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)

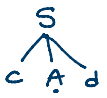
Top-down Parsing :=

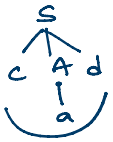
A top-down parser constructs a parse tree starting from Root (starting Nonterminal) and expands it till to the leaves (input sentence).

Top-down parser uses leftmost derivation (LMD).

Ex:- S → cAd
A → ab/a
Input: cad

i) S

ii) 

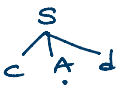
iii) 

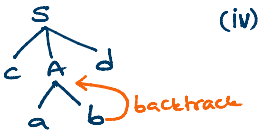
① Brute-Force Method :=

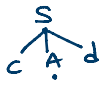
It is a top-down parser with "full backtracking".

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).

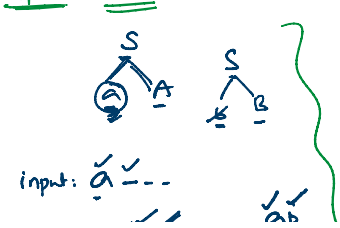
② Recursive Descent parser :-

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

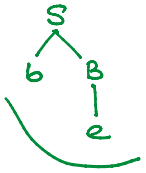
Ex:- S → aA/bB
A → d
B → e
Input: be



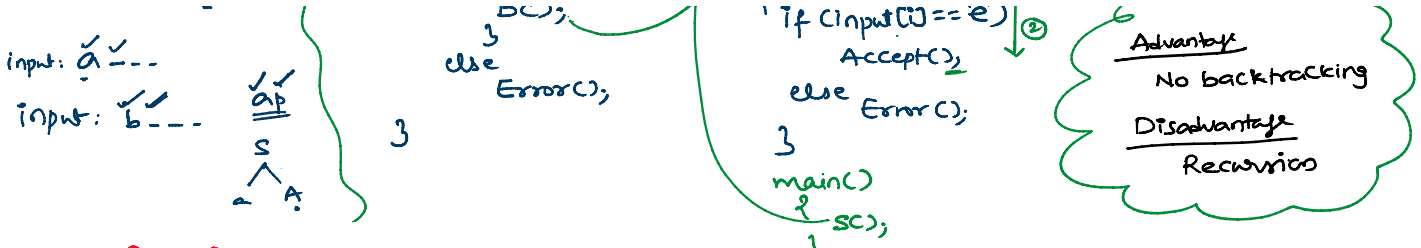

```
S()
{ if(input[i] == 'a')
  { i++;
    A();
  }
  else if (input[i] == 'b')
  { i++;
    B();
  }
  else
    Error();
}
```

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

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

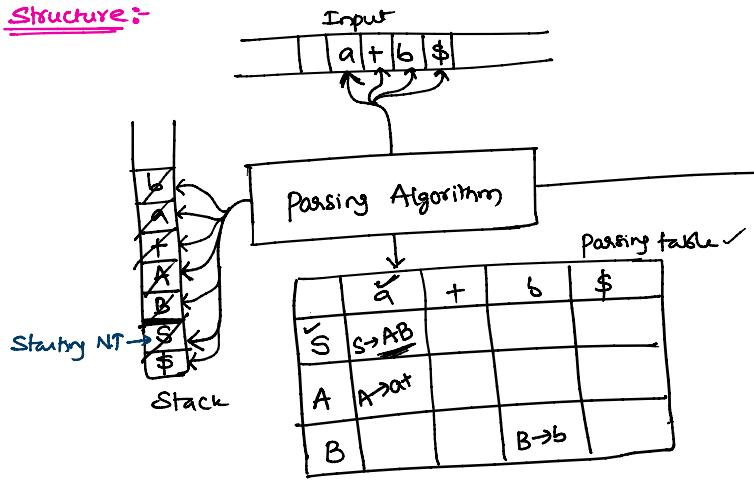


Advantage
No backtracking



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)

Construction of predictive parsing table

1. Add $A \to \alpha$ to $M[A, first(\alpha)]$
2. If first(α) contains 'ε' then add $A \to \alpha$ to $M[A, follow(A)]$
3. All the undefined entries considered as error entries.

Q:- Construct predictive parse table for the following grammar.

$S \to aA / bB$

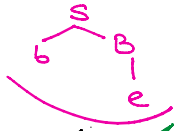
$A \to d / \epsilon$

$B \to e / \epsilon$

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

	a	b	d	e	$
S	S→aA	S→bB			
A			A→d		A→ε
B				B→e	B→ε

Stack	Input	Actions
$S	be$	S→bB
$Bb	be$	pop
$B	e$	B→e
$e	e$	pop
$	$	Accept.



Q:- $S \to aA / bB$, $A \to bb / \epsilon$, $B \to d / \epsilon$

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

input: dbd

	a	b	d	$
S	S→aA	S→AB	S→AB	S→AB
A		A→Bb	A→Bb, A→ε	A→ε
B		B→ε	B→d	B→ε

multiply defined entry.

the grammar given is not suitable for predictive parser

Q:- $S \to AA$, $A \to aA / \epsilon$

	First	Follow
S	a, ε	$
A	a, ε	a, $

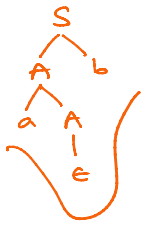
Q:- $S \to Ab / \epsilon$, $A \to aA / \epsilon$

	First	Follow
S	a, ε, b	$
A	a, ε	b

input: ab

	a	b	$
S	S→Ab	S→Ab	S→ε
A	A→aA	A→ε	

Stack	Input	Actions
$S	ab$	S→Ab
$bA	b$	A→aA
$bAa	ab$	pop
$bA	b$	A→ε
$b	b$	pop
$	$	Accept



	a, ϵ	$
A → aA/ϵ	a, ϵ	a, $

	a	$
S	S → AA	S → AA
A	A → aA A → ϵ	A → ϵ

multiply defined entry :

the grammar is not suitable for predictive parser.

LL(1) Grammar

- 1 → Number of look ahead symbols.
- L → Left most derivation
- L → Left-to-right Scanning of i/p

Rules :

1. No Ambiguous grammar is LL(1).
2. No Left recursive grammar is LL(1)
3. Any production is of the form $A \to \alpha / \beta$.
   - (i) $first(\alpha) \cap first(\beta) = \phi$
   - (ii) If $first(\alpha)$ contains 'ϵ' then $Follow(A) \cap first(\beta) = \phi$

Ex:- $S \to aA / bB \Rightarrow$ LL(1)

Ex2: $S \to abA / adB \Rightarrow$ not LL(1), but LL(2)

Ex3 $S \to abdA / abeB \Rightarrow$ not LL(1), not LL(2), LL(3).

⇒ Check the following grammars for LL(1) :-

① $S \to TETSS' / a$

$S' \to eS / \epsilon$

$E \to b$

i) $first(TETSS') \cap first(a) = \{T\} \cap \{a\} = \phi$

i) $first(eS) \cap first(\epsilon) = \{e\} \cap \{\epsilon\} = \phi$

ii) $first(eS) \cap follow(S') = \{e\} \cap \{e, \$\} = \{e\} \neq \phi$

$follow(S') = \{\$, e\}$

	e	$
S'	S' → eS S' → ϵ	S' → ϵ

not LL(1)

② $S \to AA$

$A \to aA / \epsilon$

i) $first(aA) \cap first(\epsilon) = \{a\} \cap \{\epsilon\} = \phi$

ii) $first(aA) \cap follow(A) = \{a\} \cap \{a, \$\} = \{a\} \neq \phi$

	a	$
A	A → aA A → ϵ	A → ϵ

not LL(1)

③ $E \to E+E / id$ ⇐ not LL(1) Ambiguous

④ $S \to aA$

$A \to Ab / \epsilon$ ⇐ Left recursive not LL(1)