Parse trees & Derivations:-

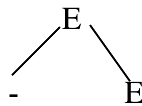
A parse tree is a graphical representation for a derivation that filters out the choice regarding replacement order.

For a given CFG a parse tree is a tree with the following properties.

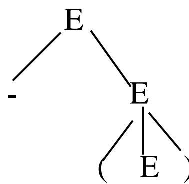
1. The root is labeled by the start symbol
2. Each leaf is labeled by a token or ε
3. Each interior node is labeled by a NT

Ex.

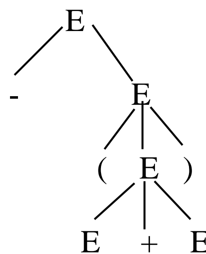
E => -E



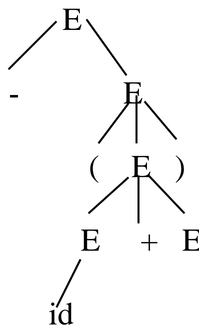
E => -(E)



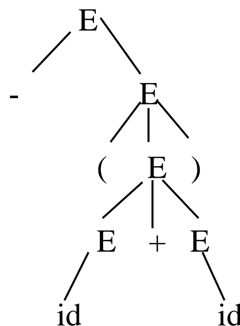
E => -(E+E)



E => -(id+E)



E => -(id+E)



Top-Down Parsing: -

Top down parser builds parse trees starting from the root and creating the nodes of the parse tree in preorder and work down to the leaves. Here also the input to the parser is scanned from left to right, one symbol at a time.

For Example,

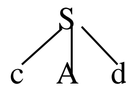
S→cAd

A→ab/a and the input symbol w=cad.

To construct a parse tree for this sentence in top down, we initially create a parse tree consisting of a single node S.

An input symbol of pointer points to c, the first symbol of w.

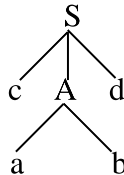
w = cad



The leftmost leaf labeled c, matches the first symbol of w. So now advance the input pointer to 'a' the second symbol of w.

w = cad

and consider the next leaf, labeled A. We can then expand A using the first alternative for A to obtain the tree.



We now have a match for the second input symbol. Advance the input pointer to d,

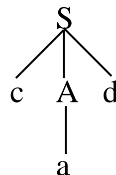
w = cad

We now consider the third input symbol, and the next leaf labeled b. Since *b does not match d,* we report failure and go back to A to see whether there is another alternative for A.

In going back to A we must reset the input pointer to position 2.

W = cad

We now try the second alternative for A to obtain the tree,



The leaf a matches the second symbol of w and the leaf d matches the third symbol. Now we produced a parse tree for w = cad using the grammar S→cAd and A→ab/a.

This is successful completion.

Difficulties of Top – Down Parsing (or) Disadvantages of Top - Down Parsing

1. Left Recursion:-

A grammar G is said to be left recursion if it has a non terminal A such that there is a derivation , $A \overset{+}{\Rightarrow} A\alpha$ for some α

This grammar can cause a top-down parser go into an infinite loop.

Elimination of left Recursion:-

Consider the left recursive pair of production

$A \rightarrow A\alpha / \beta$,

Where β does not begin with A.

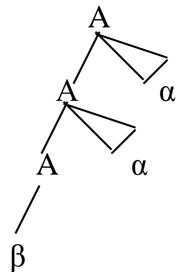
This left recursion can be eliminated by replacing this pair of production with,

$A \rightarrow \beta A'$

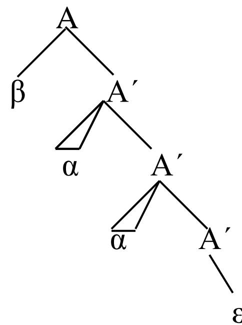
$A' \rightarrow \alpha A' / \varepsilon$

Parse tree of original Grammar:-

$A \rightarrow A\alpha / \beta$



Parse tree for new grammar to eliminate left recursion:-



Example 1:

Consider the following grammar

a. E→E+T/T
b. T→T*F/F
c. F→(E)/id Eliminate the immediate left recursions.

Solution:-

These productions are of the form A→A α/ β
A→ β A´
A´→ α A´/ε

(a) E→E + T / T
the production eliminating left recursion is,
E→TE´
E´ →+T E´/ε

(b) T → T * F / F

T→FT´

T´ →*F T´/ ε

(c) F→(E)/id This is not left recursion. So the production must be F→(E)/id

---

Example 2:- Eliminate left recursion in the following grammar.

S→Aa/b

A→Ac/Sd/e

Solution:-

1.Arrange the non terminals in order

S,A

2. There is no immediate left recursions among the S productions. We then substitute the S productions in A→Sd to obtain the following production.

A→Ac/(Aa/b)d/e

A→Ac/ Aad/bd /e now this production is in immediate left recursion form

A→A(c/ad)/bd/e

The production eliminating left recursion is,

A→(bd/e)A´ ie, A→bdA´/eA´

A´→(c/ad)A´/ε ie, A´→cA´/adA´/ε

So the production is,

1. S→Aa/b
2. A→ bdA´/eA´
3. A´→cA´/adA´/ε

2. Back Tracking:-

The two top down parser which avoid back tracking are,

1. Recursive Descent Parser
2. Predictive Parser

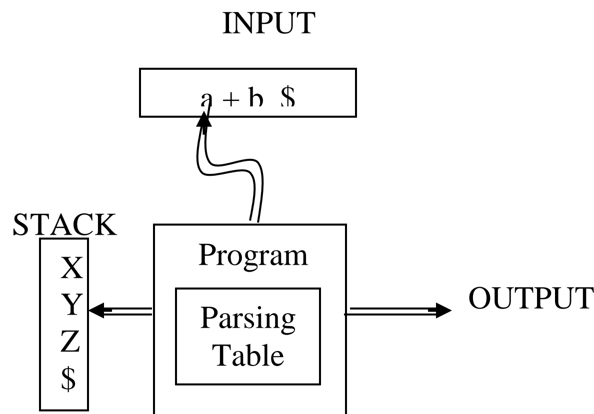
1.Recursive Descent Parser:-

A parser that uses a set of recursive procedures to recognize its input with no back tracking is called recursive descent parser.

2.Predictive Parser:-

A predictive parser is an efficient way of implementing recursive – descent parsing by handling the stack of activation records explicitly.

The picture for predictive parser is,



The predictive parser has,

1. An input – string to be parsed followed by $ (w$)
2. A stack – A sequence of grammar symbols preceded by $(the bottom of stack marker)
3. A parsing table
4. An output

References

1. J. Tremblay, P.G. Sorenson,"The Theory and Practice of Compiler Writing “, McGRAW-HILL,1985.
2. W.M. Waite, L.R. Carter,"An Introduction to Compiler Construction",Harper Collins,New york,1993
3. A.W. Appel,"Modern Compiler Implementation in, CambridgeUniversity Press,1998
4. Internet Papers
5. Aho, R. Sethi, J.D. Ullman," Compilers- Principles, Techniques and Tools"Addison-Weseley, 2007