Name : Shamon Barman

Reg. ID: 18201093

* **The Central Concepts of Automata Theory**: These concepts include the "alphabet" (a set of symbols), "strings" (a list of symbols from an alphabet), and "language" (a set of string from an alphabet).

▷ **Alphabet**: An alphabet is a finite, nonempty set of symbols. Common alphabet include:

1. $\Sigma = \{0, 1\}$, the binary alphabet.
2. $\Sigma = \{a, b, c, \ldots, z\}$, the set of all the lower-case letters.
3. The set of all ASCII characters, or the set of all printable ASCII characters.

▷ **Strings**: A string (or sometimes word) is a finite sequence of symbols chosen from some alphabet. For example, 01101 is a string from the binary alphabet.

→ The empty string is the string with zero occurrences of symbols. It is denoted by ϵ.

→ For instance, 01101 has length 5. This is accepted but not strictly correct. There are number of symbol is 2. And there are five positions for symbols. so its length is 5.

→ Powers of an Alphabet: It Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation. It is denoted Σ^k, where k is the length.

Let, $\Sigma = \{0, 1\}$, then $\Sigma^1 = \{0, 1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

→ Languages: A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a language. It Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language oven Σ.

1. The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$

2. The set of strings of 0's and 1's which an equal number of each: $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$

3. The set of binary numbers which value is a prime:

$\{10, 11, 101, 111, 1011, \ldots\}$

* DFA (Deterministic Finite Automata): DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time. In DFA, there is only one path for specific input from the current state to the next state. DFA does not accept the null move. DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

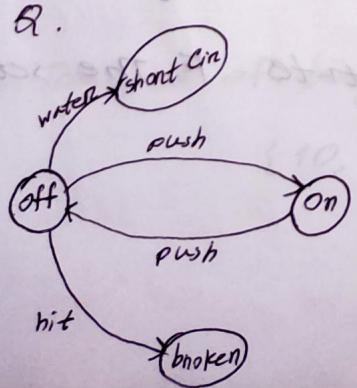
A deterministic final automata consists of:

1. A finite set of states, often denoted Q.
2. A finite set of input symbols, often denoted Σ.
3. A transition function that takes as arguments a state and an input symbol and returns a state. The transition function will commonly by denoted δ.
4. A start state, one of the states in Q.
5. A set of final or accepting state F. The set F is a subset of Q.

* NFA: NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language. The finite automata are called NFA when there exist many paths for specific input from the current state to the next state. Every NFA is not DFA, but each NFA can be translated into DFA.

An NFA is represented by, $A = (Q, \Sigma, \delta, q_0, F)$

1. Q is a finite set of states.
2. Σ is a finite set of input symbols.
3. q_0, a member of Q, is the start state.
4. F, a subset of Q, is the set of final state.
5. δ, the transition function ~~table~~ is a function that takes a state in Q and an input symbol in Σ as arguments and return a subset of Q.



states, $Q = \{off, on, short\ Cir, broken\}$

inputs, $\Sigma = \{push, water, hit\}$

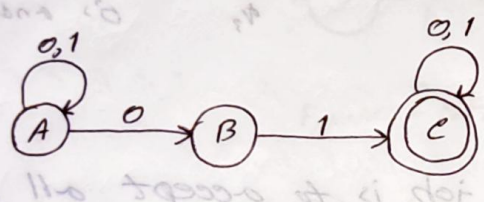
$\delta(off, push) = on$, $\delta(on, push) = off$

$= \{\epsilon, 00, 0000, 000000, \ldots\}$

★ Let us formally specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 somewhere in the string. We can write this language L as:

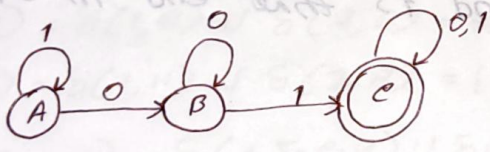
$\{w \mid w$ is of the form $x01y$ for some strings x and y consisting of 0's and 1's only$\}$

→NFA:



Example: 0111**01**00
1111**01**00
1010**11**1
0001000

→DFA:



Example: 111**01**[0,1]
11100**01**[0,1]
100**01**[01..]

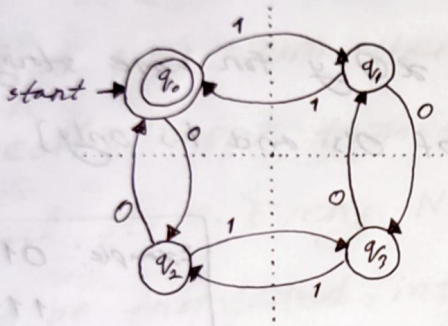
	0	1
→ A	B	A
B	B	C
*C	C	C

[Transition Table]

→NFA ϵ-transition allow করে কিন্তু DFA allow করে না।

* Let us design a DFA to accept the language.

L = {w | w has both an even number of 0's and an even number of 1's}



state	descrip
q_0	0's and 1's are even
q_1	0's even and 1's odd
q_2	0's odd and 1's even
q_3	0's and 1's odd

* Write a NFA where job is to accept all and only the strings of 0's and 1's that end in 01. Then convert it DFA.

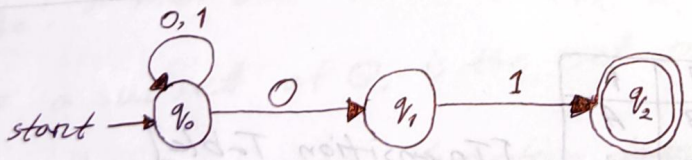


Figure - NFA

$\delta(0, q_0) = q_0, q_1$

$\delta([q_0, q_1], 0)$

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_0, q_1\} \cup \{\}$

$= \{q_0, q_1\}$

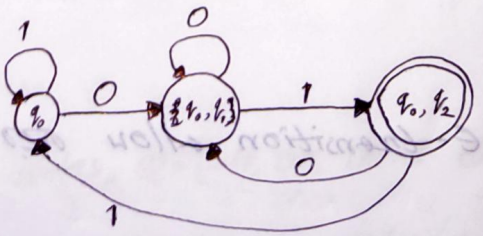
$\delta([q_0, q_1], 1)$

$= \delta(q_0, 1) \cup \delta(q_1, 1)$

$= \{q_0, q_2\}$

Transition Table

	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$



* Suppose we want to design an NFA to recognize occurrences of the words web and ebay.

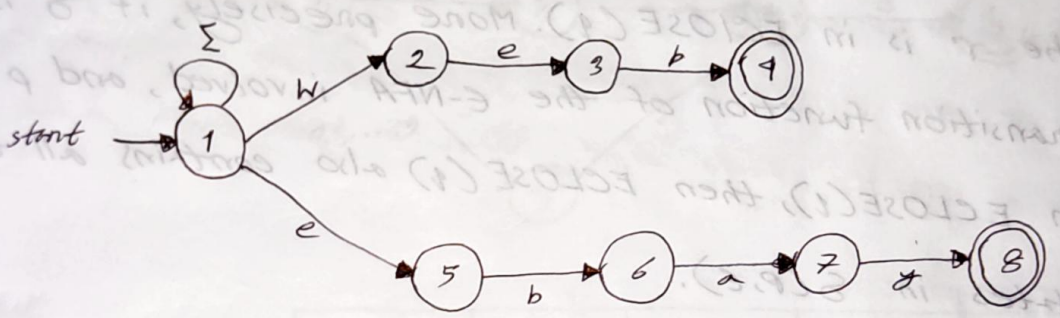


Figure - NFA

$\delta(1, W) = 1, 2 \quad \delta(1, e) = 1, 5 \quad \delta(1, \Sigma - e - W) = 1$

$\delta(\{1,2\}, e) = \delta(1,e) \cup \delta(2,e) = \{1,5\} \cup \{3\} = \{1,5,3\}$

$\delta(\{1,2\}, W) = \delta(1,W) \cup \delta(2,W) = \{1,2\} \cup \{\} = \{1,2\}$

$\delta(\{1,2\}, \Sigma - e - W) = \delta(1, \Sigma - e - W) \cup \delta(2, \Sigma - e - W) = \{1\} \cup \{\} = \{1\}$

$\delta(\{1,5\}, e) = \delta(1,e) \cup \delta(5,e) = \{1,5\} \cup \{\} = \{1,5\}$

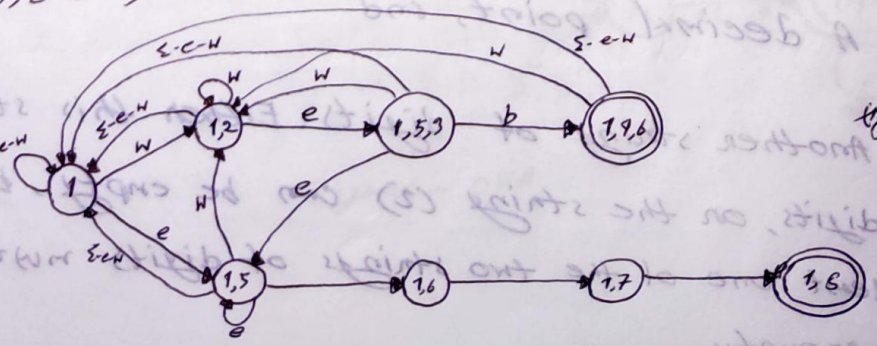
$\delta(\{1,5\}, W) = \delta(1,W) \cup \delta(5,W) = \{1,2\} \cup \{\} = \{1,2\}$

$\delta(\{1,5\}, \Sigma - e - W) = \delta(1, \Sigma - e - W) \cup \delta(5, \Sigma - e - W) = \{1\} \cup \{\} = \{1\}$

$\delta(\{1,5,3\}, e) = \delta(1,e) \cup \delta(5,e) \cup \delta(3,e) = \{1,5\} \cup \{\} \cup \{\} = \{1,5\}$

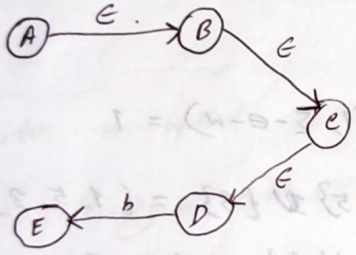
$\delta(\{1,5,3\}, W) = \delta(1,W) \cup \delta(5,W) \cup \delta(3,W) = \{1,2\} \cup \{\} \cup \{\} = \{1,2\}$

$\delta(\{1,5,3\}, \Sigma - e - W) = \delta(1, \Sigma - e - W) \cup \delta(5, \Sigma - e - W) \cup \delta(3, \Sigma - e - W) = \{1\}$



try yourself the other part

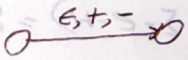
* E-Closures: If state p is in ECLOSE(q), and there is a transition from state p to state r labeled ϵ, the r is in ECLOSE(q). More precisely, if δ is the transition function of the ϵ-NFA involved, and p is in ECLOSE(q), then ECLOSE(q) also contains all the states in δ(p, ϵ).



ECLOSE(A) = {A, B, C, D}
ECLOSE(B) = {B, C, D}
ECLOSE(C) = {C, D}
ECLOSE(D) = {D}
ECLOSE(E) = {E}

* Let us design a ϵ-NFA then DFA that accepts decimal numbers consisting of:

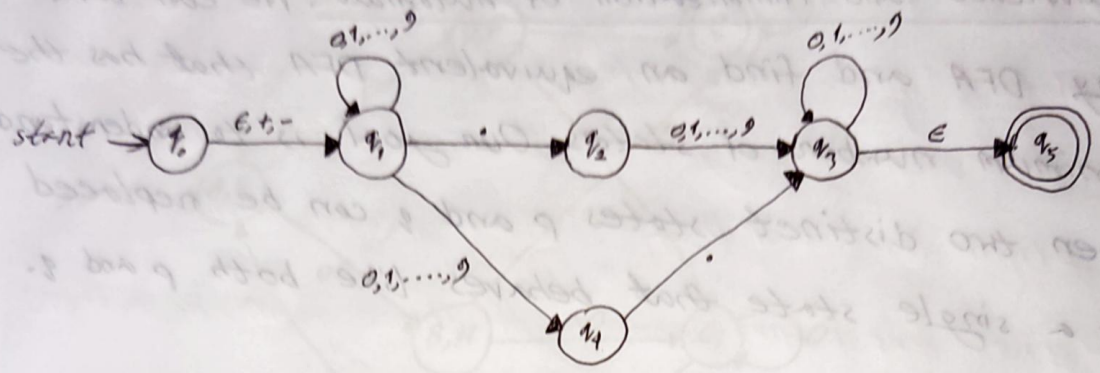
① An optional + or - sign.



② A string of digits.

③ A decimal point, and

④ Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.



ECLOSE(q_0) = {q_0, q_1}
ECLOSE(q_1) = {q_1}
ECLOSE(q_2) = {q_2}
ECLOSE(q_3) = {q_3, q_5}
ECLOSE(q_4) = {q_4}
ECLOSE(q_5) = {q_5}

	∈	+, −	.	0, 1, ..., 9
→ {q_0, q_1}	/	{q_1}	{q_2}	{q_1, q_4}
{q_1}	/	{ }	{q_2}	{q_1, q_4}
{q_2}	/	{ }	{ }	{q_3, q_5}
{q_1, q_4}	/	{ }	{q_2, q_3, q_5}	{q_1, q_4}
* {q_3, q_5}	/	{ }	{ }	{q_3, q_5}
* {q_2, q_3, q_5}	/	{ }	{ }	{q_3, q_5}

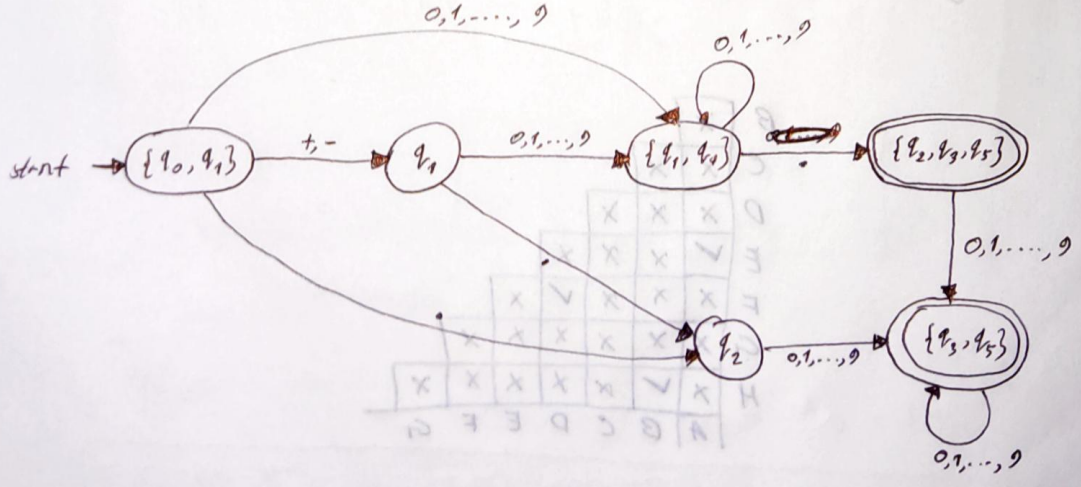


Figure – DFA

* Equivalence and Minimization of Automata: We can take any DFA and find an equivalent DFA that has the minimum number of states. Our goal is to understand when two distinct states p and q can be replaced by a single state that behaves like both p and q.

→ Example:

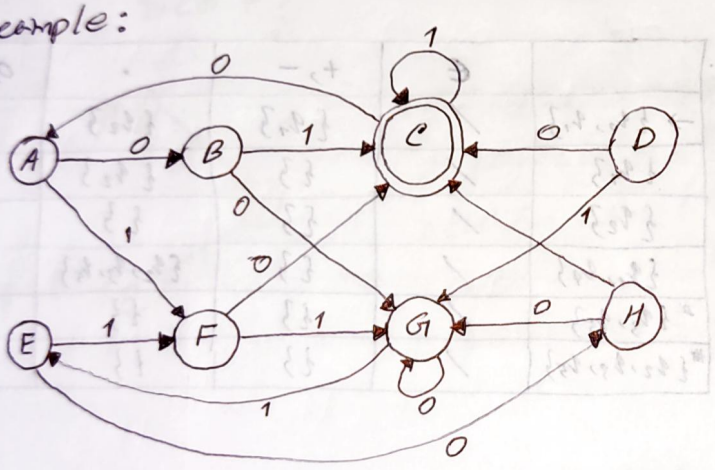


Figure: An automata with equivalent states

	A	B	C	D	E	F	G
B	X						
C	X	X					
D	X	X	X				
E	✓	X	X	X			
F	X	X	X	✓	X		
G	X	X	X	X	X	X	
H	X	✓	X	X	X	X	X

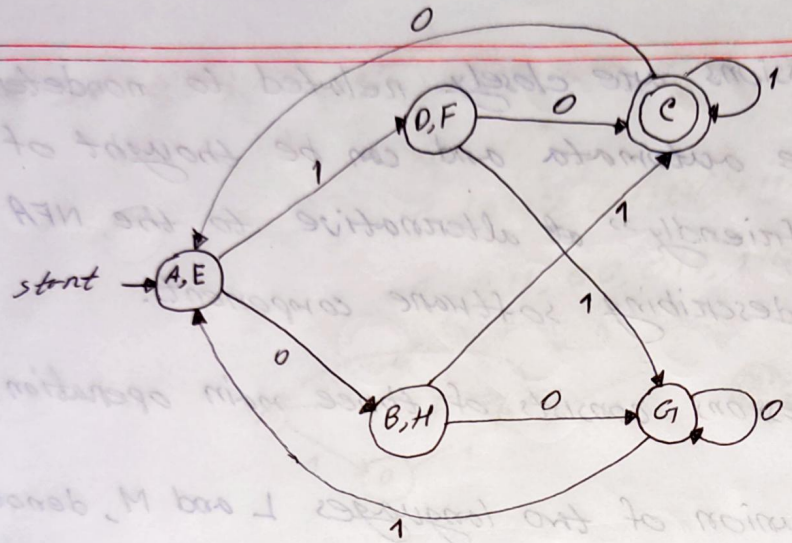


Figure: Minimum state DFA equivalent to previous figure.

Regular Expression

* Regular expressions are closely related to nondeterministic finite automata and can be thought of as a "user-friendly" alternative to the NFA notation for describing software components.

→ Regular expressions consists of three main operation:

(i) Union: The union of two languages L and M, denoted by L ∪ M. Example - if L = {001, 01, 111} and M = {∈, 001}

∴ L ∪ M = {∈, 01, 001, 111}

(ii) Concatenation: Example - if L = {001, 01, 111} and M = {∈, 001}

∴ L . M or LM = {001, 01, 111} . {∈, 001}

= {001∈, 001001, 01∈, 01001, 111∈, 111001}

= {01, 001, 111, 01001, 001001, 111001}

Note:
$$a \in = \in a = a$$
$$\in + a = a, \in$$

(iii) Closure: all possible combinations of a including ∈.

$RE = a^* = \in, a, aa, aaa, \ldots$

The concatenation RE for {0,1} is 01 or 0.1
The union RE for {0,1} is 0+1

Let, L = {0,1} ∴ $L^* = \{0,1\}^* = (0+1)^*$

$= \in, (0+1), (0+1)(0+1), (0+1)(0+1)(0+1), \ldots$

$= \in, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\ldots$

$L^* = L^0, L^1, L^2, L^3, \ldots$ where, $L^0 \rightarrow \epsilon$

$L^1 \rightarrow \{0, 11\}$

$L^2 \rightarrow \{0, 11\}\{0, 11\}$

Let, $L = \{0, 11\}$ $\therefore L^* = \{0, 11\}^*$

$= (0+11)^*$

$= \epsilon, (0+11), (0+11)(0+11), (0+11)(0+11)(0+11), \ldots$

$= \epsilon, 0, 11, 00, 011, 110, 1111, 000, 0011, \ldots$

Example : Let, $L = 01^* + 1$ or $(0(1^*)) + 1$

Here, $01^* = 0\epsilon, 01, 011, 0111, 01111, \ldots$

$\therefore L = 1, 0, 01, 011, 0111, 01111, \ldots$

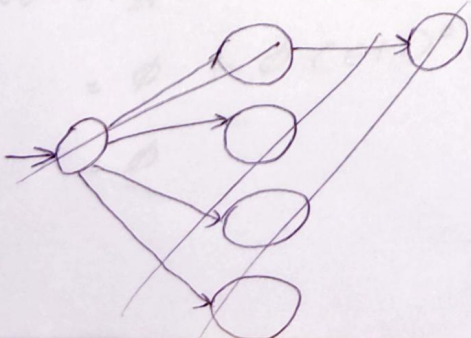
Example; Let, $RE = (01)^* + (10)^* + 0(10)^* + 1(01)^*$

Here, $(01)^* = \epsilon, 01, 0101, 010101, \ldots$

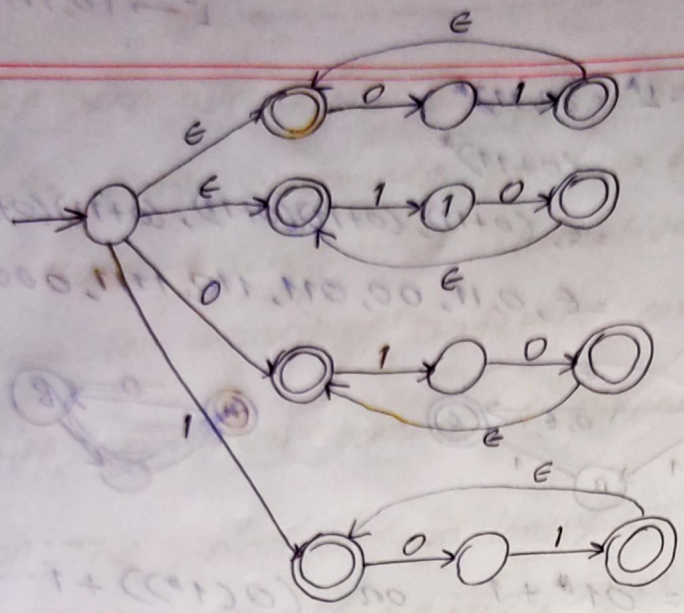
$(10)^* = \epsilon, 10, 1010, 101010, \ldots$

$0(10)^* = 0\epsilon, 010, 01010, 0101010, \ldots$

$1(01)^* = 1\epsilon, 101, 10101, 1010101, \ldots$

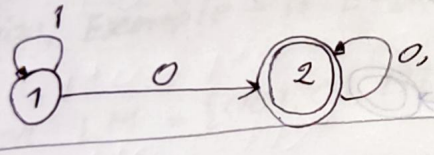


(P.T.O)



* Conversion from DFA's to Regular Expression:

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

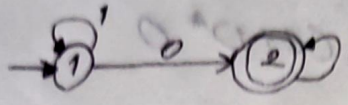


Note, formula:
$R_{ij}(k)$ where state i থেকে state j এ যাওয়ার জন্য সর্বোচ্চ কতগুলো note আছে তাই হবে k.

Main Formula:

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

→ Find Regular Expression from:



step-1:

$R_{11}^{(0)}$	$\epsilon+1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon+0+1$

[When $k=0$]

step-2: We know,

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^{*} R_{kj}^{(k-1)}$$

[When $k=1$]

So, $R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^{*} R_{11}^{(0)}$

$= (\epsilon+1) + (\epsilon+1)(\epsilon+1)^{*}(\epsilon+1)$

$= \epsilon+1+1^{*}$

$= 1^{*}$

$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^{*} R_{12}^{(0)}$

$= 0 + (\epsilon+1)(\epsilon+1)^{*} 0$

$= 0 + 1^{*} 0$

$= 1^{*} 0$

$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^{*} R_{11}^{(0)}$

$= \emptyset + \emptyset (\epsilon+1)^{*}(\epsilon+1)$

$= \emptyset$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= (\epsilon + 0 + 1) + \phi (\epsilon + 1)^* 0$$

$$= \epsilon + 0 + 1$$

step-3:

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= 1^* + 1^* 0 (\epsilon + 0 + 1)^* \phi$$

$$= 1^*$$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= 1^* 0 + 1^* 0 (\epsilon + 0 + 1)^* (\epsilon + 0 + 1)$$

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

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

$$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= \phi + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^* \phi$$

$$= \phi$$

$$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= (\epsilon + 0 + 1) + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^* (\epsilon + 0 + 1)$$

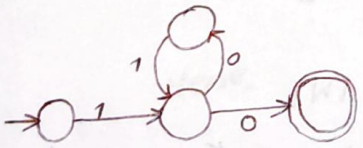
$$= (0 + 1)^*$$

Pumping Lemma

→ "pumping lemma" for showing certain languages not to be regular.

→ যে সকল language গুলো not to be regular. কিছু language গুলো regular ~~নাকি~~ নাকি যেটা pumping lemma দিয়ে বোঝা যায় না।

→ কোন একটা string বিশিষ্ট সংখ্যক (RE=01+0011+000111) হলে তাহলে NFA, DFA আঁকা যায়। বিশিষ্ট সংখ্যক নিয়ে আমাদের কোন মাথা ব্যথা নাই। আমাদের মাথা ব্যথা শুরু হয় যখন string/language টা infinite সংখ্যক হয় যাতে কোন বিশিষ্ট সংখ্যক দিয়ে সীমাবদ্ধ করা যাচ্ছে না। তখনই test করা শুরু আমরা যে language টা regular কি না regular কিনা।



⇒ 1 01 01 01 01 0 [01 যে কয় বার pump করা যাচ্ছে]

Note → কোন একটা certain string pattern যে কয় বার pump করে বাড়লে তখন decession নেওয়া যায় যে language টা regular হতে পারে এবং আরো regular নাও হতে পারে। কিন্তু pump করতে না বাড়লে তখন নিশ্চিতভাবে বলে দিবা language টা regular না।

* **Pumping Lemma for Regular Language:** Let us consider the language $L_{01} = \{0^n 1^n \mid n \geq 1\}$. This language contains all strings 01, 0011, 000111, and so on, that consist of one or more 0's followed by an equal number of 1's. We claim that L_{01} is not a regular language.

* **Theorem 4.1:** (The pumping lemma for regular languages) Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. For all $k \geq 0$, the string $xy^k z$ is also in L.

That is, we can always find a nonempty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (the case $k = 0$), keeps the resulting string in the language L.

Let, $L = \underbrace{1}_{x} \underbrace{0101010}_{y} \underbrace{1}_{z} (01)^*$

Here, $|W| = 10$, and $|xy| = 7$

Since, $|W| \geq n$, and $|xy| \leq n$ so $n \leq 10, 9$ for this language.

* Example:

Given that,

$L = 0^n 1^n$; where $n \geq 1$

Let, take a string 000111 that belongs to L.

case-1: let, $x = 00$

$y = 01$

$z = 11$

Here, $|W| = 6$

$|xy| = 4$

We know that, $|xy| \leq n \leq |W|$

$\therefore n = 4, 5, 6$

Here, $xy^k z$ will be true for $k \geq 0$ and $xy^k z$ should belong to L.

Now we will pump y 4 times. After pumping the new string is : 000101010111 which does not belong to L.

(P.T.O)

So, pumping lemma is not applicable to L. That's why 0^n1^n is not regular language.

case-2: let, $x = 0$
$y = 00$
$z = 111$

Here, $|w| = 6$ and $|xy| = 3$

We know that, $|xy| \leq n \leq |w|$

$\therefore n = 3, 4, 5, 6$

Here, xy^kz will be true for $k \geq 0$ and xy^kz should belong to L. But if we pump y 3 times, we get the string: 000000111 which does not belong to L.

~~So, L is regular.~~

So, pumping lemma is not applicable to L. That's why 0^n1^n is not regular language.

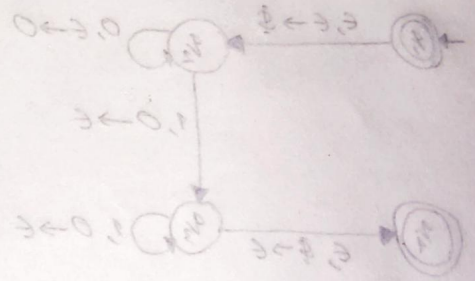
case-3: let, $x = 000$
$y = 111$
$z = \epsilon$

Here, $|w| = 6$ and $|xy| = 6$

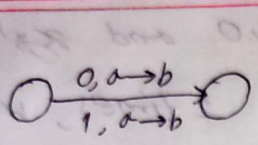
We know that, $|xy| \leq n \leq |w|$

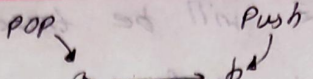
$\therefore n = 6$

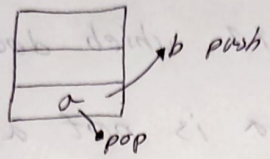
Here, xy^kz will be true for $k \geq 0$ and xy^kz should belong to L. But if we pump y 2 times, we get the string : 000 111 111 which does not belong to L. So, pumping lemma is not applicable to L. That's why 0^n1^n is not regular language.



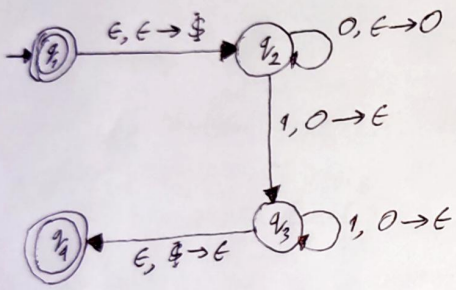
Push Down Automata







Example:



pop ε, and push $

pop ε, and push 0

pop 0, and push ε

pop $, and push ε

→A pushdown automata (PDA) is a way to implement a context free grammar in a similar way we design Finite Automata for Regular Grammar.

CFG – Context Free Grammer: CFG has two part:

① Variable / Non-Terminal ⇒ have production

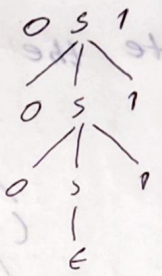
② Token / Terminal ⇒ have no production

→ 0^n1^n যেটা FA / RE / pumping lemma দিয়ে express করা যায় না, কিন্তু CFG দিয়ে possible.

Let, $S \longrightarrow 0S1$

$S \longrightarrow \epsilon$

$\therefore S \longrightarrow 0S1 \mid \epsilon$



count = left to right

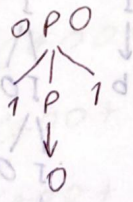
000∈111

= 000111

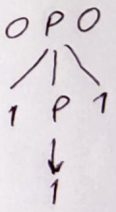
→ Example - Palindromes:

1. $P \longrightarrow \epsilon$
2. $P \longrightarrow 0$
3. $P \longrightarrow 1$
4. $P \longrightarrow 0P0$
5. $P \longrightarrow 1P1$

For 01010:



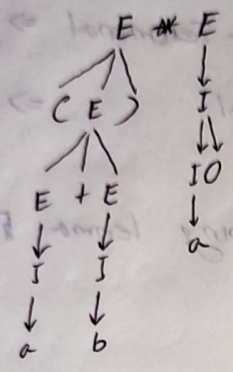
For 01110:



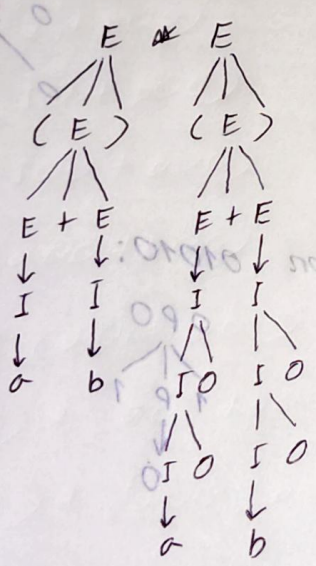
→ Example: Let us explore a more complex CFG that represents expressions in a typical programming language.

1. $E \longrightarrow I$
2. $E \longrightarrow E + E$
3. $E \longrightarrow E * E$
4. $E \longrightarrow (E)$
5. $I \longrightarrow a$
6. $I \longrightarrow b$
7. $I \longrightarrow Ia$
8. $I \longrightarrow Ib$
9. $I \longrightarrow I0$
10. $I \longrightarrow I1$

Now generate the string, (a+b)*a0



Now generate the string, (a+b)*(a00+b00)



(parse tree derivation)

এটা derivation হলো, আরো তিনটি way তে derivation করা যায়

① left-most derivation

② Right-most derivation

③ parse tree derivation

→ Write the left-most derivation of (a+b)*(a00+b00).

$E \xrightarrow{lm} E * E$

$\rightarrow (E) * E$

$\rightarrow (E+E) * E$

$\rightarrow (I+E) * E$

$\rightarrow (a+E) * E$

$\rightarrow (a+I) * E$

$\rightarrow (a+b) * E$

$\rightarrow (a+b) * (E)$

$\rightarrow (a+b) * (E+E)$

$\rightarrow (a+b) * (I+E)$

$\rightarrow (a+b) * (I0+E)$

$\rightarrow (a+b) * (I00+E)$

$\rightarrow (a+b) * (a00+E)$

$\rightarrow (a+b) * (a00+I)$

$\rightarrow (a+b) * (a00+I0)$

$\rightarrow (a+b) * (a00+I00)$

$\rightarrow (a+b) * (a00+b00)$

→ Write down the right most derivation of (a+b)*(a00+b00)

$E \xrightarrow{rm} E * E$

$\rightarrow E * (E)$

$\rightarrow E * (E+E)$

$\rightarrow E * (E+I)$

$\rightarrow E * (E+I0)$

$\rightarrow E * (E+I00)$

$\rightarrow E * (E+b00)$

$\rightarrow E * (I+b00)$

$\rightarrow E * (I0+b00)$

$\rightarrow E * (I00+b00)$

$\rightarrow E * (a00+b00)$

$\rightarrow (E) * (a00+b00)$

$\rightarrow (E+E) * (a00+b00)$

$\rightarrow (E+I) * (a00+b00)$

$\rightarrow (E+b) * (a00+b00)$

$\rightarrow (I+b) * (a00+b00)$

$\rightarrow (a+b) * (a00+b00).$

Ambiguous Grammars:

Let, $E \rightarrow E+E \mid E*E \mid id$

$id \rightarrow 0|1|2|3|4|5|6|7|8|9$

Now generate: $2+3*5$

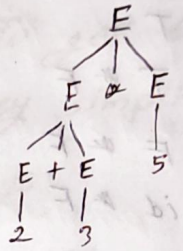
$E \xrightarrow{lm} E*E$

$\rightarrow E+E*E$

$\rightarrow 2+E*E$

$\rightarrow 2+3*E$

$\rightarrow 2+3*5$



Answer: 25

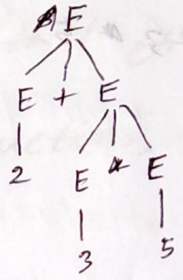
$E \xrightarrow{lm} E+E$

$\rightarrow 2+E$

$\rightarrow 2+E*E$

$\rightarrow 2+3*E$

$\rightarrow 2+3*5$



Answer: 17

We can find more than one left most derivation for $2+3*5$. So $2+3*5$ is a ambiguous grammar.

Removing Ambiguity From Grammars:

Let, $E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id$

$id \rightarrow 0|1|2|3|4|5|6|7|8|9$

← This is for unambiguous expression grammar.

(P.T.O)

Now generate : 2+3*5

$E \xrightarrow{lm} E+T$

$\rightarrow T+T$

$\rightarrow F+T$

$\rightarrow id+T$

$\rightarrow 2+T$

$\rightarrow 2+T*F$

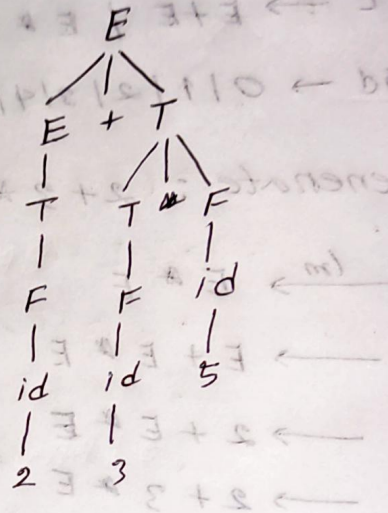
$\rightarrow 2+F*F$

$\rightarrow 2+id*F$

$\rightarrow 2+3*F$

$\rightarrow 2+3*id$

$\rightarrow 2+3*5$



Properties of Context-Free Language

* **Normal Forms for Context-Free Grammars:** Every CFL (without ϵ) is generated by a CFG in which all productions are of the form $A \to BC$ or $A \to a$, where A, B, and C are variables / non-terminals, and a is a terminal. This form is called Chomsky Normal Form. To get there, we need to make a number of preliminary simplifications, which are themselves useful in various ways:

① We must eliminate useless symbols, those variables or terminals that do not appear in any derivation of a terminal string from the start symbol.

② We must eliminate ϵ-productions, those of the form $A \to \epsilon$ for some variable A.

③ We must eliminate unit productions, those of the form $A \to B$ for variables A and B.

→ **Eliminating Useless symbols:** Consider the grammar:

$$S \to AB \mid a$$
$$A \to b$$

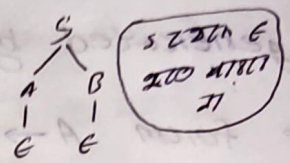
Here, B is not generating because B has no production. So, we must eliminate the production $S \to AB$, leaving the grammar:

$$S \to a$$
$$A \to b$$

Now we find that only S and a are reachable from S, So, we must eliminate $A \to b$.

→ Eliminating ε-Productions: Consider the grammer:

$$S \rightarrow AB$$
$$A \rightarrow aAA \mid \epsilon$$
$$B \rightarrow bBB \mid \epsilon$$



Now, consider production $S \rightarrow AB$

$$S \rightarrow AB \mid A \mid B$$

Next consider production $A \rightarrow aAA \mid \epsilon$

$$A \rightarrow aAA \mid aA \mid a$$

Next consider production $B \rightarrow bBB \mid \epsilon$

$$B \rightarrow bBB \mid bB \mid b$$

The two ε-productions of G yield nothing for G_1. Thus, the following productions:

$$S \rightarrow AB \mid A \mid B$$
$$A \rightarrow aAA \mid aA \mid a$$
$$B \rightarrow bBB \mid bB \mid b$$

→ Eliminating unit production: Consider the grammer:

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$
$$F \rightarrow I \mid (E)$$
$$T \rightarrow F \mid T * F$$
$$E \rightarrow T \mid E + T$$

এবারে $F \to I$, $T \to F$, and $E \to T$ রুলের মত করে যাই।

Pair	Productions
(E, E)	$E \to E+T$
(E, T)	$E \to T*F$
(E, F)	$E \to (E)$
(E, I)	$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(T, T)	$T \to T*F$
(T, F)	$T \to (E)$
(T, I)	$T \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \to (E)$
(F, I)	$F \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

so, the new grammer:

$E \to (E) \mid E+T \mid T*F \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$T \to (E) \mid T*F \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$F \to (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

→ Chomsky Normal Form: Let consider,

$A \to a$ $B \to b$ $Z \to 0$ $O \to 1$

$P \to +$ $M \to *$ $L \to (\ R \to)$

If we introduce these productions, and replace every terminal in a body that is other than a single terminal by the corresponding variable, we got the grammer:

$E \rightarrow LER \mid EPT \mid TMF \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$T \rightarrow LER \mid TMF \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$F \rightarrow LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$I \rightarrow a \mid b \mid IA \mid IB \mid IZ \mid IO$

$A \rightarrow a$

$B \rightarrow b$

$Z \rightarrow 0$

$O \rightarrow 1$

$P \rightarrow +$

$M \rightarrow *$

$L \rightarrow (\ $

$R \rightarrow)$

Now, all productions are in chomsky Normal Form except for those with the bodies of length 3: EPT, TMF, and LER. Now consider,

$C_1 \rightarrow PT \quad C_2 \rightarrow MF \quad C_3 \rightarrow ER$

The final grammer, which is in CNF, is shown below:

$E \rightarrow LC_3 \mid TC_2 \mid EC_1 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$T \rightarrow LC_3 \mid TC_2 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$F \rightarrow LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$

$I \rightarrow a \mid b \mid IA \mid IB \mid IZ \mid IO$

$A \rightarrow a$

$B \rightarrow b$

$Z \rightarrow 0$

$O \rightarrow 1$

$P \rightarrow +$

$M \rightarrow *$

$L \rightarrow ($

$R \rightarrow)$

$C_1 \rightarrow PT$

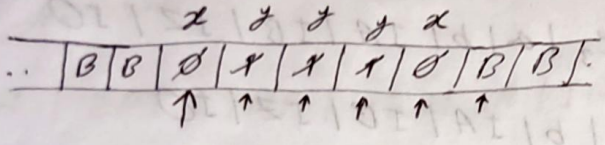
$C_2 \rightarrow MF$

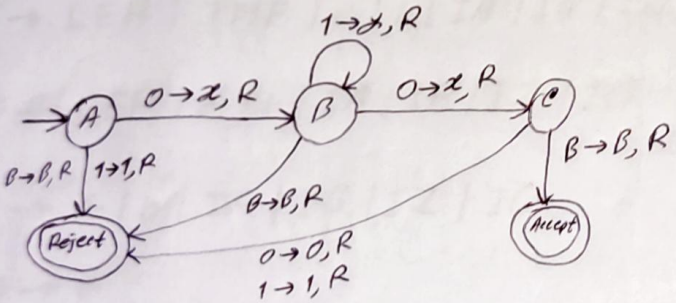
$C_3 \rightarrow ER$

* Design a turing machine for the following expression:

$$L = 01^*0$$

Let, take a string 01110 that belongs to L.





~~Design a turing machine where, L = {$0^n 1^n$ | n ≥ 1}~~

state	current symbol	effect
A	0	set state = "B" move right
A	1/B	set state = "reject" move right
B	0	set state = "C" move right
B	1	set state = "B" move right
B	B	set state = "reject" move right
C	B	set state = "accept" move right
C	0/1	set state = "reject" move right

* Design a turing machine for following expression,

$$L = \{a^n b^n \mid n \geq 1\}$$

Let take a string aaabbb that belongs to L.