Operator Precedence Parser

Date :- 17/10/2022

Shweta Tiwari

* Two Restriction for Grammar

(1) no null production (ε).

eg → A → ε

(2) No two adjacent non-terminal in its RHS of production.

eg → A → ACB

* If any grammar/production is not in operator precedence grammar then convert into operator Grammar.

Question - check a grammar/production are in operator precedence Grammar or not.

(1) T → T*T/T+T/id

so, there is no ∈ production and no two adjacent non-terminal in RHS of production

(2) S → SAS/a

A → bSb/a

In S → SAS is not a operator grammar so, convert it.

$S \rightarrow SbSbS/a/SaS$

Now, $A \rightarrow bSb/a$

All grammar are follow the opertar Precedence grammar rules..

Now, for parse a string we need a parsing construction Table.

So, we construct a passing table.

Here, Bulid a Relation Table for OPP.

There are some rules follow

check precendence but if same then left associaively.

① $\theta_1 > \theta_2$; then $\theta_1 \gtrdot \theta_2$

② $\theta_1 < \theta_2$; then $\theta_1 \lessdot \theta_2$

③ $\theta_1 = \theta_1$; then $\theta_1 \gtrdot \theta_1$

$\theta_2 = \theta_2$; then $\theta_2 \gtrdot \theta_2$

$\theta_1 = \theta_2$; then $\theta_1 \gtrdot \theta_2$

(Left Associativity).

* θ_1, θ_2 are mathematical opratar as it can be a terminal. And

* For checking the string (w) is accepted or not.

There are two operation perform.

(1) If [stack <. Input Buffer] then push(item). into stack and increase pointer.

(2) If [stack .> Input Buffer] then pop(item) from stack.

L \ R	id	+	*	$
id	—	⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	Accept

Operator Relation Table for OP parser.

- In, Table all terminals are in row and col ; And $ is also append.

Question → Construction parsing table of relation given grammar and check string w (id+id*id) is accpeted or not

- T → T*T/T*T/id.

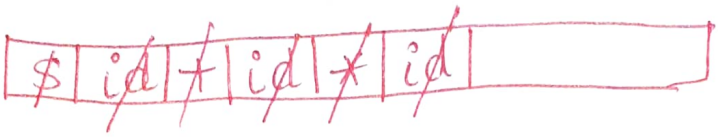
The relation table is given above.

Now, check w = id+id*id

* append $ input buffer and $ is inially add in stack

id + id * id $

Input Buffer

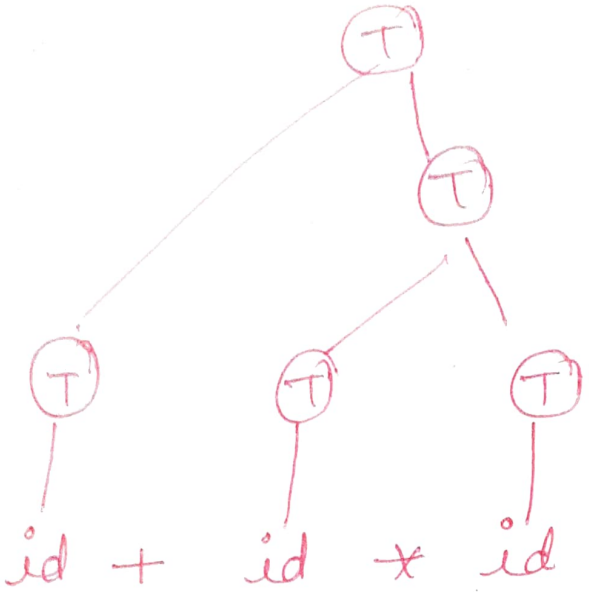


Stack

Operations by using OPP relation table.

(1) $ <· id push
(2) id ·> + pop (T → id)
(3) $ <· + push
(4) + <· id push
(5) id ·> * pop (T → id)
(6) + <· * push
(7) * <· id push
(8) id ·> $ pop (T → id)
(9) * ·> $ pop (T → T*T)
(10) + ·> $ pop (T → T+T)
(11) $ — $ Accepted

Now, make parse tree.



Parse Tree

Now, Relation table size is $4 \times 4 = 16$.
if there is n terminal, then size is $(n \times n) = (n)^2$. Complexity is $O(n^2)$.

- So, we need to reduce the size of table

The concept come of fenction

XX we apply on above question

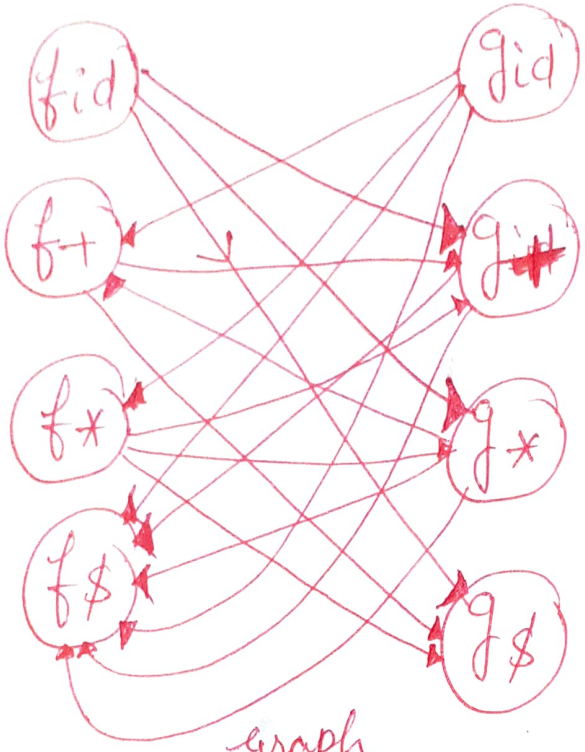
f \ g	id	+	*	$
id	→	⋱	⋱	⋱
+	⋰	⋱	⋰	⋱
*	⋰	⋱	→	⋱
$	⋰	⋰	⋰	–

$(2 \times n)$ size

$O(2n)$ complinity.

Make a Graph using relation table.

f Domain g Domain



graph

Insure no cycle form

OP Relation Graph

Now, find the longest path By using graph

Path

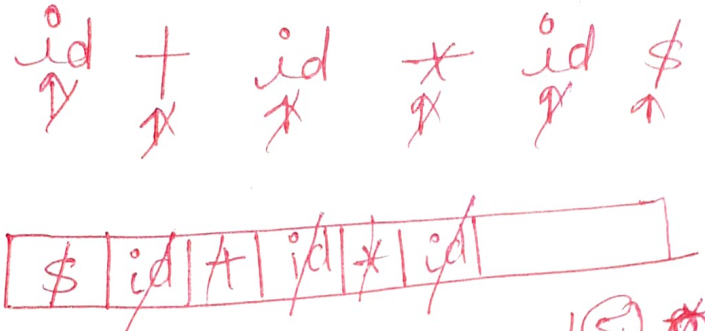
	id	+	*	$
f	4	2	4	0
g	5	1	3	0

OP. Function relation table

w = id + id * id is accepted by function table as not

So, Grammar is already above solved.

operations by using OP relation fuction table.



① $ id push

f$ < g id

0 5

② id + pop (T → id)

fid > g+

4 1

③ $ < + push

f$ g+

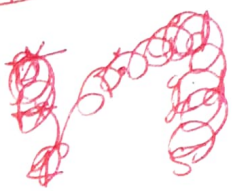
0 1

④ + < id push

f+ gid

2 5

⑤



id > * pop

fid g* (T → id)

4 3

⑥ + < * push

f+ g*

2 3

⑦ * < id push

f* gid

4 5

⑧ id > $ pop

fid g$ (T → id)

4 0

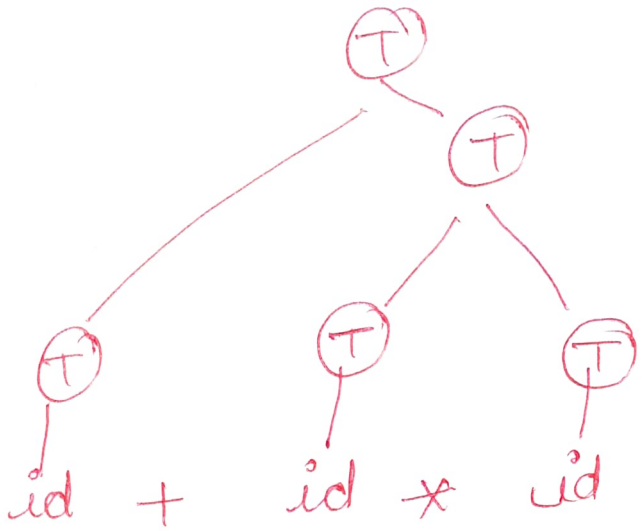
⑨ * > $ pop

f* g$ (T → T * T)

4 0

10. + > $ pop
+ $ (T → T+T)
2 0

(11)
$ $
$ — $ Accepted
0 0



parser tree

The end

Shweta Tiwari