2018

2(a) Sentence of a grammar:-

String: $-(id+id)$.

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid id$

$E \rightarrow -E$
$\rightarrow -(E)$
$\rightarrow -(E+E)$
$\rightarrow -(id+E)$
$\rightarrow -(id+id)$.

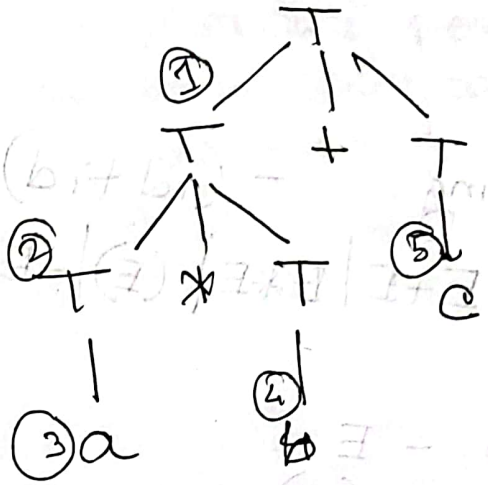
(b) Parse tree: A diagrammatic representation of the parsed structure of a sentence or string. OR. ~~A hierarchical representation of terminals or non-terminals as symbols. These symbols are~~ A hierarchical representation of symbols. The symbol can be terminal or non-terminal. In parsing, the string is derived using the start symbol. The root of the parse tree is the start symbol. Parse tree follows the precedence of operators. The deepest sub-tree traversed first. So, the operator in the parent node has less precedence over the operator on the sub-tree.

- All leaf nodes have to be terminal
- All interior nodes have to be non-terminal
- In-order traversal gives original input string.

Example: $T = T+T \mid T*T$

$T = a \mid b \mid c$

Input a*b+c



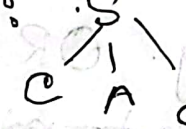
Given,

input string:- cad

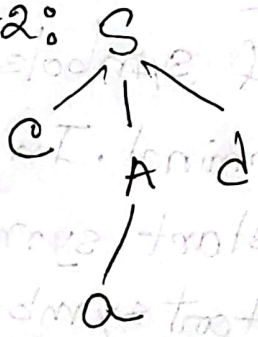
grammar → $S \to cAd$

$A \to ab \mid a$

Step-1:



Step-2:



2 © Remove left-recursion

$A \to A\alpha \mid \beta$

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

$S \to Aa \mid b$

$A \to Ac \mid Sd \mid \varepsilon$

$A \to Ac \mid Aad \mid bd \mid \varepsilon$

$A \to A' \mid bdA'$

$A' \to cA' \mid adA' \mid \varepsilon$ } Ans

$A \to A' \mid SdA'$

$A' \to cA' \mid \varepsilon$

$A \to A' \mid AadA' \mid bdA'$

$A' \to cA' \mid \varepsilon$

$A \to A'A'' \mid bdA'A''$

2018

4@ **NFA:-** Non deterministic finite automata. Many paths for specific input from current state to next state allowed. Every NFA is not DFA, but each NFA can be translated into DFA. NFA can contain ε transition.

Formal definition of ε-NFA:- [Null NFA / NFA Lambda]

5-tuple $(Q, \Sigma, \delta, q_0, f)$.

Q is Finite set of all states $(q_0, q_1, \ldots q_n)$ n is finite

Σ is " " " " symbols called alphabet

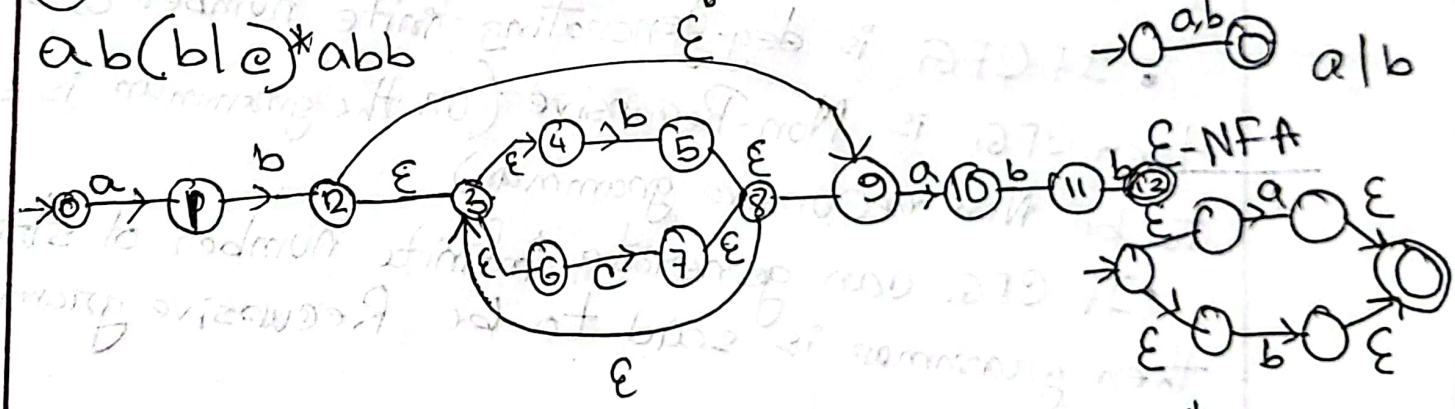
$\delta: Q \times (\Sigma \cup \epsilon) \to 2^Q$ is a total function called transition function

q_0 is initial state from where any input is processed $(q_0 \in Q)$

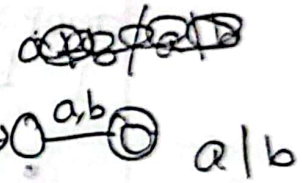
F is the final state/states where f will be subset of Q.

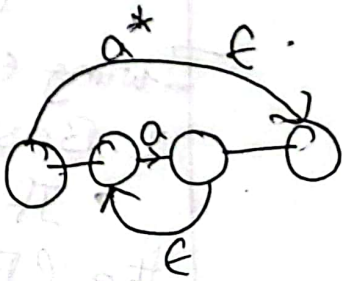
t (b) Thomson's Construction:-

ab(b|a)*abb



basic operation





2018

5 a) Context Free grammar (CFG) consisting of a finite set of grammar rules. It is a formal grammar which is used to generate all possible patterns of a string in a given formal language.

$G = (V, T, P, S)$

G = grammar which consists of a set of production rule

T = Final set of a terminal symbol, lowercase

V = Final set of a non-terminal symbol, capital letter

P = Production rules.

S = Start symbol which is used to derive the string.

CFG can be classified on the basis of following two properties:

1. Based on number of strings it generates.

- If CFG is ~~deg~~ generating finite number of string then CFG is Non-Recursive (or the grammar is said to be Non-recursive grammar).
- If CFG can generate infinite number of strings then grammar is said to be Recursive grammar

~~During compilation~~

2. Based on no. of derivation trees:-

- If there is only 1 derivation tree then the CFG is unambiguous.
- If there are more than 1 derivation tree, then the CFG is ambiguous.

Key problems of top down parsing -

To identify which ~~Parsing is to~~ production of a non-terminal must be chosen. Such that the terminal in the production body match the terminal in input string.

Advantages-

1. Very simple
2. Very easy to identify the action decision of the top-down parser.

Dis: 1. Unable to handle left recursion in the present in the grammar

2. Some recursive decent parser may require backtracking

$A \to \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3 | \alpha\beta_4$

$S \to aSSbS | aSaSb | abb | b$

Non-deterministic Grammar $\xrightarrow[\text{factoring}]{\text{Left}}$ Deterministic grammar

$S \to aS' | b$

$S' \to SSbS | SaSb | bb$

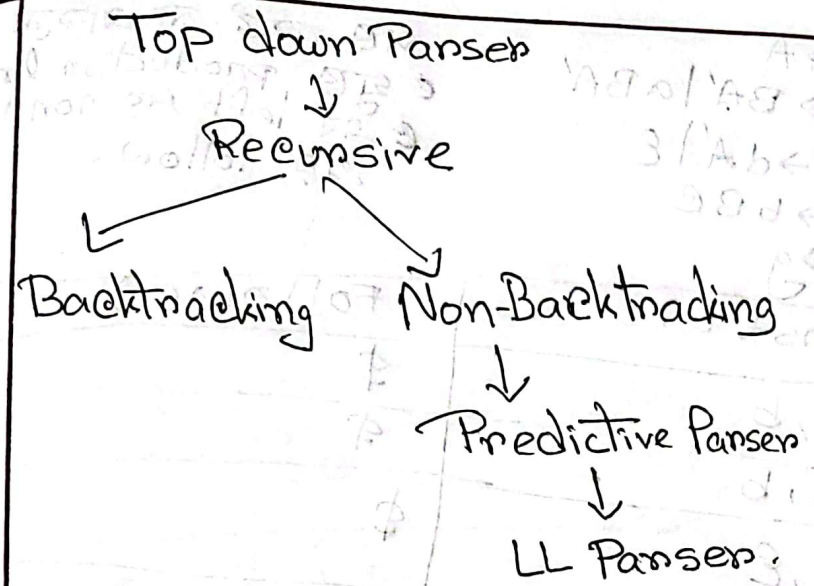
or

$S \to aS' | b$

$S' \to SS'' | bb$

$S'' \to SbS | aSb$

2018 6a



The goal of predictive parser is to construct a top-down parser that never backtracks. to do so, we transform grammar in 2 ways.

1. eliminate left recursion and 2. perform left factoring

LL(1) grammar follows top-down parsing method. For a class of grammar called LL(1). the parser which can read such grammar is predictive parser.

পরে non-terminal হলে তাহার first

2018 6b

$S \to A$
$A \to aB \mid Ad \mid B$
$B \to bBC$
$C \to g$

$S \to A$
$A \to BA' \mid aBA'$
$A' \to dA' \mid \varepsilon$
$B \to bBC$
$C \to g$

পরে কোন না থাকলে
- ঘুরে production line
- এর left এর non-terminal এর follow.

~~First(A) = {}~~	First()	Follow()
S	a, b	$
~~A~~ A	a, b	$
~~A'~~ A'	d, ε	$
~~B~~ B	b	d, $
~~C~~ C	g	d, $

2018 6b

Grammar	a	d	b	g	$
S	$S \to A$		$S \to A$		
A	$A \to aBA'$		$A \to BA'$		
A'		$A' \to dA'$			$A' \to \varepsilon$
B			$B \to bBC$		
C				$C \to g$	

2014

2014 4c

4© $S \to iEtSS' \mid a$

$S' \to eS \mid \varepsilon$

$E \to b.$

Symbol	First( )	Follow( )
S	{i, a}	{e, $}
S'	{e, ε}	{e, $}
E	{b}	{t}

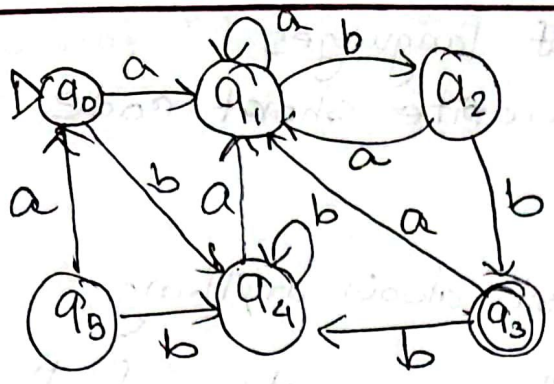
Grammar	i	t	a	e	b	$
S	S→iEtSS'		S→a			
S'				S'→eS S'→ε		S'→ε
E					E→b	

∴ There is 2 production line for same symbol.

∴ The grammar is ambiguous

17

Minimize DFA. 2017 3c



	a	b
→ q_0	q_1	q_4
q_1	q_1	q_2
q_2	q_1	q_3
(q_3)	q_1	q_4
q_4	q_1	q_4
× q_5	q_0	q_4

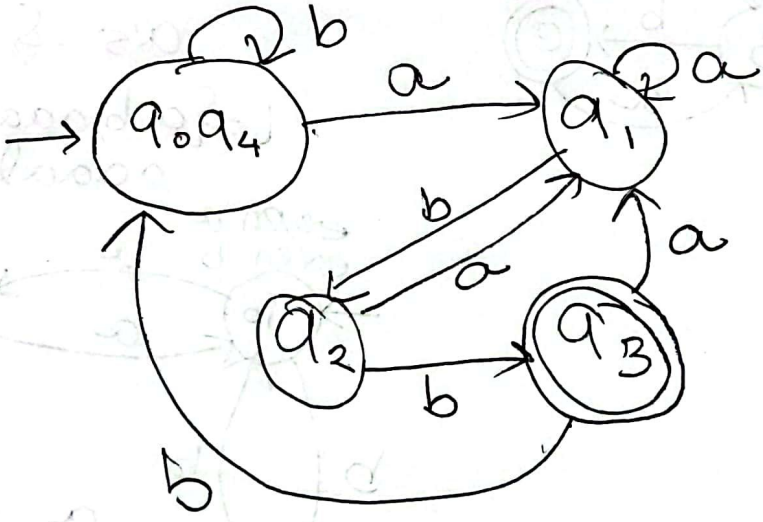
Step-1: Remove unreachable state: q_5
(from starting + direct & indirect)

Step-2: Classes → accepting & (final), non-accepting.

$\pi_0 = \{q_0\ q_1\ q_2\ q_4\}\ \{q_3\}$

$\pi_1 = \{q_0\ q_1\ q_4\}\ \{q_2\}\ \{q_3\}$

$\pi_2 = \{q_0\ q_4\}\ \{q_1\}\ \{q_2\}\ \{q_3\}$

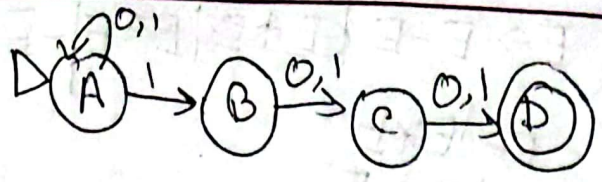


2017 4a

17

DFA from NFA

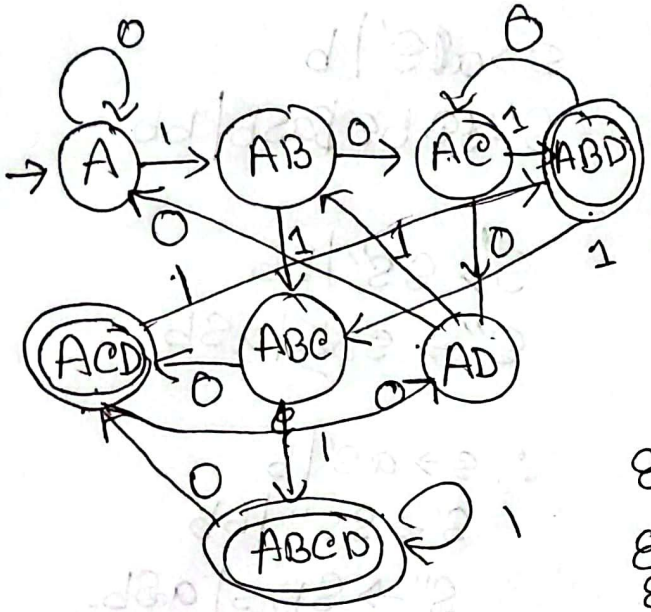
4@



	0	1
→ A	{A}	{A, B}
B	{C}	{C}
C	{D}	{D}
*D	{ }	{ }

tt for DFA

	0	1
→ [A]	[A]	[AB]
• [AB]	[AC]	[ABC]
[AC]	[AD]*	[ABD]*
[ABC]	[ACD]*	[ABCD]*
[AD]*	[A]	[AB]
[ABD]*	[AC]	[ABC]
[ACD]*	[AD]*	[ABD]*
[ABCD]*	[ACD]*	[ABCD]*



(b) Define ε-NFA:

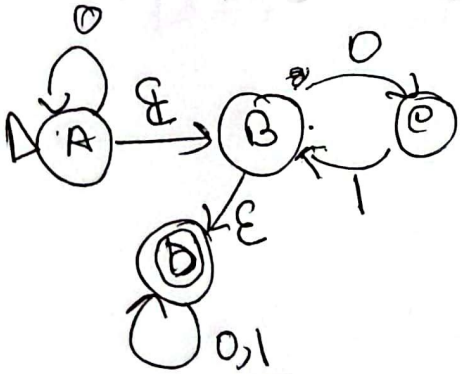
ε-closure({A}) = {A, B, D}

ε-closure({A, C, D}) = {A, B, C, D}

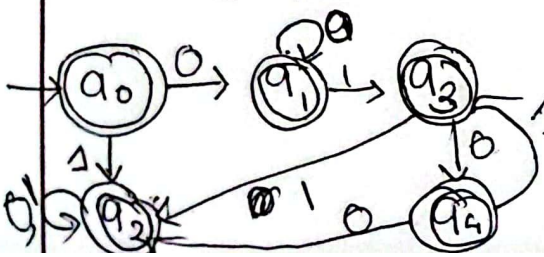
ε-closure({D}) = {D}

ε-closure({B, D}) = {B, D}

ε-closure({C}) = {C, D}



		0	1
{A,B,D}	q_0	{A,C,D}$_{q_1}$	{D}$_{q_2}$
{A,B,C,D}	q_1	{A,C,D}$_{q_1}$	{B,D}$_{q_3}$
{D}	q_2	{D}$_{q_2}$	{D}$_{q_2}$
{B,D}	q_3	{C,D}$_{q_4}$	{D}$_{q_2}$
{C,D}	q_4	{D}$_{q_2}$	{B,D}$_{q_3}$



5@

$A \to A @ B \mid B$

$B \to B \# C \mid C$

$C \to C @ D \mid D$

$D \to d$

$A \to BA'$

$A' \to @BA' \mid \varepsilon$

$B \to CB'$

$B' \to \#CB' \mid \varepsilon$

$C \to DC'$

$C' \to @DC' \mid \varepsilon$

$D \to d$

2018 5a

$E \to E - E \mid E * E \mid E \wedge E \mid - E$

$E \to -EE'$

$E' \to -EE' \mid *EE' \mid \wedge EE' \mid \varepsilon$

6. $S \to aSSbS \mid aSaSb \mid abb \mid b$

$S \to aS' \mid b$

$S' \to SbS \mid aSb \mid bb$

$S'' \to SS' \mid bb$

$S'' \to SbS \mid aSb$

$\therefore S \to aS' \mid b$

$S' \to SS'' \mid bb$

$S'' \to SbS \mid aSb.$

2014 2014 2d

2 (d) If r & s are regular expression denoting the languages L(r) and L(s), then

Union: (r)|(s) is a regular expression denoting L(r) ∪ L(s)

Concatenation: (r)(s) is a regular expression denoting L(r)L(s).

Kleene closure: (r)* is regular expression denotin (L(r))* where then language can hold 0 or many words.

(r) is regular expression denoting L(r).

2014

Role of parser

1. It varifies the structure generated by the token based on the grammar.
2. It constructs the parse tree
3. It reports the errors
4. It performs error recovery.

Func of e

Left factoring algorithm:- Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.

Input: Grammar G Output: An equivalent left factoring grammar.

Step 1: For each non terminal A find the longest prefix α common to two or more of its alternatives.

2. If $\alpha != E$ i.e there is a non-trivial common prefix, replace all the A production

$A \to \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3 | \dots | \alpha\beta_n | \gamma$ where γ represents all alternatives that do not debegin by α.

$A \to \alpha A' | \gamma$

$A' \to \beta_1 | \beta_2 | \dots | \beta_n$

Here A is new non terminal. Repeatedly apply this transformation until no two alternatives for a non-terminal has a common prefix.