Operator Procedence Parsing

Learn Vid Fun...

Operator Precedence Grammar :-

- A grammar G is said to be operator precedence if it possess following two properties -

  i) No production on the right side is ε.

  ii) There should not be any production rule possessing two adjacent non-terminals at the right hand side.

- It is a small but important class of grammars.
- An operator precedence parser is a bottom - up parser that interprets an operator precedence grammar.

Example :-

$E \rightarrow EAE \mid (E) \mid -E \mid id$ $A \rightarrow + \mid - \mid * \mid / \mid \wedge$ Not an operator precedence grammar	$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E \wedge E \mid$ $(E) \mid -E \mid id$ An operator precedence grammar

Designing Operator Precedence Parser :-

In operator precedence parsing , we first define three disjoint precedence relations between every pair of terminals and construct the operator precedence table.

- $a \lessdot b$ if b has higher precedence than a
- $a \doteq b$ if b has same precedence as a
- $a \gtrdot b$ if b has lower precedence than a

Rules to determine precedence relations :-

The determination of correct precedence relations between terminals are based on the traditional notions of associativity and precedence of operators.

Remember :-

- id has higher precedence than any other symbol.
- $ has lowest precedence.
- If two operators have equal precedence, then we check the associativity of that particular operator.

Parsing the given string :-

Step-1 :- Insert -

- \$ symbol at the start and at the end of input string
- Precedence operator in between every two symbols of the string by referring the designed precedence table.

Step-2 :- Start scanning the string from left until seeing ·> and put a pointer on its location. Now, scan backwards the string from right to left until seeing <· . Everything between the two relations <· and ·> forms the handle. Replace handle with the head of the respective production.

Repeat this step until reaching start symbol.

Problem-01 :

Construct operator precedence parser for the following grammar-

$E \rightarrow EAE \mid id$

$A \rightarrow + \mid *$

Then, parse the following string-

$id + id * id$

Solution :-

Step-1 : **Convert the given grammar to operator precedence grammar-**

The equivalent operator precedence grammar is-

$$E \rightarrow E+E \mid E*E \mid id$$

Step-2 : **Construct the operator precedence table-**

The terminal symbols we will consider are-

$$\{ id , + , * , \$ \}$$

	id	+	*	$
id		⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	

Operator Precedence Table

Parsing the given string :-

Step-1 :-

id + id * id

(Given string to parse)

Insert \$ at the two ends of the string

\$ id + id * id \$

Insert precedence operators in between the string

$\$ \lessdot id \gtrdot + \lessdot id \gtrdot * \lessdot id \gtrdot \$$

Step-2 :-

Scanning and parsing :-

$$\$ \underbrace{\lessdot id \gtrdot} + \lessdot id \gtrdot * \lessdot id \gtrdot \$$$

$$\$ E + \underbrace{\lessdot id \gtrdot} * \lessdot id \gtrdot \$$$

$$\$ E + E * \underbrace{\lessdot id \gtrdot} \$$$

$$\$ E + E * E \$$$

$$\$ + * \$$$

$$\$ \lessdot + \underbrace{\lessdot * \gtrdot} \$$$

$$\$ \lessdot + \gtrdot \$$$

$$\$ \$$$

Problem-02:

Construct operator precedence parser for the grammar-

$$S \rightarrow (L) / a$$

$$L \rightarrow L, S / S$$

Also, parse the following string-

$$(a , (a , a))$$

Solution:-

The terminal symbols in the given grammar are-

$$\{ (,) , a , , \}$$

We will build the operator precedence table for these operators.

	a	(	)	,	$
a		$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
(	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
)	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
,	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	

Operator Precedence Table

Parsing the given string :-

Step-1 :-

$$(a , (a , a))$$

(Given string)

$\Downarrow$ Insert \$ at the two ends of the string

$$\$ (a , (a , a)) \$$$

$\Downarrow$ Insert precedence operators in between the string

$$\$ \lessdot (\lessdot a \gtrdot , \lessdot (\lessdot a \gtrdot , \lessdot a \gtrdot) \gtrdot) \gtrdot \$$$

Step-2 :-

Scanning and parsing :-

$$\$ \lessdot (\lessdot a \gtrdot , \lessdot (\lessdot a \gtrdot , \lessdot a \gtrdot) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , \lessdot (\lessdot a \gtrdot , \lessdot a \gtrdot) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , \lessdot (S , \lessdot a \gtrdot) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , \lessdot (S , S) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , \lessdot (L , S) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , \lessdot (L) \gtrdot) \gtrdot \$$$

$$\$ \lessdot (S , S) \gtrdot \$$$

$$\$ \underbrace{\lessdot (L , s) \gtrdot} \$$$

$$\$ \underbrace{\lessdot (L) \gtrdot} \$$$

$$\$ \; s \; \$$$

$$\$\$$$

Advantages :-

- This type of parsing is simple to implement.
- This parser is powerful enough for expressions in programming languages.

Disadvantages :-

- It is hard to handle tokens like minus sign which has two different precedence depending on whether it is unary or binary.
- Only small class of grammars can be parsed using operator precedence parsing.

Operator Precedence Functions :-

The operator precedence parsers usually do not store the precedence table with the relations. Rather they are implemented in a special way.

Operator precedence parsers use precedence functions that map terminal symbols to integers and so the precedence relations between the symbols are implemented by numerical comparison.

Problem-03 :-

Consider the following grammar -

$$E \rightarrow E + E \ / \ E * E \ / \ id$$

A) Construct operator precedence parser

B) Find the operator precedence function.

Solution :-

The terminal symbols we will consider for constructing operator precedence table are-

$$\{ +\ ,\ *\ ,\ id\ ,\ \$ \}$$

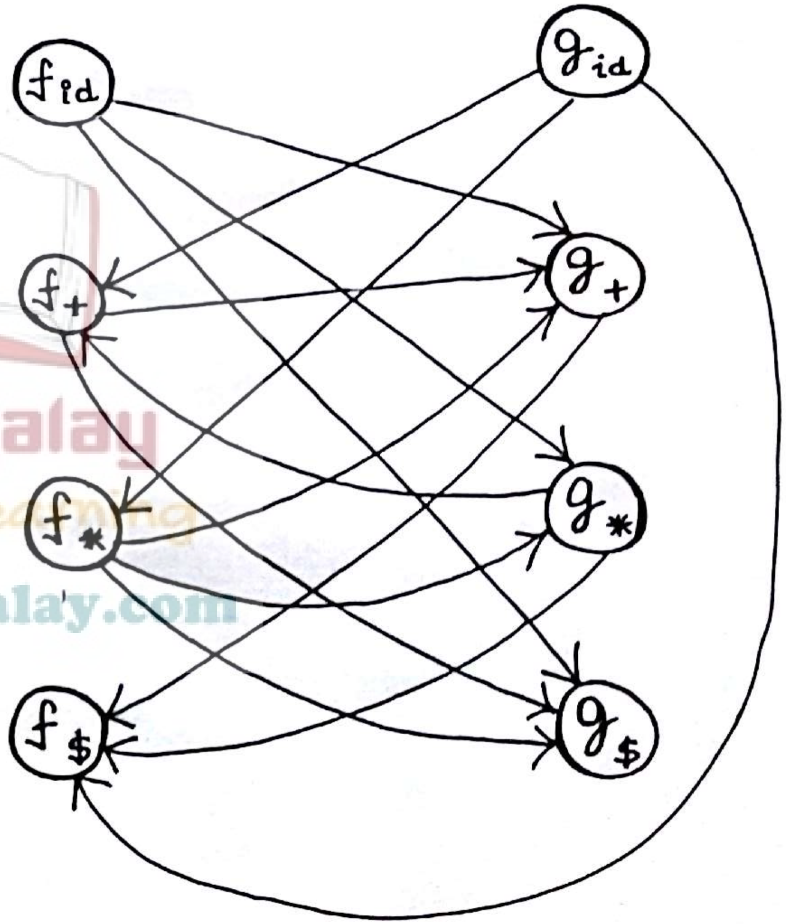
g →

f ↓

	id	+	*	$
id		⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	

Operator Precedence Table

The graph representing the precedence function is-



The longest possible paths are -

- $f_{id} \rightarrow g_* \rightarrow f_+ \rightarrow g_+ \rightarrow f_\$$
- $g_{id} \rightarrow f_* \rightarrow g_* \rightarrow f_+ \rightarrow g_+ \rightarrow f_\$$

The resulting precedence functions are -

	+	*	id	$
f	2	4	4	0
g	1	3	5	0