Operator Precedence Parsing

Dr. Meera Thapar Khanna

CSE Department

Pandit Deendayal Energy University

OPERATOR PRECEDENCE PARSING

Operator precedence grammar is kinds of shift reduce parsing method that can be applied to a small class of operator grammars. It can process ambiguous grammars also.

An operator grammar has two important characteristics:

1. There are no ε productions.

2. No production would have two adjacent non terminals.

Is below grammar operator grammar?

E → E A E |(E)|id

A → +| − | ∗

- No, it is not an operator Grammar because it does not satisfy 2nd property of operator Grammar.
- It contains two adjacent Non-terminals on R.H.S of production

  i.e. E → E A E

To convert this grammar to operator precedence grammar we can substitute the value of A in EAE. The operator grammar is:

E-> E+E |E-> E-E | E-> E*E | E-> (E) | E-> id

Is below grammar operator grammar?

S → aSbS | bSaS| ε

- No, it is not an operator Grammar because it does not satisfy 1st property of operator Grammar.
- It contains null on R.H.S of production

  i.e. S → ε

To convert this grammar to operator precedence grammar we can substitute null in the RHS of S production. The operator grammar is:

S-> aSbS | bSaS | aSb | abS | ab | bSa | baS | ba

Main Challenges in the operator precedence parsing

1. Identification of correct handles in the reduction step, such that the given input should be reduced to starting symbol of the grammar.

2. Identification of which production to use for reducing in the reduction steps, such that we should correctly reduce the given input to the starting symbol of the grammar.

Operator Precedence Relations

Relation	Meaning
a <. b	a has lower precedence than terminal b
a .> b	a has higher precedence than terminal b
a = b	a has equal precedence as terminal b

Association and Precedence Rules

- If operators have different precedence like ‘*’ has more precedence than ‘-’ operator.

Example: a+b*c;

+ <. *

- If operators have same precedence like ‘+’ and ‘-’ then apply association rule (left to right or right to left).

Example: a+b-c;

+ .> -

a^b^c;

^(b^c) .> ^ (b^c executed 1st)

Note:

- Identifiers have higher precedence than all other operators and symbols.

+ <. id	$ <. Id	id . > *
id . > $	id . >)	(<. id

- S has lower precedence than all other operators and symbols.

$ <. (	id . > $	$ <. +
). > $	$ <.*	

Components of operator precedence parser

- An input buffer that contains string to be parsed followed by a $, a symbol used to indicate the ending of input.
- A stack containing a sequence of grammar symbols with a $ at the bottom of the stack.
- An operator precedence relation table O, containing the precedence relations between the pair of terminal.
- An operator precedence algorithm.

Leading(A)

If A-> αaβ, α is single variable or ε

Leading(A) = {a}

If A-> Ba β

Leading(A) = Leading(B)

Example

S -> a|^|(T)

T -> T,S | S

Leading(S) = {a, ^, (}

Leading(T) = {, Leading(S)}

= { , a, ^,(}

Trailing(A)

If A-> αaβ, β is single variable or ε

Trailing(A) = {a}

If A-> αaB

Trailing(A) = Trailing(B)

Example

S -> a|^|(T)

T -> T,S | S

Trailing(S) = {a, ^,)}

Trailing(T) = {, Trailing(S)}

= { , , a, ^,)}

Operator Precedence Relation Table

- Compute Leading and Trailing.

1. Set $ <. Leading(S) and Trailing(S) .> $, where S is the start symbol.
2. A -> X1X2….Xn
   a. If Xi & Xi+1 Є T, then Xi = Xi+1
   b. If Xi & Xi+2 Є T and Xi+1 Є N, then Xi = Xi+2
   c. If Xi Є T and Xi+1 Є N, then Xi <. Leading(Xi+1)
   d. If Xi Є N, Xi+1 Є T, then Trailing(Xi) > Xi+1

Here T stands for Terminal, and N stands for Non-Terminal.

Example

S -> a|^|(T)

T -> T,S | S

Leading(S) = {a, ^, (}

Leading(T) = {, Leading(S)}

= { , a, ^,(}

Trailing(S) = {a, ^,)}

Trailing(T) = {, Trailing(S)}

= { , , a, ^,)}

Relation Table Entries

According to rule 1

- $ <. Leading(S), So $ <. {a, ^, (}
- Trailing(S) .> $, So {a, ^,) .> $

According to rule 2(b)

- S -> (T), so both parenthesis equal precedence (=)

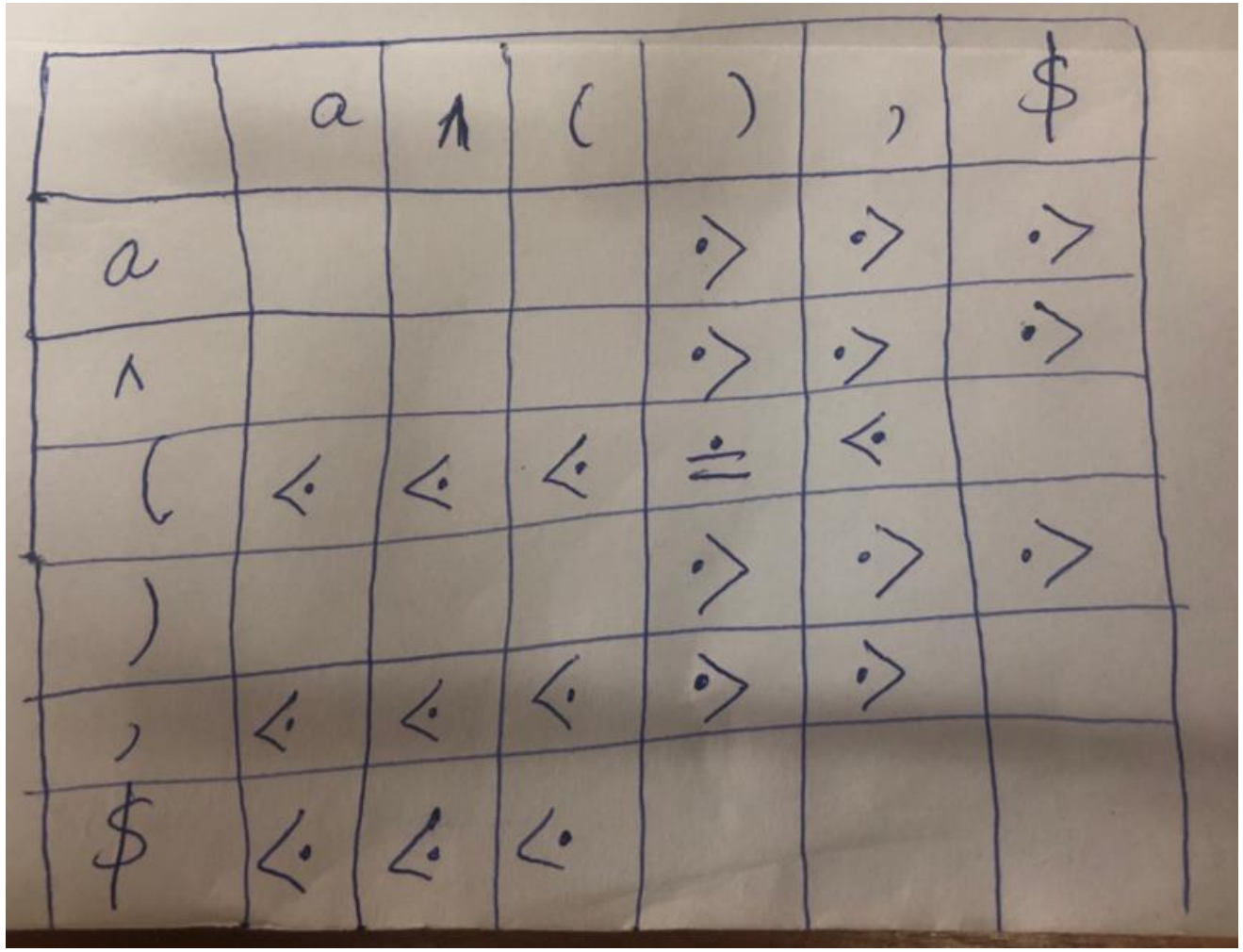
According to rule 2(c)

- S-> (T), So (<. {, ,a, ^, (}
- T-> T,S So , <. {a, ^, (}

According to rule 2(d)

- S->(T) So {, , a, ^,)} .> ,
- T->T,S So {, , a, ^,)} .> ,

Operator Precedence Relation Table



Operator Parsing Algorithm

- The operator precedence Parser parsing program determines the action of the parser depending on
    - ‘a’ is top most symbol on the Stack
    - ‘b’ is the current input symbol
- There are 3 conditions for ‘a’ and ‘b’ that are important for the parsing program

    1. a=b=$, the parsing is successful

    2. a <• b or a = b, the parser shifts the input symbol on to the stack and advances the input pointer to the next input symbol.

    3. a •> b, parser performs the reduce action. The parser pops out elements one by one from the stack until we find the current top of the stack element has lower precedence than the most recently popped out terminal

Parsing for input ‘(a, a)’

Stack	Input	Operations
$	(a, a)$	$ <• (, shift ‘(’ in to stack
$(	a, a)$	(<• a, shift ‘a’ in to stack
$(a	, a)$	a •> , pop a and reduce by S->a and reduce T->S
$(S	, a)$	
$(T	, a)$	(<• , shift ‘,’ in to stack
$(T,	a)$	, <• a, shift ‘a’ in to stack
$(T, a	)$	a •>), pop ‘a’ and reduce by S->a
$(T, S	)$	, •>), pop ‘T,S’ and reduce by T->T,S
$(T	)$	(=) shift ‘)’ in to stack
$(T)	$	) •> $, pop (T) and reduce by S-> (T)
$S	$	$=$, so parsing is successful

Example

G: E -> E+E

E -> E-E

E -> E*E

E -> E/E

E -> E^E

E -> -E

E -> (E)

E -> id

Operator Precedence Relations

	+	-	*	/	^	Id	(	)	$
+	•>	•>	<•	<•	<•	<•	<•	•>	•>
-	•>	•>	<•	<•	<•	<•	<•	•>	•>
*	•>	•>	•>	•>	<•	<•	<•	•>	•>
/	•>	•>	•>	•>	<•	<•	<•	•>	•>
^	•>	•>	•>	•>	<•	<•	<•	•>	•>
id	•>	•>	•>	•>	•>			•>	•>
(	<•	<•	<•	<•	<•	<•	<•	=	
)	•>	•>	•>	•>	•>			•>	•>
$	<•	<•	<•	<•	<•	<•	<•	<•	<•

Parsing for input ‘id*id’

Stack	Input	Operations
$	Id*id$	$ <• id, shift ‘id’ in to stack
$id	*id$	id •> *, reduce ‘id’ using E-> id
$E	*id$	$ <• *, shift ‘*’ in to stack
$E*	Id$	* <• id , shift ‘id’ in to Stack
$E*id	$	id •> $, reduce ‘id’ using E->id
$E*E	$	*•> $, reduce ‘*’ using E->E*E
$E	$	$=$=$, so parsing is successful

Summary

- It is simple and easy to implement parsing technique.
- The operator precedence parser can be constructed by hand after understanding the grammar. It is simple to debug.
- It is difficult to handle the operator like ‘-’which can be either unary or binary and hence different precedence’s and associativity's.
- It can parse only a small class of grammar.
- New addition or deletion of the rules requires the parser to be re-written.