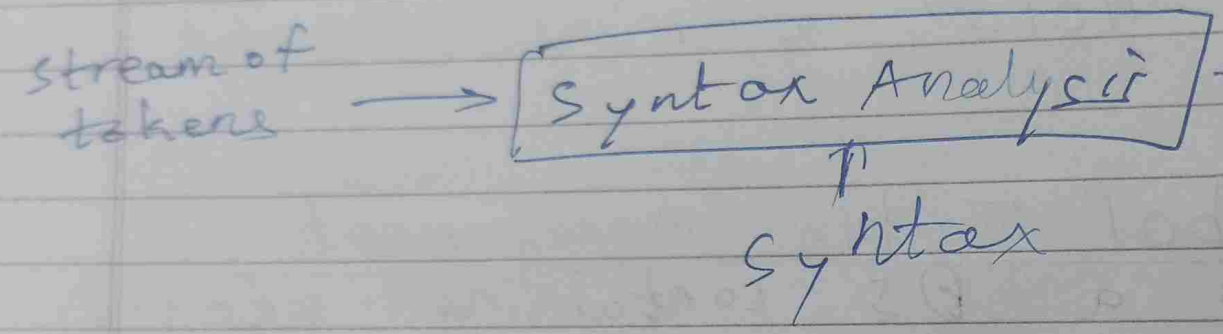
Parser :-



Parser

Topdown

Backpatching

self

Predictive Recursive descendant parsing

ii) LL(1)

bottom-up

i) shift & reduce

ii) SLR

iii) LR(1)

iv) LALR

To

R/

A

to

stru

as

seq

op

Gra

G=

Term

alphabet

Generally lowercase

Parser
↓
Grammar

Set of formal rules which checks correctness of sentence or we can gen syn correct sentence using these rules
A Grammar G is used basically for 2 purposes
1. To gen valid string for given lang. (Derivation op.)
2. To recognise a valid string for the given lang is called R/Reduction op.

A parse tree is used to depict the syntactic structure of a valid string as it emerges during the seq. of derivation or deduction op.

Grammar is quadruple of
$G = \{T, NT, P, S\}$

- T → Terminal → alphabet 0–9 Generally lowercase
- NT → Non Terminal
- P → Prod rules
- S → start symbol

$\Sigma = \{a, b, c, \ldots z, 0, \ldots 9\}$

A string is the finite seq. of symbols rep by roman symbols $\alpha, \beta, \gamma, \ldots$

$\alpha \Rightarrow a\,o\,b$

Non terminal is the name of syntax category of lang

e.g Noun, verb, adj.

Capital letters or <>

Prod rules :- Rules in Grammar The manner in which terminal and & non-terminal can be combined to form a string

NT symbol $\Rightarrow$ string of terminals & NTs

<article> = a | an | the

Every grammar must have start symbol

Chomsky Grammars are classified on the nature of production.

Type (0) - Unstructured Grammar

$\alpha \rightarrow \beta$

No restriction

Type (1) - Context sensitive

$\alpha \rightarrow \beta$

except 'ε'

start symbol does not appear to the right side of the grammar

Type (2) - context free grammar

$A \rightarrow \pi_{string}$

LHS contains only one non terminal

start symbol can appear at RHS.

Type (3) - Regular or linear

LHS only 1 NT
RHS only 1 NT

$A \rightarrow tB \mid t$ → right linear

$A \rightarrow Bt \mid t$ → left linear

Operator Grammar (?)

operation on Grammar

derivative — Reduction

let

$P_1 : A \rightarrow \alpha$

& let β be the string such that

$\beta \rightarrow \gamma A \theta$

Then replacement of A by the string α according to p_1 is called derivation op

1 step
direct
derivation
>1 steps
indirect
left Right

Let
$P_1 : A \to \alpha$
and let δ be string such that $\delta \to \gamma \alpha \Theta$ then replacement of α by NT A in δ is according to P_1 is called reduction.

Top Down (LL(1))

First & Follow set
First(α) be the set of the terminals that begins the string derived from α
First(α) $\Rightarrow \alpha a$ }. If $\alpha \Rightarrow \varepsilon$ the ε is also in the First(α)

Rules for First Set

1. If X is a terminal then First(X) $\Rightarrow$ {X}
2. If $X \Rightarrow \varepsilon$ then add ε in First(X)

3. If X is non terminal and $X \to Y_1 Y_2 \ldots$ then place a in First(X) if for some i, a is in the First(Y_i) and ε is in all of First(Y_1) upto First(Y_{i-1})

1. $E \to TE'$
2. $E' \to +TE' / \varepsilon$
3. $T \to FT'$
4. $T' \to *FT' / \varepsilon$
5. $F \to (E) / id$

First(E) = {

First(E) = First(T) = First(F)

= { (, id }

First(E') = { +, ε }

First(T') = { *, ε }

Follow set:

For non terminal a, is the set of terminals 'a' that can appear to the right of a in some sentential form, ie, 'a' s.t there exist

a derivation of the form
$S \overset{*}{\Rightarrow} \alpha A a \beta$ for some α, β

Rules:-

1. Place $ in the follow(S)
follow(S) = {$}
end marker

2. If there exists a production in the form
$A \rightarrow \alpha B \beta$
Then everything in the First(β) except ε is placed in the follow(B)

3. If $\exists$ a production αB, or $A \rightarrow \alpha B \beta$, where in the First($\beta$) contains ε then everything in the follow(A) is in the follow(B)

Follow(E) = {$,)}

Follow(E') = {$,)}

Follow(T) = Follow(T')

= {+, $}

ε → follow
else → First

Follow(F) = {+, *,), $}

Table M:-

Non Term	id	+	*	(	)	$
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)		

Bottom Up Parser

Shift & Reduce

LR

- SLR
- LR(1)
- LALR

Bottom-up Parser

Reduction

~~top down~~ eg :-

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

a b b c d e
a A b c d e rule 2
a A d e rule 2
a A B e rule 3
S rule 1

1. $E \rightarrow E + E$
2. $E \rightarrow E * E$
3. $E \rightarrow (E)$
4. $E \rightarrow id$

id + id * id.

E + E * E 4
E * E 1
E 2

shift, reduce, accept, error

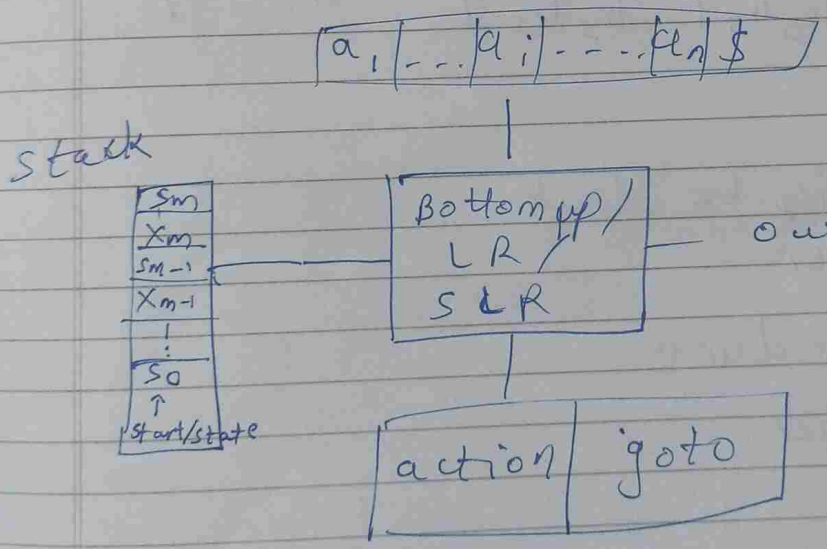
Using shift & Reduce

stack	I/p	Action
$	id+id*id$	shift
$id		Reduce 3
$E	+id*id$	shift
$E+	id*id$	shift
$E+id	*id$	Reduce 3
$E+E	*id$	shift
$E+E*	id$	shift
$E+E*id	$	Reduce 3
$E+E*E	$	Reduce 2
$E+E	$	Reduce 1
$E	$	

Diff imp

Topdown
|
derivation
|
Table M
|
First, Follow

Bottom up
|
Reduction
|
Table M
|
First, Follow,
LR(0) items/
closure ops
SLR items
goto operation



1 $E \rightarrow E + T$
2 $E' \rightarrow T$
3 $T \rightarrow T * F$
4 $T' \rightarrow F$
5 $F \rightarrow (E)$ 6 $F' \rightarrow id.$

Table M:

state	id	+	*	(	)	$	(Non-Terminal) goto F	T	F
	(Terminal) Action								
0	s_5			s_4			1	2	3
1		s_6				accept			
2		r_2	s_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4				9	3
7	s_5			s_4					10
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

s_5 = shift i/p to stack
↑
state number

r_2 = reduce
↑
rule number

Stack	I/P	Action
0	id * id + id $	.. Shift
0 id5	* id + id $	Reduce 6
0 F3	* id + id $	Reduce 4
0 T2	* id + id $	shift
0 T2 * 7	id + id $	shift
0 T2 * 7 id5	+ id $	Reduce 6
0 T2 * 7 F10	+ id $	Reduce 3
0 T2	+ id $	Reduce 2
0 E1	+ id $	shift
0 E1 + 6	id $	shift
0 E1 + 6 id5	$	Reduce 6
0 E1 + 6 F3	$	Reduce 4
0 E1 + 6 T9	$	Reduce 1
0 E 1	$	Accept

Find states

1 E → E + T
2 E → T
3 T → T * F
4 T → F
5 F → (E)
6 F → id

Augmented Grammar

Rule 0 $E' \to E$ start symbol in only 1 rule

I_0

$E' \to \circ E$
$E \to \circ E + T$
$E \to \circ T$
$T \to \circ T * F$
$T \to \circ F$
$F \to \circ (E)$
$F \to \circ id$

Dot at start

Shift dot by 1 char each time

End when dot at right in all rules

1 shift = 1 state

I_1
$E' \to E \circ$
$E \to E \circ + T$

I_2
$E \to T \circ$
$T \to T \circ * F$

I_3
$T \to F \circ$

I_4
$F \to (\circ E)$
$E \to \circ E + T$
$E \to \circ T$
$T \to \circ T * F$
$T \to \circ F$
$F \to \circ (E)$
$F \to \circ id$

If dot thru terminal and next is non terminal take rule with that NT in I_2, and all NT in I_4, $F \to \circ (E) \to I_4$
$F \to \circ id \to I_5$

I_5

$F \rightarrow id \cdot$

I_6

$E \rightarrow E + \cdot T$
$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot id$

I_7

$T \rightarrow T * \cdot F$
~~$E \rightarrow \cdot E + T$~~
~~$E \rightarrow \cdot T$~~
~~$T \rightarrow \cdot T * F$~~
~~$T \rightarrow \cdot F$~~
~~$F \rightarrow \cdot (E)$~~
$F \rightarrow \cdot id$

I_8

$F \rightarrow (E \cdot)$

Parsing Algo

i/p → grammar G
o/p → Table M

Method:

1. $C = \{I_0 \ldots I_n\}$ Collection of Canonical set for G
2. State i is constructed from I_i

The parsing action for state i are determined as follows

a] If $[A \to \alpha \cdot a \beta]$ is in I_i (a: Terminal) & goto $[I_i, a] = I_j$ then set action $[i, a]$ to "shift j"

b] If $[A \to \alpha \cdot]$ is in I_i then set action "reduce $A \to \alpha$" for all a in follow of A

A ≠ start

c] If $[S' \to S \cdot]$ is in I_i then action $[i, \$]$ to "accept"

3. The goto transition for state i is constructed only for NT

If goto $[I_i, A] = I_j$ then goto $(i, A) = j$

4. If 2 & 3 are not defined any entry then error

GOTO

M table

$\therefore$ 0E → 1

$I_0 \longrightarrow I_1$

E' → .E E' → E.

$I_0 \longrightarrow I_2$ 0T → 2

E → .T E → T.

SHIFT

$I_1 \longrightarrow I_6$ 1+ → S6

E → E.+T E → E+.T

→ dot thru NT A from I_i to I_j

$I_0 \longrightarrow I_4$ 0(→ S4

F → .(E) F → (.E)

Follow set

E' → { $ }

E → { $, +,) }

T → { $, +,), * }

F → { $, +,), * }