Methods To Remove Ambiguity

The ambiguity from the grammar may be removed using the following methods-

1. By adding the precedence rules or other context sensitive parsing rules
2. By fixing the grammar
3. By adding grouping rules.
4. By using semantics and choosing the parse that makes the most sense.

Example-1 [By precedence and associativity constraints.]

Given grammar is G :

E→E+E | E*E | (E) | id | id( )

i) Check whether the given grammar is ambiguous or unambiguous.

ii) If ambiguous , convert into an unambiguous grammar G', so that L(G')= L(G)

iii) Derive the string ***id+id*(id+id*id())*** *using LMD and RMD*, show the derivation trees.

Answer:

(i) ***Step-1 : Check Grammar Ambiguity***

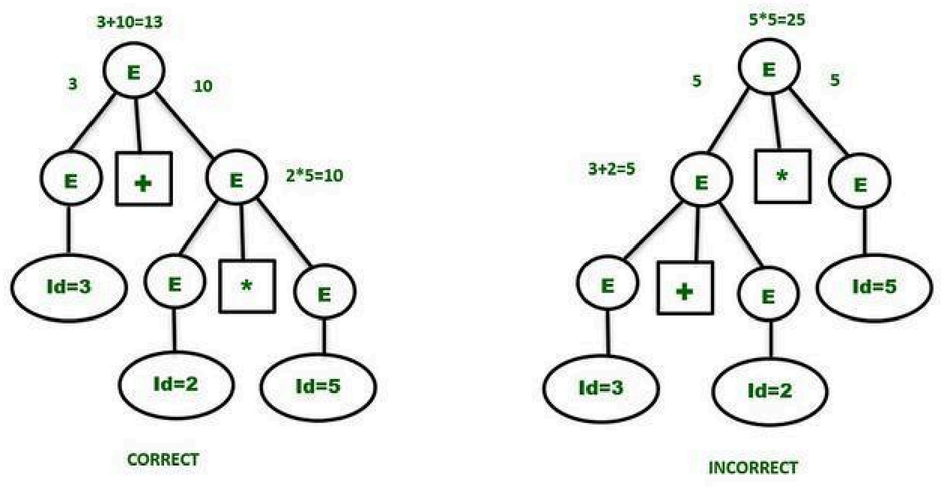
To check whether a given grammar is ambiguous or not, we need to construct the parse tree or derivation tree for any string.

If there exists at least one such string, then the grammar is ambiguous otherwise unambiguous. Let consider the string "***id+id*id***", the possible derivations and corresponding parse trees for the string "***id+id*id***" are as follows:

Derivation-1 (LMD)	Derivation-2 (LMD)
E→E+E →id+E →id+E * E →id+id * E →id+id* id	E→E*E →E+E*E →id+E * E →id+id * E →id+id* id
Derivation-3 (RMD)	**Derivation-4 (RMD)**
E→E+E →E+E*E →E+E * id	E→E*E →E*id →E+E * id

→E+id * id →id+id* id	→E+id * id →id+id* id

Consider the expression : ***3 + 2 * 5*** [“*” has more priority than “+”]
The correct answer is : ***(3+(2*5))=13***



The “+” having the least priority has to be at the upper level and has to wait for the result produced by the “*” operator which is at the lower level. So, the first parse tree is the correct one and gives the same result as expected.

(ii) ***Step-2 : Converting Ambiguous into Unambiguous Grammar***

The unambiguous grammar will contain the productions having the highest priority operator (“*” in the example) at the lower level and vice versa.

The associativity of both the operators are Left to Right. So, the unambiguous grammar has to be left recursive. The grammar will be :

E -> E + P [+ is at higher level and left associative]
E -> P
P -> P * Q [* is at lower level and left associative]
P -> Q
Q -> (Q)
Q -> G
G -> id
G -> id( )

While converting an ambiguous grammar to an unambiguous grammar, the original language shouldn't change.
The non-terminals in the ambiguous grammar are replaced with other variables to get the same language as it was derived before and also maintain the precedence and associativity rule simultaneously.

After replacing the production **(E -> P and P -> Q and Q -> id)** in the above example, because the language contains the strings **{id+id*id}** as well.

(iii) **Derive the string *id+id*(id+id*id())***

Example-2 [By precedence and associativity constraints]

Convert the following ambiguous grammar to unambiguous grammar-
R → R + R / R . R / R* / a / b
where * is kleene closure and . is concatenation.

Answer:
To convert the given grammar into its corresponding unambiguous grammar, we implement the precedence and associativity constraints.

Given grammar consists of
V= {R}
T= {+, *, ., a, b}
Operators = { + , . , * }
Operands = {a, b}

The priority order is- (a , b) > * > . > +
Where:
. operator is left associative
+ operator is left associative

Using the precedence and associativity rules,
the corresponding unambiguous grammar as-
E → E + T | T

T → T . F | F
F → F* | G
G → a | b

Example-3 [By precedence and associativity constraints.]

Convert the following ambiguous grammar into unambiguous grammar-
X → X or X | X and X | not X | T| F
where X represents Boolean expression, T represents **TRUE** and F represents **FALSE**.

Answer:

To convert the given grammar into its corresponding unambiguous grammar, we implement the precedence and associativity constraints.
Given grammar consists of
V={X}
T={ or, and, not, T, F}
Operators={or, and, not}
Operands={T, F}

The priority order is: (T , F) > not > and > or, where
and operator is left associative
or operator is left associative

Using the precedence and associativity rules,
write the corresponding unambiguous grammar as-

X → X or M | M
M → M and N | N
N → not N | G
G → T | F

Example-4 [By fixing the grammar]

Show that following grammar is ambiguous and convert into equivalent into unambiguous.
S→A | B
A→aAb | ab
B→abB | λ

Answer:

$\boxed{\begin{array}{l} S \to A \\ A \to aAb \mid ab \end{array}}$

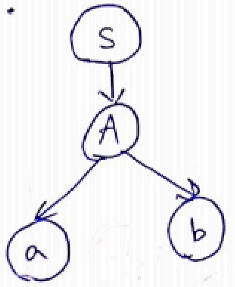
Language $= \{ab, a^2b^2, a^3b^3 \ldots\}$
$= \{a^nb^n : n \geq 1\}$

$\boxed{\begin{array}{l} S \to B \\ B \to abB \mid \lambda \end{array}}$

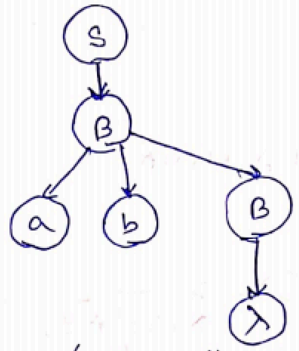
Language $= \{\lambda, ab, abab, ababab\}$
$= \{(ab)^n : n \geq 0\}$

Final language $= \{a^nb^n : n \geq 1\} \cup \{(ab)^n : n \geq 0\}$

For string "ab", there exists 2 distinct derivation tree.



(Derivation Tree - 1)



(Derivation Tree - 2)

Hence, the given grammar is ambiguous.

If we modify 'ab' to 'aabb' in A production, then language will not be changed. and 'ab' is also derived using exactly 1 derivation tree.

Hence, equivalent unambiguous grammar !

$\boxed{\begin{array}{l} S \to A \mid B \\ A \to aAb \mid aabb \\ B \to abB \mid \lambda \end{array}}$

— Ans