Chapter 2: High-Level Languages and Their Syntactic Description

Instructor: Gao Zhen
Email: gaozhen@tongji.edu.cn

Outline

1. Programming language definition

2. General features of high-level languages

    - Program structure

    - Data types & operations

    - Statements & control structures

3. Syntax description methods

Formal Languages

- To understand/define high-level programming languages is crucial for compiler construction.
- This section will introduce the **formal description** of syntactic structures.

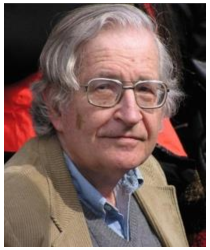
Compiler = Formal language theory + Compiler techniques

Formal Languages

- **Formal languages**
  - Described using universally recognized **symbols** and expressions, formal languages are general and nationality-neutral.
  - A formal language is a set of strings over some **alphabet** and has a **well-defined** descriptive scope.
  - The original idea came from linguist **Noam Chomsky**, who aimed to use formal methods to describe languages.
  - Started in natural language research, but found wide application in computer science, especially in theoretical computer science.

Timeline

- 1936 — Alan Turing
  - Proposed the Turing Machine, formalizing the concept of "computability."
- 1951–1956 — Stephen Kleene
  - Developed the concept of regular sets / regular expressions. Proved the equivalence regular expressions ↔ finite automata.



- 1956 — **Noam Chomsky**
  - Published *Three Models for the Description of Language*.Proposed the Chomsky hierarchy (regular, context-free, context-sensitive, unrestricted grammars).
- 1959 — Michael O. Rabin & Dana Stewart Scott
  - Introduced Nondeterministic Finite Automata (NFA) , deepening automata theory. Received the Turing Award in 1976.
- 1960–1961 — John Backus, Peter Naur, et al.
  - Used BNF (Backus–Naur Form) in programming language design, directly applying Chomsky's context-free grammar ideas.
- 1961–1963 — John E. Hopcroft, Jeffrey D. Ullman etc.
  - Systematically proved the equivalence between formal languages and automata.
- 1969 — Hopcroft & Ullman
  - Published *Formal Languages and Their Relation to Automata*, a classic textbook that unified the equivalence of grammars, automata, and languages.

Formal Definition of Grammar

- A grammar G is a 4-tuple **$G=(V_N, V_T, S, \text{₤})$**

  V_N : a finite set of non-terminal symbols

  V_T : a finite set of terminal symbols, $V_N \cap V_T = \Phi$

  S : the start symbol, and $S \in V_N$

  ₤ : a finite set of production rules of the form $P \rightarrow \alpha$

  $P \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$, $\alpha \in (V_N \cup V_T)^*$

Example of Grammar

- Let $G_1 = (\{N\}, \{0, 1\}, N, \{N \rightarrow 0N, N \rightarrow 1N, N \rightarrow 0, N \rightarrow 1\})$

  $V_N = \{N\}$, [non-terminal symbols]

  $V_T = \{0, 1\}$, [terminal symbols / alphabet]

  $S = N$, [start symbol]

  $\mathcal{P} = \{N \rightarrow 0N, N \rightarrow 1N, N \rightarrow 0, N \rightarrow 1\}$ [productions]

Explanation of Grammar

- **Terminal symbols (V_T)**
  - **basic symbols in the alphabet, individual tokens**
- **Non-terminal symbols (V_N)**
  - **need further definition, represent grammatical concept such as "arithmetic expression", "boolean expression", "procedure", etc.**
    → Non-terminals represent sets (not individual tokens)
- **Start symbol S**
  - **represents the starting point of the language, and all strings derived from it form the language**
  - **a non-terminal that appears on the left side of at least one production rule**

Explanation of Grammar

- **Production rules**Ꮬ : define the syntax structure; written as
  - $P \rightarrow \alpha$ （or $P ::= \alpha$ ）

    where "→" means "is **defined** as"

    P， α are symbol strings

    Example：$S \rightarrow 0S1$， $0S \rightarrow 01$
  - Productions with the same left-hand side ($P \rightarrow \alpha_1$, $P \rightarrow \alpha_2$, $P \rightarrow \alpha_3$, ... , $P \rightarrow \alpha_n$) can be written compactly

    $P \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$ Each α_i is a **candidate** of P.

Notation Conventions

- V_N:**uppercase** letters A, B, C, S, etc.
- V_T: **lowercase** letters, digits 0–9, operators +, –, etc.
- α、β、γ:**strings** of grammar symbols,$\in(V_T \cup V_N)^*$
- S:**start** symbol, appears in the first production
- → :**definition** symbol ("is defined as")
- | : "or"

Notation Conventions

- **To simplify, only the production part is written**

  Assume the left-hand side of the first production is the start symbol, or prefix the productions with "G[A]" where G is the grammar name, and A is the start symbol

Grammar G[N]: $N \rightarrow 0N$, $N \rightarrow 1N$, $N \rightarrow 0$, $N \rightarrow 1$

Grammar G[E]: $E \rightarrow E+E \mid E*E \mid (E) \mid i$

Direct Derivation and Reduction

- **Direct derivation**
  - If $A \rightarrow \gamma$ is a production, and $\alpha, \beta \in (V_T \cup V_N)^*$, then applying the rule $A \rightarrow \gamma$ to string $\alpha A \beta$ yields $\alpha \gamma \beta$
  - Written as: $\alpha A \beta \Rightarrow \alpha \gamma \beta$, called a **direct derivation**
- **Direct reduction**
  - Reduction is the reverse of derivation
  - If $\alpha A \beta \Rightarrow \alpha \gamma \beta$, then $\alpha \gamma \beta$ can be **directly reduced** to $\alpha A \beta$

Derivation

- Let $\alpha_1,\alpha_2,\ldots,\alpha_n$ $(n>0)\in(V_T\cup V_N)^*$, And

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$$

This sequence is a derivation from α_1 to α_n

- If such a derivation exists, α_1 can derive α_n
  - $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ (via one or more steps)
  - $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ (via zero or more steps)

Multiple Derivations

- **Given Grammer $G[N_1]$:**

$N_1 \rightarrow N \quad N \rightarrow ND|D \quad D \rightarrow 0|1|2$

Then sentence "12" can be derived in multiple ways:

(1) $N_1 \Rightarrow N \Rightarrow ND \Rightarrow N2 \Rightarrow D2 \Rightarrow 12$

(2) $N_1 \Rightarrow N \Rightarrow ND \Rightarrow DD \Rightarrow 1D \Rightarrow 12$

- **Thus, the same sentence can have different derivation sequences.**

Example – Grammar G[S]: S→AB A→A0|1B B→0|S1, Generate the sentence 101001:

Leftmost derivation:

$S \Rightarrow AB$

$\Rightarrow 1BB$

$\Rightarrow 10B$

$\Rightarrow 10S1$

$\Rightarrow 10AB1$

$\Rightarrow 101BB1$

$\Rightarrow 1010B1$

$\Rightarrow 101001$

Rightmost derivation:

$S \Rightarrow AB$

$\Rightarrow AS1$

$\Rightarrow AAB1$

$\Rightarrow AA01$

$\Rightarrow A1B01$

$\Rightarrow A1001$

$\Rightarrow 1B1001$

$\Rightarrow 101001$

Sentential Forms, Sentences, and Language

G[E]: E→E+E | E*E | (E) | i

- **Sentential Form**: Suppose G is a grammar and E is its starting symbol. If $E \Rightarrow \alpha$, then α is a sentential form of grammar G

  Example: (E + E), (i + E), (i + i), E

- **Sentence**: Sentential form consisting only of terminal characters are called sentences

  Example: (i × i + i), (i + i)

- **Language**: The whole of the sentence produced by grammar G

  $L(G) = \{\alpha | S \overset{+}{\Rightarrow} \alpha, \ \ \alpha \in V_T^*\}$

Example. With grammar G1 [S]: $S \rightarrow bA$, $A \rightarrow aA \mid a$

Try to find the **language** described by this grammar.

Solution: Because the following sentences can be deduced from the start symbol S:

$$S \Rightarrow bA \Rightarrow ba$$

$$S \Rightarrow bA \Rightarrow baA \Rightarrow baa$$

$$S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baaa$$

…

$$S \Rightarrow bA \Rightarrow baA \Rightarrow \ldots \Rightarrow baa\ldots a$$

So $L(G1)=\{ ba^n \mid n>=1\}$

Example. Construct a grammar G_3 for the language: $L(G3) = \{b^n a^n \mid n>=1\}$

This language consists of strings like:

n=1, x=ba

n=2, x=bbaa

n=3, x=bbbaaa …

L(G3)={ba,bbaa,bbbaaa,…}

Features of $L(G_3)$:

1. Symmetric strings (a's and b's in pairs)
2. Infinite set with recursive pattern
3. Alphabet: {a, b}

G3: S→ bSa|ba

G3=（{S}, {a,b}, S, {S→ bSa|ba})

Exercise

- P36
  - 第6题
    - 令文法G_6为

      N→D|ND

      D →0|1|2|3|4|5|6|7|8|9

      (1) G_6的语言L(G_6)是什么？

      (2) 给出句子0127, 34和568的最左推导和最右推导
  - 第8题
    - 令文法为

      E→T|E+T|E-T

      T→F|T*F|T/F

      F →(E)/i

      (1) 给出i+i*i, i*(i+i)的最左推导和最右推导

      (2) 给出i+i+i, i+i*i和i-i-i的语法树

Chomsky Grammar System (Review)

- **In the Chomsky hierarchy, any grammar must include:**
  - Two distinct finite sets of symbols:
    - Non-terminal set V_N
    - Terminal set V_T
  - A start symbol S
  - A finite set of formal rules $\wp$ (productions)
- Grammar $\mathbf{G=(V_N, V_T, S, \wp)}$, $\wp$: $P \rightarrow \alpha$,

  where $P \in (V_N \cup V_T)^* \, V_N \, (V_N \cup V_T)^*$, $\alpha \in (V_N \cup V_T)^*$
- Restrictions are imposed on the form of productions, and grammars are classified into four types: **Type 0, Type 1, Type 2, and Type 3**

Type 0 Grammer

- **Type 0 Grammar: Unrestricted grammar, phrase structure grammar**
  - **Corresponding Language: Recursively Enumerable Language Equivalent to Turing machines**
- **Example: The following grammar is Type 0:**

$S \rightarrow aBC|aSBC$

$CB \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bB \rightarrow b$

$bC \rightarrow bc$

$cC \rightarrow cc$

$cC \rightarrow c$

Type 1 Grammer

- **Also called Context-Sensitive Grammar (CSG)**
- **Production format: P→α, where**

  $|P| \leq |\alpha|$, $\alpha \in (V_N \cup V_T)^*$, $P \in (V_N \cup V_T)^* V_N (V_N \cup V_T)^*$
  - Corresponding language: **CSL** (Context-sensitive Language)
  - If ε is not considered, it is equivalent to a Linear Bounded Automata (**LBA**)
  - Example：$\alpha A\beta \rightarrow \alpha\gamma\beta$

Type 1 Grammer

- **Example: The following grammar is Type 1:**

  S → aBC|aSBC

  CB → BC

  aB → ab

  bB → bd

  bC → bc

  cC → ce

Type 2 Grammer

- **Also called Context-free Grammar (CFG)**
- **Production form: $P \to \alpha$, where**

  $P \in V_N$, $\alpha \in (V_N \cup V_T)^*$

  - **Corresponding language: Context-free Language (CFL)**
  - **Corresponding automaton: Pushdown Automata (PDA)**

Type 2 Grammer

- **Example. Context-free Grammar**

  $S \to 01$

  $S \to 0S1$

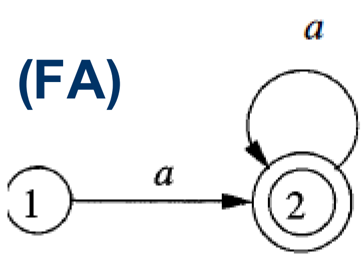
  Generated language $L = \{ 0^n1^n \mid n \geq 1\}$,

  e.g. 0011, 000111, 01 $\in$ **L, but 10, 1001 ,** ε**, 010** $\notin$ **L .**

  No finite automata can accept L.

Type 3 Grammer

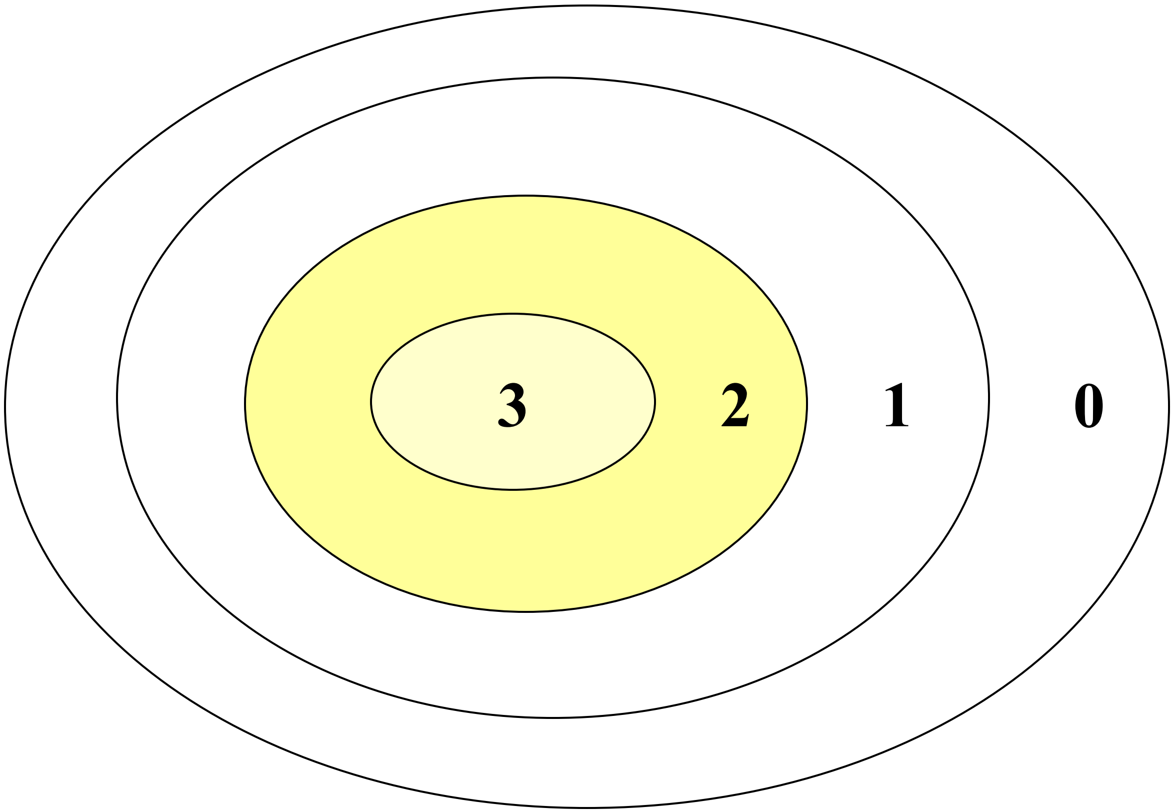
- **Also called Regular Grammar**
  - Right-linear Grammar：Productions of the form $A \to \omega B$ or $A \to \omega$, where A、B∈ V_N, ω∈ V_T^*。
  - Left-linear Grammar: Productions of the form $A \to B\omega$ or $A \to \omega$, where A、B∈ V_N, ω∈ V_T^*。
  - Equivalent to Regular Expressions
  - Corresponding language: Regular Language
  - Corresponding automaton: Finite Automaton (FA)



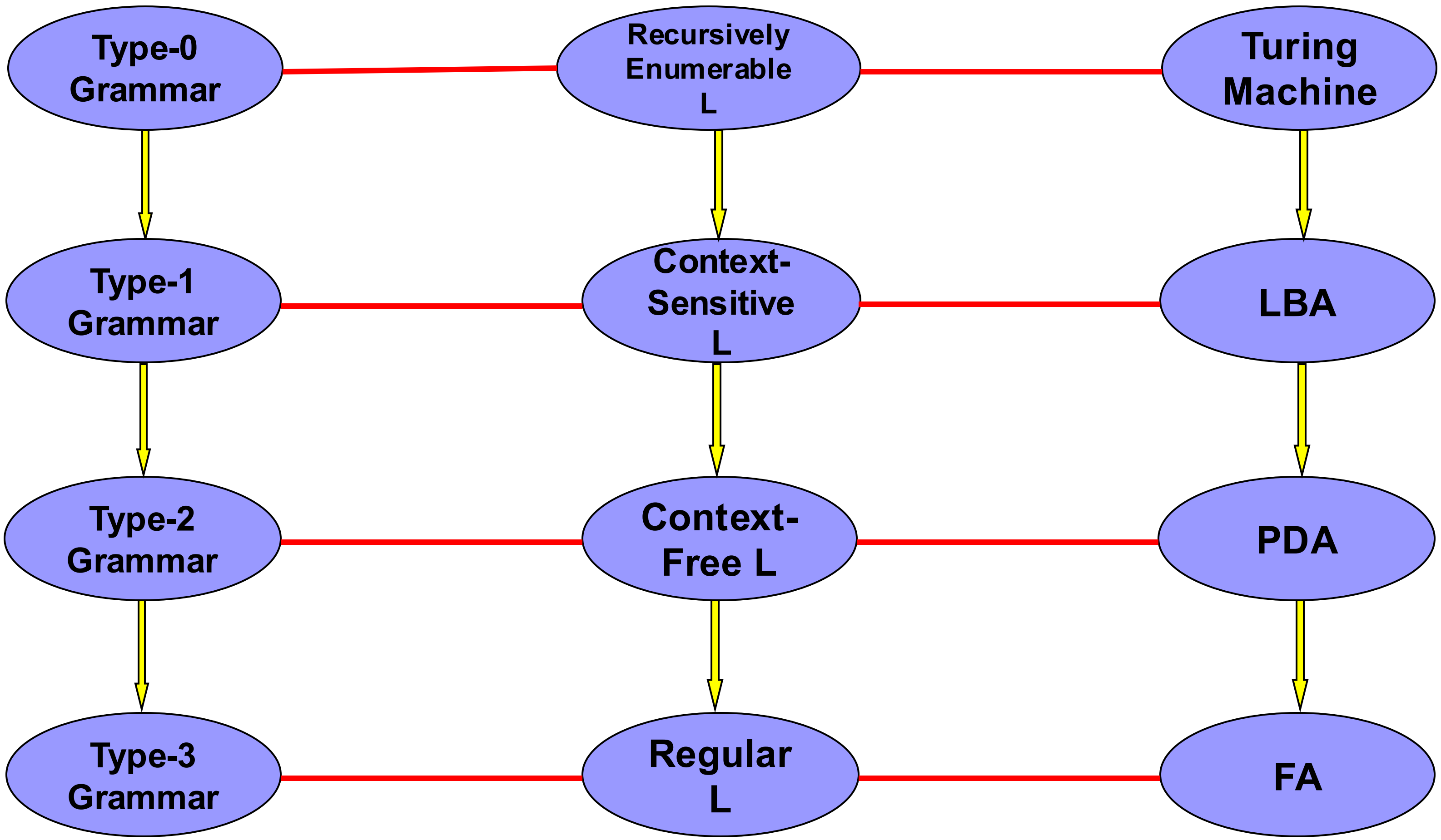
- **Example** $S \to aS$, $S \to a$

Equivalent regular expression：a^+, or a^*a

Chomsky Grammar Hierarchy



Correspondence Between Grammars, Formal Languages, and Automata



Exercise : how to draw the Finite Automata

Example. Given grammar

G1[S]: S $\rightarrow$bA , A$\rightarrow$aA | a

Task: Draw the equivalent Finite Automata

Conclusion （1）

- **Restrictions on context-free grammars for programming languages**
  - (1) Productions that form "self-loops" are NOT allowed: $P \rightarrow P$
  - (2) Every non-terminal P is useful, $P \in V_N$:
    - Appears in some derivation from start symbol: $S \Rightarrow \alpha P \beta$
    - Can derive a terminal string: $P \Rightarrow \gamma,\ \gamma \in V_T^*$
- **Lexical analysis: based on regular grammars**

Regular Grammar

$A \rightarrow lB$ $\quad\quad B \rightarrow lB|dB|\varepsilon$

- **Syntax analysis: based on context-free grammars**

Conclusion（2）

- **Context-free grammars are powerful enough to describe the syntax of most modern programming languages**
  - Arithmetic Expression
  - Assignment Statement
  - Conditional Statement
  - ……

Arithmetic Expression

Grammer G=({E}, {+, *, i, (,)}, E, P}

E → i   E → E+E

E → E*E   E → (E)

Conditional Statement

S→if E then S

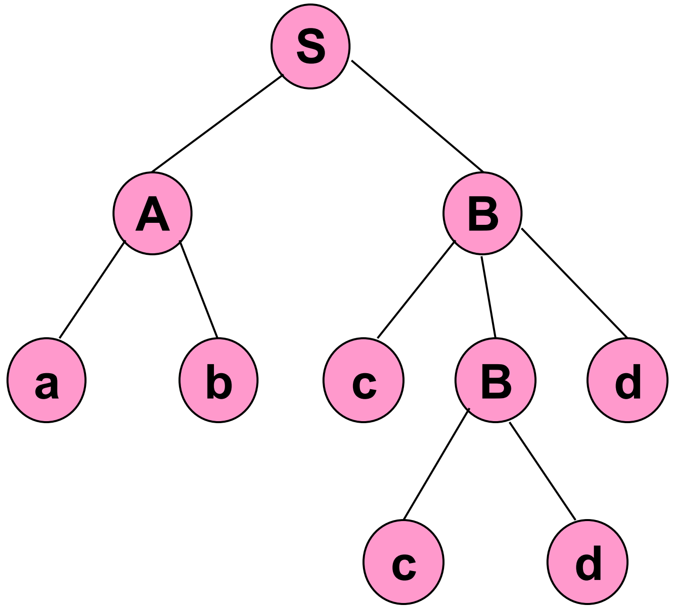
S→if E then S else S

Syntax Tree and Grammar Ambiguity

- **Parse Tree (Syntax Tree)**
- **Ambiguity**

What is a Parse Tree?

- A parse tree represents the derivation of a sentence (string) in a grammar
- It is an inverted tree (root at the top, leaves at the bottom)
  - Node
  - Edge
  - Root Node
  - Leaf Node
  - Leaf Branches： A(a,b), B(c,d)
  - Sibling Nodes



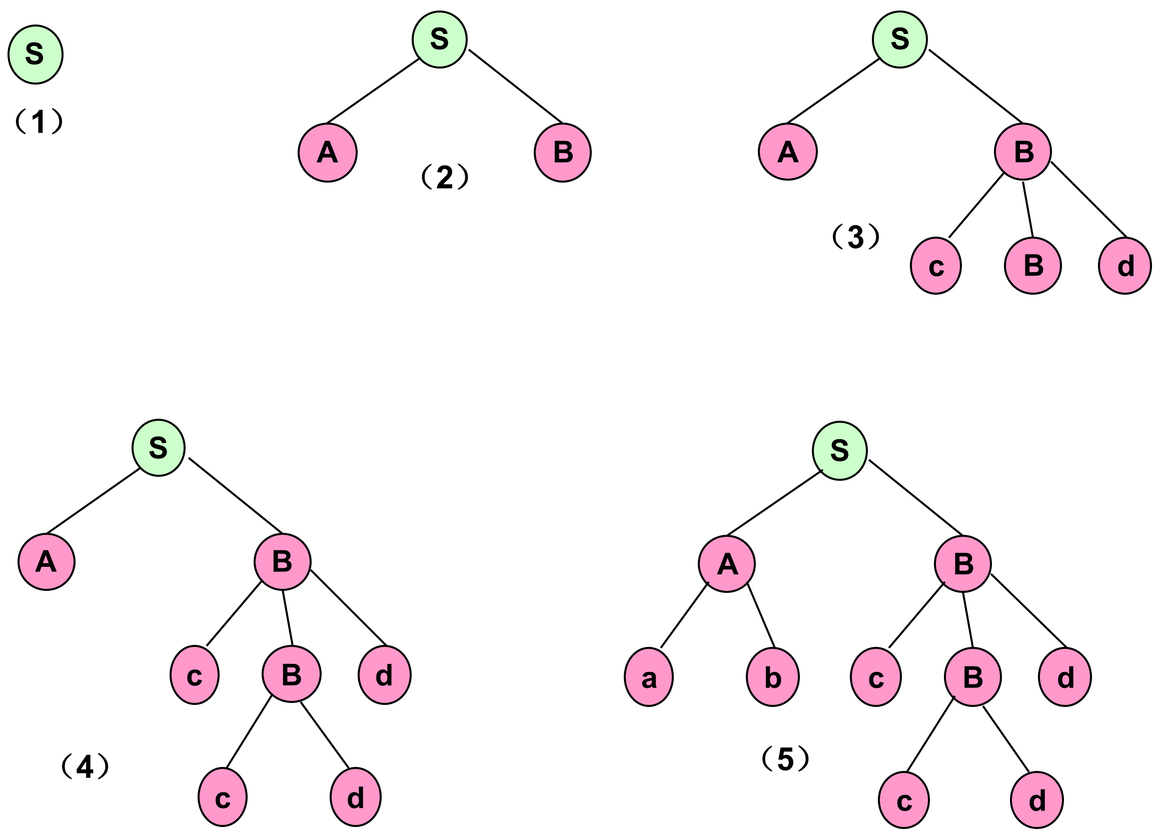
Constructing a Parse Tree from a Derivation

$S \Rightarrow AB$

$\Rightarrow AcBd$

$\Rightarrow Accdd$

$\Rightarrow abccdd$



- Example. Grammer G[E] ：

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$

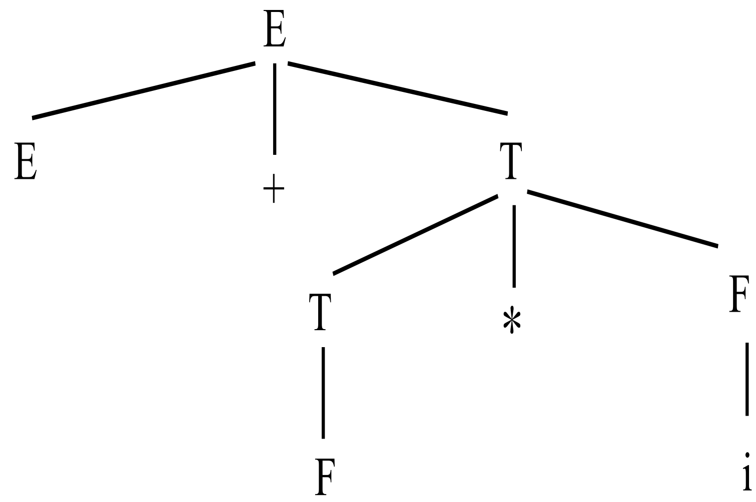
Derivation of expression E+F*i ：

$E \Rightarrow E+T$

$\Rightarrow E+T*F$

$\Rightarrow E+F*F$

$\Rightarrow E+F*i$



The syntax tree

More about parse tree

- A parse tree shows which rules are applied and on which nonterminal symbols, but it does NOT indicate the order of rule applications
- Does a sentence have a unique leftmost (or rightmost) derivation?
  - Not always
- Does a sentence correspond to a unique parse tree?
  - Not always

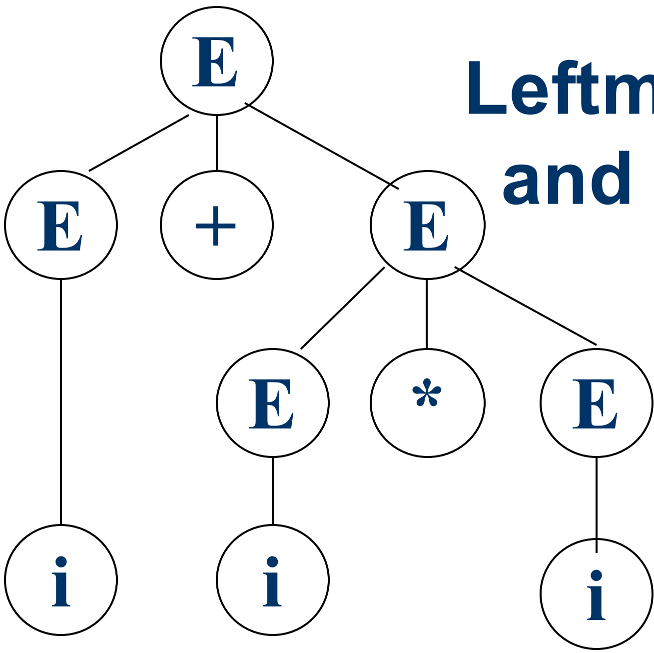
Ambiguity

- **A sentence is ambiguous if it has two parse trees**
- **A grammar is ambiguous if it generates any ambiguous sentence; otherwise, it is unambiguous**

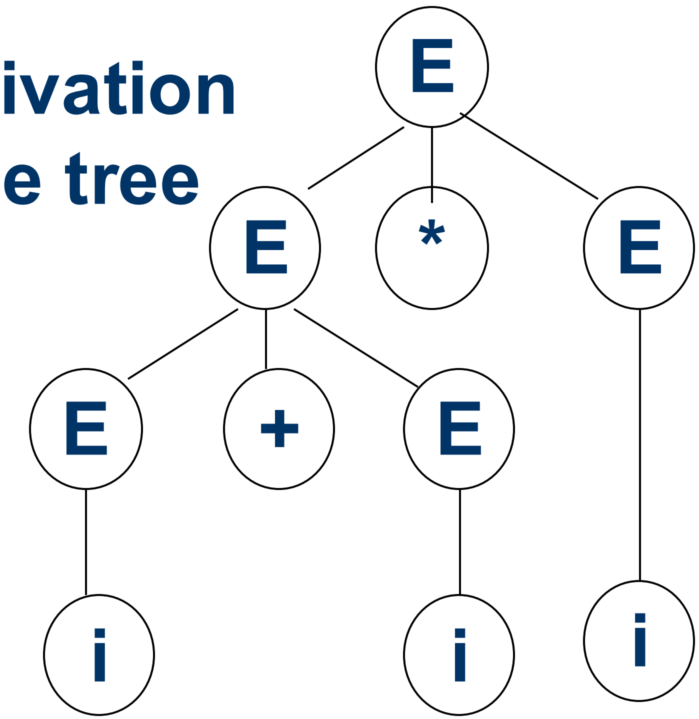
Grammer G[E] : E→E+E│E*E│(E)│i

$E \Rightarrow E+E \Rightarrow i+E$
$\Rightarrow i+E*E \Rightarrow i+i*E$
$\Rightarrow i+i*i$

$E \Rightarrow E*E \Rightarrow E+E*E$
$\Rightarrow i+E*E \Rightarrow i+i*E$
$\Rightarrow i+i*i$



Leftmost derivation and its parse tree



Similarly, the sentence's rightmost derivation and its parse tree are also different.

Transforming an ambiguous grammar into an unambiguous grammar

- **Grammer G[E] : $E \to E+E \mid E\times E \mid (E) \mid i$ is ambiguous. By defining precedence ($\times > +$) and left associativity, it can be transformed into an unambiguous grammar.**

Grammer G[E]:

$$E \to T \mid E+T$$

$$T \to F \mid T\times F$$

$$F \to (E) \mid i$$

The sentence has a unique derivation : $(i\times i+i)$

QUIZ-CANVAS

Dank u
Dutch

Merci
French

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

धन्यवाद
Hindi

감사합니다
Korean

Tack så mycket
Swedish

תודה רבה
Hebrew

Obrigado
Brazilian Portuguese

谢谢
Chinese

Thank You !

Dankon
Esperanto

ありがとうございます
Japanese

Trugarez
Breton

Danke
German

Tak
Danish

Grazie
Italian

நன்றி
Tamil

děkuji
Czech

ขอบคุณ
Thai

go raibh maith agat
Gaelic