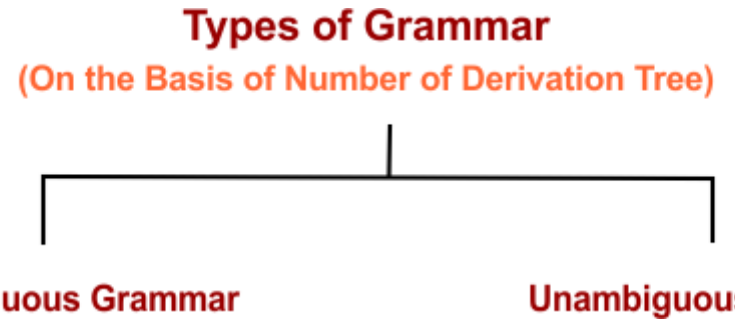
Ambiguity in CFG



1. Ambiguous Grammar

A grammar is said to be ambiguous grammar if any string generated by it produces more than one Parse tree Or syntax tree Or leftmost derivation Or rightmost derivation.

Examples of Ambiguous Grammar

Example 01

Check whether the following grammar is ambiguous or not for string w = ab

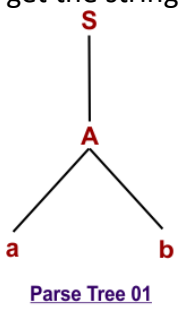
S → A / B

A → aAb / ab

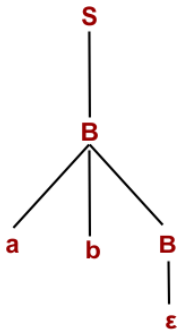
B → abB / ∈

Solution

Now, we draw more than one parse tree to get the string w = ab.



Parse Tree 01



Parse Tree 02

The original string (w =ab) can derived through two different parse trees. So, the given grammar is ambiguous.

Example-02

Check whether the following grammar is ambiguous or not for string w = aabbccdd

S → AB / C

A → aAb / ab

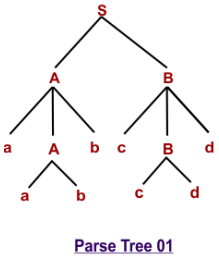
B → cBd / cd

C → aCd / aDd

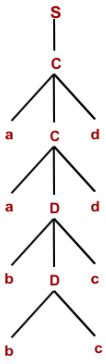
D → bDc / bc

Solution

Now we draw more than one parse tree to get string w = aabbccdd.



Parse Tree 01



Parse Tree 02

The original string (w =aabbccdd) can derived through two different parse trees. So, the given grammar is ambiguous.

2. Unambiguous Grammar

A grammar is said to be unambiguous grammar if every string generated by it produces exactly one Parse tree Or syntax tree Or leftmost derivation Or rightmost derivation.

Note: So, If we try to derive more than one tree of unambiguous grammar, then all trees will be similar.

Examples of Unambiguous Grammar

Example 01

For string "aab" the following grammar is unambiguous

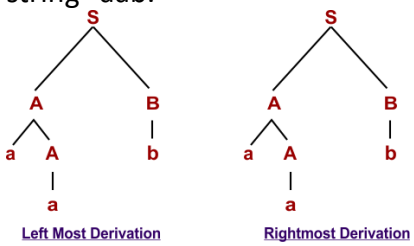
S → AB

A → Aa | a

B → b

Solution

Let's draw the leftmost and rightmost derivations of the above grammar to get the string "aab."



Because all parse trees, syntax trees, and left or right derivations will be similar for the above grammar of string "aab." So, the above grammar is unambiguous.

Example 02

For string "id+id*id," the following grammar is unambiguous

E → E + T

E → T

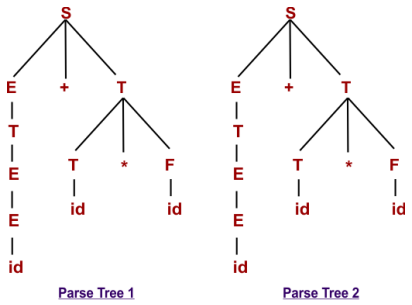
T → T * F

T → F

F → id

Solution

Because all parse trees, syntax trees, and left or right derivations will be similar for the above string grammar "id+id*id." As given below



❑ General Approach To Check Grammar Ambiguity-

To check whether a given grammar is ambiguous or not, we follow the following steps-

Step-01:

We try finding a string from the Language of Grammar such that for the string there exists more than one-

- parse tree
- or derivation tree
- or syntax tree
- or leftmost derivation
- or rightmost derivation

Step-02:

If there exists at least one such string, then the grammar is ambiguous otherwise unambiguous.

❖ **PROBLEMS BASED ON CHECKING WHETHER GRAMMAR IS AMBIGUOUS-**

Problem-01: Check whether the given grammar is ambiguous or not-

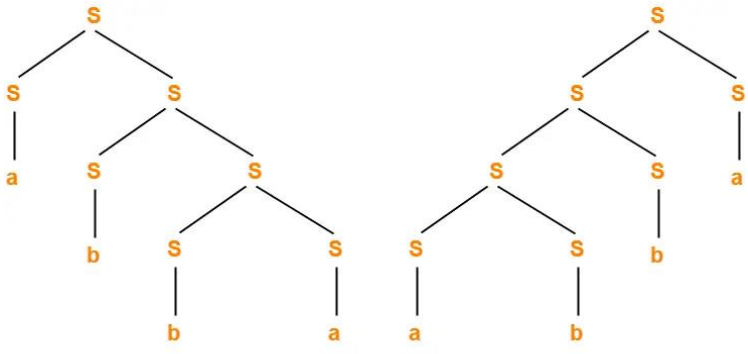
S → SS
S → a
S → b

Solution-

Let us consider a string w generated by the given grammar-

w = abba

Now, let us draw parse trees for this string w.



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-02: Check whether the given grammar is ambiguous or not-

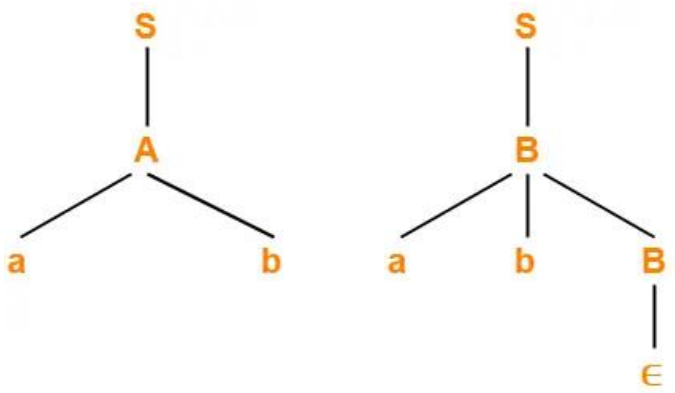
S → A / B
A → aAb / ab
B → abB / ∈

Solution-

Let us consider a string w generated by the given grammar-

w = ab

Now, let us draw parse trees for this string w.



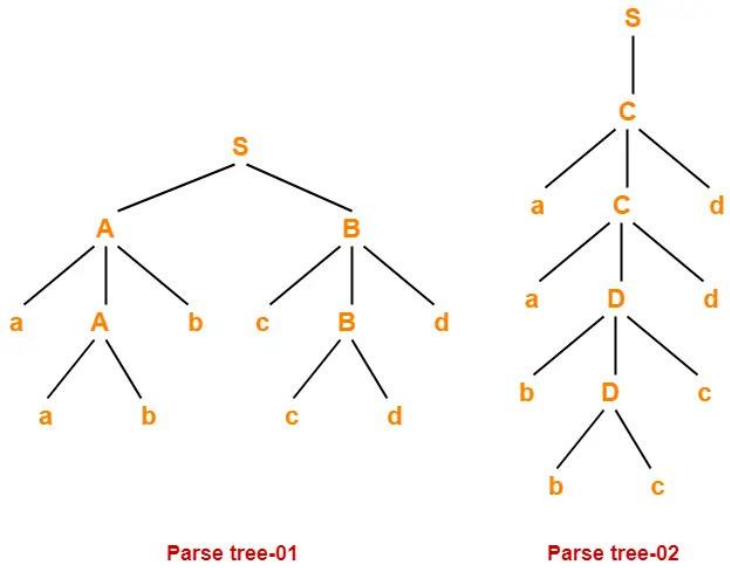
Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-03: Check whether the given grammar is ambiguous or not-

S → AB / C
A → aAb / ab
B → cBd / cd
C → aCd / aDd
D → bDc / bc

Solution-
Let us consider a string w generated by the given grammar-
w = aabbccdd
Now, let us draw parse trees for this string w.



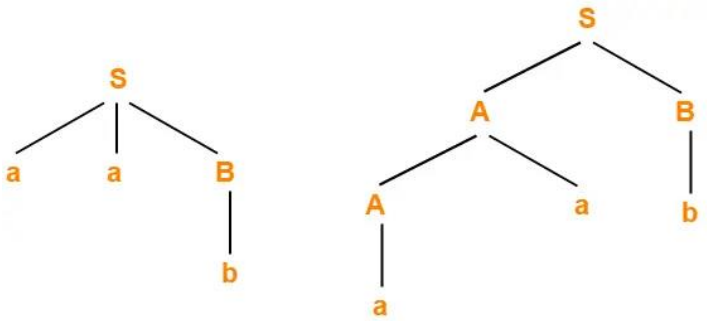
Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-04:
Check whether the given grammar is ambiguous or not-

S → AB / aaB
A → a / Aa
B → b

Solution-
Let us consider a string w generated by the given grammar-
w = aab
Now, let us draw parse trees for this string w.



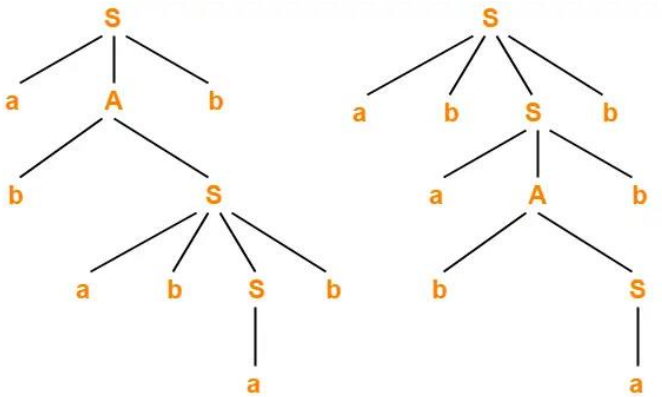
Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-05:
Check whether the given grammar is ambiguous or not-

S → a / abSb / aAb
A → bS / aAAb

Solution-
Let us consider a string w generated by the given grammar-
w = abababb
Now, let us draw parse trees for this string w.



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-06:

Check whether the given grammar is ambiguous or not-

E → E + T / T

T → T x F / F

F → id

Solution-

There exists no string belonging to the language of grammar which has more than one parse tree.

Since a unique parse tree exists for all the strings, therefore the given grammar is unambiguous.

Problem-07:

Check whether the given grammar is ambiguous or not-

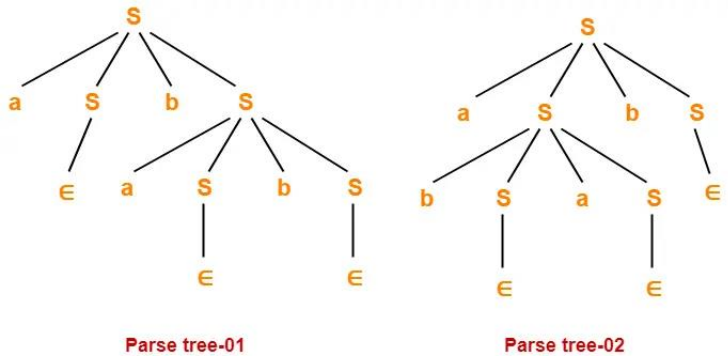
S → aSbS / bSaS / ∈

Solution-

Let us consider a string w generated by the given grammar-

w = abab

Now, let us draw parse trees for this string w.



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem-08:

Check whether the given grammar is ambiguous or not-

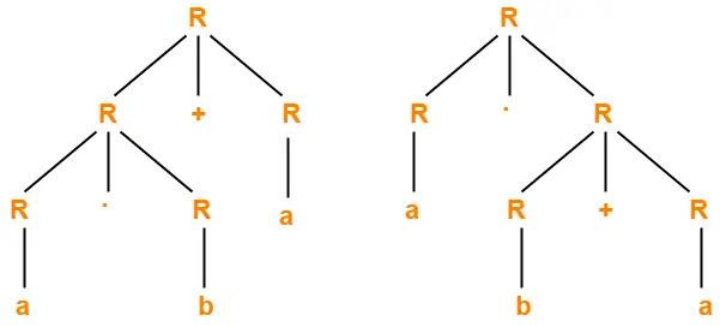
R → R + R / R . R / R* / a / b

Solution-

Let us consider a string w generated by the given grammar-

w = ab + a

Now, let us draw parse trees for this string w.



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

❑ Ambiguity and Limitations of CFG

One big challenge with using CFGs in compilers is ambiguity. Ambiguity happens when the same piece of code can be interpreted in more than one way, which makes it unclear what the actual structure should be.

Statement → if Expression then Statement | if Expression then Statement else Statement

Figure 2: A common example is the "if-then-else" problem.

If you have two if statements followed by an else, the compiler might get confused about which if the else is supposed to match with. This makes it hard to figure out the intended structure of the code.

To deal with this, compiler designers use **disambiguation rules**. A typical rule is that an else clause always pairs with the closest if statement. Another way to resolve ambiguity is using advanced parsing techniques, like operator precedence, to handle tricky cases and make sure things are interpreted correctly.

Another limitation of CFGs is that they can't handle everything in modern programming languages. Some languages have context-sensitive rules, meaning that how certain parts of the code should be understood depends on the surrounding context. Since CFGs are context-*free*, they can't directly handle those cases. To get around this, compilers use additional methods like attribute grammars or extra checks during the semantic analysis phase.

Another limitation of CFG is that it cannot handle all the features of modern programming languages. Some languages have **context-sensitive** rules, which require the meaning of a part of the code to depend on the context in which it appears. Since CFG is **context-free**, it cannot handle such cases directly. To address this, compilers often use **attribute grammars** or perform additional checks during semantic analysis (*6.0 Semantic Analysis Translation and Attribute Grammars*, 2024).

3. Removal of Ambiguity from CFG

Removing Ambiguity By Precedence & Associativity Rules-

An ambiguous grammar may be converted into an unambiguous grammar by implementing-

- Precedence Constraints
- Associativity Constraints

Associative:

Left associative: *,/ > . > +,-

Right associative: ↑

Precedence:

@ > # > $

PROBLEMS BASED ON CONVERSION INTO UNAMBIGUOUS GRAMMAR-

Problem-01:

Convert the following ambiguous grammar into unambiguous grammar-

R → R + R

R → R . R

R → R*

R → a | b

Solution-

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

R → R + T | T

T → T . F | F

F → F* | G

G → a | b

Problem-02:

Convert the following ambiguous grammar into unambiguous grammar-

bexp → bexp or bexp

bexp → bexp and bexp

bexp → not bexp

bexp → T | F

where bexp represents Boolean expression, T represents True and F represents False.

Solution-

bexp → bexp or M / M

M → M and N / N

N → not N / G

G → T / F

Problem-03:

Convert the following ambiguous grammar into unambiguous grammar-

E → E+E
E → E *E
E → id

Solution-

Equivalent Unambiguous Grammar
E → E+T | T
T → T*F | F
F → id

Problem-04:

Convert the following ambiguous grammar into unambiguous grammar-

E→E+E
E→E *E
E→E ^ E
E→ id

Solution-

Equivalent Unambiguous Grammar
E → E + T | T
F → G ^ F |G
G→ id