Finite-state Machine with NO OUTPUT

- Used for language recognition
- Finite-state machine with output can also be used for language recognition, but requires customization

- Finite-state machine with no-output is specifically designed for the language recognition.
  - Instead of producing output, these machines have final states
  - A string is recognized if and only if it takes the starting state to one of the final states

Some back-ground:

Def^n:

Suppose A and B are subsets of V^*, where V^* is the vocabulary.

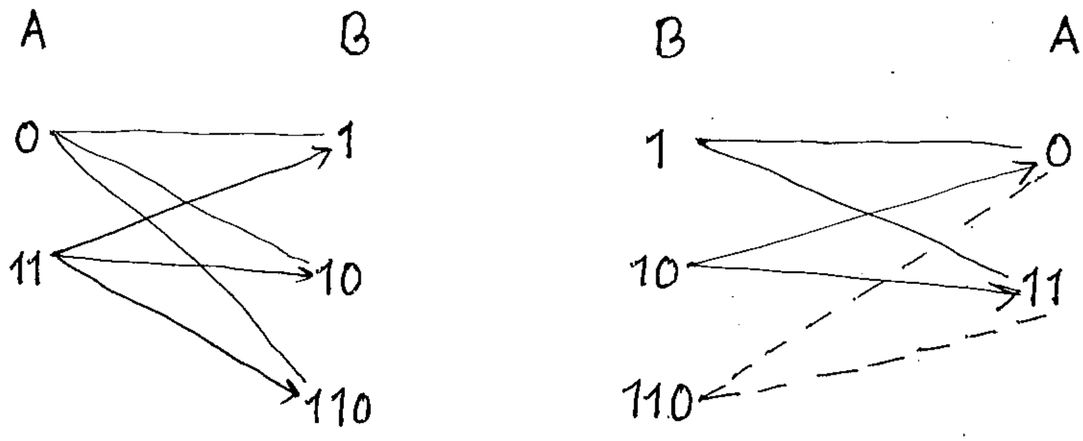
Concatenation of A and B, denoted by AB, is the set of all strings of the form xy, where

x is a string in A
y is a string in B

Example: Given $A = \{0, 11\}$ and $B = \{1, 10, 110\}$

Find AB and BA

AB　　　　　　　　BA



so, $AB = \{01, 010, 0110, 111, 1110, 11110\}$

$BA = \{10, 111, 100, 1011, 1100, 11011\}$

That is, when A and B are subsets of V^*, where V is an alphabet — it is not necessary that $AB = BA$

For def^n of concatenation, we can define

$A^0 = \{\lambda\}$ empty string $\to \varepsilon$

$A^{n+1} = A^n A$ for $n = 0, 1, 2, 3 \ldots$

Example: $A = \{1, 00\}$, Find A^n for $n = 0, 1, 2, 3$

$A^0 = \{\lambda\}$

$A^1 = A^0 . A = A = \{1, 00\}$

$A^2 = A^1 . A = \{11, 100, 001, 0000\}$

$A^3 = A^2 . A = \{111, 1100, 1001, 10000, 011, 00100, 00001, 000000\}$

Definition :

Suppose that A is a subset of V^*. Then, Kleene closure of A, denoted as A^*, is the set consisting of concatenations of arbitrarily many strings from A. So,

$$A^* = \bigcup_{K=0}^{\infty} A_K$$

example : Kleene closures of the sets

$$A = \{0\}.$$

$$A^* = \{0^n \mid n = \{0, 1, 2, \dots \dots\}\}$$

Concatenation of string 0 with itself for an arbitrary finite times.

$$B = \{0, 1\}$$

So,

$$B^* = \bigcup_{K=0}^{\infty} B^K = \bigcup_{K=0}^{2} B^K, \text{ for } K_{max} = 2$$

$$= B^0 \cup B^1 \cup B^2$$

$$B^0 = \{\lambda\}$$

$\hookrightarrow$ empty

$$B^1 = B^0 B = \{0, 1\}$$

$$B^2 = B^1 . B = \{0,1\} . \{0,1\} = \{00, 01, 10, 11\}$$

So, $B^* = \{0, 1, 00, 01, 10, 11\}$

FINITE-STATE AUTOMATA

Finite-state machine with no output

↓ known as

Finite-state automata

→ They don't produce output

→ Instead, they do have a set of final states.

Finite-state automata recognize strings that take system state to any of the final states.

Definition :

→ singular of automata

A finite-state automaton is denoted/ described by a five-element Tuple

$$M = (S, I, f, s_0, F)$$

Where,

S : finite set of states

I : set of input alphabet

f : Transition function f

s_0 : Initial or start state

F : Set of final/accepting states

~~So~~, Here, $f : S \times I \rightarrow S$ Which means that f assigns a next state to every pair of state and input.

Finite-state Machine

↓

Finite Automata (FA)

- With output
- Without output

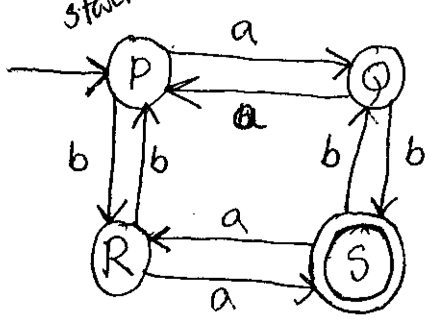
* Finite-state machine has a fundamental role in the design and construction of compilers.

* There are finite machines that are specially designed for language recognizing.
  - These machines have final states
    - If a string starts from starting state and reach to one of the final states

We specifically cover the Deterministic Finite Automata. (DFA)

Structures:

start state / initial state



Final state / accepting state

Recall: Finite-state automaton is a five-element Tuple

So, What is Tuple ?

→ Ordered sequence of elements

→ n-tuple is described as

$$(a_1, a_2, a_3, \dots \dots a_n)$$

W Two tuples —

$(a_1, a_2, \dots a_n)$ and $(b_1, b_2, \dots \dots b_n)$

They will be equal if and only if

$a_1 = b_1, \quad a_2 = b_2, \quad a_3 = b_3 \quad \dots \dots \quad a_n = b_n$

Tuple vs. Set :

- Tuple can contain multiple copies of same element and they are not treated as same element

  So, $(1, 2, 2, 3, 4) \neq (1, 2, 3, 4)$

  Whereas, in set

  $\{1, 2, 2, 3, 4\} = \{1, 2, 3, 4\}$

- Elements in Tuple are ordered

  $\{1, 2, 3, 4\} = \{4, 3, 2, 1\}$ in Set

  but

  $(1, 2, 3, 4) \neq (4, 3, 2, 1)$ are not equal for Tuple

Considering the example network again:

here,

$S = \{P, Q, R, S\}$

$I = \{a, b\}$

$f \equiv$ from $S \times I \rightarrow S$

$s_0 = A$

$F = \{S\}$

when

$S = \{P, Q, R, S\}$

$I = \{0, 1\}$

$f =$ Trasition fn^c from $(S \times I) \rightarrow S$

$s_0 = A$

$F = \{S\}$

State transition

	a	b
P	Q	R
Q	P	S
R	S	P
S	R	Q

Example: Let's construct a Deterministic Finite Machine (DFA) that accepts strings of length 2 with $\{0, 1\}$ as the alphabets.

Here, $I =$ Set of input alphabets $= \{0, 1\}$

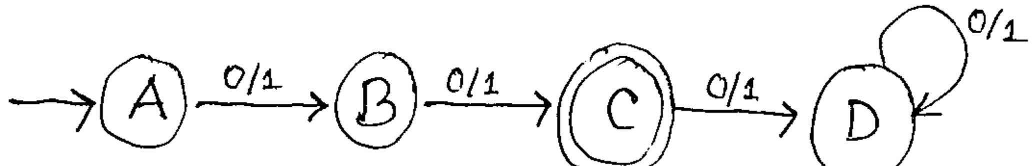
Possible strings of length 2 ?

↳ Use concatenation

$I^{n+1} = I^n I$

$\Rightarrow I^2 = I^1 I = \{0, 1\}\{0, 1\} \equiv \{00, 01, 10, 11\}$

Let's assume that the initial state is A



initial state

Final state

Trap/Dead state

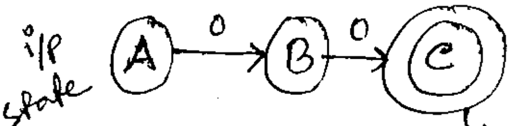
↳ by definition, it is a non-acceptible state that goes to itself for every possible input symbol.

↓

it is a reachable non-accepting state from which no accepting state is reachable.

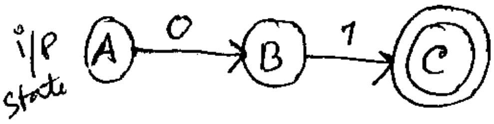
For instance,

00



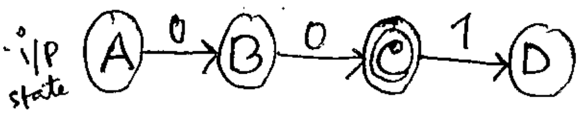
↳ Final state is reached after the input sequence is fed.

01



Again, we reach to the accepting state.

001



not reachable, we are at trap state

As we see here, string length of two is accepted and the rest are discarded.

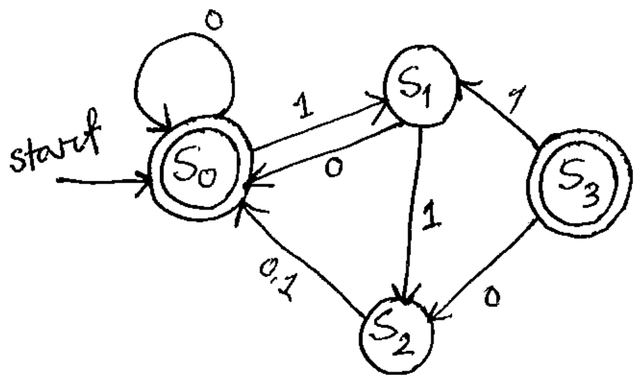
* At the end of string, we must be in an accepting state.

↳ also, known as final state.

Another DFA example —

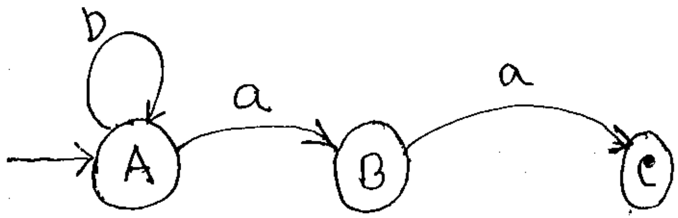
Construct a state diaram for $M = (S, I, f, s_0, F)$
Where,
$$S = \{s_0, s_1, s_2, s_3\}$$
$$I = \{0, 1\}, \quad F = \{s_0, s_3\}$$
and transition fnc is given as :

state	f Input	
	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1



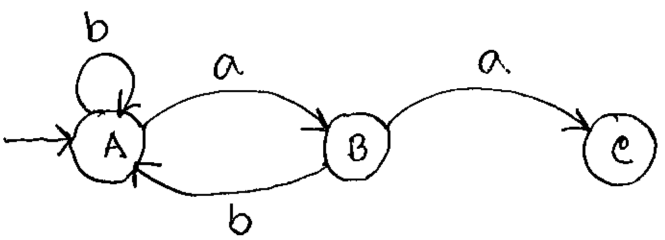
ANOTHER DFA EXAMPLE

Let's construct a DFA that accepts any string with $\boxed{aabb}$ in it for alphabets $\{a, b\}$



When a is at input, it matches with the 1st string of aabb

When b is at input, it remains at A



at B When we get a, it makes aa
When we get b as i/p, the formed string ab doesn't match with aabb
and with ab, we cannot form aabb.

So, at B, for i/p 'b', it comes back to A

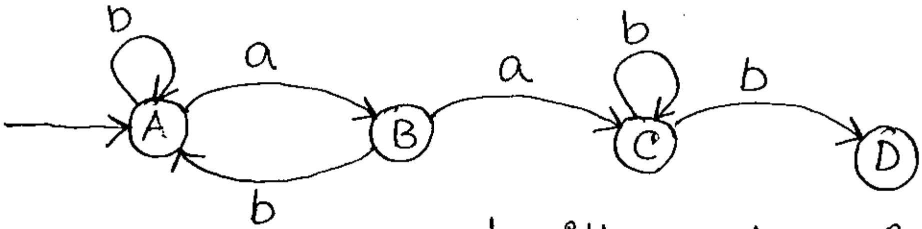
While at C, an input b, takes it to next state D
an input a, makes

aaa which doesn't match aabb

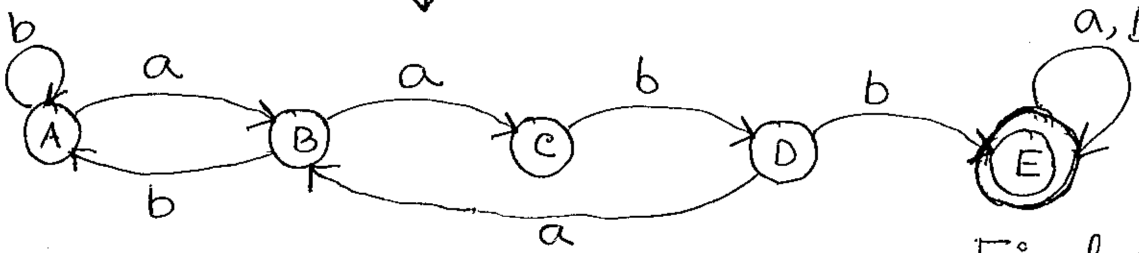
But

we still can keep aa and expect b in next step. So, staying at C for an input a keeps the possibility of the string aabb still open.

So,



with another input **a** or **b**, the DFA takes the following form.



Final state

at D
aabaa
aabab
at D

LANGUAGE RECOGNITION

By Finite-state Machines

We can construct Finite-State Machines that can recognize a given set of strings.

↓ can be done by

carefully adding states and determining the final states from the added states.

Definition:

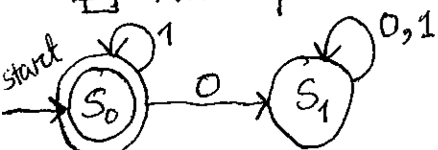
A string x is said to be recognized or accepted by the machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(, x)$ in F.

$L(M)$: Language Recognized/Accepted by machine M

$L(M)$ is the set of all strings that are recognized by machine M.

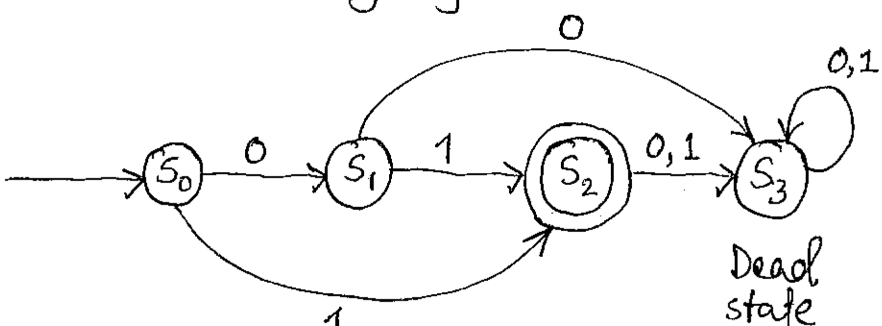
Two machines are equivalent if they recognize the same language

Example:



Language it recognizes

$L(M) = \{1^n \mid n = 0, 1, 2 \ldots\}$



Languages it recognizes

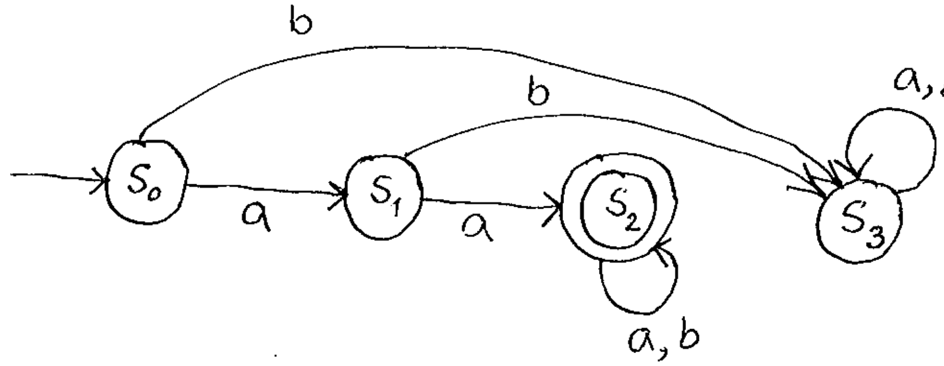
$L(M) = \{0, 01\}$

Example :

Let's construct a machine (DFA) that recognize a language M, which is the set of strings that contain two a's.

- We consider S_0 as the start state
  S
  S_2 as the Final state
  S_3 as the Dead state

- We move to S_1 from S_0 if the first charater is a.



- From S_1, we move to the final state S_2 on receiving a as the input.

- So, as the two characters received so far to reach the final state is a, we remain at S_2 regard of any new inputs.

Now, what happens if we receive b initially ?
We don't accept the string and hence, we move to S_3 from S_0 and S_1 upon recei b.

→ non-final state / Dead state / Trap state