SE2052: PROGRAMMING PARADIGMS

BNF AND EBNF

BY JEEWAKA PERERA

Updated on: 11/2/2025

WHY STUDY GRAMMAR IN PROGRAMMING?

- Defines valid syntax for programming languages.
- Used in compilers, interpreters, and parsers.
- Helps prevent syntax errors.

Think of grammar in programming like rules in natural languages.

WHAT IS SYNTAX IN PROGRAMMING?

- Syntax defines the structure of a programming language.
- Example: Grammar rules ensure valid statements.
- Think of it like grammar in natural languages.
- Syntax of a programming language is the form of its expressions, statements, and program units.
- Semantics is the meaning of those expressions, statements, and program units.

TYPES OF GRAMMARS (CHOMSKY HIERARCHY)

1. **Type 0**: Unrestricted Grammar
2. **Type 1**: Context-Sensitive Grammar
3. **Type 2**: Context-Free Grammar (CFG)
4. **Type 3**: Regular Grammar

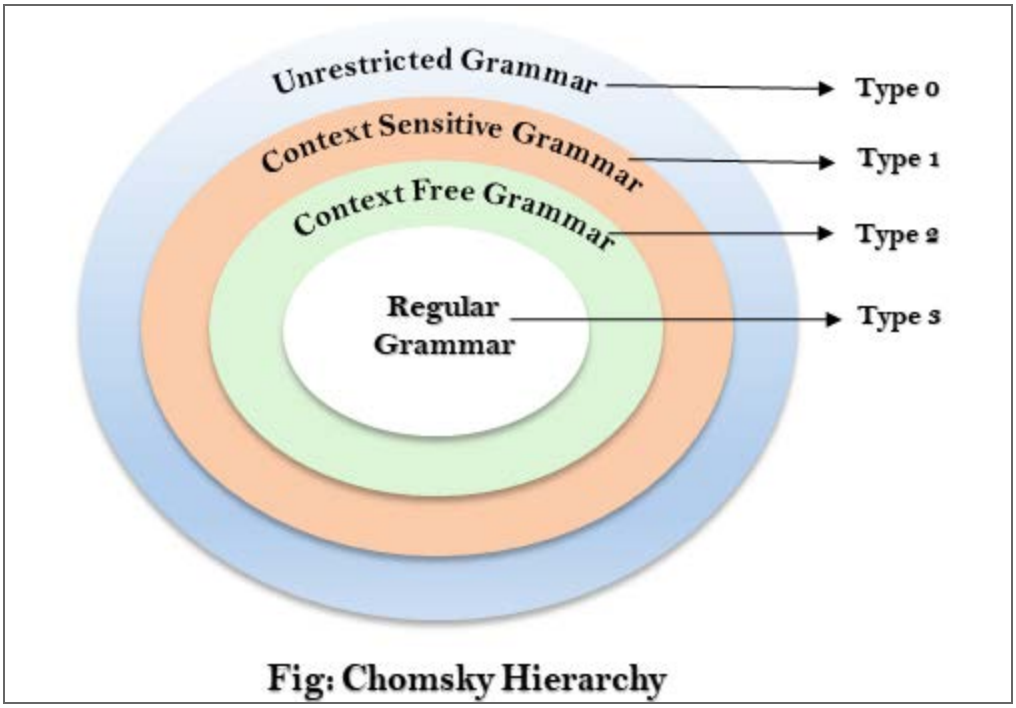


Fig: Chomsky Hierarchy

EXAMPLES OF DIFFERENT GRAMMAR TYPES

Regular Grammar (Type 3):

```
S → aS | bS | ε
```

Used for Analysis (Tokens in Regex).

Context-Free Grammar (Type 2):

```
S → S + S | S * S | (S) | number
```

Used for Syntax Analysis (Parsing).

DARK AGE BEFORE BNF

- Before BNF, programming language syntax was described informally.
- Each compiler was built manually without a structured grammar.

MOTIVATION FOR BNF

- Before BNF, ambiguities in syntax caused compiler issues.
- BNF introduced precise and structured grammar rules.
- Enabled the development of automated parsers for compilers.

INTRODUCTION OF BNF

John Backus introduced BNF in1959.

- First used to define the ALGOL 60 programming language.
- Created a notation for programming languages.

BNF allowed programming languages to be clearly specified for the first time.

COMPONENTS OF BNF

- Terminals – The actual symbols (e.g., keywords, numbers, operators).
- Non-terminals – Abstract names for patterns (e.g., ``, ``).
- Production Rules – Define how non-terminals expand into terminals.

TERMINALS IN BNF

> Terminals are the basic symbols in a language.

- They appear exactly as written in the language.
- Examples of terminals:

```
+   -   *   /   (   )   0-9   if   while   int
```

Think of terminals as fixed words or symbols in a language.

NON-TERMINALS IN BNF

- Non-terminals represent patterns or structures.
- They are enclosed in angle brackets (`< >`).
- Examples:

```
<expression>   <term>   <statement>   <loop>
```

Think of non-terminals as placeholders for complex patterns.

PRODUCTION RULES

- Production rules define how non-terminals expand.
- Uses the `::=` operator.
- Example rule:

```
<expression> ::= <term> | <expression> "+" <term>
```

This means an `expression` is either a `term` or an `expression` followed by `+` and another `term`.

FULL EXAMPLE OF BNF

```
<expression> ::= <term> | <expression> "+" <term> | <e
<term> ::= <factor> | <term> "*" <factor> | <term> "/"
<factor> ::= <number> | "(" <expression> ")"
<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

✅ This defines a basic arithmetic expression.

ACTIVITY: IDENTIFY COMPONENTS

Given the rule:

```
<statement> ::= "if" "(" <condition> ")" "{" <body> "}
```

Identify:

- Terminals
- Non-terminals
- Production rule structure

QUICK QUIZ

Which of the following is not a valid BNF rule?

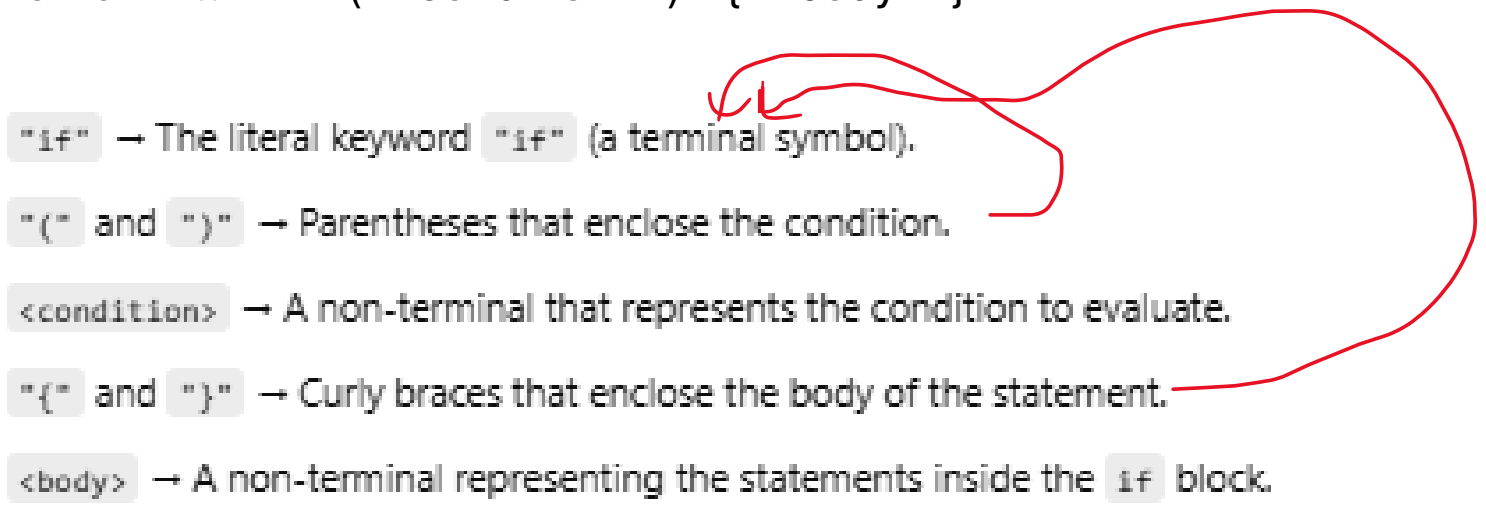
```
1. <expr> ::= <expr> "+" <term>
2. term ::= number | "(" expr ")"
3. <expr> = <expr> "+" <term>
4. <factor> ::= "0" | "1" | ... | "9"
```

Discuss and explain your answer!

BNF vs EBNF

IF statement structure:

<statement> ::= "if" "(" <condition> ")" "{" <body> "}“



`"if"` → The literal keyword `"if"` (a terminal symbol).

`"("` and `")"` → Parentheses that enclose the condition.

`<condition>` → A non-terminal that represents the condition to evaluate.

`"{"` and `"}"` → Curly braces that enclose the body of the statement.

`<body>` → A non-terminal representing the statements inside the `if` block.

Expanding it to if-else statement:

<statement> ::= "if" "(" <condition> ")" "{" <body> "}" <optional_else>
<optional_else> ::= "else" "{" <body> "}" | ε

Note : Eepsilon represents an empty production (i.e., nothing), which simulates the optional behavior

Standard BNF does not support optional elements directly; everything must be explicitly defined using separate rules.

EBNF

Extended Backus-Naur Form (EBNF) is an improved version of **Backus-Naur Form (BNF)** used to describe the syntax of programming languages in a more **concise and readable** way. It introduces additional symbols to make grammar definitions easier to write and understand.

EBNF extends BNF by:

- ✅ **Making rules more compact**
- ✅ **Adding optional elements** (without requiring extra rules)
- ✅ **Allowing repetition without recursion**
- ✅ **More concise syntax. Hence, improved readability**

Key Features of EBNF

1)Allow optional parts directly. Square brackets [...] indicate optional parts. The else block is optional below.

<if-statement> ::= "if" "(" <condition> ")" "{" <body> "}" ["else" "{" <body> "}"]

2) Repetition using curly braces. Following mean a number consists of at least one digit, followed by zero or more digits

<number> ::= <digit> { <digit> }

3) Parentheses are used to group elements.

<term> ::= <factor> (("*" | "/") <factor>)*
A <term> is a <factor> followed by zero or more occurrences of "*" or "/" with another <factor>.

BNF

<expression> ::= <term> | <expression> + <term> | <expression> - <term>

<term> ::= <factor> | <term> * <factor> | <term> / <factor>

<factor> ::= <number> | (<expression>)

<number> ::= <digit> | <number> <digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

EBNF

<expression> ::= <term> (("+" | "-") <term>)*

<term> ::= <factor> (("*" | "/") <factor>)*

<factor> ::= <number> | "(" <expression> ")"

<number> ::= <digit> { <digit> }

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

If – else if - else

<statement> ::= "if" "(" <condition> ")" "{" <body> "}"

{ "else if" "(" <condition> ")" "{" <body> "}" }

["else" "{" <body> "}"]

Which is invalid BNF rule?

1) <digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9“
2) <number> ::= <digit> | <number> <digit>
3) <expression> ::= <term> { ("+" | "-") <term> }
4) <statement> ::= "while" "(" <condition> ")" "{" <body> "}“

🚀 GUIDED ACTIVITY

We need to define a BNF grammar for an **Aircraft Autopilot Alert System**.

- Monitors **altitude**, **pitch**, and **error margin**.
- Auto-corrects minor errors.
- Sends alerts for serious issues.

Question: What are the key components we need to model?

🛠 STEP 1: IDENTIFY INPUTS

Inputs are measurable factors:

- **Altitude**: "low", "normal", "high".
- **Pitch**: "up", "level", "down".
- **Error Margin**: "low", "moderate", "high".

Now, let’s write the BNF for this!

✍️ WRITING BNF FOR INPUTS

```
<altitude> ::= "low" | "normal" | "high"
<pitch> ::= "up" | "level" | "down"
<error> ::= "low" | "moderate" | "high"
```

✅ These are our terminals because they are fixed values.

🔍 STEP 2: IDENTIFY SYSTEM MESSAGES

The system needs to generate messages:

- **Auto-correct**: Adjust flight path if error is low.
- **Alert**: Warn if error is high.

How do we express this in BNF?

✍️ WRITING BNF FOR SYSTEM MESSAGES

```
<decision> ::= <autocorrect> | <alert>
<autocorrect> ::= "Auto-correcting flight path..."
<alert> ::= "ALERT! Manual intervention required."
```

✅ The system selects a decision based on the error level.

🖇️ STEP 3: COMBINE STATUS & DECISION

We now define the full message structure:

- The status includes altitude, pitch, and error.
- The decision follows based on error margin.

Let’s write this in BNF.

📌 FULL BNF (BASIC VERSION)

```
<message> ::= <status> <decision>

<status> ::= "Altitude:" <altitude> "Pitch:" <pitch> "

<altitude> ::= "low" | "normal" | "high"
<pitch> ::= "up" | "level" | "down"
<error> ::= "low" | "moderate" | "high"

<decision> ::= <autocorrect> | <alert>
<autocorrect> ::= "Auto-correcting flight path..."
<alert> ::= "ALERT! Manual intervention required."
```

🔷 EXAMPLE OUTPUTS

```
Altitude: high Pitch: down Error: low
Auto-correcting flight path...
```

```
Altitude: normal Pitch: level Error: high
ALERT! Manual intervention required.
```

🚀 STUDENT ACTIVITY

We need to define a **BNF grammar** for an aircraft autopilot alert system.
The system monitors:

- 🌍 **Altitude**
- 🛩️ **Pitch**
- ⚠️ **Error margin**
- 🌦️ **Weather conditions**

Based on these, the system decides whether to **auto-correct** or trigger an **alert**.

🛠 STEP 1: IDENTIFY INPUTS

These are the monitored factors:

- **Altitude:**

  ```
  low | normal | high
  ```

- **Pitch:**

  ```
  up | level | down
  ```

- **Error Margin:**

  ```
  low | moderate | high
  ```

- **Weather:**

  ```
  clear | storm | turbulence
  ```

Task: How do we write this in BNF?

📌 STEP 2: DEFINE SYSTEM BEHAVIOR

The system should:

- ✅ Auto-correct if error is `low`.
- ⚠️ Trigger an alert if error is `high`.
- 🛩️ Allow pilot override.

How do we represent these decisions in BNF?

🚨 STEP 3: DEFINE ALERT CONDITIONS

The system triggers different alerts based on conditions:

- ⛽ **Low Fuel Warning**:

  ```
  Altitude = low & Error = high
  ```

- 🛑 **Stall Warning**:

  ```
  Pitch = up & Error = (moderate | high)
  ```

- 🌪️ **Severe Turbulence Alert**:

  ```
  Weather = turbulence & Error = high
  ```

- ⛈️ **Critical Weather Alert**:

  ```
  Weather = storm
  ```

- 🎮 **Manual Override**: If pilot takes control.

Task: Write BNF rules to capture these alerts.