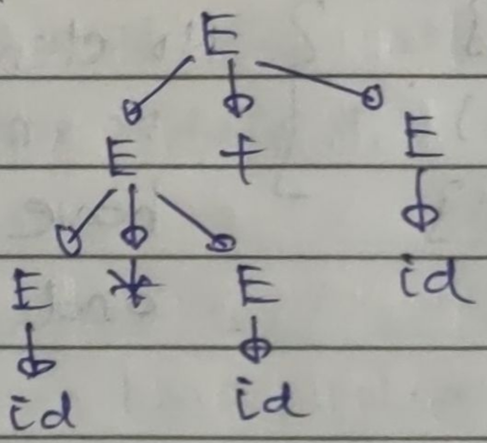
Ambiguous to Unambiguous

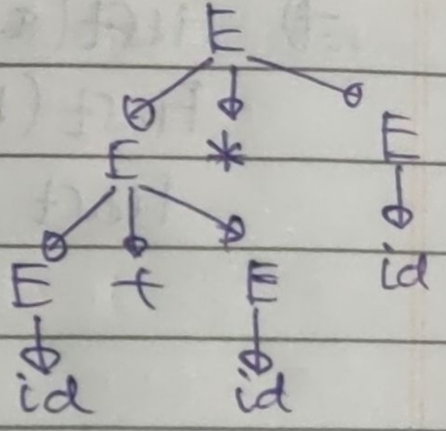
- Ambiguity will be removed by precedence and associativity rules.

(a) If different operators are present, then the operator with higher precedence must be present below the lower precedence.

{/,*} > {+,-}
^
{^}



(✓)

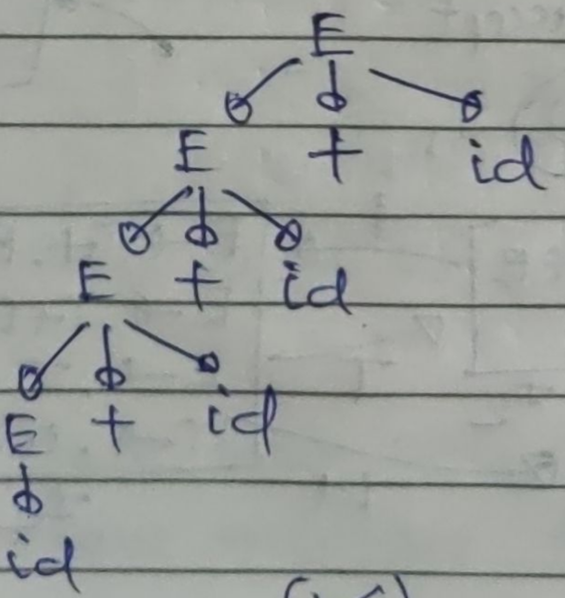


(×)

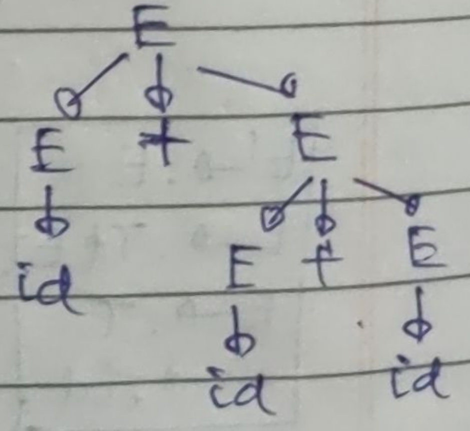
(b) If same (precedence) operators are involved, then:

(i) for left to right associative operators, left recursion will be preferred {+, -, /, *}

(ii) For right to left associative operators, right recursion will be preferred. {^}



(✓)



(×)

Ex :-

(a) E → E-E | id

⇒ Only a left associative present; left recursion will be preferred :-

E → E-P | P
P → id

OR

E → E-id | id

(b) E → E^E | id

⇒ Right recursion will be preferred :-

E → P^E | P
P → id

OR

E → id^E | id

(c) E → E+E | E*E | id

⇒ Precedence(*) > Precedence(+) ⇒ all the '+' operations be completed before '*' operations.

⇒ Also left-recursion will be favoured for multiple '+' and '*' operations :-

E → E+T | T
T → T*F | F
F → id

Ans

(d) E → E-E | E*E | E^E | id

⇒ '-' will be above '*' and '*' will be above '^' in the parse tree and hence :-

E → E-T|T
T → T*F|F } → left recursion for {'-', '*'}
F → P^F|P → right recursion for "^"
P → id

Note :- Take care that language generated by grammar mustn't change on removing ambiguity.

(d) bexp → bexp or bexp | bexp and bexp | not bexp | true | false

⇒ bexp → bexp or P | P
P → P and Q | Q
Q → not Q | true | false

where "or", "and" → left associative and precedence = {"not" > "and" > "or"}.

(e) R → R+R | R.R | R* | a|b

⇒ Precedence = * > . > +

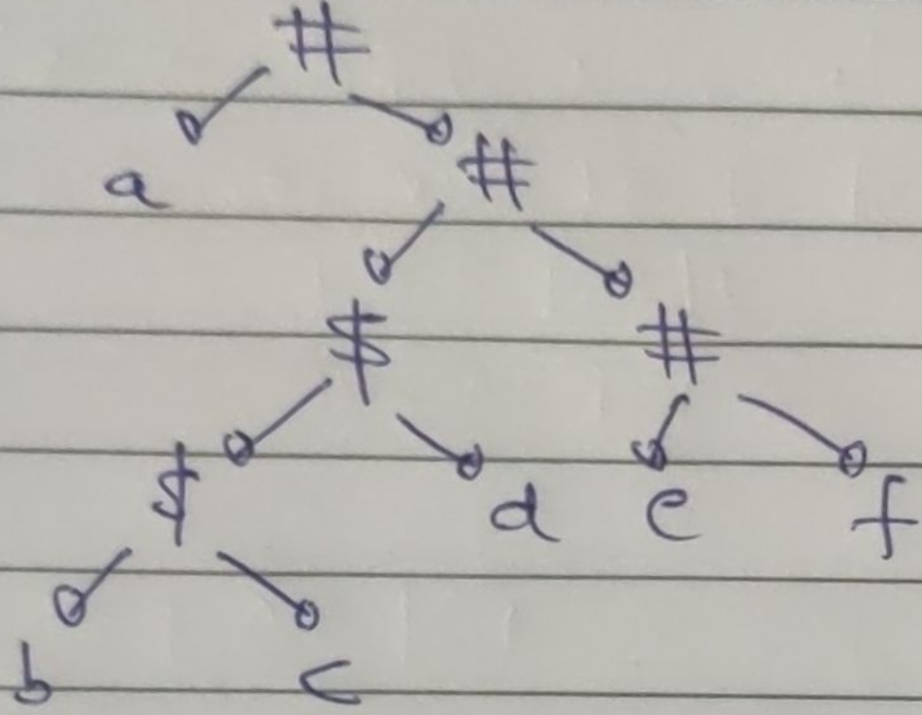
⇒ '+' and '.' are left associative.

R → R+P|P
P → P.Q|Q
Q → Q*|a|b

Ans

Example:- Consider string: "a#bcd#e#f"

Parse tree will be:



Which tells that:

(a) Precedence($) > Precedence(#)

(b) $ → left associative

(c) # → right -"-.