Top-Down Parser

Thursday, March 31, 2022 12:02 PM

Parsing :- We can define a parser formally a program 'P', for a CFG 'G', which take a string w as input and outputs

i) a parse tree, if w is a sentence of G or

ii) an error message, if $w \notin L(G)$.

Parsing
- Top-down parsing
  1. Brute force method
  2. Recursive descent parser
  3. predictive parser
     - → LL(1) parser.
- Bottom-up parsing
  1. Shift-Reduce parser
  2. Operator precedence parser
  3. LR(K) parser
     - LR(0)
     - SLR(1)
     - LALR(1)
     - CLR(1)/LR(1)

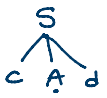
Top-down Parsing :=

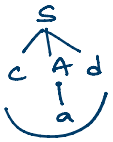
A top-down parser constructs a parse tree starting from Root (starting Nonterminal) and expands it till to the leaves (input sentence).

Top-down parser uses leftmost derivation (LMD).

Ex:- S → cAd
A → ab/a
Input: cad

i) S

ii) 

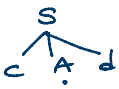
iii) 

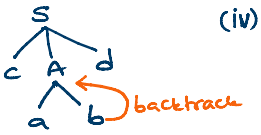
① Brute-Force Method :=

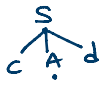
It is a top-down parser with "full backtracking".

Ex:- S → cAd
A → ab/a
Input: cad

(i) S

(ii) 

(iii) 

(iv) 

(v) 

Advantage:- It is easy to implement.

Disadvantage: It is time consuming process (Backtracking).

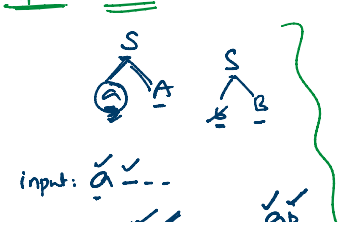
② Recursive Descent parser :-

It is a top-down parser with "no backtracking".

Backtracking is avoided by realizing each nonterminal as one function call.

	a	b
S	S→aA	S→bB

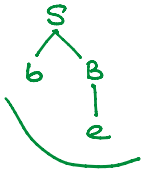
Ex:- S → aA/bB
A → d
B → e
Input: be



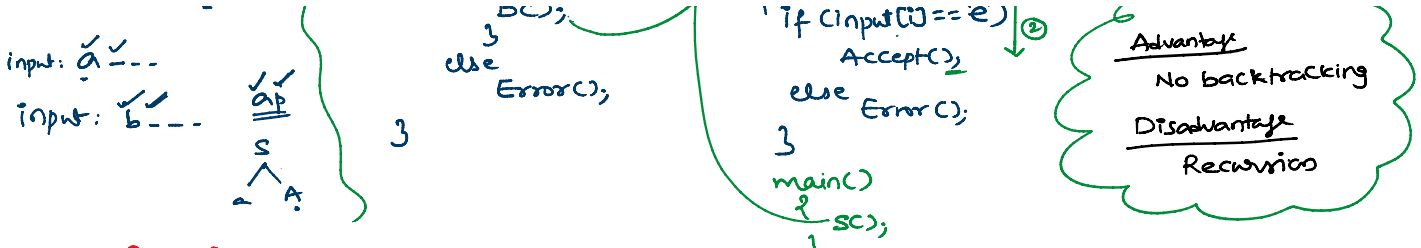

```
S()
{ if(input[i] == 'a')
  { i++;
    A();
  }
  else if (input[i] == 'b')
  { i++;
    B();
  }
  else
    Error();
}
```

```
A()
{ if(input[i] == 'd')
    Accept();
  else
    Error();
}
```

```
B()
{ if (input[i] == 'e')
    Accept();
  else
}
```

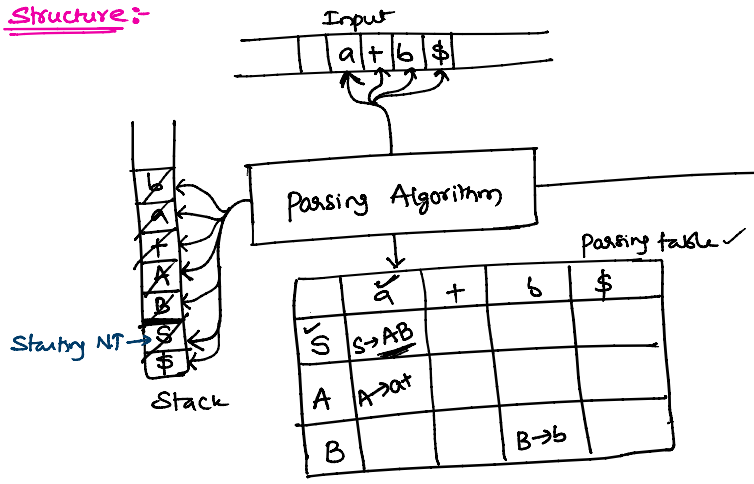


Advantage
No backtracking



③ **Predictive Parser :-** It is a tabular representation of recursive descent parser.

Structure :-



Initial Configuration

Stack	input
$S	w$

	Stack	input	Actions
1.	terminal (a)	terminal (a)	→ pop
2.	$	$	→ Accept
3.	Nonterminal	terminal	→ pop LHS and push RHS in reverse.
4.	otherwise		→ Error

First and follow set :- The construction of a predictive parse table is aided by two functions called First and follow set.

First set :- first(α) be the set of terminals that begin the strings derived from 'α'.

Rules: 1. If $X \to a$ is a production; then first(X) = {a}

2. If $X \to \epsilon$ is a production; then first(X) = {ϵ}

3. If $X \to Y_1 Y_2 \dots Y_n$ is a production then

(i) first(X) = first(Y_1)

(ii) If first(Y_1) contains 'ϵ' then add first(Y_2) to first(X).

(iii) If first(Y_i) contains 'ϵ' for all i = 1 to n then add 'ϵ' to first(X).

① $S \to aA / AB / \epsilon$, $A \to Bb / \epsilon$, $B \to d / \epsilon$

	First	Follow
S	a, d, b, ϵ	$
A	d, b, ϵ	$, d
B	d, ϵ	$, b

② $S \to aA / Bb$, $A \to BC / \epsilon$, $B \to d / \epsilon$, $C \to e / \epsilon$

	First	Follow
S	a, d, b	$
A	d, e, ϵ	$
B	d, ϵ	b, e, $
C	e, ϵ	$

③ $S \to AA / \epsilon$, $A \to aA / \epsilon$

	First	Follow
S	a, ϵ	$
A	a, ϵ	a, $

④ $S \to ACB / Cbb$, $A \to da / BC$, $B \to g / \epsilon$, $C \to h / \epsilon$

	First	Follow
S	d, g, h, e, b	$
A	d, g, h, ϵ	h, g, $
B	g, ϵ	$, h, g
C	h, ϵ	g, $, b, h

⑤ $S \to aA$, $A \to BC / d$, $B \to Sb / \epsilon$, $C \to Aa / \epsilon$

	First	follow
S	a	$, b
A	d, ϵ, a	$, b, a
B	ϵ, a	a, d, b, $
C	ϵ, d, a	$, b, a

⑥ $S \to bSa / Sd / \epsilon$

	First	Follow
S	b, ϵ, d	$, a, d

(3) $S \to AA$, $A \to aA / \epsilon$

	First	Follow
S	a, ϵ	$
A	a, ϵ	a, $

(6) $S \to bSa / Sd / \epsilon$

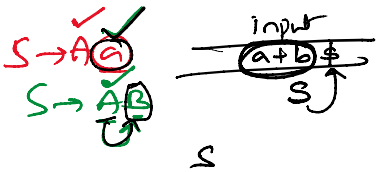
	First	Follow
S	b, ϵ, d	$, a, d

Follow set := Follow(A) for nonterminal A, to be the set of terminals that can appear immediately to the right of A in some sentential form.

Rules: 1) Follow(S) = {$} where 'S' is starting nonterminal

2) If $A \to \alpha B\beta$; then (i) Follow(B) = First(β)

ii) If First(β) contains 'ϵ'
then Follow(B) = Follow(A)



Follow(C) = First(D)