Parsing

The parsing is the process of determining whether a given input string can be derived from respective context free grammar.
In other words, the parser is a program that accepts strings of token as input and produces the parse tree from respective context free grammar.

- Top down parsing (recursive descent parser)
- Bottom up parsing (Shift Reduce parser)

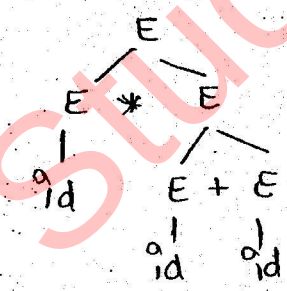
eg. $E \to E*E \mid E+E \mid E-E \mid id$

$w = id*id+id$

$E \to E*E$
$\to id*E$
$\to id*E+E$
$\to id*id+E$
$\to id*id+id$

$w = id*id+id$
$= E*id+id$
$= E*E+id$
$= E+id$
$= E+E$
$= E.$

parse tree,



Note :- There are 2 parsing approaches :-
1. Top down parsing (TDP)
2. Bottom up parsing (BUP)

* In **Top down parsing** we attempt to construct a parse tree start from root node and leads towards leaf nodes.

The top down parser can be viewed as left most derivation.

A top down parser is also known as **recursive descent parser**.

* **Bottom up parser** :- In which we attempt to construct a parse tree starting from leaf and & work towards root node.

The bottom up parser can be viewed as Right most derivation in reverse order.

Bottom up parser is also called as **Shift Reduce parser**

Parsing Approaches

- Top down parser
  - Brute force (Recursive descent parser)
  - Predictive parser
- Bottom-up parser
  - operator parsing
  - LR-parsir
    - SLR
    - CLR
    - LAL

Top down parser

* Brute force Method (trial and error method).

Alternate :- Recursive descent parser

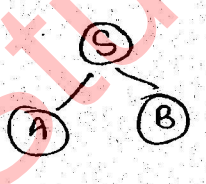
(OR) Back tracking Approach.

In this approach, we start with start symbol first production, then we expand left most variable by its first production. If generated terminal doesn't match with first symbol, then replace with second production. When all combinations fails, then backtrack to the start symbol. In this method, the parser consist of set of procedures, hence it is called recursive descent parser or backtracking approach.

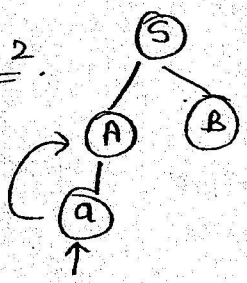
eg. $S \rightarrow AB \mid CA$
$A \rightarrow a \mid b$
$B \rightarrow c \mid d$
$C \rightarrow e \mid f$

Let $w = bd$.

Step1

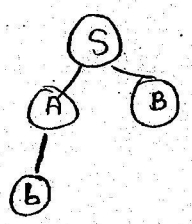


Step2.



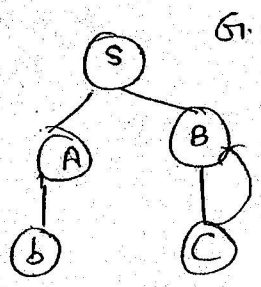
$G.S : a \neq b$.

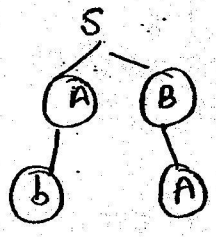
Step3



$G.S = b$.

$G.S = c$; $c \neq d$.





G : b↑d↑. ∴, Success

Q. Consider the following grammar :-

$S \rightarrow AB$

$A \rightarrow Aa \mid b$ → Left Recursive grammar.

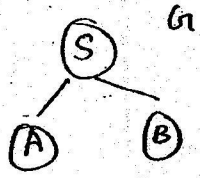
$B \rightarrow d$.

Traverse w = bd using Brute force Method.

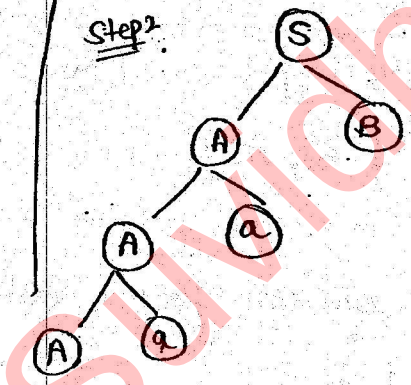
⇒ w = b d

GS :- —

Step1



Step2.



runs infinite.

Limitation of this method :- Not working for Left recursive grammar.

* Left Recursion Grammar :-

A grammar G is said to be left recursive if it is of the form, $A \rightarrow A\alpha \mid \beta$

Due to the left recursive grammar the top down parser enters into ∞ loop.

We can eliminate left recursion from the given grammar with equivalent production :-

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

Solution for left recursion

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

Right Recursion is not a problem at all.

General form of Left Recursion grammar is :-

$$A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \beta_3 \dots \mid \beta_s$$

We can construct an equivalent grammar without left recursion as :-

$$A \to \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \dots \mid \beta_s A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \dots \mid \alpha_n A' \mid \varepsilon.$$

Q. Eliminate left Recursion from following grammar :-

(1). $E \to \overset{A}{E} + \overset{\alpha}{n} \mid \overset{\beta}{a}$

$\Rightarrow$ $E \to aE'$
$E' \to +nE' \mid \varepsilon$

(2). $E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid a$

$\Rightarrow$ $E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid a.$

Q. Gate2008

$S \rightarrow Sa \mid Sb \mid a \mid b$

$\Rightarrow$ ~~$S \rightarrow Sa \mid a \mid b$~~.

$S \rightarrow Sb \mid a \mid b$.

~~$\varepsilon \rightarrow as'$~~

~~$s' \rightarrow \varepsilon$~~

$\Rightarrow S \rightarrow Sa \mid Sb \mid a \mid b$.

$\Rightarrow S \rightarrow as' \mid bs'$

$S' \rightarrow as' \mid bs' \mid \varepsilon$.

Q. Eliminate left Recursion :-

$E \rightarrow E + T \mid E * M \mid id$

$T \rightarrow T * A \mid T - B \mid a$

$A \rightarrow b$

$B \rightarrow d$.

$M \rightarrow bd$

$\Rightarrow$

$E \rightarrow id\,E' \mid \varepsilon$

$E' \rightarrow +TE' \mid *ME' \mid \varepsilon$

$T \rightarrow aT'$

$T' \rightarrow *AT' \mid -BT' \mid \varepsilon$.

$A \rightarrow b$

$B \rightarrow d$

$M \rightarrow bd$.

* Left Factor grammar :-

Left factor means factoring out common prefixes in the given grammar. A grammar G said to be containing common prefix if it is of the form $A \to \alpha\beta_1 \mid \alpha\beta_2$

due to this type of grammar, the parser may get confuse while selecting the production containing the common prefix.

This can be eliminated by using following production :-

This is known as Left factoring.

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

Solution

To avoid multiple comparisons we try to ~~avoid~~ remove common prefixes.

Q. Left factored the following grammars.

Left prefix.

① $S \to abA \mid abB \mid ab$

$A \to a$

$B \to b$.

This all is done to make derivation shorter.

$\Rightarrow S \to aBA' \mid \epsilon$.

$A' \to A \mid B \mid \epsilon$

$A \to a$

$B \to b$.

unit productions, remove them

$S \to abA'$

$A' \to a \mid b \mid \epsilon$ → null production is there, remove it.

$A \to a$

$B \to b$.

~~$S \rightarrow abA' \mid ab$~~

$S \rightarrow$

$S \rightarrow abA' \mid ab$

$S' \rightarrow a \mid b$

$A \rightarrow a$

$B \rightarrow b.$

I star

(2). $S \rightarrow iCtS$

$S \rightarrow iCtSeS$

$C \rightarrow b.$

$\Rightarrow$. $S \rightarrow iCtSS'$

$S' \rightarrow eS \mid \varepsilon.$

$C \rightarrow b.$

$\Rightarrow$ Removing ε production,

$S \rightarrow iCtSS' \mid iCtS$

$S' \rightarrow eS$

$C \rightarrow b.$

(3). $S \rightarrow abcdA \mid abcBA' \mid abcA'$

$A \rightarrow a \mid d$

$A' \rightarrow b.$

$B \rightarrow b.$

$\Rightarrow$. $S \rightarrow abcS'$

$S' \rightarrow dA \mid BA' \mid \varepsilon A'$

$A \rightarrow a \mid d$

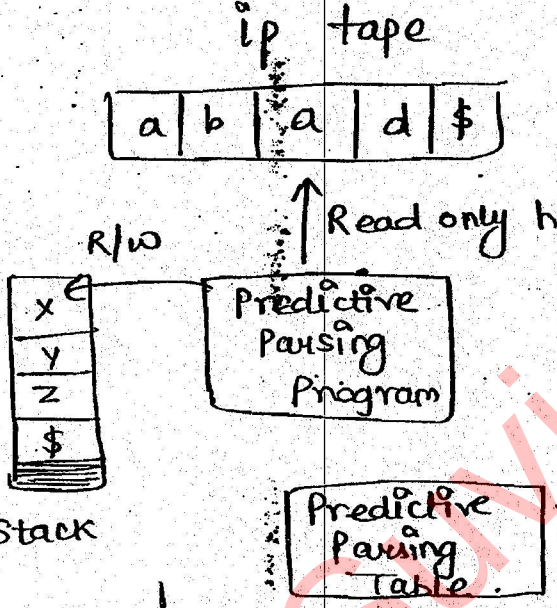
$A' \rightarrow b.$

$B \rightarrow b.$

★ PREDICTIVE PARSING techniques.

A predictive parsing technique is also called as **Non - backtracking parsing method**.

The model of predictive parser as shown in the figure.



The predictive parsing technique requires a CFG in the table format.

The construction of the predictive parsing table procedure depends on 2 concepts -:

1. first set rules
2. follow set rules.

A predictive parsing algorithm or program ~~Consider~~ to consult the predictive parsing table during parsing process.

* FIRST Set Rules :

1. FIRST (X) = X if X is a terminal.
2. If X is non terminal, $X \to a\alpha$ is X-production, then
   FIRST ~~First~~ (X) = {a}
3. If X is non-terminal and $X \to \varepsilon$ is X-production, then
   FIRST (X) = {ε}
4. If X is non terminal and $X \to Y_1 Y_2 Y_3$ is X production then
   FIRST (X) = FIRST ($Y_1 Y_2 Y_3$).
   Where Y_1 is Non Terminal.

To compute FIRST set for the string $Y_1 Y_2 Y_3$

FIRST ($Y_1 Y_2 Y_3$) = FIRST (Y_1) if FIRST (Y_1) don't contain ε;

= FIRST (Y_1) − ε ∪ FIRST (Y_2), if FIRST (Y_2) don't contain ε.

= FIRST (Y_1) − ε ∪ FIRST (Y_2) − ε ∪ FIRST (Y_3), if FIRST (Y_3) don't Contain ε.

= FIRST (Y_1) − ε ∪ FIRST (Y_2) − ε ∪ FIRST (Y_3) − ε ∪ {ε} (if F(Y_3) also contains ε).

Q. find first.

(a) $S \to AB$
$A \to a \mid \varepsilon$
$B \to b.$

Sol. FIRST(S) = a, b
FIRST(A) = a, ε
FIRST(B) = b.

X	FIRST(X)	Follow(X)
S	{a, b}	$
A	{a, ε}	b
B	{b}	a $

(b) $S \to AB$
$A \to a \mid \varepsilon$
$B \to bB \mid \varepsilon$.

X	FIRST(X)	follow(X)
S	{a, b, ε}	{$}
A	{a, ε}	{b, $}
B	{b, ε}	{$}

* FOLLOW - SET RULES.

(1). Follow(S) = {$} if S → Start symbol of G1;

(2). if $A \to \alpha B \beta$ is A production $\beta \neq \varepsilon$ then
FOLLOW(B) = FIRST(β) except ε.

(3). if $A \to \alpha B$ or $A \to \alpha B \beta$ is a A product and FIRST(β) contains ε then,
FOLLOW(B) = FOLLOW(A).

follow set will never have an ε - element.

Q. $S \to ABC$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$.

X	FIRST(X)	FOLLOW(X)
S	{a, b, c}	{$}
A	{a, ε}	{b, $}
B	{b, ε}	{c}.

Q. $S \to ABC$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$
$C \to a \mid \varepsilon$

X	FIRST(X)	FOLLOW(X)
S	{a, b, c, ε}	{$}
A	{a, ε}	{b, $}
B	{b, ε}	{a, $}
C	{a, ε}	{$}

Q Compute first and follow sets for the following Grammar variables.

① $S \to aAb \mid AC$
$A \to a \mid \varepsilon$
$B \to b \mid \varepsilon$
C

FIRST
FOLLOW

⇒

X	FIRST(x)	FOLLOW(x)
S	{a, b, ε}	{$}
A	{a, ε}	{b, $}
C	{b, ε}	{$}

(2). $S \rightarrow ABeB$

$A \rightarrow aAb \mid \varepsilon$

$B \rightarrow Aa \mid b \mid \varepsilon$

⇒

X	FIRST(X)	FOLLOW(X)
S	{a, b, e}	{$}
A	{a, ε}	{a, b, e, $}
B	{a, b, ε}	{e, $}

FOLLOW(A)

⇒ FT(Beb)

U FT(b)

FT(a)

⇒ {a, b} − ε U e

⇒ {a, b, e}

(3). $E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

⇒

X	FIRST(X)	FOLLOW(X)
E	{(, id}	{$,)}
E'	{+, ε}	{$,)}
T	{(, id}	{+, $,)}
T'	{*, ε}	{$,), +}
F	{(, id}	{*, +, $,)}