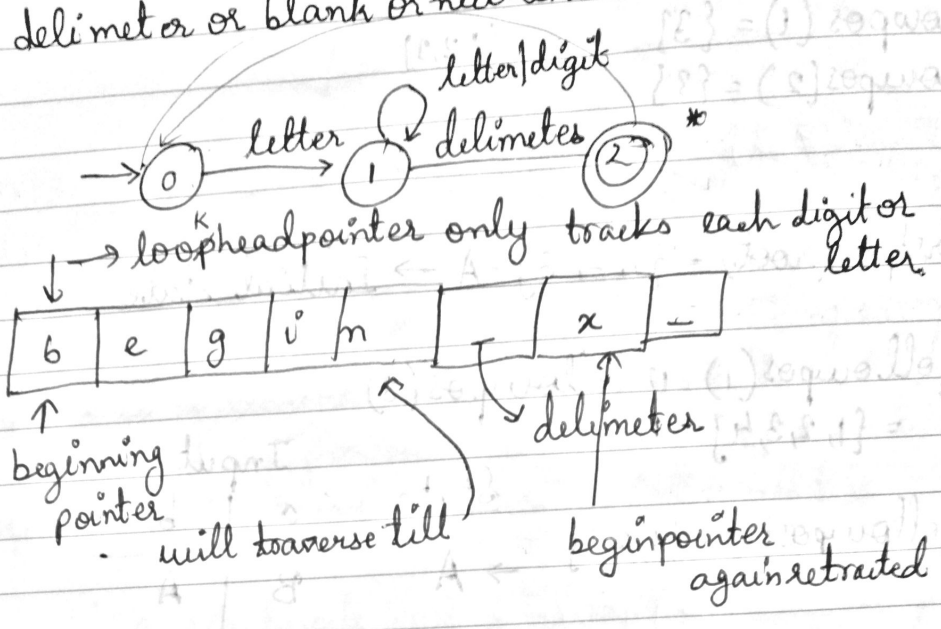
Design of a lexical analyser

1) Token → identifer



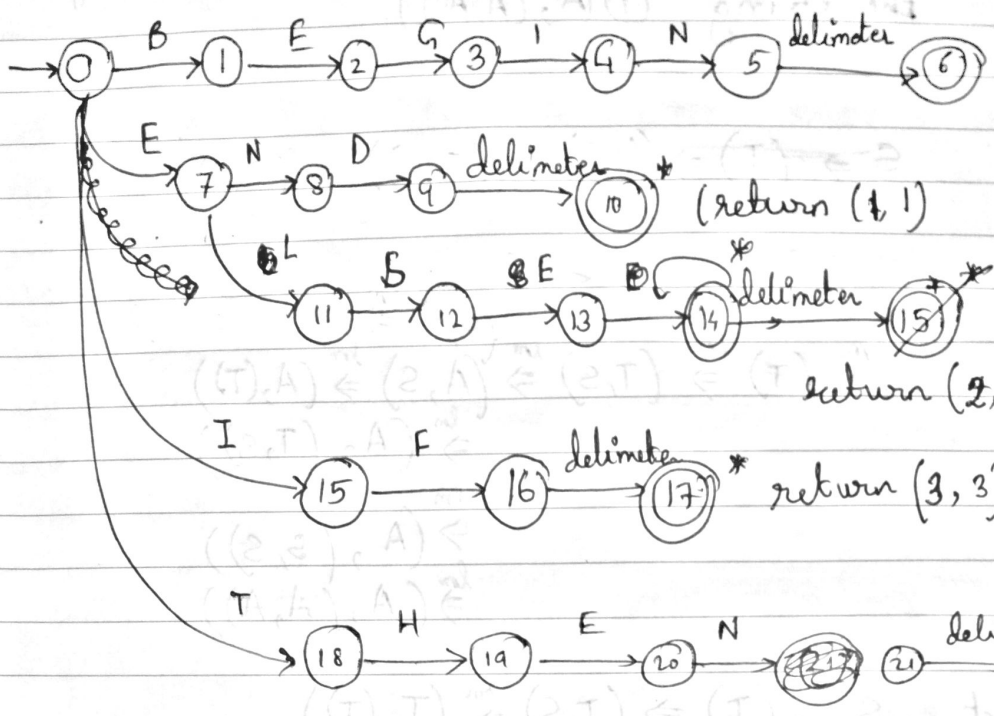
State 0: c = GETCHAR()
if LETTER(c) then
goto State 1
else
FAIL()

State 1: c = GETCHAR()
if LETTER(c) or DIGIT(c) then
goto state 1
else if(DELIMETER (c))
goto State 2
else
FAIL()

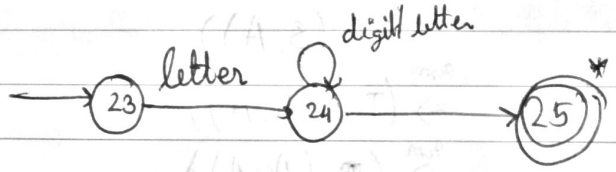
State 2: RETRACT()
return (id. installed)

Token → Keyword:

begin, if, end, else, then



If THENA (if it is not keyword then it is identifier)

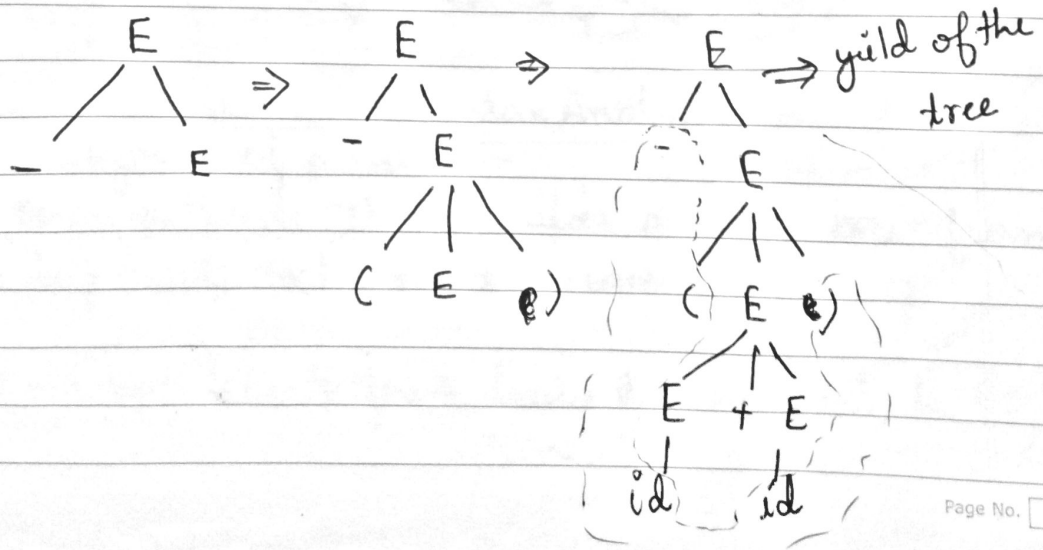


$(a+b+c)^*$

aabbcc

$E \overset{lm}{\Rightarrow} -E \overset{lm}{\Rightarrow} -(E) \overset{lm}{\Rightarrow} -(E+E)$
$\overset{lm}{\Rightarrow} -(id+E) \overset{lm}{\Rightarrow} -(id+id)$

Parse Tree



Date ___/___/_____

8/09/2022

Consider the grammar

$S \to A \mid \wedge \mid (T)$

$T \to T, S \mid S$

Find the leftmost & rightmost derivation of the string (i) (A, (A, A))

~~$S \Rightarrow (T) \Rightarrow ((T)) \Rightarrow ((T, S)) \Rightarrow ((S, S))$~~

~~$\Rightarrow ((A, S)) \Rightarrow ((A, A))$~~

leftmost $S \overset{lm}{\Rightarrow} (T) \overset{lm}{\Rightarrow} (T,S) \overset{lm}{\Rightarrow} (S,S) \overset{lm}{\Rightarrow} (A,S) \overset{lm}{\Rightarrow} (A,(T))$

$\overset{lm}{\Rightarrow} (A,(T,S))$

$\overset{lm}{\Rightarrow} (A,(S,S))$

$\overset{lm}{\Rightarrow} (A,(A,S))$

$\overset{lm}{\Rightarrow} (A,(A,A))$

rightmost $S \overset{rm}{\Rightarrow} (T) \overset{rm}{\Rightarrow} (T,S) \overset{rm}{\Rightarrow} (T,(T))$

$\overset{rm}{\Rightarrow} (T,(T,S))$

$\overset{rm}{\Rightarrow} (T,(T,A))$

$\overset{rm}{\Rightarrow} (T,(S,A))$

$\overset{rm}{\Rightarrow} (T,(A,A))$

$\overset{rm}{\Rightarrow} (S,(A,A))$

$\overset{rm}{\Rightarrow} (A,(A,A))$

(ii) $(((A,A),\wedge,(A)),A)$

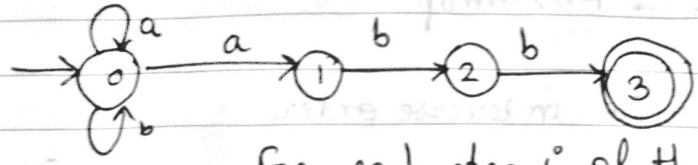
Date ___/___/_____

saathi

RE → RG

re: (a|b)*abb

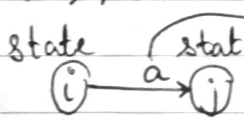
NFA:



For each step 'i' of the NFA create a non terminal

Step 1: $i \rightarrow A_i$

Step 2:



Add the production $A_i \rightarrow aA_j$

Step 3: if i is the final state, then

$A_i \longrightarrow \epsilon$

Step 4: if i is the start state, then

A_i is the start symbol of grammar.

<id,1>

$$P \Rightarrow \left\{ \begin{array}{l} A_0 \text{ is the start symbol} \\ A_0 \rightarrow a\,A_0 \mid a\,b\,A_0 \\ A_0 \longrightarrow a A_1 \\ A_1 \longrightarrow b A_2 \\ A_2 \longrightarrow b A_3 \\ A_3 \longrightarrow \epsilon \end{array} \right\}$$

$$RG = \{ \underbrace{\{A_0, A_1 A_2, A_3\}}_{V}, \underbrace{\{a,b\}}_{T}, \underbrace{\{A_0\}}_{S}, \underbrace{P}_{P} \}$$

Parser → Syntax Analyser

There 2 types of parser: →

Top down method: It builds a parse tree from the top (root) to the bottom (leaves)

Bottom up: starts from leaves and builds it to top.

Date ___/___/_____

left to right scan at a time one symbol.

LL parser → Top down (→ leftmost derivation)

LR parser → Bottom up

↘ rightmost derivation in reverse order.

LL parser

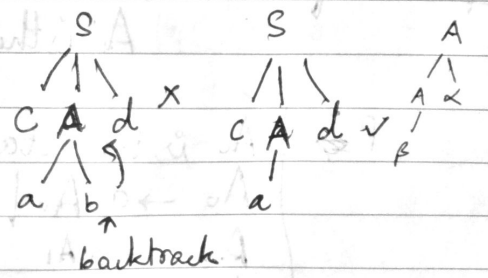
w is a string

Top down - Parser

$S \rightarrow CAd$

$A \rightarrow ab \mid b \mid a$

All productions are checked using a parser tree.



Left factoring

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

Left recursive

$A \rightarrow A\alpha \mid \beta$

↳ we always eliminate left recursion

Left recursive

↳ $A \rightarrow A\alpha \mid \beta \quad \Rightarrow \quad A \Rightarrow A\alpha \Rightarrow \beta\alpha$

removing left recursion

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \epsilon$

$A \Rightarrow \beta A' \Rightarrow \beta\alpha A' \Rightarrow \beta\alpha$

Production Rules:

$E \rightarrow E+T \mid T \qquad F \rightarrow (E) \mid id$

$T \rightarrow T * F \mid F$

A α β

Date

$E \to TE'$	$E \to TE'$
$E' \to E+T \mid \epsilon$	$E' \to +TE' \mid \epsilon$
$T \to FT'$	$T \to FT'$
$T' \to T*F \mid \epsilon$	$T' \to *FT' \mid \epsilon$
$F \to (E) \mid id$	$F \to (E) \mid id$

~~$E \Rightarrow TE' \Rightarrow T+TE' \Rightarrow T+TE' \Rightarrow (E)$~~

Q) $A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \beta_3 \ldots \mid \beta_n$

Sol: $A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$
$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \alpha_m A' \mid \epsilon$

Q) $S \to Aa \mid b$
$A \to Ac \mid Sd \mid e$

$S \to Aa \mid b$
$A \to Ac \mid Aadbd \mid \epsilon$

$S \to Aa \mid b$
$A \to bdA' \mid eA'$
$A' \to cA' \mid adA' \mid \epsilon$

Q) $S \to Aa \mid b$
$A \to bdA' \mid eA'$
$A' \to cA' \mid adA' \mid \epsilon$

$A \to \alpha\beta_1 \mid \alpha\beta_2$

equivalent left factoring
$A \to \alpha A'$
$A' \to \beta_1 \mid \beta_2$

$S \to ictS \mid ictSeS \mid a$
$C \to b$

$S \to ictSS' \mid a$
$S' \to eS \mid \epsilon$
$C \to b$

Page No.

Date __/__/____ 22/08/22

Saa

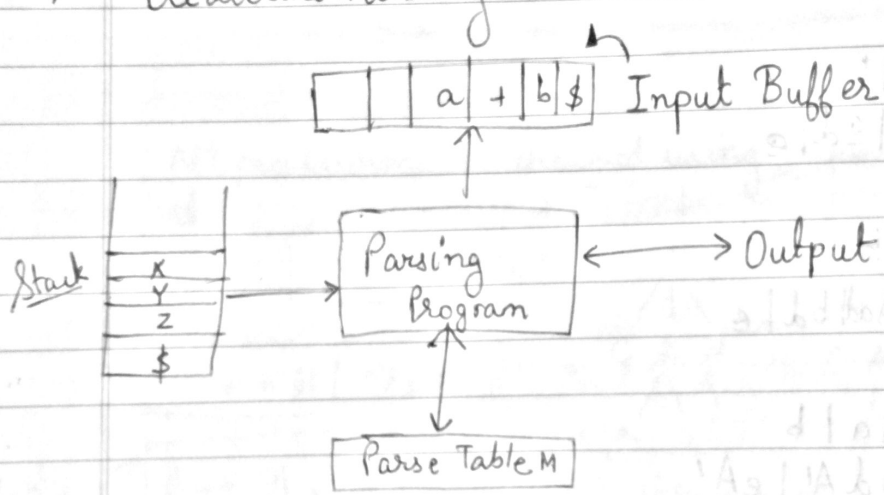
* Recursive Descent Parsing

Example:

statement - If condition then
statement
else
statement

Statement - while condition do statement
| begin statement - list end.

* Predictive Parsing → Non Recursive



What is parsing Table?

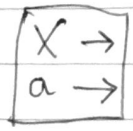
Sol:

Non Terminal	Input Symbol id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'					
T'		T'→ϵ	T'→*FT'		T'→ϵ	T'→ϵ
F	F→id			F→(E)		

What is parsing program?

Sol: A Parser is controlled by a program that behave as follows

The program considered X, the symbol on top of stack and a the current input symbol.



M[A,a]

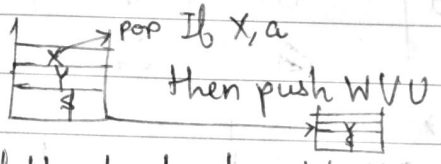
There are 3 possibilities

1) If X = a = $
the parser halts, and announces a succesful completion of parsing.

2) If X = a ≠ $
the parser pops X from the stack and advances the input pointer to the next input symbol.

3) If X is a nonterminal, the program consult entry M[X, a] of the parsing table. This entry will be either an X production of the grammar or an error entry.

M[X, a] = { X → UVW }



The parser replaces X on the top of the stack by WVU

With U on the top as output the parser prints the production used.

If M[X, a] = error blank entry and error recovery routine.

Consider the grammar:

$E \to TE'$

$E' \to +TE' \mid \epsilon$

$T \to FT'$

$T' \to *FT' \mid \epsilon$

$F \to (E) \mid id$

and w is id + id * id

Stack (Initial: $S)	Input (Initially: w$)	Output
$S	id+id*id$	
$E	id+id*id$	
$E'T	id+id*id$	$E \to TE'$
$E'T'F	id+id*id$	$T \to FT'$
$E'T'id	id+id*id$	$F \to id$
$E'T'	+id*id$	
$E'	+id*id$	$T' \to \epsilon$
$E'T+	+id*id$	$E' \to +TE'$
$E'T	*id*id$	
$E'T'F	id*id$	$T \to FT'$
$E'T'id	id*id$	$F \to id$
$E'T'	*id$	
$E'T'F*	*id$	$T' \to *FT'$
$E'T'F	id$	
$E'T'id	id$	$F \to id$
$E'T'	$	
$E'	$	$T' \to \epsilon$
$	$	$E' \to \epsilon$

Predictive Parsing Table

FIRST FOLLOW

If α is any string of grammar symbol then First of α FIRST(α) the set of terminal that begins the string derived from α.

To compute FIRST(X)* (terminal or nonterminal) for all grammar symbol X, apply the following until no more terminal or ϵ can be added to any FIRST set.

1. If X is a terminal, then FIRST(X) = {X}
2. If X is a ϵ ie $X \to \epsilon$, then add ϵ to FIRST(X).
   FIRST(X) = {ϵ}
3. If X is a non-terminal and $X \to Y_1 Y_2 Y_3 \ldots Y_k$, then

FIRST(X) = FIRST($Y_1 Y_2 \ldots Y_k$) = FIRST(Y_1)

If Y_1 is ϵ

FIRST(X) = {ϵ}

FIRST(X) = FIRST($Y_2 Y_3 \ldots Y_k$) = FIRST(Y_2)

We construct FIRST(E) = FIRST(T) = FIRST(F) = FIRST(C)
= {c, id}

FIRST(E') = FIRST(+) = {+} ∪ {ϵ}
FIRST(E') = FIRST(ϵ) = {+, ϵ}

FIRST(T') = FIRST(*) = {*} ∪ {ϵ}
= {*, ϵ}

S → iEtSeS' | a FOLLOW(S) = FIRST(S)
E → b
S' → c E

FIRST(S) = {i, a}
FIRST(E) = {b}
FIRST(S') = {b}

FOLLOW(A)

Define FOLLOW(A) for non-terminal A to be the set of terminals a that can appear immediately to the right of A if some sentential form

To conclude FOLLOW(A) for all non-terminal A, apply the following rules until nothing can be added to the follow set.

1. Place $ in FOLLOW(S) where S is the start symbol and $ input right endmarker.

   FOLLOW(S) = {$}

2. If there is a production $\alpha B \beta$ then everything in FIRST(β) except for ϵ is placed in FOLLOW(B)

3. If there is a production $A \to \alpha B$ or $A \to \alpha B \beta$ where FIRST(β) contains ϵ (ie $\beta \Rightarrow \epsilon$), then everything in follow(A) is in FOLLOW(B)

FOLLOW(A)

Fo(E) = {$,)}

FOLLOW(E') = FOLLOW(E) = {), $}

FOLLOW(T) = FIRST(E') = {+, $,)}

FOLLOW(F) = FIRST(T') = {), *, +, $}

FOLLOW(T') = FOLLOW(T) = {+, $,)}

Fo(E) = {), $}

Parsing table contains a Nonterminal | Input Symbol

$S \to iEtSS' \mid a$

$S' \to eS \mid \epsilon$

$E \to b$

First(S) = {i, a}

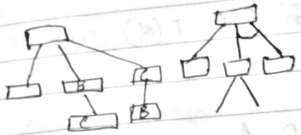
First(S') = {e, ε}

First(E) = {b}

Fo(S) = {e, $}

Fo(S') = {$, e}

Fo(E) = {t}



26/08/2022

LL Parsing Table

Input Symbol

Nonterminal	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'				
T	T→FT'			T→FT'		
T'			T'→*FT'			
F	F→id			F→(E)		
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

check this one

① FIRST(E) ⇒ FIRST(T) ⇒ FIRST(F) = {(, id}

② FIRST(E') ⇒ {+, ε}

③ FIRST(T') ⇒

Date __/__/____

Saathi

Construction of Predictive Parsing Table

Input: Grammer G

Output: Parsing Table M

Step1: For each ~~modifi~~ production $M \to \alpha$ of the grammar Do steps 2 and 3

Step2: For each terminal a in FIRST(α) and at $A \to \alpha$ to M[A, a]

Step3: If ϵ is in FIRST(α) ADD $A \to \alpha$ to M[A, b] for each terminal b in fo(A). If ϵ is in FIRST(α) and \$ is in FOLLOW(A), Add $A \to \epsilon$ to M[A, \$].

Step4: Make each undefined entry of M be error

Construct a parsing table for the grammar: →

$S \to iEtSS' \mid a$

$S' \to eS \mid \epsilon$

$E \to b$

FIRST(S) → {i, a} FOLLOW(S) → {e, \$}

FIRST(E) → {b} FOLLOW(S') → {\$, e}

FIRST(S') → {e} FOLLOW(E) = {t}

LL(1)

Nonterminal	i	t	e	a	b	\$
S	$S \to iEtSS'$			$S \to a$		
S'			$S' \to eS$, $S' \to \epsilon$			$S' \to \epsilon$
E					$E \to b$	

LL(1) – Parsing table entry is single

Page No.

Date __/__/____

Bottom up Parsing LR:

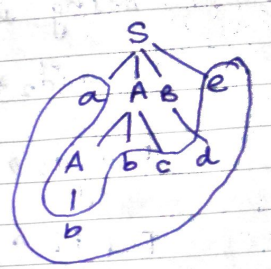
L → left to right scan
R → rightmost derivation in Reverse order.

Given a grammar $S \to aABe$ $A \to Abc \mid b$
$B \to d$

$w \to abbcde$

First check right-hand side of the production

$S \to aABe$ $w \Rightarrow abbcde$
$\Rightarrow aAbcde$
$\Rightarrow aAde$
$\Rightarrow aABe$
$\Rightarrow S$



Rightmost derivation →
$S \overset{rm}{\Rightarrow} aABe \overset{rm}{\Rightarrow} aAde \overset{rm}{\Rightarrow} aAbcde \overset{rm}{\Rightarrow} abbcde$

* **Handle:**
A Handle of a string is a substring that matches the right side of a production and who's reduction to the nonterminal on the left side of the production that represents one step along the reverse of a rightmost derivation

$S \to aABe$
$A \to Abc \mid b$
$B \to d$

abbcde
aAbcde — $A \to b$ — this b is a handle.
aAbcde — $A \to Abc$ → this is the handle
aAde
$B \to d$ → this substring is the ha