CSF 425 / L-14 / 16.10.2024

Syntax Analyzer:

- Structure of a sentence → source code
- Grammar
  - ↳ Context Free Grammar (CFG)
    - ↳ BNF ⇒ meta language
      - ⇒ language that is used to explain other language

input for CFG:
- Lexems ⇒ smallest syntactical sub-unit
- Tokens ⇒ Group of Lexems

BNF → (concise) → EBNF → Produce Parse Tree → (concise) → Abstract Syntax tree (AST)

while (x ≥ 5)
x = x - 6 ;

→ Lexems Analyzer → output

while ⇒ KEYWORDS
(⇒ LPAREN
x ⇒ ID
≥ ⇒ OP
5 ⇒ NUMBER
) ⇒ RPAREN
x ⇒ ID
= ⇒ EQ
x ⇒ ID
- ⇒ OP
6 ⇒ NUMBER
; ⇒ SEMICLOLNE

Longest sub-string:
- until get a white space.

⇒ do <space> if → KEYWORDS

doif ⇒ valid ID

Meta language: ⇒ BNF

⇒ Meta symbol ⇒ | ⇒ OR

→ ⇒ defined as

::= ⇒ ↗

<non-terminal>

terminal

⇒ Lexems & tokens

<non-terminal>

⇒ Abstraction used in BNF description.

Grammar:

<program> → begin <statement_list> end

<statement_list> → <statement> | <statement> ; <statement_list>

<statement> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var> | <var> - <var> | <var>

or after fix.

Code:

```
begin  A = B + C;  B = C  end
```

Meta languages:

<Program> → begin <stmt_list> end

→ begin <stmt> ; <stmt_list> end

→ begin <var> = <expression> ; <stmt_list> end

→ begin A = <expression> ; <stmt_list> end

→ begin A = <var> + <var> ; <stmt_list> end

→ begin A = B + <var> ; <stmt_list> end

→ begin A = B + C ; <stmt_list> end

→ begin A = B+C ; <stmt> end

→ begin A = B+C ; <var> = <expression> end

→ begin A = B+C ; B = <expression> end

→ begin A = B+C ; B = <var> ± <var> end

not valid for this grammar
- we need to another option OR show error message.

→ begin A = B+C ; B = <var> end

→ begin A = B+C ; B = C end.

* Left most derivation:
  - replacing left most non-terminal at every derivation step.

* Right most derivation:
  - replacing right-most non-terminal .....

* Assignment Statement: Grammar:

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <id> + <expr> |
<id> * <expr> |
(<expr>) |
<id>

⇒ Statement:

$A = B * (A+C)$

⇒ Meta language:

<assign> → <id> = <expr>

→ A = <expr>

→ A = <id> * (<expr>)

→ A = B * <expr>

→ A = B * (<expr>)

→ A = B * (<id> + <expr>)

→ A = B * (A + <expr>)

→ A = B * (A + <id>)

→ A = B * (A + C)

Grammar will be given and a statement.

→ write down the meta language

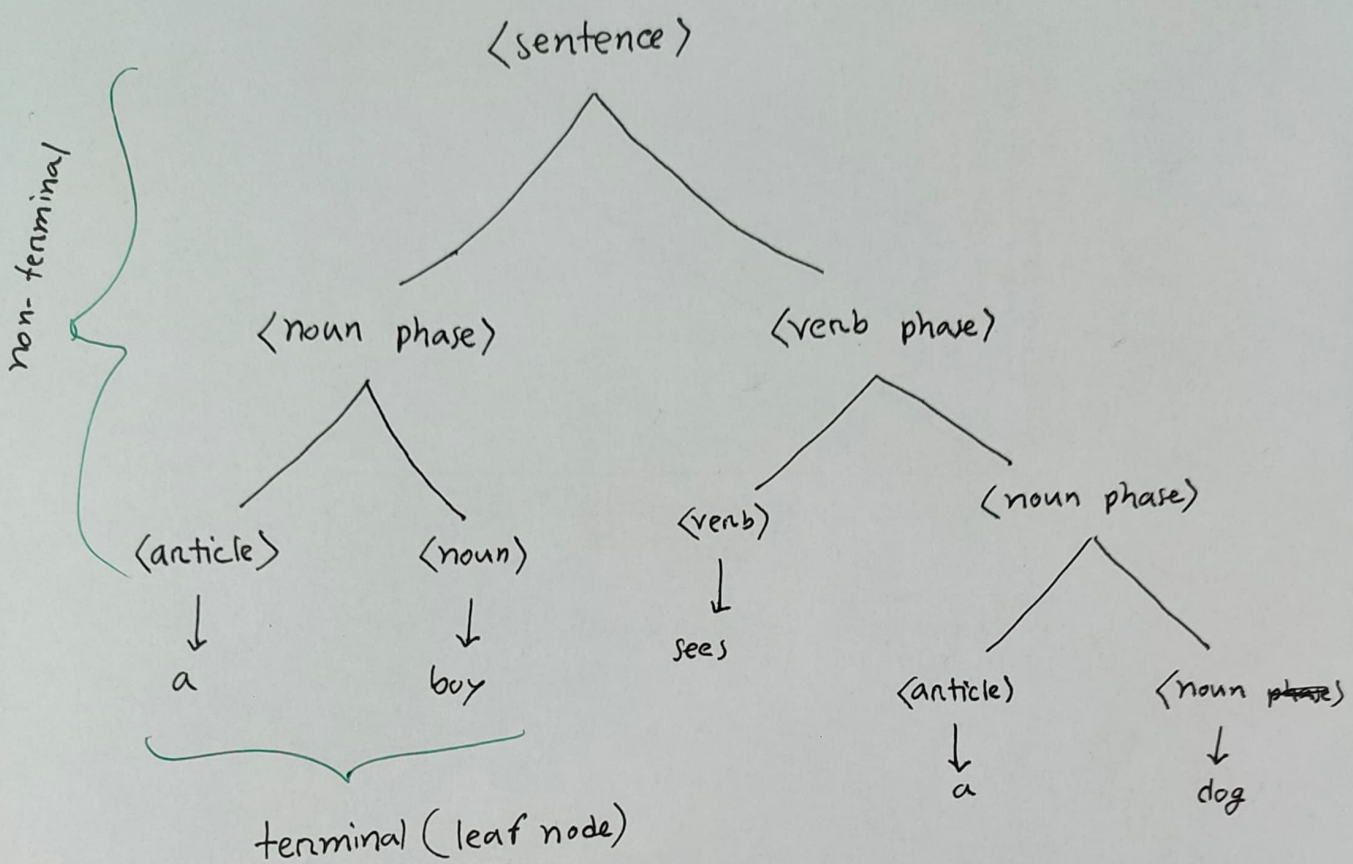
OR

vise-versa (fin grammar or write grammar)

L-15/ 21.10.2024

CFG Derivation of the language construct → Parse Tree → AST: Abstract Syntax Tree

Parse Tree:



Grammar:

$$\langle \text{number} \rangle \longrightarrow \langle \text{number} \rangle \langle \text{digit} \rangle \mid \langle \text{digit} \rangle$$

$$\langle \text{digit} \rangle \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

⇒ String: 234

⇒ Meta-language:

$\langle \text{number} \rangle \longrightarrow \langle \text{number} \rangle \langle \text{digit} \rangle$

$\longrightarrow \langle \text{number} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle$

$\longrightarrow \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle$

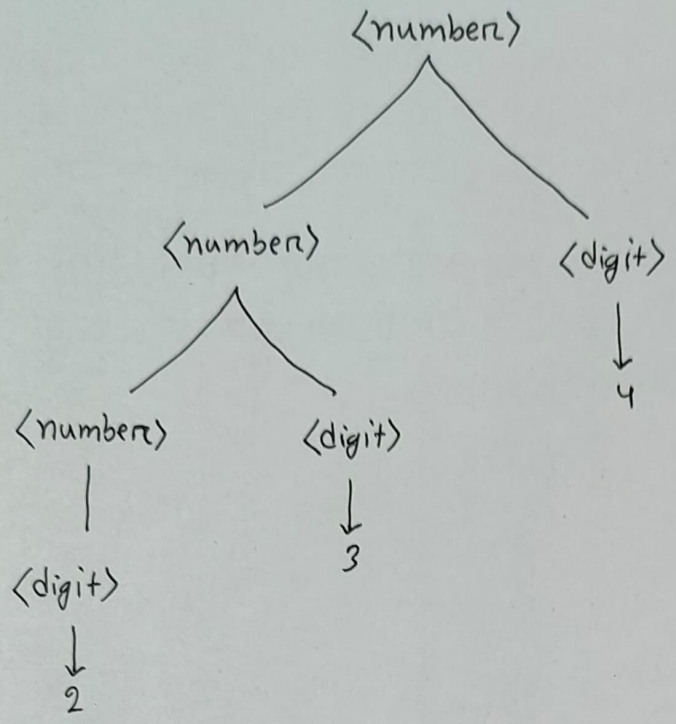
$\longrightarrow 2 \langle \text{digit} \rangle \langle \text{digit} \rangle$

$\longrightarrow 2\ 3 \langle \text{digit} \rangle$

$\longrightarrow 2\ 3\ 4$

Left most derivation

⇒ Parse tree:



Right most derivation:

<number> ⟶ <number> <digit>

⟶ <number> 4

⟶ <number> <digit> 4

⟶ <number> 3 4

⟶ <digit> 3 4

⟶ 2 3 4

Parse tree: Whatever ~~you~~ we use left most or right most derivation, parse tree will be the same for a perfect grammar.

Grammar:

$$\langle expr \rangle \longrightarrow \langle expr \rangle * \langle expr \rangle \mid \langle expr \rangle + \langle expr \rangle \mid (\langle expr \rangle) \mid \langle number \rangle$$

$$\langle number \rangle \longrightarrow \langle number \rangle \langle digit \rangle$$

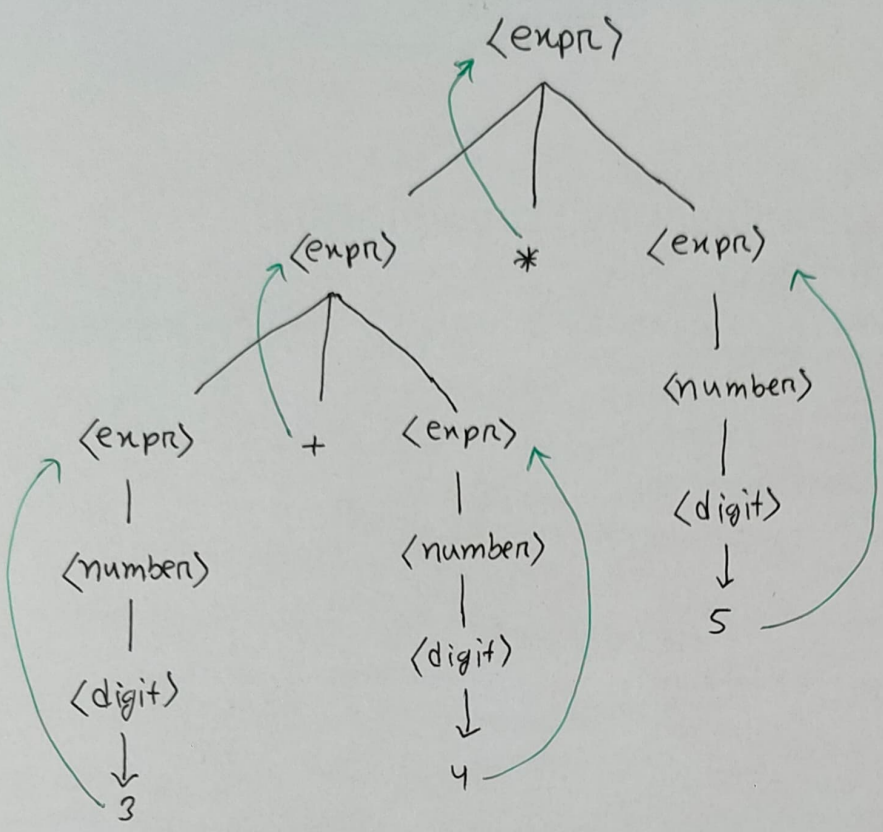
$$\langle digit \rangle \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Source/String: $3 + 4 * 5$

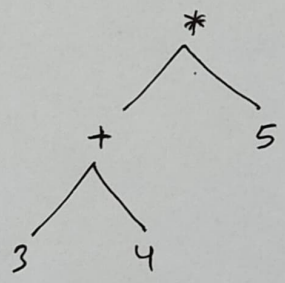
Meta-language: Left most derivation

$$\begin{aligned}
\langle expr \rangle &\longrightarrow \langle expr \rangle * \langle expr \rangle \\
&\longrightarrow \langle number \rangle + \langle expr \rangle * \langle expr \rangle \\
&\longrightarrow \langle digit \rangle + \langle expr \rangle * \langle expr \rangle \\
&\longrightarrow 3 + \langle expr \rangle * \langle expr \rangle \\
&\longrightarrow 3 + \langle number \rangle * \langle expr \rangle \\
&\longrightarrow 3 + \langle digit \rangle * \langle expr \rangle \\
&\longrightarrow 3 + 4 * \langle expr \rangle \\
&\longrightarrow 3 + 4 * \langle number \rangle \\
&\longrightarrow 3 + 4 * \langle digit \rangle \\
&\longrightarrow 3 + 4 * 5
\end{aligned}$$

⇒ Parse Tree:



⇒ AST :



⇒ 35

⇒ Meta language : Another left most derivation

<expr> ⟶ <expr> + <expr>

⟶ <number> + <expr>

⟶ <digit> + <expr>

⟶ 3 + <expr>

⟶ 3 + <expr> * <expr>

⟶ 3 + <number> * <expr>

→ 3 + <digit> * <expr>

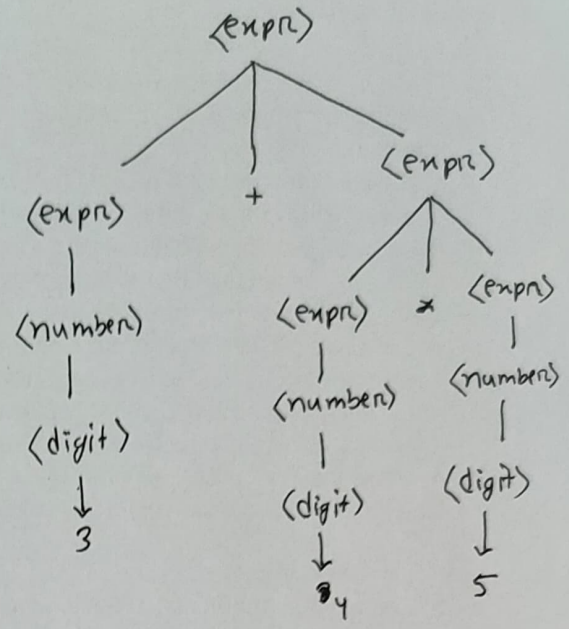
→ 3 + 4 * <expr>

→ 3 + 4 * <number>

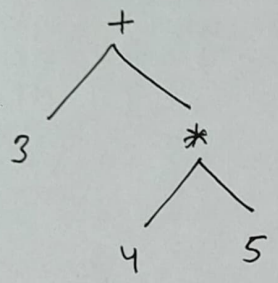
→ 3 + 4 * <digit>

→ 3 + 4 * 5

Parse tree:



AST:



⇒ 23

✳ Is there more than one AST/ Parse tree exist, then the grammar is ambiguous.

✳ Question: Check is the grammar is ambiguous or not?

✳ Given a language construct, if the designed grammar has more than one left-most derivation (or right-most) then, the grammar is ambiguous.

⇒ Each left-most has its unique Parse Tree.

✳ If same precedence operator exist, then follow the left to right.

Midterm
30.10.2024
upto 28.10.2024

Ambiguity Source:

$\langle expr \rangle \longrightarrow \langle expr \rangle + \langle expr \rangle \mid \langle expr \rangle * \langle expr \rangle$

↓ revised

$\langle expr \rangle \longrightarrow \langle expr \rangle + \langle expr \rangle \mid \langle term \rangle$

$\langle term \rangle \longrightarrow \langle term \rangle * \langle term \rangle \mid (\langle expr \rangle) \mid$

$\langle number \rangle$

L-16 / 23.10.2024

Language Grammar (CFG):

From the derivation → Left most derivation
From the derivation → Right most derivation

⇒ Ambiguity : multiple parse tree for a given language construct.

Example: Operators precedence.

⇒ each left-most (Right-most) has its unique parse tree.

Associativity:

$4-2=2$ →

$8-4-2=?$

← $8-2=6$

⇒ assured through → left recursive for left-associative
⇒ assured through → right recursive for right-associative

Grammar:

$\langle expr \rangle \longrightarrow \langle expr \rangle + \langle expr \rangle \mid \langle term \rangle$

↓ modifying further

Recursive defined in left so, left associative

$\langle expr \rangle \longrightarrow \langle expr \rangle + \langle term \rangle \mid \langle term \rangle$

$\langle term \rangle \longrightarrow \langle term \rangle * \langle factor \rangle \mid \langle factor \rangle$

$\langle factor \rangle \longrightarrow (\langle expr \rangle) \mid \langle number \rangle$

$\langle number \rangle \longrightarrow \langle number \rangle \langle digit \rangle \mid \langle digit \rangle$

$\langle digit \rangle \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Left associative: left recursive

$\langle expr \rangle \longrightarrow \langle expr \rangle + \langle term \rangle \mid \langle term \rangle$

Right associative: Right recursive

$\langle expr \rangle \longrightarrow \langle term \rangle + \langle expr \rangle \mid \langle term \rangle$

left, right

L.H.S. ⟶ R.H.S.

⇒ we need to follow any one in the entire grammar.

Exam Question:

upper case ⇒ ⟨non-terminal⟩

lower case ⇒ terminal

Ambiguous or not ongoing research topic - Automata

Grammar:

empty string

$$S \rightarrow AS \mid \epsilon$$

$$A \rightarrow A1 \mid 0A1 \mid 01$$

⇒ check if the above grammar is ambiguous or not?

⇒ Let's consider a string,

00111

⇒ left most derivation:

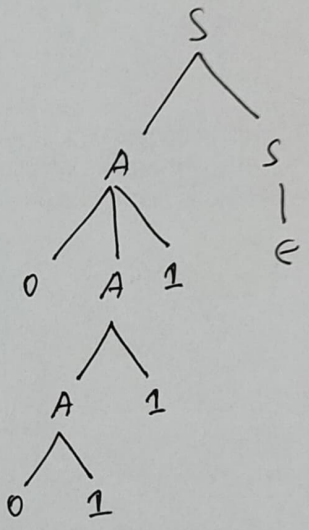
$S \rightarrow AS$

$\rightarrow \boxed{0A1}S \quad [A = 0A1]$

$\rightarrow 0\boxed{A1}1S \quad [A = A1]$

$\rightarrow 0\boxed{01}11S \quad [A = 01]$

$\rightarrow 00111 \quad [S = \epsilon]$



⇒ Another left most derivation:

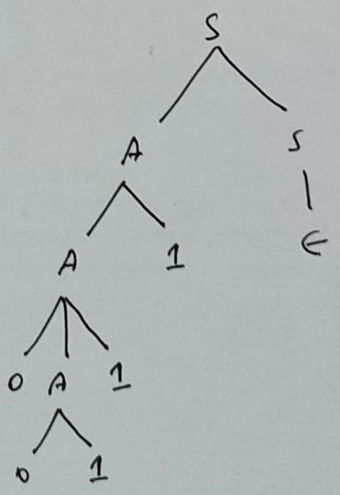
$S \rightarrow AS$

$\rightarrow \boxed{A1}S \quad [A = A1]$

$\rightarrow \boxed{0A1}1S \quad [A = 0A1]$

$\rightarrow 0\boxed{01}11S \quad [A = 01]$

$\rightarrow 00111 \quad [S = \epsilon]$



⇒ Therefore the grammar is ambiguous.

Grammar:

$S \rightarrow AB \mid C$

$A \rightarrow aAb \mid ab$

$B \rightarrow cBd \mid cd$

$C \rightarrow aCd \mid aDd$

$D \rightarrow bDc \mid bc$

String: aabbccdd

Left most derivation:

$S \rightarrow AB$
$\rightarrow aAbB$
$\rightarrow aabbB$
$\rightarrow aabbcBd$
$\rightarrow aabbccdd$

Another left most derivation:

$S \rightarrow C$
$\rightarrow aCd$
$\rightarrow aaDdd$
$\rightarrow aabDcdd$
$\rightarrow aabbccdd$

$\Rightarrow$ two parse tree available.

$\Rightarrow$ ambiquous.

Another question could be what is the string accepted or not?

$A \rightarrow Ba \mid bc$

$B \rightarrow d \mid eBf$

$C \rightarrow gC \mid g$

$\Rightarrow$ bffd, faae, bg, bggg ... } accepted or rejected?

Σ = alphabets

= $\{a, b\}$

⇒ Design a grammar that achieve palindrome of any length.

⇒

$S \rightarrow a \mid b \mid aSa \mid bSb$

EBNF (Extended Backus Naür Form):

⇒ $\langle number \rangle \rightarrow \langle numbers \rangle \langle digit \rangle$

$\rightarrow \langle number \rangle \langle digit \rangle \langle digit \rangle$

$\rightarrow \langle digit \rangle \langle digit \rangle \langle digit \rangle \cdots\cdots$

↓ EBNF

$\langle number \rangle \rightarrow \langle digit \rangle \{ \langle digit \rangle \}$

0 or any number of repeatation.

⇒ reduced the long chain by defining some new meta symbol. Zts just for convenience.

$\langle expr \rangle \rightarrow \langle expr \rangle + \langle term \rangle$

$\rightarrow \langle expr \rangle + \langle term \rangle + \langle term \rangle + \cdots$

$\rightarrow \langle term \rangle + \langle term \rangle + \langle term \rangle + \cdots\cdots$

↓ EBNF

$\langle expr \rangle \rightarrow \langle term \rangle \{ + \langle term \rangle \}$

0 or any number of repeatation.

Anything within the

(----) ⇒ means: options

⇒

⟨term⟩ → ⟨term⟩ * ⟨factor⟩ |

⟨term⟩ + ⟨factor⟩ |

⟨term⟩ - ⟨factor⟩ |

⟨term⟩ / ⟨factor⟩

↓ EBNF

⟨term⟩ → ⟨term⟩ (* | + | - | /) ⟨factor⟩

Grammar: EBNF

⟨expr⟩ → ⟨expr⟩ + ⟨term⟩ |

⟨expr⟩ - ⟨term⟩ |

⟨term⟩

⟨term⟩ → ⟨term⟩ * ⟨factor⟩ |

⟨term⟩ / ⟨factor⟩ |

⟨factor⟩

⟨factor⟩ → ⟨exp⟩ ** ⟨factor⟩ |

⟨exp⟩

⟨exp⟩ → (⟨expr⟩) | id

The algebra of Language

Article

⇒ EBNF:

⟨expr⟩ → ⟨term⟩ { (+|-) ⟨term⟩ }

⟨term⟩ → ⟨factor⟩ { (*|/) ⟨factor⟩ }

⟨factor⟩ → ⟨exp⟩ { ** ⟨exp⟩ }

⟨exp⟩ → (⟨expr⟩) | id