Top-Down Parsing

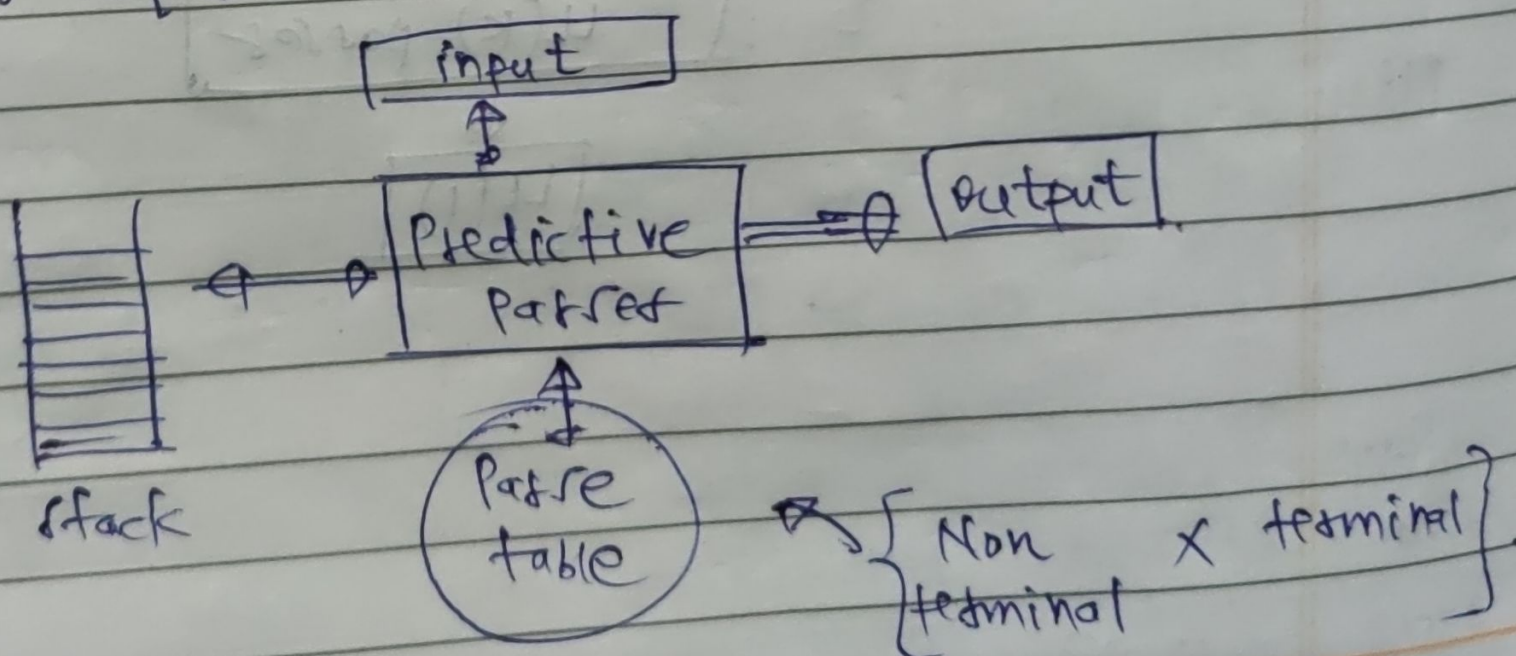
- Apply :
  (a) Removal of left-Recursion
  (b) left-factoring
  to the grammar, before doing anything.
  (Because both are problems for top-down parsers).

(1) Recursive-descent Parser :-

- uses recursive procedures for each Non-terminal or a non-recursive equivalent.
- and an extra match() procedure.
- No stack required at all.

(2) Predictive Parser :-

- is a recursive decent parser which doesn't need back-tracking or backup.
- This parser accepts LL(K) languages.
- It required a stack and parse table



Construction of parse table:- (for LL(1) parsing) (LL(K) ≡ Predictive Parsing)

- Remove left-recursion and left-factoring
- Rules:- for each production $A \to \alpha$ do:

[Compute first and follow first]

(i) $M[A, first(\alpha)] = (A \to \alpha)$
where first(α) not contains $\in$.

(ii) If $\in$ is in first(α), then:
$M[A, follow(A)] = (A \to \alpha)$

(iii) If $\in$ is in first(α) and $ in follow(A), then:
$M[A, \$] = A \to \alpha$

- If these are no multiple entries in the table, then the grammar is LL(1).

Example:- bexpr → bexpr or bterm | bterm
bterm → bterm and bfactor | bfactor
bfactor → not bfactor | (bexpr) | true | false.
Construct LL(1) parse table for above grammar.

Solution:-

(a) bexpr → bterm E'
E' → or bterm E' | ∈
bterm → bfactor T'
T' → and bfactor T' | ∈
bfactor → not bfactor | (bexpr) | true | false.

(b) First(bterm) = First(bexpr) = First(bfactor)
= {not, (, true, false}

First(E') = {ϵ, or}

First(T') = {ϵ, and}

(c) Computation of Follow set:-

	bexpr	bterm	bfactor	E'	T'
rule 1:	$				
rule 2:	)	or	and		
rule 3:				$,)	or
rule 4:		$,)	or, $,)		$,)

(d)

	or	and	not	(	)	true/ false	$
bexpr			bexpr → bterm E'	bexpr → bterm E'		bexpr → bterm E'	
E'	E' → or bterm E'				E' → ϵ		E' → ϵ
bterm			bterm → bfactor T'	bterm → bfactor T'		bterm → bfactor T'	
T'	T' → ϵ	T' → and bfactor T'			T' → ϵ		T' → ϵ
bfactor			bfactor → not bfactor	bfactor → (bexpr)		bfactor → true/ false	