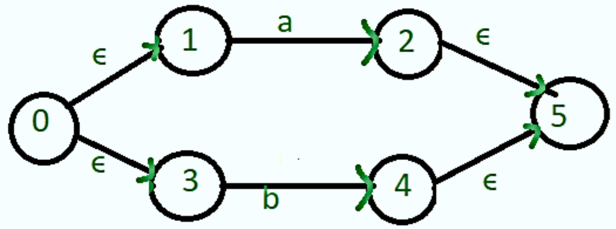


3. a*

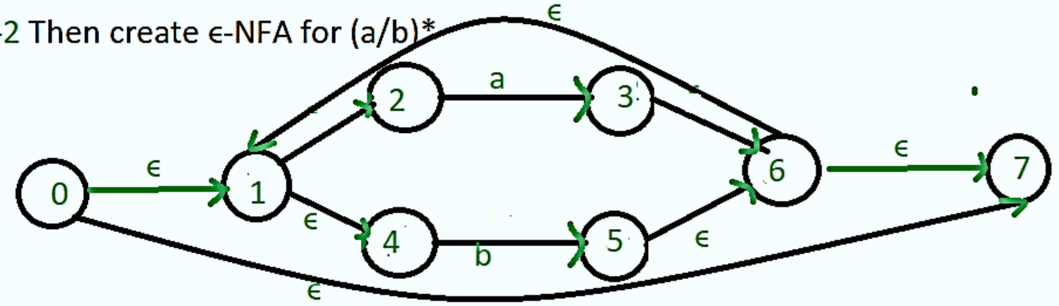


Example: Create a ∈-NFA for Regular Expression: (a/b)*a

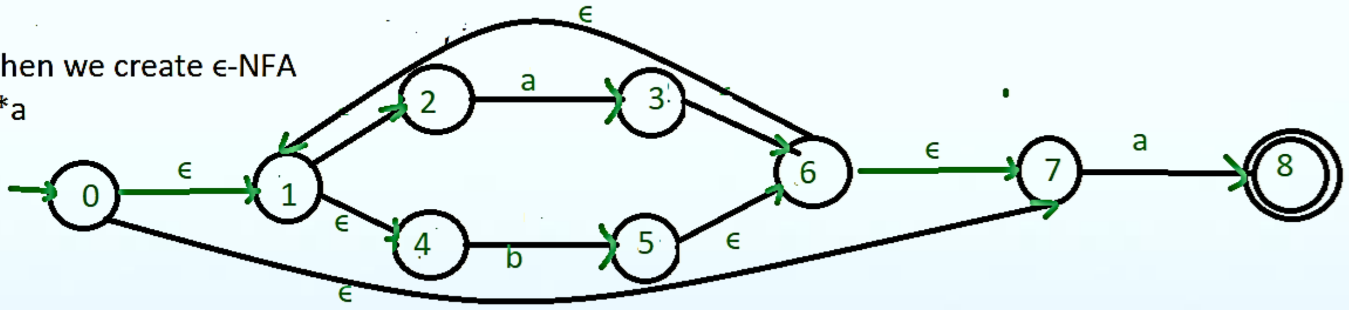
Step-1 First we create ϵ-NFA for (a/b)



Step-2 Then create ϵ-NFA for (a/b)*



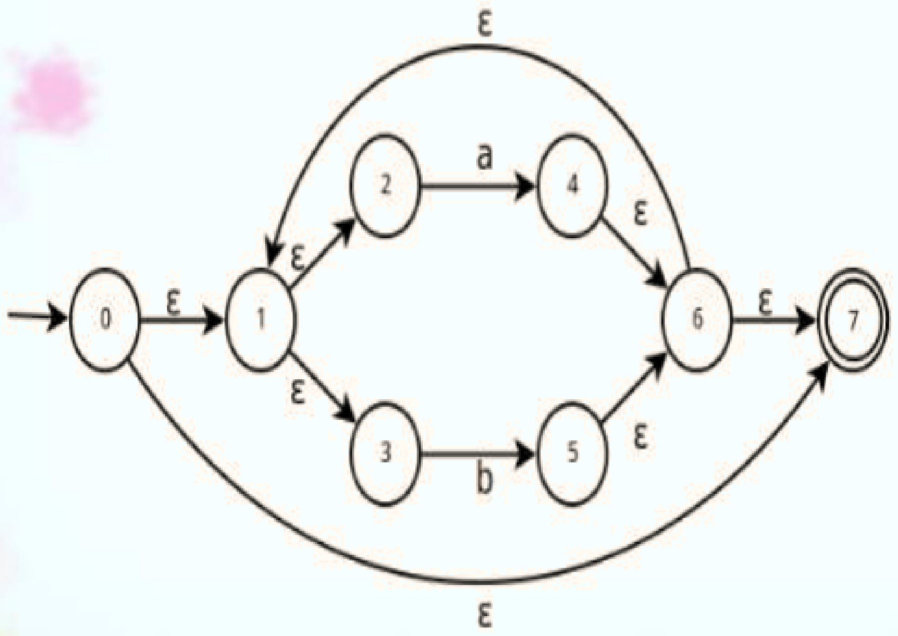
Step-3 Then we create ϵ-NFA for(a/b)*a



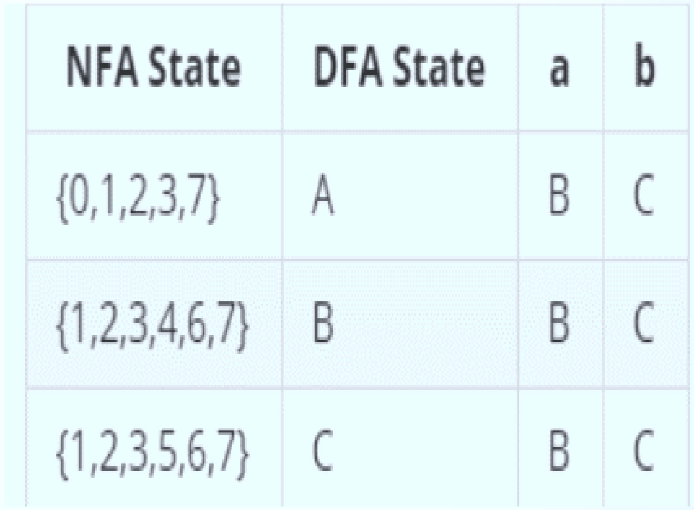
Conversion from NFA to DFA

An NFA can have zero, one or more than one move from a given state on a given input symbol. An NFA can also have NULL moves (moves without input symbol). On the other hand, DFA has one and only one move from a given state on a given input symbol. The general idea behind the subset construction is that each state of the constructed DFA corresponds to a set of NFA states.

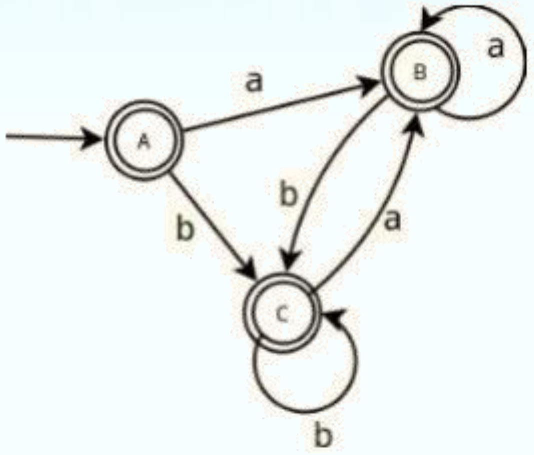
Example: NFA for the regular expression (a|b)*



Transition Table

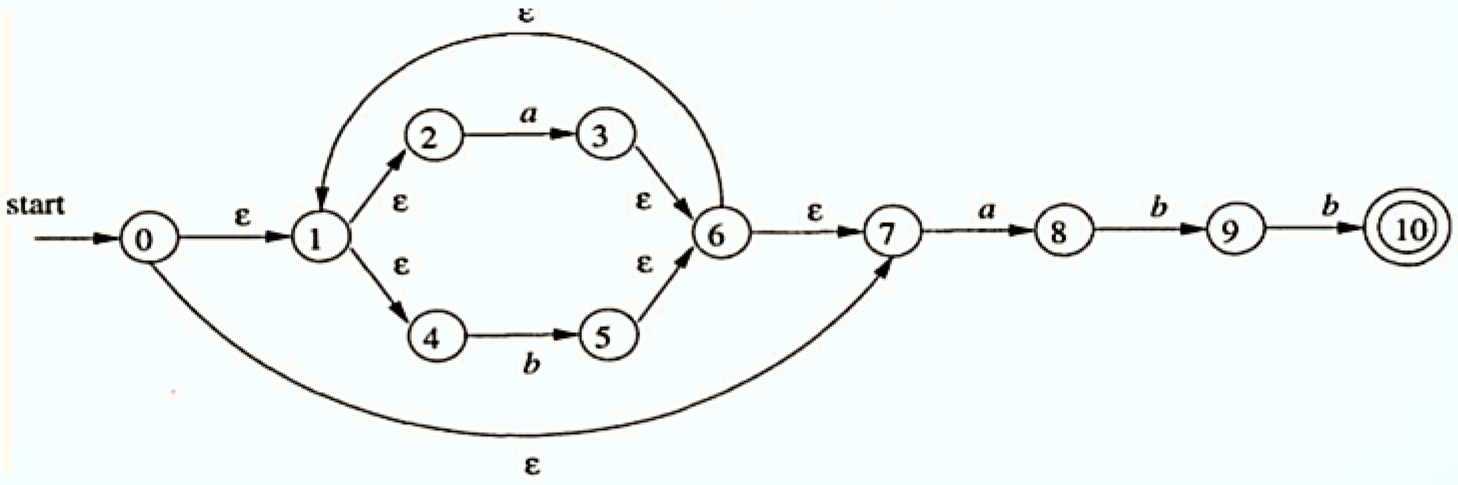


NFA State	DFA State	a	b
{0,1,2,3,7}	A	B	C
{1,2,3,4,6,7}	B	B	C
{1,2,3,5,6,7}	C	B	C



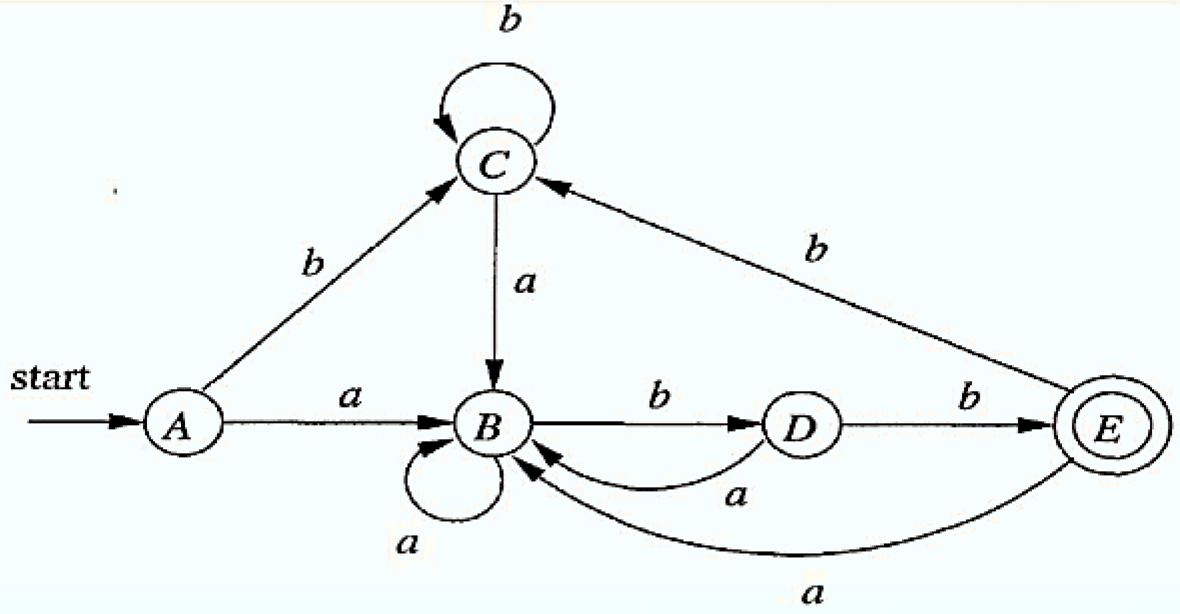
DFA for the regular expression (a|b)*

Example: NFA for the regular expression (a|b)*abb



Transition Table

NFA STATE	DFA STATE	a	b
$\{0,1,2,4,7\}$	A	B	C
$\{1,2,3,4,6,7,8\}$	B	B	D
$\{1,2,4,5,6,7\}$	C	B	C
$\{1,2,4,5,6,7,9\}$	D	B	E
$\{1,2,3,5,6,7,10\}$	E	B	C



DFA for the regular expression (a|b)*abb