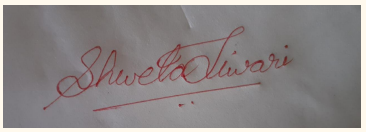
CD: COMPILER DESIGN
CD: UNIT-2 09/2022

SEPTEMBER 2022 / IT-3rd year, Vth semester
FALL SEMESTER, YEAR (Vth, 3rd)
FALL SESSION (2022-23)
(CD)
MS. SHWETA TIWARI
Published: SEPTEMBER, 2022

PREPARED FOR
Engineering Students
All Engineering College



INSTRUCTOR: Ms. SHWETA TIWARI
shwetatiwari08@recabn.ac.in
shwetatiwari08aug@gmail.com

CD: COMPILER DESIGN

TOPIC On : Removing Ambiguity

Ambiguous to Unambiguous

—

By SHWETA TIWARI

Under On: Unit-2

TOPIC On : Removing Ambiguity
Ambiguous to Unambiguous

Grammar Ambiguity

We will discuss how to convert ambiguous grammar to unambiguous grammar.

Converting Ambiguous Grammar To Unambiguous Grammar-

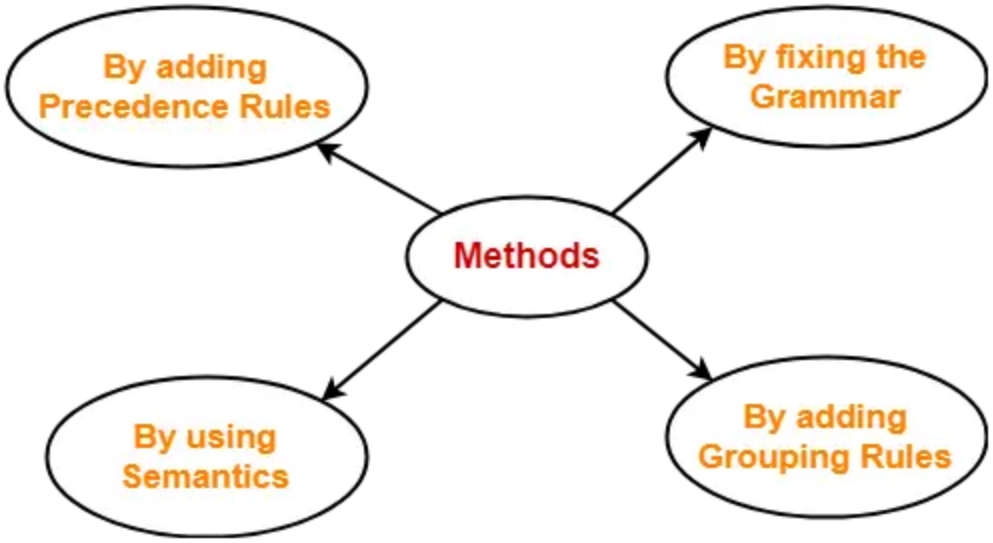
- Causes such as left recursion, common prefixes etc makes the grammar ambiguous.
- The removal of these causes may convert the grammar into unambiguous grammar.
- However, it is not always compulsory.

NOTE

It is not always possible to convert an ambiguous grammar into an unambiguous grammar.

Methods To Remove Ambiguity-

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



- By fixing the grammar
- By adding grouping rules
- By using semantics and choosing the parse that makes the most sense
- By adding the precedence rules or other context sensitive parsing rules

Removing Ambiguity By Precedence & Associativity Rules-

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

- Precedence Constraints
- Associativity Constraints

These constraints are implemented using the following rules-

Rule-01:

The precedence constraint is implemented using the following rules-

- The level at which the production is present defines the priority of the operator contained in it.

- The higher the level of the production, the lower the priority of the operator.
- The lower the level of the production, the higher the priority of the operator.

Rule-02:

The associativity constraint is implemented using the following rules-

- If the operator is left associative, induce left recursion in its production.
- If the operator is right associative, induce right recursion in its production.

PROBLEMS BASED ON CONVERSION INTO UNAMBIGUOUS GRAMMAR-

Problem-01:

Convert the following ambiguous grammar to unambiguous grammar-

$$R \rightarrow R + R \mid R \,.\, R \mid R^* \mid a \mid b$$

where * is kleene closure and . is concatenation.

Solution-

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

We have-

- Given grammar consists of the following operators-

$$+ \,,\, . \,,\, *$$

- Given grammar consists of the following operands-

a , b

The priority order is-

$$(\mathbf{a}\ ,\ \mathbf{b}) > * > . > +$$

where-

- **.** operator is left associative
- + operator is left associative

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

$$E \rightarrow E + T \ / \ T$$

$$T \rightarrow T\ .\ F \ / \ F$$

$$F \rightarrow F^* \ / \ G$$

$$G \rightarrow a \ / \ b$$

Unambiguous Grammar

OR

$$E \rightarrow E + T \ / \ T$$

$$T \rightarrow T\ .\ F \ / \ F$$

$$F \rightarrow F^* \ / \ a \ / \ b$$

Unambiguous Grammar

Problem-02:

Convert the following ambiguous grammar to unambiguous grammar-

bexp → bexp or bexp / bexp and bexp / not bexp / T / F

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

Solution-

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

We have-

- Given grammar consists of the following operators-

or , and , not

- Given grammar consists of the following 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, we write the corresponding unambiguous grammar as-

bexp → bexp or M / M

M → M and N / N

N → not N / G

G → T / F

Unambiguous Grammar

OR

bexp → bexp or M / M

M → M and N / N

$$N \rightarrow \text{not } N \mid T \mid F$$

Unambiguous Grammar