⊛ Introduction of Parsers

A parser for any grammar is a program (and parsing is a technique) that takes as input string 'w' and produces as output either a parse tree for w,

if w is a valid sentence of grammar, or an error message indicating that w is not a valid sentance of given grammar.

There are two way of identifying an elementry subtree.

(a) By driving a string from a non-terminal, or.
(b) By Reducing a string of symbols to a non-terminal.

Based on these two fundamental approaches to parsing namely:

(i) Top Down Parsing (TDP)
(ii) Bottom up Parsing (BUP)

are defined.

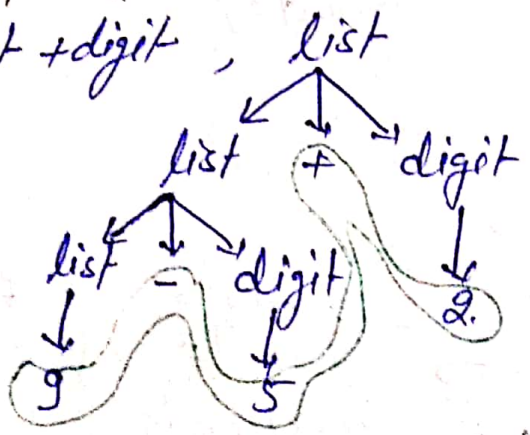
Example:- The production rule of grammar G is:

list → list + digit / list - digit / digit
digit → 0/1/2/ . . . 19.

Then find the parse tree for the expression: 9-5+2.

Solution:- Parse tree is:

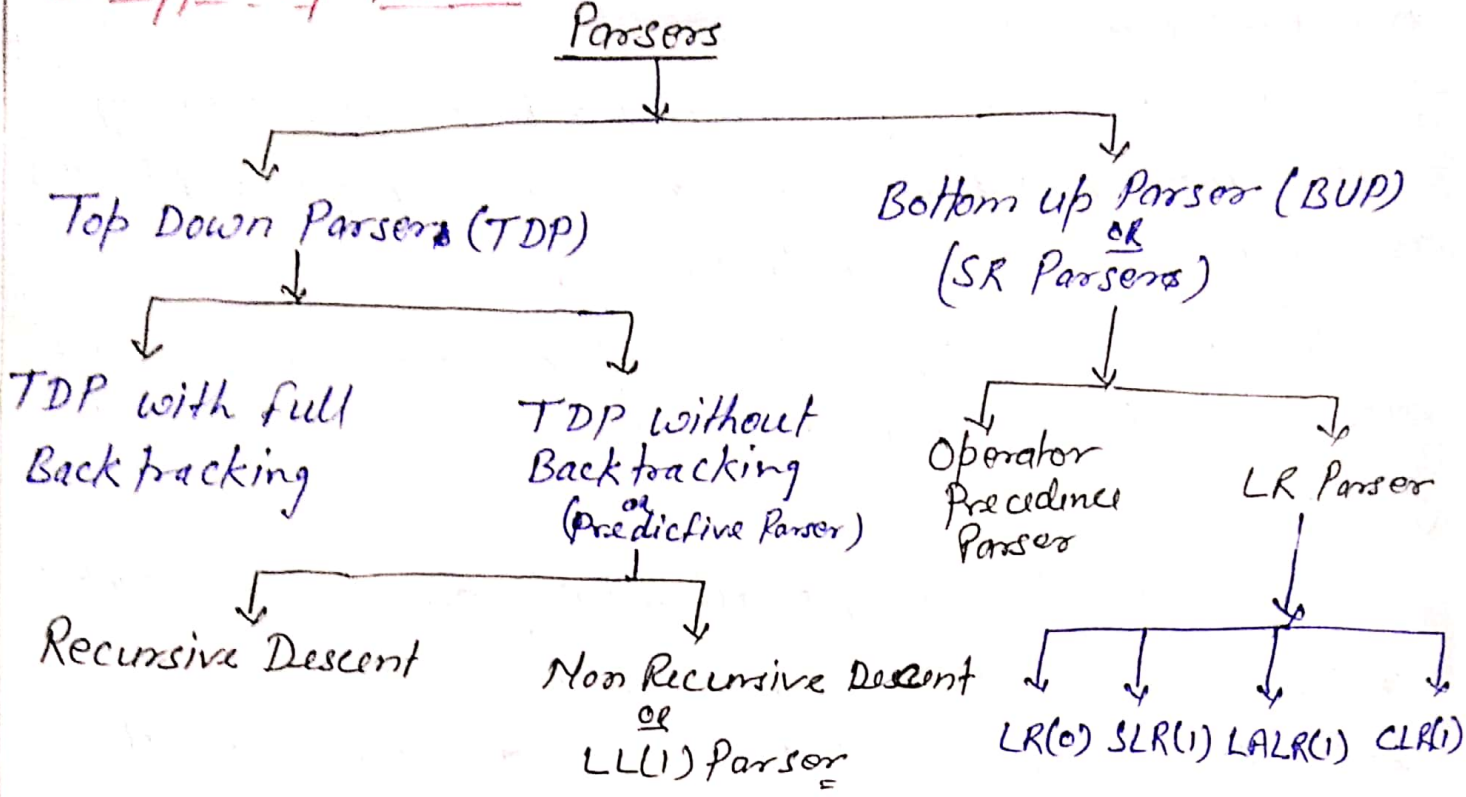
list → list + digit , list



= 9-5+2 ✓ Ans

Scanned with CamScanner

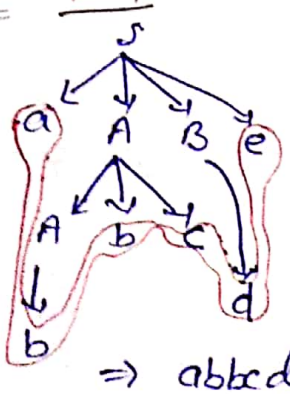
⊛ Types of Parsers



⊛ Basic Difference Between TDP and BUP:-

Example:- $S \to aABe$
$A \to Abc/b$
$B \to d$
for $\omega = abbcde$

Sol^{n}:- TDP

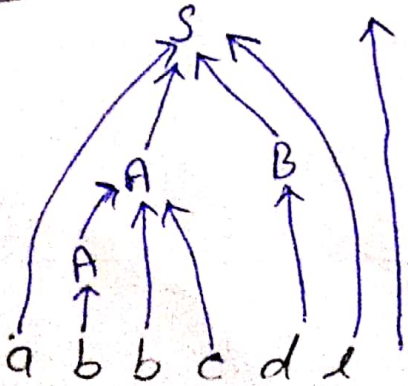


$\Rightarrow abbcde$

$S \to aABe$
$\to aAbcBe$
$\to abbcBe$
$\to abbcde$

} TDP is also used LMD.

BUP



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

} BUP is use Reverse of RMD.

Scanned with CamScanner

⊛ Top Down Parsing :-

Basically top-down parsing attempts to find the left most derivations (LMD) for the input string w, since string w is scanned by the parser. Left to Right, one symbol/token at a time, and the LMD generate the leaves of the parse tree in left to right order, which matches the input scan order.

Example:- Consider the following grammar

$V_n = \{expr, term, rest\}$

$V_t/T = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \varepsilon\}$

expr → term rest

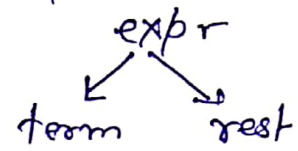
rest → + term rest / - term rest / ε

term → 0/1/2/3/4/5/6/7/8/9

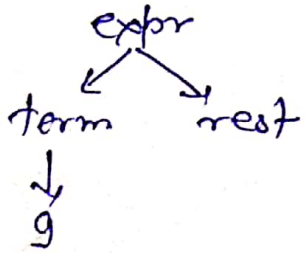
and show the Construction of the parse tree for i/p string: 9-5+2.

Sol^n:- Initialization:- The root of parse tree must be starting s/b (non-terminal of the G., expr

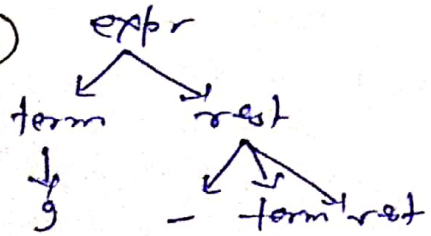
Step ①- expr → term rest



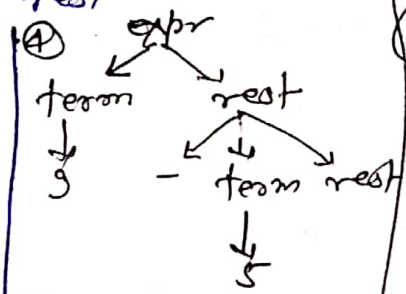
②



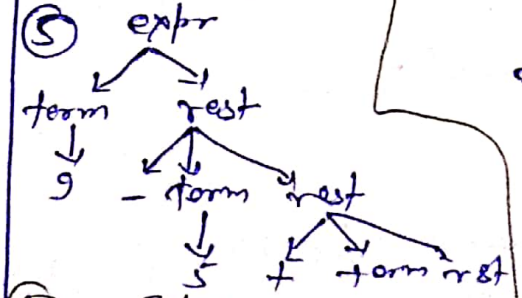
③



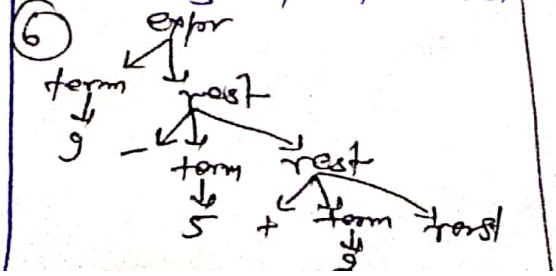
④



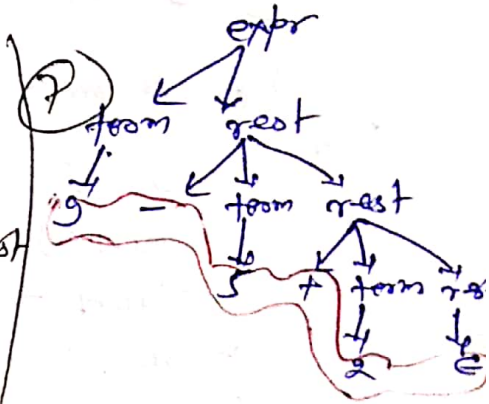
⑤



⑥



⑦



⇒ 9 - 5 + 2 ε

⇓

9 - 5 + 2

Ans

Scanned with CamScanner

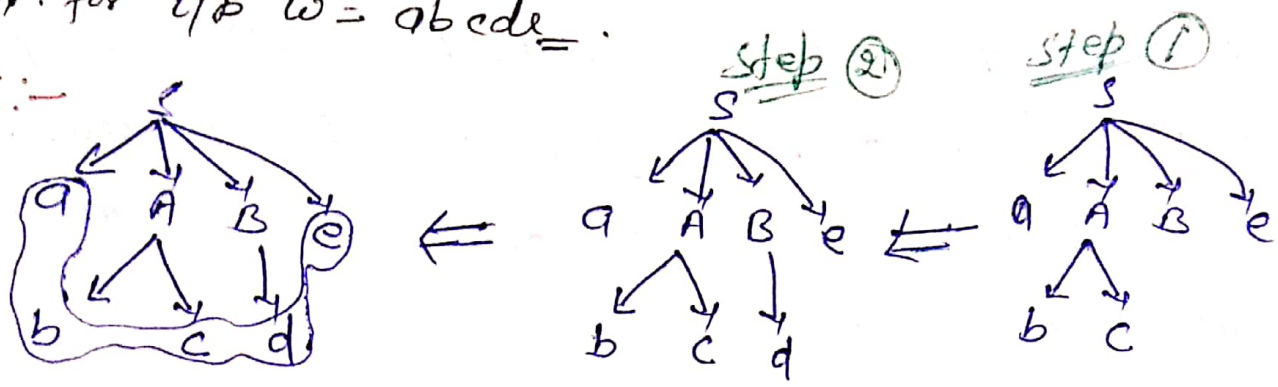
Example:- $S \to aABe$

$A \to bc$

$B \to d$

and Show the Construction of Parse tree for i/p $w = abcde$.

Soln:-



$= abcde$ Ans

Recursive Descent Parsing :-

A TDP that executes a set of recursive procedures to process the input without backtracking is called recursive-descent parser, and parsing is called recursive descent parsing.

Example: suppose i/p stream is : a + b $

i/p stream : a + b $

Then

Lookhead == a

match()

lookhead == +

match()

lookhead == b

.

.

.

In this manner parsing can be done.

Scanned with CamScanner

Example:- Write a code for Recursive-descent parsing of the following grammar:

expr → term rest
rest → +term rest | -term rest | ε
term → 0 | 1 | ... | 9.

Soln:- We know that in a recursive-descent parsing, we write code for each non-terminal of a grammar.

In the above grammar, we should have three procedures correspond to non-terminal: expr, rest, term.

There is only one procedure for non-terminal: expr.

the procedure expr is:

```
expr()
  {
   term();
   rest();
   return;
  }
```

procedure rest uses a global variable i "lookhead" to select correct production or simply select no action. ie ε

```
rest()
  {
   if (lookhead == '+')
     {
      match();
      term();
      rest();
      return;
     }
  else if (lookhead == '-')
     {
      match();
      term();
      rest();
      return;
     }
  else
     {
      return;
  } }
```

The procedure term Check whether global variable is a digit.

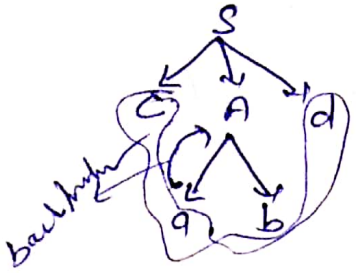
```
term() {
  if (is digit (lookhead))
    {
     match();
     return;
    }
  else
     {
      Error();
     }
```

Scanned with CamScanner

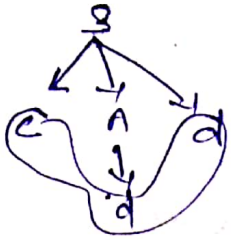
* **Backtracking** :- The parse tree is started from root node and input string is matched against production rules for replacing them.

Example:- Let grammar G be
$S \rightarrow cAd$
$A \rightarrow ab/d$
i/p string $\omega = cdd$.

Sol^n:- $S \rightarrow cAd$



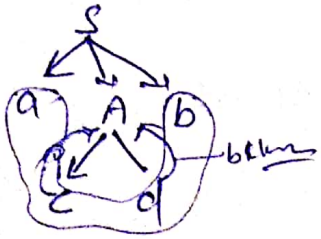
$= cabd \neq cdd$ so
need back tracking, and use another production rule.



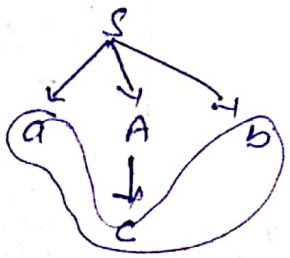
$= cdd = cdd$ ✓ Ans

Example:- Consider the given grammar
$S \rightarrow aAb$
$A \rightarrow cd/c$ and show the backtracking for string $\omega = acb$.

Sol^n:- $S \rightarrow aAb$



$=$



$= acb$ Ans

Scanned with CamScanner

⊛ Top down Parser / Top down Predictive Parser / LL(1)

A predictive parser is an efficient way of implementing recursive-descent parsing since a stack is maintained in predictive parsing for handling the activation records.

In top down predictive parsers, the grammar will be able to predict the right alternative for the expansion of non-terminal during the parsing process; and hence, it need no backtrack.

Note⇒ The parser is initialized with the start symbol of grammar on the stack top and the input pointing to the first token.

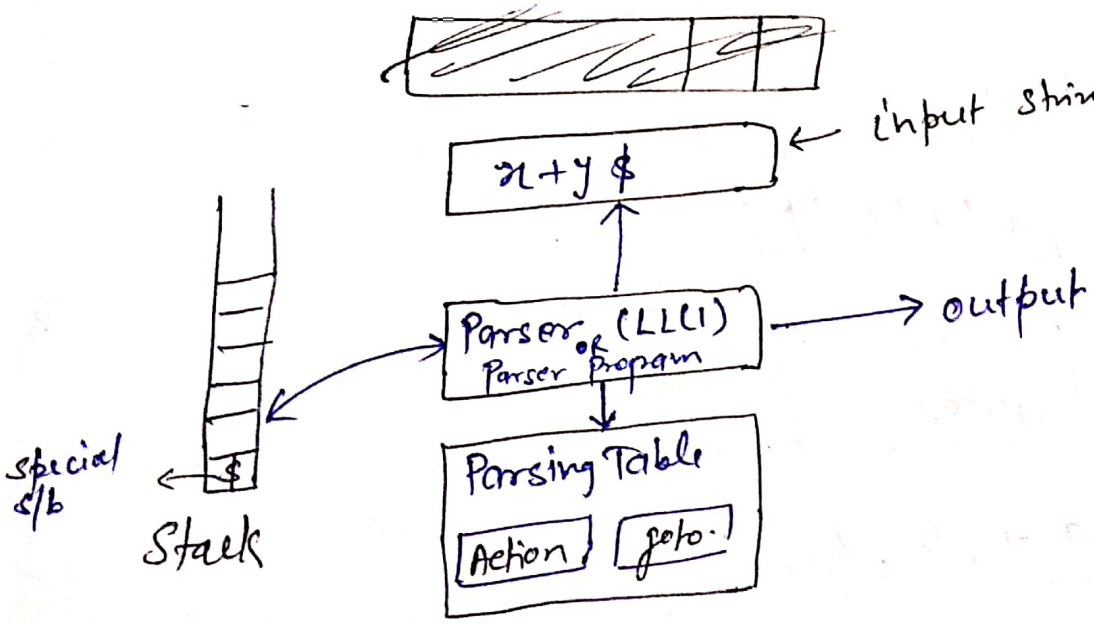


fig: Predictive Parser

⇒ The action of the parser depends on the grammar symbol on the top of the stack,

Scanned with CamScanner

⊛ LL(1) Parser :-

- it accepts LL grammers.
- it is denoted as LL(k)

- A grammar G is LL(1) if there are two distinct productions. eg:- $A \rightarrow \alpha / \beta$

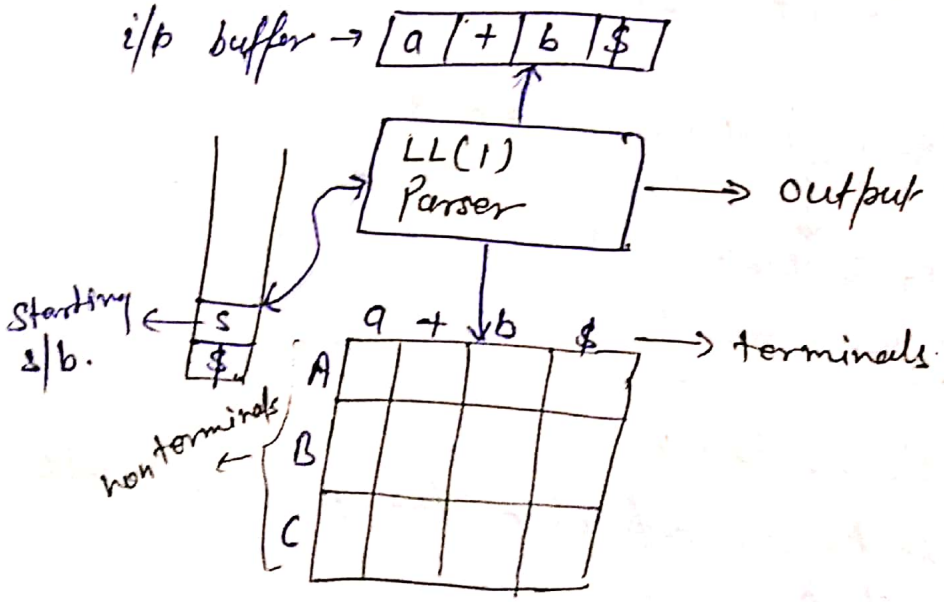
Conditions :-

① for no terminal α and β drive string begining with 'a'.

② At most one of α and β can drive Empty string.

③ if $\beta \rightarrow \epsilon$ then 'a' does not drive any string begining with 'a' terminal in FOLLOW(A).

Structure of LL(1) :-



⊛ Construction of Predictive of LL(1) :-

① FIRST()
FOLLOW()

② Predictive parsing table by using first & follow functions.

③ Stack Implimentation

④ Parse the input stream with the help of the table.

Scanned with CamScanner

⊛ Finding FIRST and FOLLOW :-

(i) FIRST function :-

- First(α) is a set of terminal symbol that begins in strings derived from α / production rule.

eg :- A → abc / def / ghi

then

first(A) = {a, d, g}

(OR)

first() :- is a function which gives the set of terminals that begin the strings derived from the production rule.

- A symbol 'c' is in FIRST(α) if and only if $\alpha \Rightarrow c\beta$ for some sequence of β of grammar symbols.

⊛ Rules for calculating first function :-

(OR)

Algorithm to calculate the first(X) for a Grammar Symbol X.

Rule
1. if X is a terminal, first(X) is {X}. (i.e first(X) = {X}
2. if X → ε is a production, then add ε to the set of first(X).
3. if X is a non-terminal, X → Y
   (i) - if ε ∉ first(Y) is an element of the set of first(X)
   (i.e first(X) = first(Y))
   (ii) if ε ∈ first(Y), & X → YZ then
   first(X) = first(Y) − {ε} ∪ first(Z)

Scanned with CamScanner

Example:- if given grammar is

$E \rightarrow TA$
$A \rightarrow +TA \mid \varepsilon$
$T \rightarrow FB$
$B \rightarrow *FB \mid \varepsilon$
$F \rightarrow (E) \mid id$

then let us calculate first(E), first(A), first(T), first(B) and first(F).

Solution:- first(E) = first(TA) (since $E \rightarrow TA$)
= first(T)
= first(FB)
= first(F)
= { (, id }

first(A) = { +, ε }
first(T) = { (, id }
first(B) = { *, ε }
first(F) = { (, id } Ans

Scanned with CamScanner

⊛ Follow function :-

follow function is a function which gives the set of terminals ~~that begin the strings derived from the production rule~~ that can appear immediatly to the right of a given symbol.

A terminal symbol a is in follow(N) if and only if there is a derivation from the start symbol S of the grammar such that $S \rightarrow \alpha N a \beta$, where α and β are (possibly empty) sequences of grammar symbols.

OR

In other words, a terminal c is in FOLLOW(N) if c may follow N at some point in a derivation.

⊛ Rules for calculating follow function :-

OR

Algorithm to Calculate the follow(X) for a Non-terminal X.

Rule ① :- $ is an element of follow(S), where S is the ~~the~~ start symbol and $ indicates the end of the input.

Rule ② :- if $A \rightarrow \alpha X$ or $A \rightarrow \alpha X Y$ are productions, and first(Y) has an element ε, then set follow(X) is in the set follow(A). i.e. $\boxed{follow(X) = follow(A)}$

Rule ③ :- if $A \rightarrow \alpha X Y$ is a production, the set first(Y) is in the set follow(X), excluding ε.

OR

Scanned with CamScanner

Example:- Calculate the first & follow of given grammar.
if given grammar is:

$E \rightarrow TA$
$A \rightarrow +TA/\varepsilon$
$T \rightarrow FB$
$B \rightarrow *FB/\varepsilon$
$F \rightarrow (E)/id$

Solution: Now Calculate first of A, E, T, B, F.

if $E \rightarrow TA$, then first(E) = {(, id}

$A \rightarrow +TA/\varepsilon$, then first(A) = {+, ε}

$T \rightarrow FB$, then first(T) = {(, id}

$B \rightarrow *FB$; then first(B) = {*, ε}

$F \rightarrow (E)/id$; then first(F) = {(, id}

Now calculate follow of E, A, T, B, F.

follow(E) = {$,)}

follow(A) = {$,)}

follow(T) = {first(A) − ε} ∪ follow(E)
= {+, ε − ε} ∪ {$,)}
= {+, $,)}

follow(B) = follow(T)
= {+, $,)}

follow(F) = {first(B) − ε} ∪ follow(T) ∪ follow(B)
= {*, ε − ε} ∪ {+, $,)} ∪ {+, $,)}
= {*, +, $,)} = Ans

Scanned with CamScanner