Recursive Descent Parsing :-

* The general form of top-down parsing is recursive descent parsing.
* Backtracking may be involved in recursive descent parsing. i.e) making repeated scans of the input.
* The special case of recursive descent parsing is predictive parsing where no backtracking is needed.

* Example:

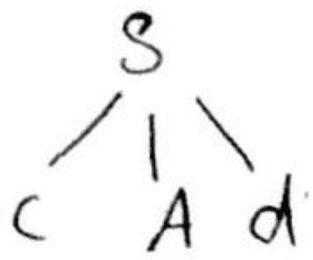
Consider the grammar,

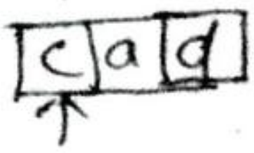
$S \rightarrow cAd$

$A \rightarrow ab \mid a$

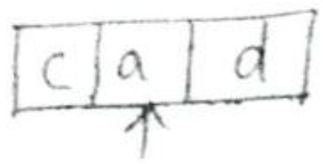
and the input string $w = cad$

* To construct a parse tree for this string,
* We initially create a tree consisting of a single node labeled S.
* An input pointer points to c, the first symbol of w.
* We then use the first production for S to expand the tree and obtain the tree

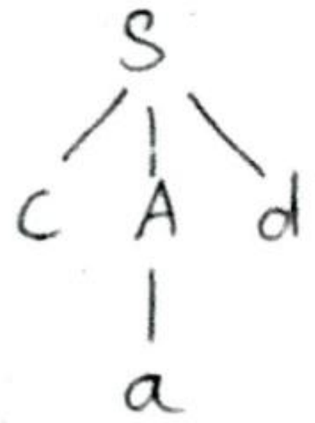


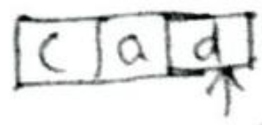


* The left most leaf labeled c, matches the first symbol of w, so we now advance the input pointer to a, the next symbol of w, and consider the next leaf labeled A.
* We can then expand A using the first alternative for A to obtain the tree,



* We now have a match for the second input symbol so we advance the input pointer to d, and compare the next leaf against d.
* 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
* We now try the second alternative for A to obtain the tree





* The leaf a matches the second symbol of w and the leaf matches the third symbol d
* Since we have produced a paise tree for w we halt and announce successful completion of parsing.

Predictive Parsers:

* Eliminating left recursion from it and left factoring the resulting grammar, we can obtain a grammar that can be parsed by a recursive descent parser with no backtracking i.e) a predictive parser

* Example:

$S \rightarrow cAd$

$A \rightarrow ab \mid a$

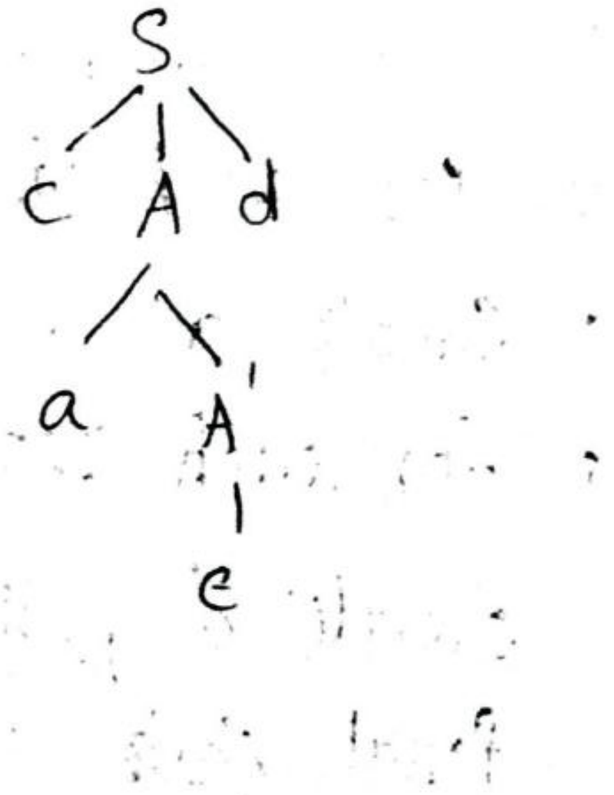
left factoring the grammar

$S \rightarrow cAd$

$A \rightarrow aA'$

$A' \rightarrow b \mid \epsilon$

* Now the parse tree can be constructed as following with no backtracking:



Transition Diagram for Predictive Parser:

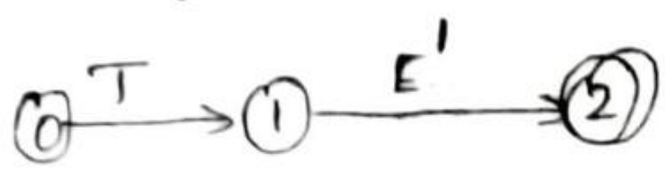
* In parsing there is one diagram for each non-terminal
* The labels of the edges are tokens and non-terminals.
* A transition on a token means we should take that transition if that token (terminal) is the next input symbol.
* A transition on a non-terminal A is a call of the procedure for A
* To construct a transition diagram of a predictive parser from a grammar first eliminate left-recursion from the grammar and then left factor the grammar.
* Then for each non-terminal A do the following
    1. Create an initial and final state
    2. For each production $A \rightarrow X_1 X_2 \ldots X_n$. create a path from the initial to the final state with edges labeled $X_1, X_2, \ldots X_n$

* Example, consider the grammar

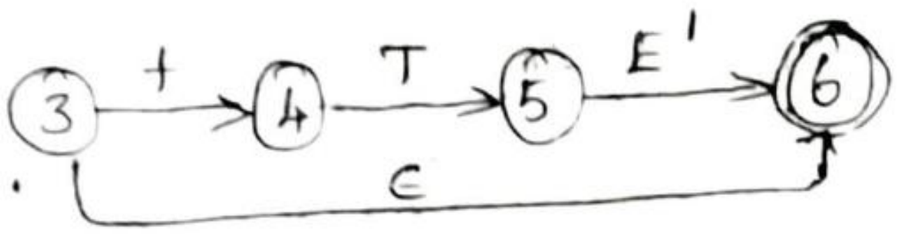
$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow (E) \mid id$

* The following grammar contains a collection of transition diagrams for the above grammar

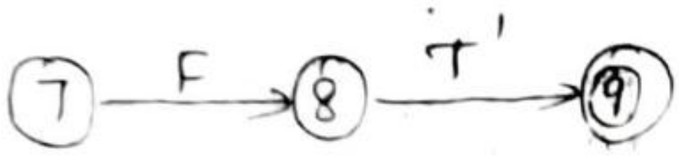
E :



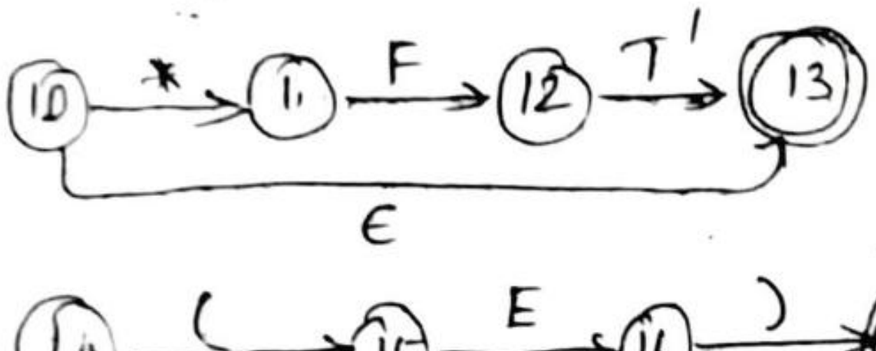
E' :



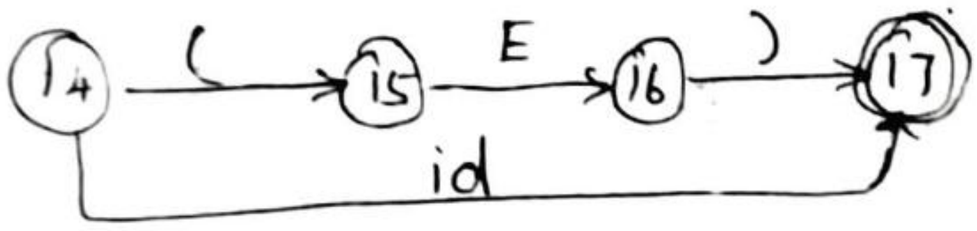
T :



T' :

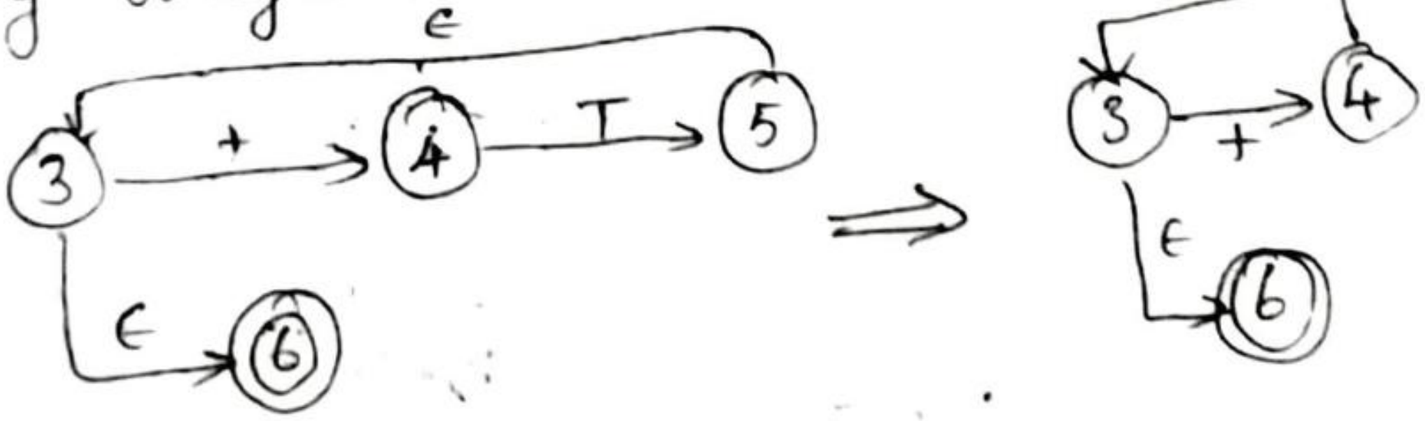


F :

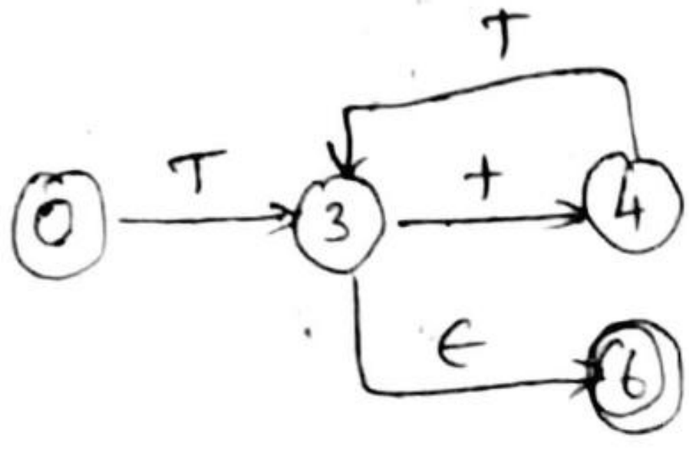


* Transition diagrams can be simplified by substituting diagrams in one another.

E' :



E :



* A C implementation of this simplified predictive parser runs 20-25% faster than a C implementation of the normal predictive parsers