Grammars **(Lecture 3-1)**

– A grammar G is defined as a 4-tuples: G = (*V, T, S, P*)

Where:

- ***V*** : is a finite set of objects called variables (non terminals)
- ***T*** : is a finite set of objects called terminal symbols
- ***S*** : S ∈ *V* is a special symbol called the Start symbol
- ***P*** : is a finite set of productions or "production rules"

Sets ***V*** and ***T*** are nonempty and disjoint.

– Production rules have the form:

$$\mathbf{x \rightarrow y}$$

- where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$
- Given a string of the form

$$\mathbf{w = uxv}$$

And a production rule

$$\mathbf{x \rightarrow y}$$

We can apply the rule, replacing x with y, giving

$$\mathbf{z = uyv}$$

We can then say that

$$\mathbf{w \rightarrow z}$$

Read as "w derives z", or "z is derived from w"

If $\mathbf{u \rightarrow v}$, $\mathbf{v \rightarrow w}$, $\mathbf{w \rightarrow x}$, $\mathbf{x \rightarrow y}$, and $\mathbf{y \rightarrow z}$, then we say:

$$\mathbf{u \overset{*}{\rightarrow} z}$$

This says that **u** derives **z** in an unspecified number of steps.

Along the way, we may generate strings, which contain variables as well as terminals. These are called **sentential forms**.

– What is the relationship between a language and a grammar?

Let G = (*V*, *T*, *S*, *P*)

The set

$$\mathbf{L(G) = \{w \in T^* : S \overset{*}{\rightarrow} w\}}$$

is the language generated by G.

Consider the grammar G = (*V*, *T*, *S*, *P*), where:

V = {S}

T = {a, b}

S = S,

P =

$S \rightarrow aSb$
$S \rightarrow \lambda$

– What are some of the strings in this language?

$S \Rightarrow \lambda$

$S \Rightarrow aSb \Rightarrow ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

It is easy to see that the language generated by this grammar is:

$L(G) = \{a^n b^n : n \geq 0\}$

Let's go the other way, from a description of a language to a grammar that generates it. Find a grammar that generates:

$$\mathbf{L = \{a^n b^{n+1} : n \geq 0\}}$$

So the strings of this language will be:

b (0 a's and 1 b)

abb (1 a and 2 b's)

aabbb (2 a's and 3 b's) . . .

In order to generate a string with no **a**'s and **1 b**, you might want to write rules for the grammar that say:

$$\mathbf{S \rightarrow ab}$$

$$\mathbf{a \rightarrow \lambda}$$

But you can't do this; **a** is a terminal, and you can't change a terminal, only variables

So, instead of:

$$\mathbf{S \rightarrow ab}$$

$$a \rightarrow \lambda$$

We create another variable, **A** (we often use capital letters to stand for variables), to use in place of the terminal, **a**:

$$S \rightarrow Ab$$
$$A \rightarrow \lambda$$

Now you might think that we can use another **S** rule here to generate the other part of the string, the $a^n b^n$ part

$$S \rightarrow aSb$$

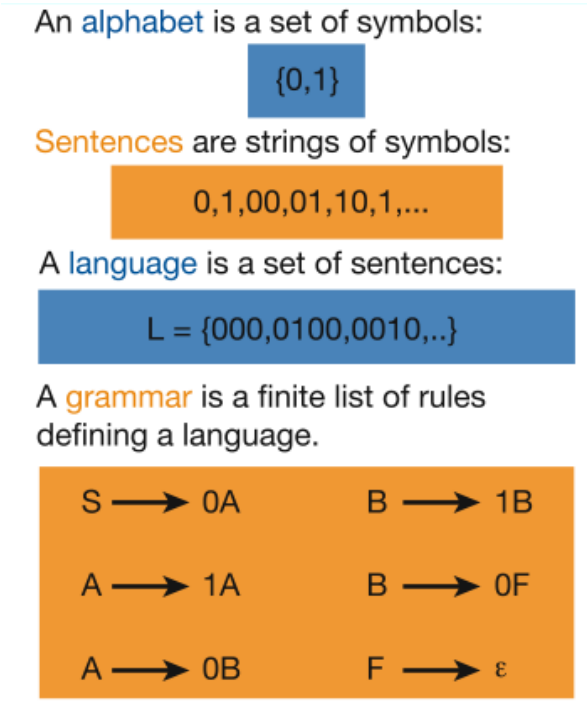
But you can't, because that will generate **ab**, **aabb**, etc.
Note, however, that if we use **A** in place of **S**, that will solve our problem:

$$A \rightarrow aAb$$

So, here are our rules:

$$S \rightarrow Ab$$
$$A \rightarrow aAb$$
$$A \rightarrow \lambda$$

The $S \rightarrow Ab$ rule creates a single b terminal on the right, preceded by other strings (including possibly the empty string) on the left.
The $A \rightarrow \lambda$ rule allows the single b string to be generated.
The $A \rightarrow aAb$ rule and the $A \rightarrow \lambda$ rule allows **ab**, **aabb**, **aaabbb**, etc. to be generated on the left side of the string.



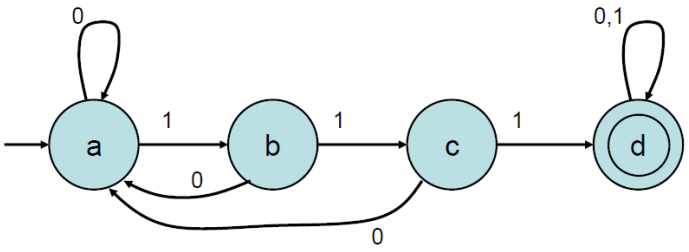
Finite Automata (FA) (Lecture 3-2)

- One of the powerful models of computation that are restricted model of actual computer is called finite automata.
- These machines are very similar to CPU of a computer .But they lack memory.
- Finite Automata (FA) is the simplest machine to recognize patterns.
- Finite automation is called finite because number of possible states and number of letter in alphabet are both finite and automation because the change of state is totally governed by the input.
- A finite automata is a collection of three things:
- A Finite Automata is a 5-tuple (**Q**, **Σ**, **δ**, **q0**, **F**), where:

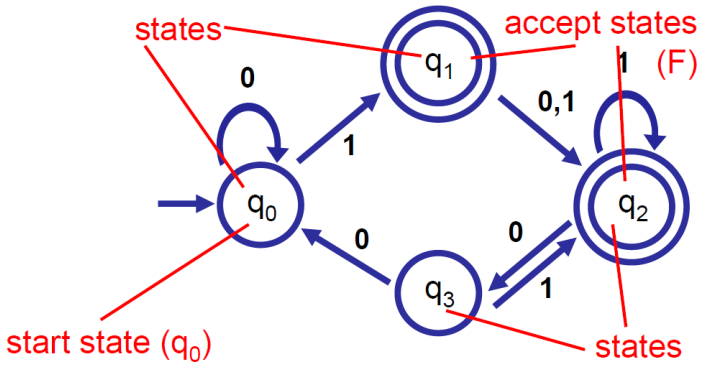
  Q: Finite set of states.
  - one of which is designed as the initial state, called the start state,
  - In addition, some (may be none) of which are designed as final states.

  ∑: is a finite set (alphabet) of Input Symbols.
  δ *(delta):* represents the set of transitions that *FA* can take between its states. Q ×Σ→Q is the transition function.
  $\mathbf{q_0}$: ∈ Q is the start state (Initial state).
  F: ⊆ Q is the set of accepting, or final states.
- Formal specification of machine is:
  - {Q, ∑, q_0, F, δ}.

An FA diagram, machine M



Conventions:



How does FA work?

1. Starts from a start state.
2. Loop

   Reads a sequence of letters

3. Until input string finishes
4. If the current state is a final state then

   Input string is accepted.

5. Else

   Input string is NOT accepted.

Acceptability of a string

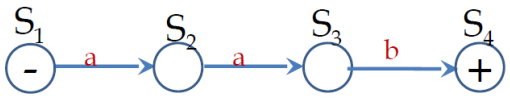
– A string is accepted by a transition system if
- There exist a path from initial state to final state.
- Path traversed is equal to w.

But how can FA be designed and represented?

Let see the following examples

Example

Build an FA that accepts only aab

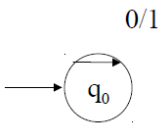


	a	b
S_1	S_2	?
S_2	S_3	?
S_3	?	?
S_4	?	?

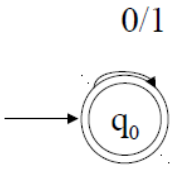
Example

Let Σ = {0, 1}. Give DFAs for {}, {ε}, Σ*, and Σ+.

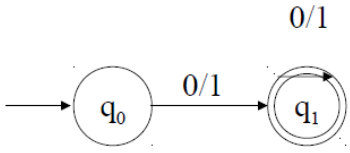
For {}:



For Σ*:



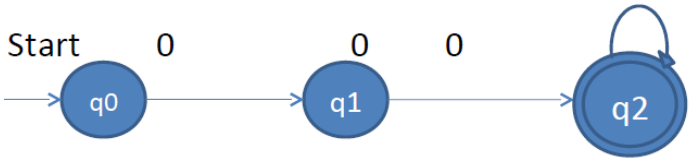
For Σ+:



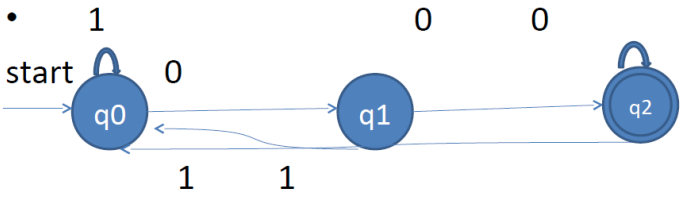
Example

Design a FA that accepts set of strings such that every string ends in 00, over the alphabet {0,1} i,e Σ={0, 1}

- In order to design any FA, first try to fulfill the minimum condition.



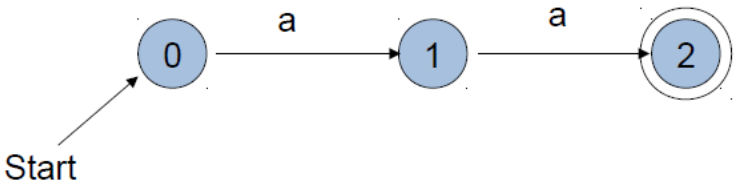
- Being DFA, we must check every input symbol for output state from every state. So we have to decide output state at symbol 1 from q0,q1 and q2. Then it will be complete FA



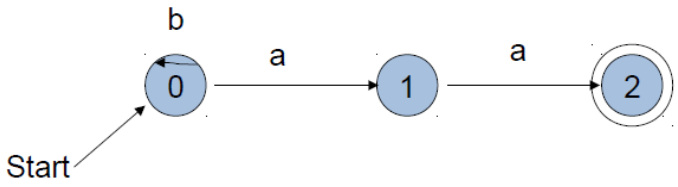
Example

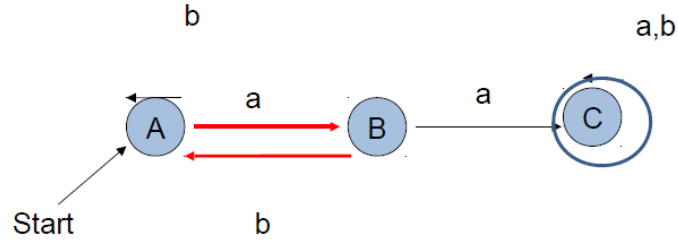
- Construct a DFA that accepts a's and b's and 'aa' must be substring

Minimal condition: aa



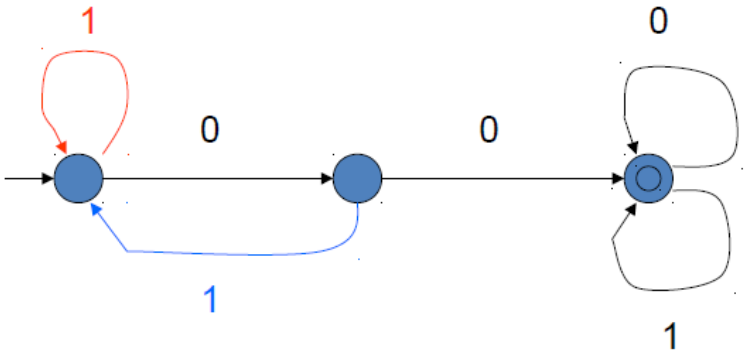
There may be aabbaa.bbbbaa,aa,aab,aabb,….





Example

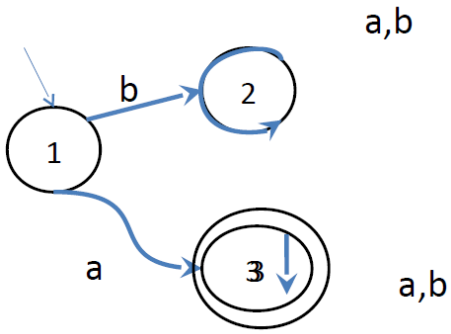
$(0+1)^*00(0+1)^*$



Example

All words that start with "a" over the alphabet {a,b}

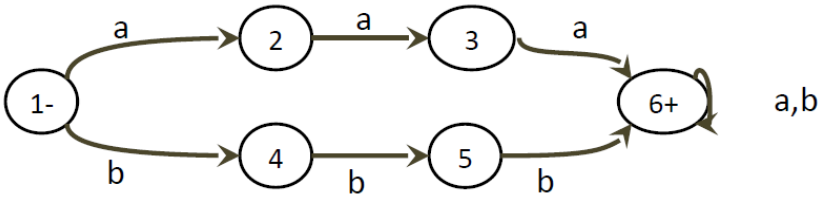
$a(a+b)^*$



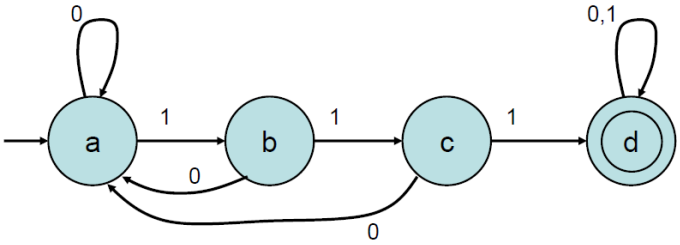
Example

All words that start with triple letter

$(aaa+bbb)(a+b)^*$



Example

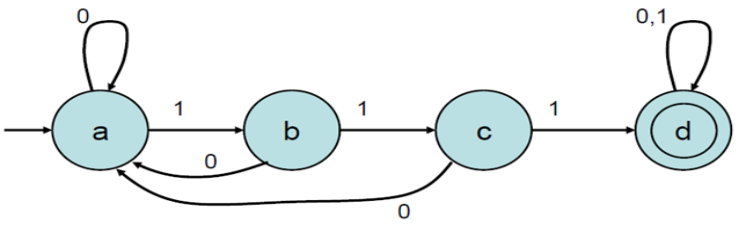


Example computation:

- Input word w: 1 0 1 1 0 1 1 1 0
- States: a b a b c a b c d d
    - We say that M accepts w, since w leads to d, an accepting state.
- A FA M accepts a word w if w causes M to follow a path from the start state to an accept state.
- Some terminology and notation:
    - Finite alphabet of symbols, usually called Σ.
    - In Example 1 (and often), Σ= {0, 1}.
    - String (word) over Σ: Finite sequence of symbols from Σ.
    - Length of w, | w |
    - ε, placeholder symbol for the empty string, | ε| = 0
    - Σ^*, the set of all finite strings of symbols in Σ

- Concatenation of strings w and x, written w ◦ x or w x.
- L(M),language recognized by M: { w | w is accepted by M }.

Example



What is L(M) for the above machine?

- { w ∈{ 0,1 }* | w contains 111 as a substring }
- Note: Substring refers to consecutive symbols.

Example

What is the 5-tuple (Q, Σ, δ, q_0, F)?

- **Q** = { a, b, c, d}
- **Σ**= { 0, 1 }
- **δ** is given by the state diagram, or alternatively, by a table:

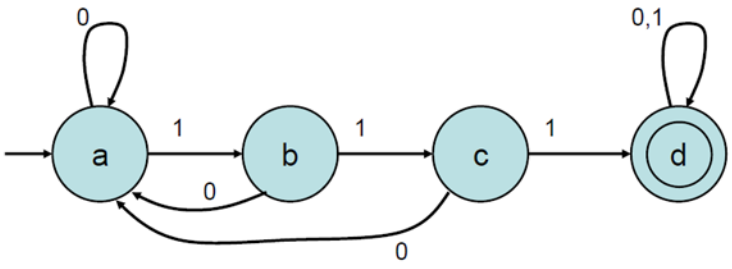
	0	1
a	a	b
b	a	c
c	a	d
d	d	d

- $\mathbf{q_0}$ = a
- **F** = { d}

Example

Design an FA M with

L(M) = { w∈{ 0,1 }* | w contains 101 as a substring }.



- Failure from state b causes the machine to remain in state b.

FA is characterized into two types:

1) Deterministic Finite Automata (DFA)

2) Nondeterministic Finite Automata (NFA)