Bottom-Up Parser

Thursday, March 31, 2022 12:03 PM

Bottom-up parser :-

It constructs parsetree starting from bottom/leaves/ input sentence and reduce it to top/Root/ starting Nonterminal.

It uses rightmost derivation in reverse.

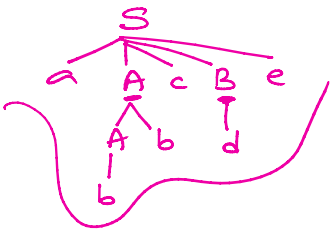
Ex:- S→aAcBe
A→Ab/b
B→d
input: abbcde

abbcde
aAbcde
aAcde
aAcBe
S

A→b ✓
A→Ab ✓
B→d ✓
S→aAcBe ✓

"Handle" :- It is RHS of a production

"Handle pruning" : Replacing RHS of a production with its LHS.



Reduction : Each replacement of the RHS of a production by the LHS in the process of reducing it to the starting Nonterminal is called Reduction.

Shift-Reduce Parser :-

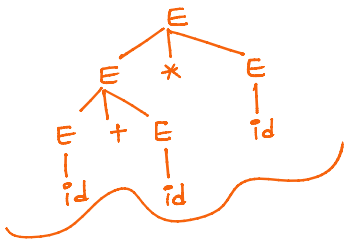
It can be implemented using stack to hold the grammar symbols and input buffer to hold the input sentence. Initial configuration is

Stack	input
$	w$

4 possible actions :- 1. Shift :- push i/p onto the stack.
2. Reduce :- pop RHS and push LHS
3. Accept :- if stack contains $S and input contains $
4. Error :- otherwise

Ex:- E→E+E/E*E/id
input: id+id*id

Stack	input	Action
$	id+id*id$	Shift
$id	+id*id$	E→id ✓
$E	+id*id$	Shift
$E+	id*id$	Shift
$E+id	*id$	E→id ✓
$E+E	*id$	E→E+E ✓
$E	*id$	Shift
$E*	id$	Shift
$E*id	$	E→id ✓
$E*E	$	E→E*E ✓
$E	$	Accept



Note :- Shift-Reduce parser is possible for some of the Ambiguous grammars.

Operator precedence parser :-

→ It is an efficient Shift-Reduce parser that can delimit the handles by giving precedence relations (<, =, >) between pair of terminals.

→ It is an efficient Shift-Reduce parser that can delimit the handles by giving precedence relations between pair of terminals. (<, =, >)

→ It accepts an operator grammar :=

└ It is a ∈-free grammar and no two nonterminals are adjucent.

Ex:- i) ~~S → ∈~~ ii) ~~S → AB~~ iii) S → a (iv) S → AaB v) S → AabB

→ There are three precedence relations

i) $a < b$ ⇒ a gives precedence to b. ('b' reduce first)

ii) $a = b$ ⇒ a has same precedence as b ('a' and 'b' reduce together)

iii) $a > b$ ⇒ a takes precedence over b ('a' reduce first).

⟹ These precedence relations guide the selection of handles with '<' marking the left end, '=' appearing in the interior and '>' marking the right end.

Q:- Construct an operator precedence parser for the following grammar

E → E+E / E*E / id

With an assumption that id > * > + and both * and + are left associative.

	id	+	*	$
id	–	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	accept

id - id +<id *<id $<id

id > + +>+ *>+ $<+

id > * +<* *>* $<*

id > $ +>$ *>$ $-$

Q:- Construct a parse tree for the input id + id * id using operator precedence parse table above.

Input

$ < id > + < id > * < id > $

$ < E + < id > * < id > $

$ < E + < E * < id > $

$ < E + < E * E > $

$ < E + E > $

$ E $

E → id ✓

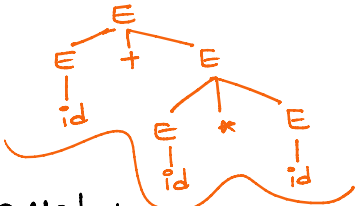
E → id ✓

E → id ✓

E → E * E

E → E + E

Accept



Lead and Trail Set :=

Finding the precedence relations is based on two functions called Lead and Trail set.

Lead set :=

1. If the production is of the form $A \to \alpha a \beta$ then Trail(A) = {a} iff α is single nonterminal or '∈'.
2. If the production is of the form $A \to B\alpha$ ($A \to \alpha B$) then Lead(A) = Lead(B) (Trail, Trail).

Q:-

	Lead	Trail
S → AaBeC	a, b, f	e, h
A → Bbd	b, f	d
B → fCgD	f	g, i
C → h	h	h
D → i	i	i

Q:-

	Lead	Trail
S → AaBb	a, d	b
A → dBe	d	e
B → fg	f	g

Computation of Precedence relations

D → i i | I

Computation of precedence relations

Let us consider the production is of the form $A \rightarrow \alpha_1 \alpha_2 \cdots \alpha_n$ where α_i is either single Nonterminal or single terminal.

i) if α_i and α_{i+1} are terminals then $\alpha_i = \alpha_{i+1}$

ii) if α_i and α_{i+2} are terminals and α_{i+1} is non terminal then $\alpha_i = \alpha_{i+2}$

iii) if α_i is terminal and α_{i+1} is nonterminal then $\alpha_i < Lead(\alpha_{i+1})$

iv) if α_i is nonterminal and α_{i+1} is terminal then $Trail(\alpha_i) > \alpha_{i+1}$

v) $\$ < Lead(S)$ and $Trail(S) > \$$ where 'S' is starting nonterminal.

$S \rightarrow aAb$