Theory of Computation

Theory of computation বা গণনা তত্ত্ব হল কম্পিউটার বিজ্ঞান ও গণিতের একটি শাখা। Computational model বা গণনাকারী মডেলে algorithm প্রয়োগ করে দক্ষতার সাথে সমস্যা সমাধানের পদ্ধতি যে তত্ত্বে আলোচনা করা হয়, সেই তত্ত্বকে Theory of computation বা গণনা তত্ত্ব বলে।

Theory of Computation বা গণনা তত্ত্বের পরিধিকে তিনটি গুরুত্বপূর্ণ শাখায় ভাগ করা যায়। এর শাখা গুলো হল ঃ

i. Automata Theory
ii. Computational Theory
iii. Computational Complexity theory

Automata Theory: Automata Theory তে সচরাচর ব্যবহৃত গুরুত্বপূর্ণ Terminology গুলো হলঃ
i) Symbol ii) Alphabet iii) String iv) Empty string v) Length of string vi) Power set of string vii) Concatenation of string viii) Language ইত্যাদি।

Terminology গুলোর বর্ণনা নিম্নরূপঃ

i. Symbol: গণনা বা Computational Machine তৈরি করার অন্যতম উপাদান বা মৌলিক উপাদান হল Symbol । এতে ব্যবহৃত Symbol গুলো হল-
a. Letter– a, b ..... z.
b. Digits-- 0, 1 ..... 9 ইত্যাদি।

ii. Alphabet: Symbol এর সসীম (Finite) set (যা empty হবে না) কে Alphabet বলে। একে Σ দ্বারা প্রকাশ করা হয়।
1. Σ = {0, 1} এটি binary symbol এর একটি Alphabet.

iii. String: Alphabet এর Symbol গুলোর সসীম ধারা (Finite sequence) কে String বলে। একে Word ও বলা হয়। একে ω (ওমেগা) দ্বারা প্রকাশ করা হয়।
উদাহরণঃ ধরি, Σ = {0, 1} একটি বাইনারি সংখ্যার Alphabet, এর String সমূহ {0, 1, 01, 10, 00, 11, 101,.....} ইত্যাদি ।

iv. Empty String: যে String এ কোন symbol থাকে না, ঐ String কে Empty String বলে। একে E দ্বারা প্রকাশ হয়।
Example: ω = ∈, φ

v. Length of String:

একটি string এ symbol গুলোর মোট সংখ্যাকে length of string বলে। একে পরম মান ওমেগা (|ω|) দ্বারা প্রকাশ করা হয়।
উদাহরণঃ ω= 0101011 হলে the length of string |ω| = 7 হবে।

vi. Power of an Alphabet:
যদি ∑ একটি alphabet হয়, তাহলে পাওয়ার অফ আলফাবেট হবে ∑ এর এক্সপোটেনশিয়াল ফর্ম, আমরা তাকে প্রকাশ করতে পারি ∑n দ্বারা, এখানে n হলো স্ট্রিং এর লেংথ।

For example: If ∑ = {0, 1}, then ∑1 = {0, 1}, similarly ∑2 = {00, 01, 10, 11} or ∑3 = {000, 001, 010, 011, 100, 101, 110, 111} and so on.

vii. Concatenation of strings:
Concatenation হল দুই বা ততোধিক string এর মধ্যে সংযুক্তি। যদি x একটি string হয় , i সংখ্যক symbol থাকে (x=a, a1 ...ai) এবং y একটি string হয় ও y এ J সংখ্যক symbol থাকে (y=b1, b2 -----bj) তবে নতুন string xy এর length হবে i+j। string টি হবে, xy = a1 a2------ ai b_1, b_2 ....bj
Length of $|xy| = i + j$

উদাহরণ-১ঃ x = 01101 এবং y = 110 হলে xy এর length কত হবে?
Solution: x=01101, y=110
$|x| = 5$ $|y| = 3$
Length = |xy|=|x|+|y|=8
String = xy = 01101110

viii. Language: একটি Alphabet এর Symbol সমূহ সমন্বয়ে গঠিত নির্দিষ্ট কতগুলো String এর set কে Language বলে।
অর্থাৎ একটি Language হল Alphabet দিয়ে গঠিত String সমূহের Subset, Language $\subseteq \Sigma$
উদাহরণ:
1. n সংখ্যক তারপর n সংখ্যক 1 বসিয়ে উৎপন্ন সকল সমূহ String Language হবে = {∈ ,01,0011,000111} এখানে, n>0।
ii. সমান সংখ্যক 0 এবং 1 নিয়ে গঠিত string সমূহের Language হবে=∈ ,01,10,1001,.....?
iii) Prime সংখ্যাগুলো binary digit দিয়ে গঠিত string সমূহ Language = {10,11,101,111,1001,.....}
iv) Σ^* (সিগমা) কে যেকোন Alphabet 'Σ' (সিগমা) র Language বলা যায়।
v) যেকোন Language এর ক্ষেত্রে 'φ' ফাই হল Empty Language.
Note: ∈ হল → *Empty string*
φ হল → Empty Language
String হল → কতগুলো symbol এর সসীম ধারা।
Language হল →কতগুলো string এর সমষ্টি।
vi. {∈}ইহা Empty String নিয়ে গঠিত একটি Language যেকোন Alphabet এই Language টি বিদ্যমান থাকে।
[Note:φ ≠ {∈}]

Set গঠন পদ্ধতি ব্যবহার করে Language গঠনঃ
একটি Language বর্ণনা করার সাধারণ (Common) পদ্ধা হল Set গঠন পদ্ধতি বা set formers,
$\{\omega, something\ about\ \omega\}, w = set\ of\ words$
উদাহরণঃ
i. {ω| ω গঠন হবে সমান সংখ্যক 0 এবং 1 নিয়ে}

ii. {ω| ω হবে prime সংখ্যাগুলোর binary রুপ}
Language তৈরির আরো একটি উপায় নিচে আলোচনা করা হলঃ Expression এর Parameter ব্যবহার করে ω কে Replece করা যায়। এখানে Parameter সমূহের উপর শর্ত আরোপ করে। Language তৈরি করা যায়। যেমন-
i. $\{1^n 0^n In \geq 1\}$
Solution: এখানে, n = যতগুলো 1 হবে 1এরপর ততগুলো 0 হবে। n এর সাইজ 1 এর চেয়ে বেশি বা সমান হতে হবে।
set of strings, $\Sigma^* = \{10, 1100, 111000\}$
ii. $\{ 0^i 1^j | 0 \leq i \leq j\}$
এই Language এর struing সমূহ যে কোন সংখ্যক 0 এর পর যে কোন সংখ্যক 1 নিয়ে গঠিত হয়।

প্রশ্ন ১. What is finite automata? What are the components of finite automata model?
Answer: Finite Automata: Finite Automata এক ধরনের (Simplest) machine এটি কত গুলো symbol নিয়ে গঠিত Pattern সমূহ input হিসেবে গ্রহন করে এবং input অনুযায়ী state পরিবর্তন করে outupt প্রদান করে।

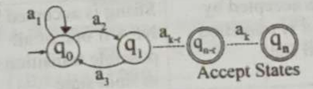


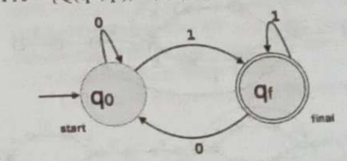
Fig: Finite Automata

এই Machine টি Starting এর Operation শেষে দুটি অবস্থা বুঝায়। যার একটি হল Accept State এবং অপরটি হল Reject State। এতে String সমূহ Input State থেকে শুরু করে Final State এ Successfully পৌছালে Accpet State বুঝায়, অন্যথায় Reject State বুঝায়।

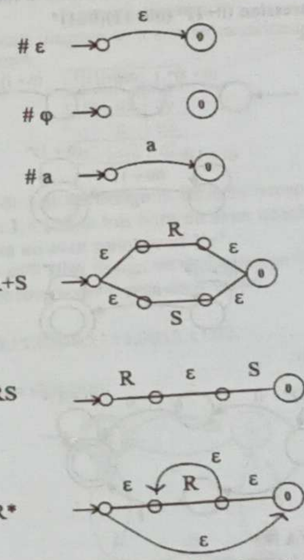
প্রশ্ন ২. Finite Automata তে কয়টি Tuple বিদ্যমান?
উত্তরঃ একটি Finite Automata তে ৫ ধরনের Tuple (Q, Σ, δ, q_0, F) বিদ্যমান। যথাঃ
i) Q = সসীম State এর Set (Finite set of State)
ii) Σ = সসীম Input Symbol এর Set (Finite set of the input Symbols)
iii) δ= Transmission Function (কোন State থেকে কোন State থেকে কোন State এ যাবে তা নির্দেশ করে।)
iv) q_0= Initial State (Machine এর শুরু State নির্দেশ করে)
v) F = Final State (Machine টি এই State কার্য শেষ করে)

Type of Automata:
দুই ধরনের Finite Automata রয়েছে। যথাঃ
i) DFA (Deterministic Finite Automata)
ii) NFA (Non deterministic Finite Automata)

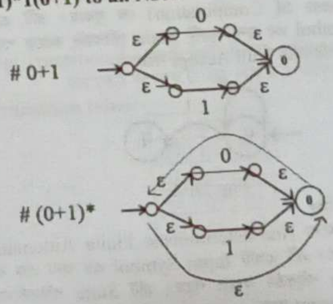
উদাহরণঃ
মনে কর, FA তিন digit এর যেকোন binary value accept করতে পারে, যার শেষ digit টি 1। এখানে,
FA = {Q(q0,qf), Σ(0,1), q0, qf, δ}

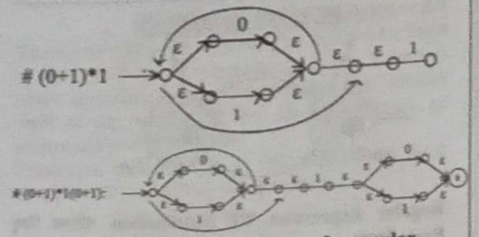


Regular Expression হতে Automation গঠনের কিছু Basic নিয়মঃ



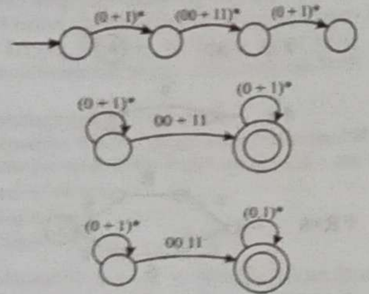
প্রশ্ন ৩. Let us convert the Regular expression (0+1)*1(0+1) to an NFA



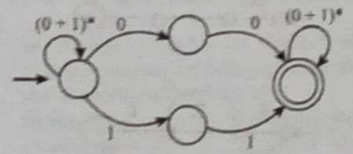


প্রশ্ন ৪. Construct the finite automata for regular expression (0+1)* (00+11)(0+1)*

Ans:



Finally,



প্রশ্ন ৫. DFA কি?

উত্তরঃ DFA হলো Deterministic Finite Automata এর সংক্ষিপ্ত রূপ। Deterministic শব্দটি দ্বারা গণনার অদ্বিতীয়তা (Uniqueness of Computation) কে বুঝায়। এটি একটি Input Symbol এর জন্য একটি State পরিবর্তন করতে পারে। এটি State পরিবর্তনে Null Accept করে না।

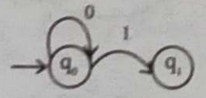


Fig: DFA

প্রশ্ন ৬. NFA কি?

উত্তরঃ NFA হলো Non-deterministic Finite Automata এর সংক্ষিপ্ত রূপ। এটি একটি Input Symbol এর জন্য এক বা একাধিক State পরিবর্তন করতে পারে। এটি State পরিবর্তনের ক্ষেত্রে Null Accept করে।

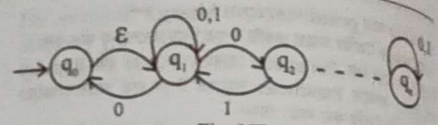


Fig: NFA

প্রশ্ন ৭. Difference between NFA and DFA are given bellow:

DFA	NFA
DFA stands for deterministic finite automata	NFA stands for non-deterministic finite automata
Back tracking is allowed in DFA	Back tracking is not allowed in NFA
More difficult to construct	Easier to construct
DFA cannot use empty string transition	NFA can use empty string transition
Requires more memory space	Requires less memory space
String is accepted by DFA if it is transition into final state	String is accepted by NFA if one of all possible transition ends in final state
DFA can be understood as one machine	NFA can be understood as multiple little machine
Next possible state is distinctly set	Each pair of state and input symbol can have many possible next state

প্রশ্ন ৮. DFA এবং NFA এর মধ্যে সম্পর্ক দেখাও।

উত্তরঃ DFA এবং NFA এর মধ্যে সম্পর্ক (Relationship) নিম্নরূপ

i) সকল DFA কে NFA বলা যায়, কিন্তু সকল NFA কে DFA বলা যায় না।

ii) DFA এবং NFA উভয় Machine এ একাধিক Final State থাকতে পারে।

iii) DFA Compiler এর Lexical Analysis এর ক্ষেত্রে ব্যবহৃত হয়।

iv) NFA একটি তাত্ত্বিক (Theoretical) ধারণা (Concept)

❖ Transition diagram/state Diagram: DFA ও NFA এর Transition diagram/state Diagram হল একটি Directed graph। নিম্নলিখিত উপায়ে এটি তৈরি করা যায়

i. Q তে অবস্থিত প্রতিটি state এর node কে circle এর মধ্যে প্রকাশ করা হয়। যেমন- q_i

ii. Directed edge a ব্যবহার করে node p তে যাওয়া কে $\delta(q,a) = p$ দ্বারা প্রকাশ করা হয়।

iii. $q_0 \rightarrow$ *start state* কে Arrow (→) সহ লেখা হয়।
→ q_0 *or* $\bar{q}_0$ [→ q_0 or q_0]

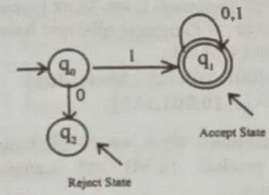
iv. F→ Final state কে double circle এর মধ্যে লিখা হয়। q_n [q_n]

v. Σ → input symbol এর Finite set ।

প্রশ্ন ৯. DFA এর NFA এর কিছু সমস্যা ও সমাধানঃ

Example-1: DFA with Σ= {0,1} accepts all strings starting with 1 । অথবা Σ= {0,1} হলে, একটি DFA design কর যার সকল String 1 দ্বারা শুরু হবে।

Sol^n:



এই DFA টি ঐ সকল strting গুলো accept করবে যাদের প্রথম digit `1' হবে, অন্যথায় Reject হবে।

$\Sigma^* = \{10,110,111000, \ldots\ldots\ldots\ldots\}$

Example-2: Design a DFA that accepts all an only the strings of o's and 1's that have the sequence '01' Somewhere in the string. Show the transition table.

অথবা, এমন একটি DFA design কর, যার string সমূহ 0 এবং 1 নিয়ে গঠিত এবং String এর যেকোন স্থানে '01' থাকবে। এবং transition table দেখাও।

Sol^n: এমন একটি DFA design করতে হবে যার String সমূহ হবে,

$\Sigma^* = \{01, 0101,$
$001, 0010, 0011 \ldots\ldots\ldots\ldots\ldots\}$

DFA এর Transition diagram:

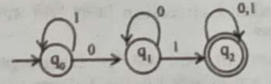


Fig: এটি DFA এর Transition diagram , যা sub-string হিসেবে '01' কে accept করে।

Transition Table:

	0	1
q0	q1	q0
q1	q1	q2
q2	q2	q2

Fig: উপরোক্ত DFA এর Transition table.

❖ Transition table:

একটি Function δ (q, a) = p এর একটি Conventional tabular উপস্থাপন (যেখানে, দুটি argument (q,a) মিলে একটি value return করে) কে transition table বলে।

* Row দিয়ে State উপস্থান করা হয়।
* Column দিয়ে Input Symbol উপস্থাপন করা হয়।

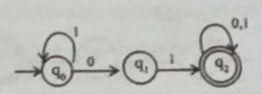


Fig: state diagram

উপরোক্ত state diagram এর জন্য state/transition table টি হবে নিম্নরূপ:

	0	1
q0	q1	q0
q1	φ	q2
q2	q2	q2

Example-3: Let us design a DFA to accept the language: L= {ωI ω has both an even number of 0' s and/on an even number of 1's[4]

অথবা, এমন একটি DFA design কর যার Language টি হবে L = {ω|ω তে জোড় সংখ্যক 0 অথবা এবং 1 থাকবে}

Soln:

$\Sigma^* = \{00,11,0000,1111,0011,1100, \ldots\ldots\ldots\ldots\}$

Transition diagram:

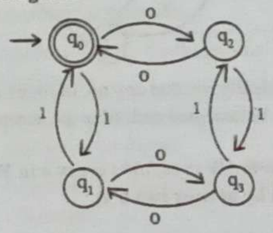


Fig: Transition diagram (এটি ঐ সমস্ত string accept করে যার জোড় সংখ্যক 0 এবং /অথবা 1 থাকে)

Transition tabel:

	0	1
q0	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

Example-4: একট NFA design কর, যা 0 এবং 1 দ্বারা গঠিত ঐ সকল string accept করবে। যাদের শেষে 01 থাকে। Or, Draw an NFA, Whose job is to accept all and only the strings of 0's and 1's that end in 01.
Sol^n: $\Sigma^* = \{01,001,10,00 \ldots\ldots.01 \ldots\ldots.01, \ldots.\}$
Transition diagram:

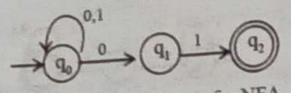
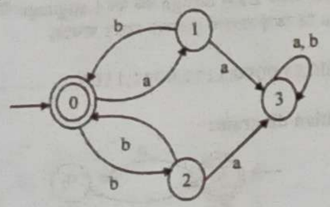


Fig: Transition diagram for NFA

Transition Table:

	0	1
->q0	{q0,q1}	{q0}
q1	φ	{q2}
*q2	φ	φ

Example-5: Construct DFA for this language: (ab+bb)*
Soln:



we can clearly see that any no. of loops of 'ab' and 'bb' can be accepted and '0' is the accepting state.

Example-6: {W ⊂ (a, b)* | every a in W is followed by at least two b's}
Ans:

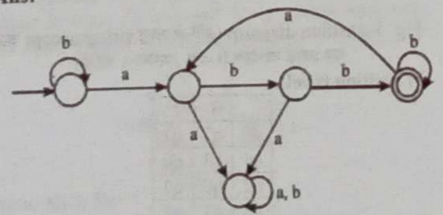


Fig: DFA

Regular Expressions

Regular Expressions: রেগুলার এক্সপ্রেশন হচ্ছে স্ট্রিং ম্যানিপুলেশন বা স্ট্রিং নিয়ে কাজ করার জন্য অন্যতম বহুল ব্যবহৃত একটি টুল। এটা একধরনের ডোমেইন স্পেসিফিক ল্যাঙ্গুয়েজ। রেগুলার এক্সপ্রেশন(Regular Expression) ব্যবহার করে আমরা কোন স্ট্রিং(String) থেকে কোন টেক্সট সার্চ অথবা কোন টেক্সট রিপ্লেজমেন্টস করতে পারি।
Regular Expression হল NFA এবং DFA এর Algebraic Equation. একটি Automata একই জাতীয় language এর সকল Form define করতে পারে Regular Expression এর মাধ্যমে তা প্রকাশ করা হয়।

❖ **RE এর Operator সমূহের বর্ণনা নিম্নরুপ:**
1. Union: দুটি Language L এবং M এর Union কে LUM দ্বারা প্রকাশ করা হয়। যা Language দুটির সকল উপাদানের সমন্বয়ে গঠিত একটি সেট LUM হবে।
ধরি, L= {001, 10,111} , M={ϵ,001}
∴ LUM= {ϵ,10,001,111}

2. Concatenation: যদি L এবং M দুটি Language হয় তবে 'dot' product (L.M) হবে Language দুটির Concatenation.
ধরি, L= {001,10,111} , μ = {ϵ,001}
L.M = {001, 10, 111, 001001, 10001, 11100}

3. Closure (star, or Kleene Closure): Closure এ Language L কে L^* দ্বারা প্রকাশ করা হয়। এটি Language L থেকে অসংখ্য String তৈরি করতে পারে। এতে string এর repetition এবং String এর মতো concatenation ও করা যায়। তবে একই string একবারই select করা যাবে।
যদি L={0,1} হয়, তবে L^* এর String সমূহ যেকোন সংখ্যক 0 এবং 1 দ্বারা তৈরি করা যাবে। অর্থাৎ
$L^* = 0,1,01,001,011,\ldots\ldots\}$

❖ **RE ব্যবহৃত Operator গুলোর Presidency:**
i. সর্বোচ্চ Precedency: star '*' Operator.
ii. ২য় Precedency: dot '.' Operator.
iii. ৩য় বা শেষ Precedency: union '+' Operator.

❖ **Regular Expression তৈরির ভিত্তি গুলো নিম্নে আলোচনা করা হল:**
i. Constant 'ϵ' এবং 'φ' Language দুটির RE হবে {ϵ} এবং φ অর্থাৎ $L(\epsilon) = \{\epsilon\}$ এবং $L(\varphi) = \varphi$
ii. Symbol a নিয়ে গঠিত একটি Language {a} হলে এর RE হবে {a} অর্থাৎ L(a) ={a}
iii. Variable: RE এর একটি variable কে সাধারণত বড় হাতের অক্ষর বা italic form এ প্রকাশ করা হয়। যেমন 'L' হল একটি Variable 'L' এর সাহায্যে RE এ ব্যবহৃত Language প্রকাশ করার জন্য ব্যবহৃত হয়।

[নিচের টেবিল এর নিয়ম গুলো দেখলে সহজেই রেগুলার এক্সপ্রেশন করা যাবে।]

Regular Expression

Regex	Description
^	The caret symbol matches the start of a string without consuming any character.
$	The dollar symbol matches the end of a string without consuming any character.
[abc]	Matches either a, b or c from within the square brackets [ ].
[a-z]	Matches any lowercase character from a to z.
[A-Z]	Matches any uppercase character from A to Z.
[0-9]	Matches any whole numbers from 0 to 9.
[a-zA-Z0-9]	Matches any character from a to z or A to Z or 0 to 9.
[^abc]	Matches any character except a, b or c.
[^A-Z]	Matches any characters except those in the range A to Z.
(apple)	The grouping character ( ) matches anything that is within the parenthesis.
\| | A vertical bar matches any element separated. |
\ | A back slash is used to match the literal value of any metacharacter (e.g. try using \. or \@ or \$ while building regex). |
\number | Matches the same character as most recently matched by the nth (number used) capturing group. |
\s | Matches any space or tab. |
\b | Matches, without consuming any characters immediately between a character matched by \w and a character not matched by \w (in either order). \b is also known as the word boundary. |
\d | Matches any equivalent numbers [0-9] |
\D | Matches anything other than numbers (0 to 9). |
\w | Matches any word character (i.e. a to z or A to Z or 0 to 9 or _). |
\W | Matches anything other than what \w matches (i.e. it matches wild cards and spaces). |
? | A question mark used just behind a character matches or skips (if not required) a character match. |
* | An asterisk symbol used just behind a character matches zero or more consecutive character. |
+ | The plus symbol used just behind a character matches one or more consecutive character. |
{x} | Matches exactly x consecutive characters. |
{x,} | Matches at least x consecutive characters (or more). |
{x,y} | Matches between x and y consecutive characters. |

Examples related to use of numbers
For all text type questions that use numbers, do not forget to type numbers under the appearance column.

XLSForm Regex	Description
regex(., '^[0-9]{10}$') or regex(., '^\d{10}$')	Restrict mobile number to ten digits

XLSForm Regex	Description
regex(., '^[0-9]{4}.[0-9]{2}.[0-9]{2}$') or regex(., '^\d{4}\.\d{2}\.\d{2}$')	Restrict an input to 1234.56.78
regex(., '^[01-99]{2}$') and (. >= 01)	Restrict an input between 01 to 99 digits where input format of a single number (like 1 or 2) is not allowed
regex(., '^(12|345)$')	Restrict an input to either to 12 or 345
regex(., '^[1-9][0-9]{8}$') or regex(., '^[^0][0-9]{8}$')	Restrict an input of nine digits where the first number can't be 0
regex(., '^\d$')	Restrict an input to one digit in between 0 to 9
regex(., '^\d{5}$')	Restrict an input to five digits in between 0 to 9
regex(., '^\d{2}\.\d{3}$')	Restrict an input to two digits and three decimals (e.g. 12.345)
regex(., '^\d{2}(\.\d{3})?$')	Restrict an input to two digits and three decimals (while the decimals are optional) (e.g. 12 or 12.345)

Examples related to use of letters

XLSForm Regex	Description
regex(., '^[a-z]{1,6}$')	Restrict an input to any lowercase letters (up to 6 characters long)
regex(., '^[A-Z]{1,10}$')	Restrict an input to any uppercase letters (up to 10 characters long)
regex(., '^(Apple|Orange|Banana)$')	Restrict an input to only either to Apple or Orange or Banana
regex(., '^p(ea|ai)r$')	Restrict an input to only pear or pair
regex(., '^[A-Za-z0-9._%+-]+@[A-Za-z0-9-]+[.][A-Za-z]{2,}$')	Restrict an input with a valid email address
regex(., '^[A-Z]{1}[a-z]{1,}[ ]{1}[A-Z]{1}[a-z]{1,}$')	Restrict an input of the beneficiaries name where the initials of the first name and last name are uppercase e.g. Kobe Bryant
regex(., '^\w{1,}\s(\w{1,})?(\s)?\w{1,}$')	Restrict an input of the beneficiaries name with first name, middle name (if any) and last name e.g. Kobe Bean Bryant
regex(., '^([A-Z]{1}[a-z]{1,}\s)([A-Z]{1}[a-z]{1,}\s?)+$')	Restrict an input of the beneficiaries' full name where the initials of the names are in uppercase and the name are quite long (often greater than 3 words) e.g. Samayamantri Venkata Rama Naga Butchi Anjaneya Satya Krishna Vijay (this is an example of south Indian names)
regex(., '^(\D+)\s(\D+)\s?\1$')	Restrict an input of the beneficiaries' first name (so that you are able to capture the exact spelling) where the enumerators are forced to enter the beneficiaries first name twice e.g. Kobe Bryant Kobe. (This could be helpful when you are trying to document beneficiaries details where a typo error could cost you heavy)
regex(., '^(\D+)\s(\D+)\s?\2$')	Restrict an input of the beneficiaries' last name (so that you are able to capture the exact spelling) where the enumerators are forced to enter the beneficiaries last name twice e.g. Kobe Bryant Bryant. (This could be helpful when you are trying to document beneficiaries details where a typo error could cost you heavy)
regex(., '^colou?r$')	Restrict a character within a word by using the ? (quantifier) e.g. allow either color or colour as an input
regex(., '^ah*!$')	Restrict a character within a word by using the * (quantifier) e.g. allow either a! or ah! or ahh! or ahhh! and so on as an input
regex(., '^ah+!$')	Restrict a character within a word by using the + (quantifier) e.g. allow either ah! or ahh! or ahhh! and so on as an input
regex(., '^\D$')	Restrict an input to a non-digit character (e.g. a or c or ! or # or % etc.)
regex(., '^\D{5 }$')	Restrict an input to five non-digit character (e.g. aZcB!#% etc.)

Examples related to use of a combination of letters and numbers

XLSForm Regex	Description
regex(., '^\w$')	Restrict one character which matches between a to z or A to Z or 0 to 9 or _ (i.e. match one character from [a-zA-Z0-9_])
regex(., '^\w{3}$')	Restrict three character which matches between a to z or A to Z or 0 to 9 or _ (i.e. match one character from [a-zA-Z0-9_])
regex(., '^[A-Z]{3}[_][A-Z]{3}[_][0-9]{4}[_][0-9]{4}$')	Restrict your beneficiary ID to a specific format e.g. CAR_PRC_2020_0048
regex(., '^CAR-PRC-2020-[0-9]{4}$')	Restrict your beneficiary ID to a specific format e.g. CAR-PRC-2020-0048 (where the enumerators should enter an exact match from CAR to - i.e. CAR-PRC-2020- and can enter any 4 digit serial number)
regex(., '^[$\|£]\d{3}$')	Restrict a currency input of three digits with a currency sign (either dollar or pound) in front (e.g. $999 or £500)

XLSForm Regex	Description
regex(., '^\W*(\w+\b\W*){3}$')	Restrict an exact input of number of words (e.g. to restrict exactly 3 words I love you.)
regex(., '^\W*(\w+\b\W*){3,5}$')	Restrict an input of number of words (e.g. to restrict a range of words say 3 to 5)

❖ FA থেকে Regular Expression তৈরি করার নিয়মঃ

Finite Autamata কে Regular Expression এ রুপান্তর করার ক্ষেত্রে Path (**state** থেকে state) কে $R_{ij}^{(k)}$ দ্বারা উপস্থাপন করা হয়। label k=0 ধরি state 1 থেকে n পর্যন্ত RE তৈরি করার ক্ষেত্রে path গুলোর জন্য দুটি শর্ত যাচাই করতে হয়।

শর্ত গুলো হলঃ

1. node (state) i থেকে j পর্যন্ত (→) তীর চিহ্ন আছে কিনা।
2. শুধুমাত্র কয়েকটি node 'i' নিয়ে গঠিত, ∈ path এর length হবে '0'

যদি i≠j হয়, তবে শর্ত '1' সত্য হবে। এবং input symbol এর জন্য state i থেকে state j পর্যন্ত transition নির্ণয় করতে হবে।

i. যদি কোন Symbol (a) না থাকে, তবে $R_{ij}^{0} = \varphi$

ii. Exactly একটি symbol (a) থাকে, তবে $R_{ij}^{0} = a$

iii. যদি অনেকগুলো symbol (a, a_2 .....a_k) থাকে তবে $R_{ij}^{0} = a1 + a2 + \ldots\ldots + a_k$

যদি i=j হয়, তবে path length 0 হবে এবং সবগুলো loop i এর মধ্যে ঘুরতে থাকবে।

Path length '0' কে RE এ ∈ দ্বারা প্রকাশ করা হয়।

Expression (i) থেকে (iii) পর্যন্ত ∈ যোগ করে ক্ষেত্রফল expression পাওয়া যায় সেগুলো হলঃ-

i. [no symbol a] $R_{ij}^{0} = \in$

ii. [one symbol a], $R_{ij}^{0} = \in + a$

iii. [multiple symbols a_1+a_2 .....+a_k];

$$R_{ij}^{0} = \in + a_1 + a_2 + \ldots\ldots + a_k$$

❖ **Remember** : Path i থেকে j পর্যন্ত Regular Expression টি হবে,

$$R_{ij}^{(k)} = R_{ik}^{(k-1)} + R_{ij}^{(k-1)}(R_{ij}^{(k-1)}R_{kj}^{(k-1)}$$

Example-1: Let u convert the DFA of the following figure to a regular expression.

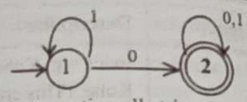


Fig: A DFA, accepting all strings that have at least one '0'

Solution: ধরি, K=0 এবং $R_{ij}^{(0)}$ অর্থাৎ state '1' থেকে state '2' input symble '0' এবং '1' নির্ণয় করি।

$R_{11}^{(0)}$	∈ +1	∈ state '1' এর beginning এ ending state same বুঝায়। state '1' এ input symbol '1' আছে তাই এ একটি term হবে, '1'।
$R_{12}^{(0)}$	θ	state '1' থেকে state '2' তে যাওয়ার তীরক '0' দ্বারা চিহ্নিত করা হয়েছে তাই এ একটি term হবে '0'
$R_{21}^{(0)}$	φ	state '2' থেকে state '1' এ যাওয়ার জন্য যেহেতু কোন path নাই সেহেতু $R_{21}^{(0)} = \varphi$ হবে।
$R_{22}^{(0)}$	∈ +0 + 1	' ∈ ' হয় যদি '0' পর আর কোন input symbol না থাকে। term '1' বা '0' হয় যদি একটি '0' পর বা একাধিক input symbol 0 '1' থাকে।

আবার উপরের DFA এর জন্য $R_{ij}^{(1)}$ গননা করতে চাইলে নিম্ন সাধারণ নিয়ম অনুসরণ করতে হয়ঃ

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)}\left(R_{11}^{(0)}\right)^{*}R_{11}^{(0)}$$

	By direct substitution	simplified
$R_{11}^{(1)}$	$\in +1 + (\in +1)(\in +1)^{*}(\in +1)$	1*
$R_{12}^{(1)}$	$(\in +1)\ (\in +1)^{*}0$	1*0
$R_{21}^{(1)}$	$\varphi + \varphi(\in +1)^{*}(\in +1)$	φ
$R_{22}^{(1)}$	$(\in +0+1)^{*}0$	(∈ +0+1)

[**Note:** DFA থেকে RE হবে $R_{ij}^{(K)}$,

i. k=0 হলে state 'i' থেকে state 'j' পর্যন্ত arc (→) এর উপর ভিত্তি করে RE, $R_{ij}^{(0)}$ নির্ণয় করতে হবে।

ii. k = 1, 2,... হলে RE টি হবে।

$$R_{ij}^{(0)} = R_{ij}^{(k-1)} + R_{ik}\left(R_{kk}^{(k-1)}\right)^{*}R_{Kj}^{(k-1)}$$

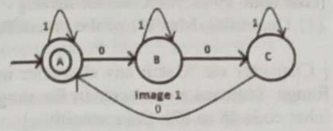
আরোও দুটি নিয়ম হলঃ

i. $\varphi R = R\varphi = \varphi$[annihilator for concatenation]

ii. $R + \varphi = \varphi + R = R$ [*identity for union*]

প্রশ্ন ১. State diagram of DFA using binary strings having 0 with multiple of 3 also showing reguular expression.

Ans: Regular Expression : (1*01*01*01*)*



Example

Example-1: Let us write a regular expression for the set of strings that consist of alternating 0's and 1's. [অথবা, চল এমন একটি **Regular Expression** তৈরি করি, যাতে একটি শুন্যর পর একটি এক নিয়ে অথবা একটি একের পর একটি শূণ্য নিয়ে গঠিত **String** সমূহের **set** পাওয়া যাবে।]

Solution: Alternating 0's and 1's অথবা, একটি শূন্যের পর একটি এক অথবা একটি একের পর একটি শূণ্য নিয়ে গঠিত RE টি হল- $(01)^{*} + (10)^{*} + 1(01)^{*} + 0(10)^{*}$

বর্ণনায় এখানে দুটি Symbol 0 এবং 1 RE এর basic rules হতে পাই 0 ও 1 এর Language হল {0} ও {1} এবং RE হল 0 ও 1। {0} ও {1} কে Concatenate করে {01} একটি Language পাওয়া যায়। আর {01} এর Regular Expression হবে 01।

আবার, 01 নিয়ে গঠিত সকল String (যেমন- 0101, 01,∈ 010101 .....) এর RE হবে $(01)^{*}$ *। (01) কে প্রথম বন্ধনি দ্বারা আবদ্ধ করা হয়েছে, যেন* $(01)^{*}$ *এর সাথে মিলে যায়। 01* বলতে বুঝায় একটি শূণ্যের পর একাধিক 1 হতে পারে। আমরা বলতে পারি Language টি হবে* $L((01)^{*})$ *। কিন্তু* $((01)^{*})$ *Language টি exact উত্তর না। কারণ একটি String '0' অথবা '1' দিয়ে শুরু করে, '0' অথবা '1' দ্বারা শেষ করা যায়। যেমন- (01,10,010, 1010,101) যা alternating '0' এবং '1' বুঝায়।*

'1" দিয়ে শুরু '0' দিয়ে শেষ Language টি হবে, $L((10)^{*})$ *এবং Regular expression টি হল* $(10)^{*}$*. '1' দিয়ে শুরু '1' দিয়ে শেষ RE টি হবে-* $1(01)^{*}$ *'0' দিয়ে শুরু '0' দিয়ে শেষ RE টি হবে* $0(10)^{*}$

এখন, সম্পূর্ণ RE টি হবে সবগুলো RE এর Concatenation -

$\therefore RE = (01)^{*} + (10)^{*} + 1(01)^{*} + 0(10)^{*}$

Example-2: The set of strings over alphabet {a,b,c} containing at least one 'a' and at least one 'b' write an RE for the above language.

অথবা, alphabet {a,b,c} হতে কমপক্ষে একটি 'a' ও একটি 'b' নিয়ে গঠিত string এর set গঠন করা গেলে,এর RE টি লেখ।

Solution: alphabet {a,b,c} হলে কমপক্ষে একটি 'a' ও একটি 'b' নিয়ে গঠিত RE টি হবে

c*a(a+c)*b(a+b+c)+c*b(b+c)*a(a+b+c)*

Example-3: The set of string of 0' s and 1' s whose tenth symbol from the right end is 1. write an RE for the above language.

অথবা, Symbol '0' এবং '1' নিয়ে গঠিত string সমূহের set, যেখানে string সমূহের ডান থেকে ১০ম symbol '1' হবে। তার জন্য একটি RE লিখ।

Solution: '0' এবং '1' নিয়ে গঠিত string যার ১০ম symbols ডান দিক থেকে '1' হবে তার RE টি হবে।

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

বিগত পরীক্ষার প্রশ্ন

প্রশ্ন ১. Write a Regular Expression for user considers the condition:

i. The string 'PGCB' following by

ii. 5 Digit Number following by

iii. 4 Digit Postal code following by

Answer: Regular Expression: (PGCB)([0-9]{5})[0-9]{4}

প্রশ্ন ২. Give regular expression for set of strings which either have 'a' followed by some b's or all b's also containing 'Ɛ'

Ans: The regular expression is ReX = ab*|b*| Ɛ = b*(a| Ɛ)| Ɛ.

প্রশ্ন ৩. Design a regular expression for inputs a and b {a, b} that contains exactly 2 a's.

Sol. Here, the description defines that there will be only 2 a's. In between number of b's is not defined so b can be present in string or b may not be present in the string. This can be represented as b * thus,

R = b * ab * ab *

প্রশ্ন ৪. Design a regular expression over strings a and b that contains at most 2 a's.

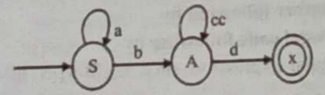
Sol. Here, the string can have either 1 a or 2 a's or string will have no a's. For string containing 2 a's regular expression will be

b* ab* ab*

For string containing 2 a's the regular expression will be b* ab* ab* for string containing no as the regular expression will be.
Thus, the regular expression for language containing at most 2 a's will be
R = b* ab* + b* ab* ab* + b*

প্রশ্ন ৫. Find the regular expression containing the sub-string aa.
Sol. The regular expression will be as
(a + b)* aa(a + b)*

প্রশ্ন ৬. What is the regular expression for the language generated by
S->aS|bA
A->d|ccA
Ans:



We can clearly see that X is final state and the corresponding language regular expression is a*b(cc)*d.

প্রশ্ন ৭. TGTDCL decided to go for a systematic coding of their customers and they recommended that the customer code from now on will consist of:
i. three letter alphabetic district code, followed by,
ii. five-digit customer serial, the first digit cannot be a zero, followed by,
iii. single digit customer type, 0-5 will indicate the six types of the customers, followed by,
iv. optional last character (D,E, or F) indicating previous customer defaults. Write down the necessary regular expression for the above-described customer code.

Ans: RegEx: [a-zA-Z]{3}[1-9]{1}[0-9]{4}[0-5]{1}[DEF]{0,1}

Explanation:
[Character set. Match any character in the set. a—z Range. Matches a character in the range "a" to "z" (char code 97 to 122). Case sensitive. A—Z Range. Matches a character in the range "A" to "Z" (char code 65 to 90). Case sensitive.]
{3} Quantifier. Match 3 of the preceding token.

[Character set. Match any character in the set. 1-9 Range. Matches a character in the range "1" to "9" (char code 49 to 57). Case sensitive.]
{1} Quantifier. Match 1 of the preceding token.

[Character set. Match any character in the set. 0-9 Range. Matches a character in the range "0" to "9" (char code 48 to 57). Case sensitive.]
{4} Quantifier. Match 4 of the preceding token.

[Character set. Match any character in the set. 0-5 Range. Matches a character in the range "0" to "5" (char code 48 to 53). Case sensitive.]
{1} Quantifier. Match 1 of the preceding token.

[Character set. Match any character in the set.
D Character. Matches a "D" character (char code 68). Case sensitive.
E Character. Matches a "E" character (char code 69). Case sensitive.
F Character. Matches a "F" character (char code 70). Case sensitive.]
{0,1} Quantifier. Match between 0 and 1 of the preceding tokens.

Context Free Grammar

প্রশ্ন ১. What is Context Free Grammar?
Ans: Context-free grammar বা সংক্ষেপে CFG হচ্ছে একটি বিধিগত ব্যাকরণ (formal grammar) যেখা প্রতিটি উৎপাদনী সূত্র (production rule) নিম্নোক্ত রূপের হ

A->α

যেখানে, A একটি অসমাপ্তি জ্ঞাপক প্রতীক (non-termina symbol) এবং α হচ্ছে সমাপ্তি জ্ঞাপক (termina symbol) অথবা অসমাপ্তিজ্ঞাপক প্রতীক দিয়ে গঠিত একটি স্ট্রি।

A context free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N,T,P,S) where

N = is a set of Non-terminal symbol
T = is a set of terminals, where N∩T = NULL
P = is a set of rules, P:N-> (NUT); i.e. the left hand side of the production rule P does have any right context or left context.*
S = is the start symbol.

[**Note:**
➔ Grammar এর Left side এ যা থাকবে সেগুলো Non-terminal/Variable.
➔ Grammar এর Right side এ Non-terminal/Variable বাদে যা থাকবে সেগুলো Terminal। যেমনঃ অপারেটর, lower case letter, digit etc.
➔ First Non-terminal কে Start Variable বলে।]

প্রশ্ন ২. Give a context-free grammar (CFG) for each of the following languages over the alphabet Σ ={a,b}:

(a) All strings in the language L:{a^n b^m a^(2n) | n,m≥0}
S→aSaa|B
B→bB| ε

(b) All nonempty strings that start and end with the same symbol.
S→aXa|bXb|a|b
X→aX|bX| ε

(c) All strings with more a's than b's.
S→Aa|MS|SMA
A→Aa| ε
M→ ε |MM|bMa|aMb

(d) All palindromes (a palindrome is a string that reads the same forwards and backwards).
S→aSa|bSb|a|b| ε

প্রশ্ন ৩. Identify the terminal, non-terminals, start variable from the following grammar.
E->E+T|T
T->T*F|F
F->(E)|ID
Ans:
Non-terminal(V) = {E,T,F}
Terminal (T) = {+,*,id, (,)}
Start Variable = E

প্রশ্ন ৪. Give context-free grammars that generate the following languages.
(a){w∈ {0,1}*|w contains at least three 1's}
Answer: G= (V,Σ, R, S)with set of variables V={S, X}, where S is the start variable; set of terminals Σ ={0,1}; and rules
S→X1X1X1X
X→0X|1X|ε

(b){w∈ {0,1}*|w=w^R and |w| is even}
Answer: G= (V,Σ, R, S)with set of variables V={S}, where S is the start variable; set of terminals Σ ={0,1}; and rules
S→0S0|1S1|ε

(c){w∈ {0,1}*|the length of w is odd and the middle symbol is0}
Answer: G= (V,Σ, R, S)with set of variables V={S}, where S is the start variable; set of terminals Σ ={0,1}; and rules
S→0S0|0S1|1S0|1S1|0

প্রশ্ন ৫. Write a Context free grammar for user considers the condition:
i. The string 'PGCB' following by
ii. 5 Digit Number following by
iii. 4 Digit Postal code following by
Answer:
Regular Expression: (PGCB) ([0-9] {5}) [0-9] {4}
S→PS / GX / Y
X → CX / Y
Y→YB / ε

প্রশ্ন ৬. What is Production ?
Production: Production হল একটি Grammar নির্দিষ্ট করার কিছু নিয়ম-নীতি। Terminal এবং Non-terminal সমূহ একত্রিত করে Production গুলো String আকারে লেখা হয়।

Production এর উপকরণ গুলো হল:
i. Non-terminal (Production এর right side এ বসে)
ii. Arrow (→)
ii. Terminal (Production এর left side এ বসে)

Left Recursion (LR): একটি Grammar কে Left Recursion বলা হবে যদি এতে একটি non-terminal 'A' থাকে এবং Derivation যদি A $\xrightarrow{+}$ A ∝ হয়। Top-down parsing method ব্যবহার করে LR grammar কে handle করা যায় না।

নিম্নোক্ত নিয়ম ব্যবহার করে Left Recursion eliminate করা যায়:
i.যদি, A→A∝ |β তবে একে নিম্নোক্ত ভাবে Replace করা যায়, A→βA′
A→∝A′ | ∈

উদাহরণ:

1.	2.	3.
E→E+T | T	T→T*F | F	F→(E) | id
E→TE′	T→FT′	
E′→T E′ | ϵ	T′→*FT′ | ϵ	

উদাহরণ -১:
S→Aa | b
A→Ac | Sd | ϵ

Solution:

S→Aa | b	অথবা,
S→Sda | b	A=Ac | sd | ϵ
S→bS′	A=Ac | Ada | ϵ
S′→daS′ | ϵ	A=ϵA′
	A′=CA′ | adA′ | ϵ

Eliminate left-recursion from the following grammar:

A→ A + B | B
B→ int | (A)

Solution: A→BA′
A′→ +BA′ | ε
B→ int | (A)

Left Factoring (LF): Left Factoring হল একটি grammar এর রূপান্তর (transformation), যা predication অথবা Top-down Parsing এর জন্য উপযুক্ত grammar তৈরিতে ব্যবহৃত হয়।

Left Factoring এর নিয়মাবলী:
যদি, $A \to \alpha\beta_1 \mid \beta_2$ হয়,
তবে $A' \to \alpha A'$
$A' \to \beta_1 \mid \beta_2$ লেখা যায়।

উদাহরণ-১: S→iE+S | iE+SeS | a এবং E→b হলে, Left factoring ব্যবহার করে এর Grammar নির্ণয়।

Solution:

S→iE+SS′ | a	অথবা,
S′→ϵ | eS	S→ib+S | ib+SeS | a
E→b	S→ib+SS′ | a
S→ib+SS′ | a	S→ϵ | eS
S′→ϵ | eS	

উদাহরণ-২: S→bSSaaS | bSSaS | b | bSb | a হলে, Left factoring ব্যবহার করে grammar নির্ণয় কর।

Solution: Left factoring ব্যবহার করে Grammar টি নিম্নরূপ:
S→bSS′ | a
S′→SaaS | SaSb | b
S′→SaS" | b
S"→aS | sb

উদাহরণ-৩: Left Factoring ব্যবহার করে A→aAB | a ও B→bB | b এর জন্য একটি grammar নির্ণয় কর।

Solution: Grammar using (LF):
A→aAA′
A'→B | ϵ
B→bB′
B'→B | ϵ

উদাহরণ -৪: Left factoring ব্যবহার করে নিম্নের expression গুলোর Grammar নির্ণয় কর।

i. A→aAB | aA | a
ii. E→T+E | T

i. Solution: Grammar using LF:

A→aA′	অথবা, A→aAA′ | a
A′→AB | A | ϵ	A′→B | ϵ
A′→AA"	
A"→B | ϵ	

ii. Solution: Grammar using LF:
E→TE′
E'→+E | ϵ

উদাহরণ -৫ Left factor the following grammar:
E→ int | int + E | int – E | E – (E)

Solution: E→ int E' | E – (E)
E′→ ε | +E | –E

LR Parser: LR Parser হলো এক ধরনের bottom-up pasring, এর context যা language এর grammar বিশ্লেষণ করে থাকে। এখানে L হল input string এর left-to-Right scaning, R হল Right Most Derivation in reverse.

LR Parser কে নিম্নোক্ত শ্রেণীতে ভাগ করা যায় যথা:
i. LR (o)
ii. SLR (1)
iii. LALR
iv. CLR(1)

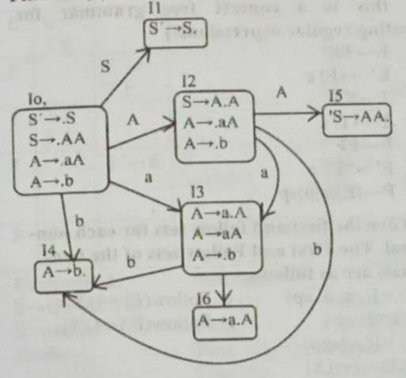
LR (o) Parser:
"L"→ Left-to-Right Scanning
"R"→Right Most Derivation
"O"→number of inputs symbol of look ahead.

LR(o) parsing table তৈরির ধাপ:
i. Augmented grammar তৈরি করা।
ii. Dot (.) Forward এর দিকে ১ ঘর move করে Item সংখ্যা নির্ণয় করা।
iii. Parsing table এর ২ ফাংশন চিহ্ন নির্ণয় করন।
a) go to (terminal এর list তৈরি করতে হবে)
b) action (non-terminal এর list তৈরি করতে হবে)

[**Notes:** Augmented grammar: একটি Grammar কে augmented Grammar বলা হবে যদি Grammar 'a' তে একটি অতিরিক্ত 'a' Production যুক্ত করে প্রতিটি **production** এর ডান পাশের শুরুতে **dot** (.) যুক্ত করা হয়। যেমন a→AA হলে 'a' এর Augmented Grammar হবে a →.a, a→.AA]

Example: construct a LR Parsing table from the given context free grammar (উক্ত CFG থেকে একটি LR parsing table তৈরি/গঠন কর:
S→AA
A→aA | b
Solution: Augmented grammar:
S′→.S
S→.AA
A→.aA
A→.b

Find LR(o)collection of items:



Terminal of this grammar are→{a,b}
Non-terminal of this grammar are→{A,S}

[**Note:**
i. যদি item নতুন তৈরি করার সময় dot(.) এরপরে Non-terminal পাওয়া যায়, তবে উক্ত Non-terminal এর সবগুলো production ঐ item এ যুক্ত করতে হয়।
ii. Production এর Terminal বা Non-terminal visite/scam হয়ে গেলে dot(.) একঘর সামনে move করতে হয়।]

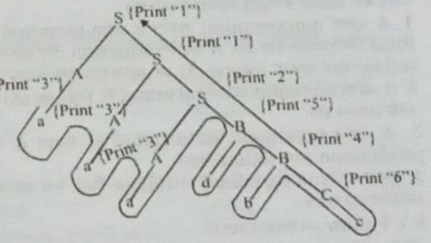
LR Parser table:
i. It a state is going to some other state on a terminal then hits correspondence to a shift move.
ii. It a state is going to some offer state on a variable or no a terminal then it's corresponds to go to move.
iii.If a state contain the final state in the particular raw then write the reduce mode completely.

	Action			goto	
	a	b	$	A	S
0	s3	s4		2	1
1			Accept		
2	s3	s4		5	
3	s3	s4		6	
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

প্রশ্ন ১. A shift-reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar
S —> AS {print "1"}
S —> AB {print "2"}
A —> a {print "3"}
B —> bC { print "4"}
B —> dB {print "5"}
C —> c {print "6"}

This syntax directed translation scheme translates a language whose terminal symbols are a, b, c and d into another language whose terminal symbols are 1, 2, 3, 4, 5 and 6.

প্রশ্ন ২. What is the translation of "aadbc"?
Ans: The syntax directed tree for the given grammar can be represented as for given input "aaadbc"



Output printed will be: 333645211

প্রশ্ন ৩. Consider the grammar
S->ABSc|Abc
BA->AB
Bb->bb
Ab->ab
Aa->aa

Which of the following sentences can be derived by this grammar?

Ans: The grammar has the sentencial forms as

S-> Abc-> abc

↓

ab

Hence, Only "abc" can be derived.

First and follow

First: First () হল Terminal সমূহের set, যা ০ε থেকে string সমূহ Derived করা হয়।

Computing First: Grammar যে সকল প্রতীক X এর জন্য First (X) নির্ণয় করার জন্য নিম্নের নিয়ম অনুসরণ করতে হয়। যেখানে, First set এ আর কোন terminal বা ϵ যুক্ত করার জন্য অবশিষ্ট থাকবে না।

1. X যদি terminal হয়, তবে First (X)= {X}
2. X যদি non-terminal k >=1 এর জন্য একটি Production $X \rightarrow y_1, y_2 \ldots . y_n$ হয়, তবে যতক্ষণ পর্যন্ত Production এর শুরুতে terminal পাওয়া যাবে না ততক্ষণ পর্যন্ত production এর শুরুতে যে non-terminal পাওয়া যাবে তার production যাচাই করতে হবে। Terminal পাওয়া গেলে তা First (X) এর set এ যুক্ত হবে।
3. ϵ যদি X এর Production হয়, তবে তা First (X) এ যুক্ত হবে।

Example:	**Solution:**
E→TE′	First (E) = First (T)=First
E′→+TE′ | ϵ	(F)={C,id}
T→FT′	First (E′) = {+, ϵ}
T′→* FT′ | ϵ	First (T′) ={*, ϵ}
F→(E) | id	

Follow: ধরি, A একটি non-terminal । A এর Follow নির্ণয় করার জন্য নিম্নোক্ত ধাপগুলো অনুসরণ করতে হবে-

1. A এরপর non-terminal থাকলে ঐ non-terminal এর Frist নির্ণয় করতে হবে এবং A যার Production তার যদি কোন follow থাকে তবে তা follow (A) এর সেটের সাথে যুক্ত করতে হবে।
2. A এরপর terminal থাকলে terminal টি follow (A) এর একটি উপাদান হবে।
3. A এরপর terminal/non-terminal না থাকলে A যার production তাকে follow করবে।
4. শুরুতে যে Non-terminal এর follow নির্ণয় করা হয় তার একটি উপাদান হবে $।
5. 'ϵ' Follow এর উপাদান হবে না।

Example:	**Solution:**
E→TE′	Follow (E)={$,)}
E′→ + TE′ | ϵ	Follow (E′)= Follow (E)={$,
T→FT'	)}
T′→ * FT′ | ϵ	Follow (T)={+,$,)}
F→(E) | id	Follow (T′)={+$,)}
	Follow (=)={*,+,$,)}

প্রশ্ন ১. The First and Follow sets for the grammar:
S-> SS+/SS*/a
Ans: First(S) = {a}
Follow (S) = {+,*,$}

প্রশ্ন ২. Consider the grammar
E->TE'
E->+TE'|Ɛ
T->FT'
T'->*FT'| Ɛ
F->(E)|id.
What is the Follow(F)?
Ans: {+,*,),$}

প্রশ্ন ৩. Consider the following LL(1) grammar, which has the set of terminals T={a, b, ep,+,*,(,)}. This grammar generates regular expressions over {a, b}, with + meaning the RegExp OR operator, and ep meaning the ε symbol. **(Yes, this is a context free grammar for generating regular expressions!)**

E→TE′
E′→+E| ε
T→FT′
T′→T| ε
F→PF′
F′→*F′| ε
P→(E)|a|b|ep

প্রশ্ন ৪. Give the first and follow sets for each non-terminal. The First and Follow sets of the non-terminals are as follows.

First(E) ={(, a, b, ep} Follow(E) ={),$}
First(E′) ={+, ε } Follow(E′) ={),$}
First(T) ={(,a,b,ep}
Follow(T) ={+,),$}

First(T′) ={(,a,b,ep, ε }
Follow(T′) ={+,),$}

First(F) ={(,a,b,ep}
Follow(F) ={(,a,b,ep,+,),$}

First(F′) ={*, ε }
Follow(F′) ={(,a,b,ep,+,),$}

First(P) ={(,a,b,ep}
Follow(P) ={(,a,b,ep,+,),*,$}

LL(1) Grammar:

* 1st L means Left to Right (scanning input)
* 2nd L means Left Most Derivation
* And 1 stands for using one input for look ahead.

LL(1) Grammar নির্ণয়:
1.যে সকল Production দেওয়া থাকবে তাদের LR বা LF হলে তা বের করতে হবে।
2. First এবং Follow এবং Follow নির্ণয় করতে হবে।
3. First এ ϵ থাকলে Follow ϵ দ্বারা replace হবে।

Example 1:	**Solution:**	
E→TE′	First (E)={(, id}	Follow(E)={$,)}
E′→+TE′ | ϵ	First (E′)={+,ϵ}	Follow (E′)={$,)}
T′→*FT′ | ϵ	First (T)={ (, id}	Follow (T)={+,$,)}
F→ (E) | id	First (T′)={*,ϵ}	Follow (T′)={+,$)}
	First (F)={ (, id}	Follow (F)={*,+$,)}

	id	+	*	(	)	$
E	E→ T E′			E→T E′		
E′		E′→+ T E′			E′→ ϵ	E′→ ϵ
T	T→ F T′			T′		
T′		T′→ϵ	T′ →*FT		T′→ ϵ	T′→ ϵ
F	F→i d			F→(E)		

Example -2:
S→iE+SS′ | a
S′→eS | ϵ
E→b

Solution:
First (s) ={i,a}
First ={e, ϵ}
First (E) ={b}
Follow (S)={$,e}
Follow (S′)={$,e}
Follow (F)={+}

	i	+	a	e	b	$
S	S→iE+S S′		S→a			
S′				S′→eS S′→ ϵ		S′→ ϵ
E					E→b	

Directed Acyclic Graph (DAG): A DAG is a directed graph that contains on cycles. A DAG is constructed from three address statement.

Example-1: Contact DAG for the given expression- (a+b)*(a+b+c).

Solution: Three address code for expression is-
$t_1 = a+b$
$t_2 = t_1+c$
$t_3 = t_1*t_2$

the DAG is:

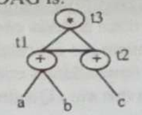
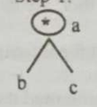


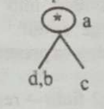
Fig: DAG of expression (a+b) * (a+b+c)

Example-2: Construct DAG-
a=b*c
d=b
e=d*c
b=e
f=b+c
g=f+d

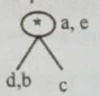
Step-1:



Step-2:

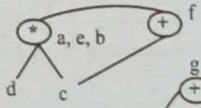


Step-3:

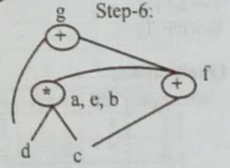


Step-4:

Step-5:



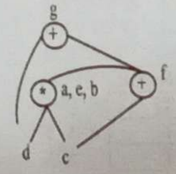
Step-6:



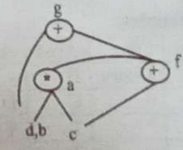
Example-3: Optimized the block following the expression-

	The optimized basic block
a=b*c	a=b*c
a=b	d=b
e=d*c	f=a+c
b=e	g=f+d
f=b+c	
g=f+d	

Solution: The DAa is



Now, DAG is-



Three Address Code:

* A three address code has at three address location calculate the expression.
* A three address code can be represented in three from this are –Quadruples

These are: Quadruples, Triples, Indirect triples

Quadruples: In quadruples representation each instructions is divided into four fields such as operator, Arg1, Arg2, result.

*Operator field → represent the internal code for operation.
*Arg1 and Arg2 field→ represented the two operant of an expression.
*Result fields→ store the result of an expression

Example: a+b*c/e↑f+b*a

Solution:

T_1=e↑f
T_2=b*c
T_3=T_2/T_1
T_4=b*a
T_5= a+T_3
T_6=T5+T_4

Quadruples:

Loc	OP	Arg1	Arg2	Result
1	↑	e	f	T_1
2	*	b	c	T_2
3	/	T_2	T_1	T_3
4	*	b	a	T_4
5	+	a	T_3	T_5
6	+	T_5	T_4	T_6

Example: A=-B*(c+d)

Solution:

T_1= - B
T_2= c+D
T_3=T_1*T_2
A=T_3

Quadruples:

Loc	OP	Arg1	Arg2	Result
1	Unary	B	-	T_1
2	+	C	D	T_2
3	*	T_1	T_2	T_3
4	=	T_3	-	A

বিশেষ শর্ত (Exception):

1. X= ++Y হলে, operation field a unary operation, ১ম এ বসবে। y ২য় argument field এ থাকবে এবং x result field এ store হবে।
2. Unconditional conditional এবং jump statement এর ক্ষেত্রে target labels সমূহ result field এ store হবে।

Example: if A<B than 1 else 0.

Solution:

1. If A<B go to (4)
2. T= 0
3. go to (5)
4. T= 1

Quadruples:

Loc	Operation	Arg1	Arg2	Result
1	<	A	B	(4)
2	=	0		T
3	Go to			(5)
4	=	1		T
5				

Triples: In triples representation the use of temporary variable is avoided and instead reference to instruction are made.

Example 1: a+b*c/e↑f+b*a

Triples:

Loc	Operation	Arg1	Arg2
0	↑	c	f
1	*	b	c
2	/	(1)	(0)
3	*	b	a
4	+	a	(2)
5	+	(4)	(3)

Indirect triples: This is representation age an enhancement over triples representation

x=(a+b)^c/d

	Loc	Operation	Arg1	Arg2
(0)	(1)	+	a	b
(1)	(2)	^	c	
(2)	(3)	*	(1)	(2)
(3)	(4)	/	(3)	d
(4)	(5)	=	x	(4)

Statement	
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)

Operator Grammar: A grammar is side to be an operator grammar if no production right side age is ϵ or has two adjacent non-terminals.

Example 1: E→E-E | E+E | E*E | E | E | E↑ | -E | (E) | id

অপারেটর grammar এবং operator precedence টেবিল n.Z operator এর precedencey নির্ণয়:

Grammar: E→E+E | EE*E | id

Table:

	id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	

কোন operator এর presidence কেমন হবে তা উপরের টেবিল থেকে নির্ণয় করা সম্ভব। এখানে * operator এর presidence সর্বোচ্চ (+<*>+)। পুল সম্পন্ন না হওয়া পর্যন্ত যোগ করা সম্ভব নয়।

Example 2: নিম্নোক্ত grammar ও operator precedence table ব্যবহার করে id+id*id spring টির operation দেখাও।

E→E+E | E•E | id

Solution:

	Id	+	*	$
id		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	

$<id>+< id>*< id >$
$<+< id>*< id >$
$<+<*< id >$
$<+<* >$
$<+ >$
$$$

E→ id $ id + id*id $
E →id $E+id * id$
E →id$ E+ E* id$
E →id$ E+ E* E$
E →E *E $E+ E$
E →E +E E

Syntax Directed Translation (SDT): A national trim work for intermediate code generation that is an extension CFG. This framework is called SDT. It allows semantic actions to be attached to the production of a CFG.

Translation: Associated with grammar symbol is called transaction of the symbol.

Example: x is grammar symbol.
So, x. Value
x. Turn.

Semantic action or Errors:

1. Type of mismatch.
2. Undeclared variable.
3. Reserved identifier misluses.
4. Multiple declaration of variable of in a scope.
5. Accessing an out of scope variable.

Expansion and Reduction for SDT.

Expansion: Grammar এর নিম্ন অনুপাতে non-terminal গুলোকে expand করতে হয়।

(a) Construct an LL(1) parsing table for the left-factored grammar.

Ans: Here is an LL(1) parsing table for the grammar.

	(	)	a	b	ep	+	*	$
E	TE'		TE'	TE'	TE'			
E'		ϵ				$+E$		ϵ
T	FT'		FT'	FT'	FT'			
T'	T	ϵ	T	T	T	ϵ		ϵ
F	PF'		PF'	PF'	PF'			
F'	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	$*F'$	ϵ
P	(E)		a	b	ep			

(b) Show the operation of an LL(1) parser on the input string

ab*.

Stack	Input	Action
$E$$	$ab*$	TE'
$TE'$$	$ab*$	FT'
$FT'E'$$	$ab*$	PF'
$PF'T'E'$$	$ab*$	a
$aF'T'E'$$	$ab*$	*terminal*
$F'T'E'$$	$b*$	ϵ
$T'E'$$	$b*$	T
$TE'$$	$b*$	FT'
$FT'E'$$	$b*$	PF'
$PF'T'E'$$	$b*$	b
$bF'T'E'$$	$b*$	*terminal*
$F'T'E'$$	$*$	$*F'$
$*F'T'E'$$	$*$	*terminal*
$F'T'E'$$	$	ϵ
$T'E'$$	$	ϵ
$E'$$	$	ϵ
$	$	*ACCEPT*