



第二章 高级语言及其文法

重点：文法的定义与分类，CFG的语法树及二义性、
程序设计语言的定义。

难点：程序设计语言的语义。

编译器如何处理无穷多的程序
语言的有穷描述



第2章 高级语言及其文法

2.1 语言概述

2.2 基本定义

2.3 文法的定义

2.4 文法的分类

2.5 CFG的语法树

2.6 CFG的二义性

2.7 本章小结



2.1 语言概述

语言是**一定**的群体用来进行
什么是语言？
信息交流的工具。



2.1 语言概述

■ 信息交流的基础

- 按照共同约定的**生成规则**和**理解规则**去生成“句子”和理解“句子”

- 例：

- “今节日上课始开译第一编”

- “今日开始上第一节编译课”

- 语言的描述涉及最小单位的**“字”**和组合这些字的**“规则”**

2.1 语言概述

■ 语言的特征

■ 自然语言(Natural Language)

- 是人与人的通讯工具

- 语义(semantics):环境、语境——难以形式化

■ 计算机语言(Computer Language)

- 计算机系统间、人机间通讯工具

- 严格的语法(Grammar)、语义(semantics) ——易于形式化

这份报XX

研究表明，汉字的序顺并不定一能影响阅读，比如当你看完这句话后，才发这现里的字全是都乱的。



2.1 语言概述

- **语言的描述方法——现状**
 - **自然语言：自然、方便 - 非形式化**
 - **数学语言（符号）：严格、准确 - 形式化**
 - **形式化描述**
 - **高度的抽象，严格的理论基础和方便的计算机表示。**



2.1 语言概述

- **语言——形式化的内容提取**
 - **语言(Language): 满足一定条件的句子集合**
 - **句子(Sentence): 满足一定规则的单词序列**
 - **单词(Token): 满足一定规则的字符串**



2.1 语言概述

- **程序设计语言——形式化的内容提取**
 - 程序设计语言(Programming Language): 组成程序的**所有语句**的集合。
 - 程序(Program): 满足**语法规则**的语句序列。
 - 语句(Sentence): 满足**语法规则**的单词序列。
 - 单词(Token): 满足**词法规则**的字符串。
- **例:**
 - 赋值语句: 变量:=表达式
 - 条件语句: if 条件表达式 then 语句
 - 循环语句: while 条件表达式 do 语句
 - 过程调用语句: call 过程名(参数表)

2.1 语言概述

- **描述形式**

- **词法**

- **单词的组成**

- **描述方法：BNF范式、正规式**

- **语法——语句**

- **语句的组成规则**

- **描述方法：BNF范式、语法(描述)图**

约翰 巴克斯，计算机科学家，哥伦比亚大学数学硕士学位，1977年图灵奖得主，FORTRAN语言之父

彼得 诺尔，计算机科学家，哥本哈根大学天文学博士学位，2005年图灵奖得主，ALGOL 60发明者

形式语言与自动机理论的产生与作用

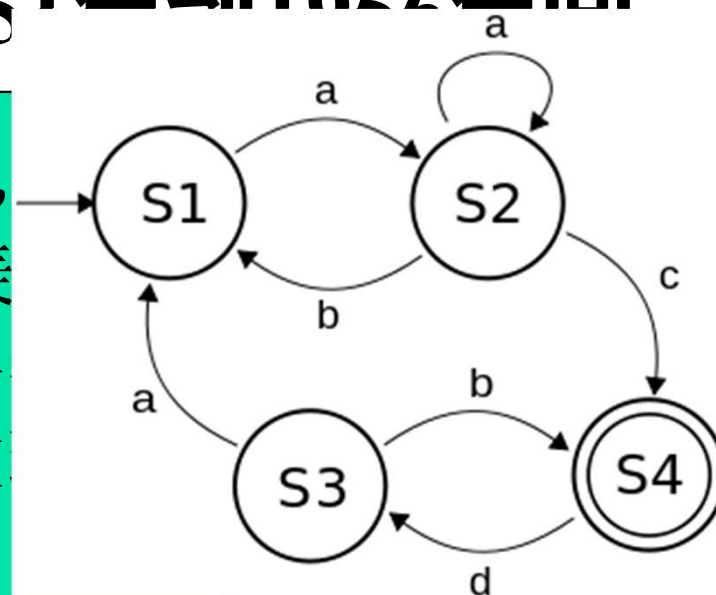
■ 语言学家Chomsky最初从产生语言的角度研究语言。

- 1955年宾夕法尼亚大学语言学博士，1955年执教麻省理工学院，著作《生成语法》被认为是20世纪理论语言学上最伟大的贡献。该语法定义为是由一
- 在所能语言。

形式语言与自动机理论的产生与作用

■ 数理逻辑学家Kleene在1951年到1954年间

Kleene(克林), 数理逻辑学家, 1934年普林斯顿大学博士, 美国科学院院士、美国艺术与科学院院士, 主要贡献在递归函数和有效的可计算性方面



从一

种
的任
语

语言由该自动机所能识别的所有句子组成。



形式语言与自动机理论的产生与作用

- 1959年，Chomsky通过深入研究，将他本人的研究成果与Kleene的研究成果结合了起来，不仅确定了文法和自动机分别从生成和识别的角度去表达语言，而且**证明了文法与自动机的等价性。**



形式语言与自动机理论的产生与作用

- 20世纪50年代，人们用**巴科斯范式**（Backus Nour Form 或 Backus Normal Form，简记为 BNF）成功地对高级语言ALGOL-60进行了描述。
- 实际上，巴科斯范式就是上下文无关文法（Context Free Grammar）的一种表示形式。这一成功，使得形式语言在20世纪60年代得到了大力的发展。



形式语言与自动机理论的产生与作用

- 形式语言与自动机理论除了在计算机科学领域中的直接应用外，更在计算学科人才的**计算思维的培养**中占有极其重要的地位
- 计算思维能力的培养，主要是由基础理论系列课程实现的，该系列主要由从**数学分析**开始到**形式语言**结束的一些数学和抽象程度比较高的内容的课程组成。
 - 它们构成的是一个梯级训练系统。在此系统中，**连续数学**、**离散数学**、**计算模型**等三部分内容要按阶段分开，三个阶段对应与本学科的学生在大学学习期间的思维方式和能力的变化与提高过程的三个步骤。

2.2 基本定义

- 定义2.1 **字母表** (Alphabet) Σ 是一个非空有穷集合，字母表中的元素称为该字母表的一个字母 (Letter)，也叫字符 (Character)
- 例 以下是不同的字母表：
 - $\{a, b, c, d\}$
 - $\{a, b, c, \dots, z\}$
 - $\{0, 1\}$
 - **ASCII字母表**

字符具有如下特点：
整体性：不可划分
可辨认性：可区分，
两两不同，明确区分



2.2 基本定义

- **定义2.2** 设 Σ_1 、 Σ_2 是两个字母表, Σ_1 与 Σ_2 的**乘积** (Product)定义为 $\Sigma_1\Sigma_2=\{ab|a\in\Sigma_1, b\in\Sigma_2\}$
- **例:** $\Sigma_1=\{0,1\}$, $\Sigma_2=\{a,b\}$, $\Sigma_1\Sigma_2=\{0a,0b,1a,1b\}$
- **定义2.3** 设 Σ 是一个字母表, Σ 的 **n 次幂**(Power)递归地定义为:
 - (1) $\Sigma^0=\{\varepsilon\}$
 - (2) $\Sigma^n=\Sigma^{n-1}\Sigma \quad n\geq 1$
- **例:** $\Sigma_1^3=\{000,001,010,011,100,101,110,111\}$

2.2 基本定义

- **定义2.4** 设 Σ 是一个字母表, Σ 的**正闭包** (Positive Closure)定义为:

- $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \dots$

- Σ 的**克林闭包** (Kleene Closure)为:

- $\Sigma^* = \Sigma^0 \cup \Sigma^+$

- $= \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots$

克林闭包也
包括空串



2.2 基本定义

■ 例

$$\{0,1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \dots\}$$

$$\{a, b, c, d\}^+ = \{a, b, c, d, aa, ab, ac, ad, ba, bb, bc, bd, \dots, aaa, aab, aac, aad, aba, abb, abc, \dots\}$$



2.2 基本定义

■ 例

$$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 11, 000, 001, 010, 011, 100, \dots\}$$

$$\{a, b, c, d\}^* = \{\epsilon, a, b, c, d, aa, ab, ac, ad, ba, bb, bc, bd, \dots, aaa, aab, aac, aad, aba, abb, abc, \dots\}$$



2.2 基本定义

- 定义2.5 设 Σ 是一个字母表, $\forall x \in \Sigma^*$, x 称为字母表 Σ 上的一个**句子** (sentence), ε 叫做 Σ 上的一个**空句子**。
- 定义2.6 设 Σ 是一个字母表, 对任意的 $x, y \in \Sigma^*$, $a \in \Sigma$, 句子 xay 中的 a 叫做字符 a 在该句子中的一个**出现**(occurrence)。



2.2 基本定义

■ **定义2.7** 设 Σ 是一个字母表, $\forall x \in \Sigma^*$, 句子 x 中字符**出现**的总个数叫做该字符串的**长度** (length), 记作 $|x|$ 。

■ **字母表** $\Sigma = \{a, b\}$

■ $|\varepsilon| = 0$

■ $|a| = 1$

■ $|b| = 1$

■ $|ababaa| = 6$



2.2 基本定义

■ **定义2.8** 设 Σ 是一个字母表, $\forall x, y \in \Sigma^*$, x, y 的**并置**(concatenation)是由串 x 直接连接串 y 所组成的。记作 xy 。并置也叫做**联结**。

■ $x=001, xx=x^2=001\ 001$

■ 对于 $n \geq 0$, 串 x 的 n 次幂(power)定义为:

■ (1) $x^0 = \varepsilon$;

■ (2) $x^n = x^{n-1}x$ 。

2.2 基本定义

■ 定义2.9 设 Σ 是一个字母表, 对 $\forall x, y, z \in \Sigma^*$, 如果 $x=yz$, 则称 y 是 x 的**前缀**(prefix), 如果 $z \neq \varepsilon$, 则称 y 是 x 的**真前缀**(proper prefix); z 是 x 的**后缀**(suffix), 如果 $y \neq \varepsilon$, 则称 z 是 x 的**真后缀**(proper suffix)。

- 字母表 $\Sigma=\{a, b\}$ 上的句子 $abaabb$ 的前缀、后缀、真前缀和真后缀如下:
- 前缀: $\varepsilon, a, ab, aba, abaa, abaab, abaabb$
- 真前缀: $\varepsilon, a, ab, aba, abaa, abaab$
- 后缀: $\varepsilon, b, bb, abb, aabb, baabb, abaabb$
- 真后缀: $\varepsilon, b, bb, abb, aabb, baabb$



2.2 基本定义

- **定义2.10** 设 Σ 是一个字母表, 对 $\forall x, y, z, w, v \in \Sigma^*$
 - 如果 $x=yz, w=yv$, 则称 y 是 x 和 w 的**公共前缀**(common prefix), 如果 x 和 w 的任何公共前缀都是 y 的前缀, 则称 y 是 x 和 w 的**最大公共前缀**(maximal common prefix)。
 - 如果 $x=zy, w=vy$, 则称 y 是 x 和 w 的**公共后缀**(common suffix), 如果 x 和 w 的任何公共后缀都是 y 的后缀, 则称 y 是 x 和 w 的**最大公共后缀**(maximal common suffix)。



2.2 基本定义

- **定义2.11** 设 Σ 是一个字母表, 对 $\forall w, x, y, z \in \Sigma^*$, 如果 $w=xyz$, 则称 y 是 w 的**子串**。
- **定义2.12** 设 Σ 是一个字母表, 对 $\forall t, u, v, w, x, y, z \in \Sigma^*$
 - 如果 $t=uyv, w=xyz$, 则称 y 是 t 和 w 的**公共子串** (common substring)。
 - 如果 y_1, y_2, \dots, y_n 是 t 和 w 的公共子串, 且 $|y_j| = \max\{|y_1|, |y_2|, \dots, |y_n|\}$, 则称 y_j 是 t 和 w 的**最大公共子串** (maximal common substring)。



2.2 基本定义

■ **定义2.13** 设 Σ 是一个字母表, $\forall L \subseteq \Sigma^*$, L 称为字母表 Σ 上的一个**语言** (Language), $\forall x \in L$, x 叫做 L 的一个**句子**。

■ **例: 字母表 $\{0, 1\}$ 上的语言**

$\{0, 1\}$

$\{00, 11\}$

$\{0, 1, 00, 11\}$

$\{0, 1, 00, 11, 01, 10\}$

$\{00, 11\}^*$

$\{01, 10\}^*$



2.3 文法的定义

如何实现语言结构的 形式化描述？

考虑赋值语句的形式：

左部量 = 右部表达式



2.3 文法的定义

左部量：简单变量或下标变量等，假设简单变量包括a、b、c，下标变量包括：m[1]、m[2]、m[3]

右部表达式：由+和-运算符连接的两个简单变量或下标变量

考虑赋值语句的形式：

左部量 = 右部表达式

合法的赋值语句：

$a = a + a$

$b = m[3] + b$

$m[1] = a + m[2]$



2.3 文法的定义

左部量：简单变量或下标变量等，假设简单变量包括a、b、c，下标变量包括：m[1]、m[2]、m[3]

右部表达式：由+和-运算符连接的两个简单变量或下标变量

考虑赋值语句的形式：

左部量 = 右部表达式

不合法的赋值语句：

$a = b + d$

$b = a + b + m[2]$

$a = b$

句子的组成规则

箭头表示“定义为”，
或者左侧可以表示
为箭头右侧的形式

■ $\langle \text{赋值语句} \rangle \rightarrow \langle \text{左部量} \rangle = \langle \text{右部表达式} \rangle$

■ $\langle \text{左部量} \rangle \rightarrow \langle \text{简单变量} \rangle$

■ $\langle \text{左部量} \rangle \rightarrow \langle \text{下标变量} \rangle$

■ $\langle \text{简单变量} \rangle \rightarrow a$

■ $\langle \text{简单变量} \rangle \rightarrow b$

■ $\langle \text{简单变量} \rangle \rightarrow c$

■ $\langle \text{下标变量} \rangle \rightarrow m[1]$

■ $\langle \text{下标变量} \rangle \rightarrow m[2]$

■ $\langle \text{下标变量} \rangle \rightarrow m[3]$

左部量：简单变量或下标变量等，假设简单变量包括a、b、c，下标变量包括：m[1]、m[2]、m[3]

句子的组成规则

- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle$
- $\langle \text{运算符} \rangle \rightarrow +$
- $\langle \text{运算符} \rangle \rightarrow -$

右部表达式：由+和-运算符连接的两个简单变量/下标变量

定义句子的规则的语法组成

——终结符号集，非终结符号集，语法规则，开始符号

非终结符号集 $V =$

每个非终结符号对应一个元素集合

$\{ \langle \text{赋值语句} \rangle, \langle \text{左部量} \rangle, \langle \text{右部表达式} \rangle, \langle \text{简单变量} \rangle, \langle \text{下标变量} \rangle, \langle \text{运算符} \rangle \}$

终结符号集 $T =$

$\{ a, b, c, m[1], m[2], m[3], +, - \}$

每个终结符号对应一个元素

语法规则集 $P =$

$\{ \langle \text{赋值语句} \rangle \rightarrow \langle \text{左部量} \rangle = \langle \text{右部表达式} \rangle, \dots \}$

开始符号 $S = \langle \text{赋值语句} \rangle$

每个语法规则表示产生句子的过程



文法G的形式定义

定义2.16 文法G为一个四元组:

$$G = (V, T, P, S)$$

- **V : 非终结符集**
 - 每个非终结符称为一个语法变量——代表某个语言的各种子结构
- **T : 终结符集**
 - 语言的句子中出现的字符, $V \cap T = \emptyset$
- **S : 开始符号, $S \in V$**
 - 代表文法所定义的语言, 至少在产生式左侧出现一次



文法G的形式定义

■ P : 产生式(Product)集合

$\alpha \rightarrow \beta$, 被称为产生式 (也称为定义式), 读作:
 α 定义为 β 。其中 $\alpha \in (V \cup T)^+$, 且 α 中至少有 V
中元素之一出现。 $\beta \in (V \cup T)^*$ 。

α 称为产生式 $\alpha \rightarrow \beta$ 的左部(Left Part)

β 称为产生式 $\alpha \rightarrow \beta$ 的右部(Right Part)。

产生式定义各个语法成分的结构 (组成规则)

例2.9 赋值语句的文法

- $V = \{ \langle \text{赋值语句} \rangle, \langle \text{左部量} \rangle, \langle \text{右部表达式} \rangle, \langle \text{简单变量} \rangle, \langle \text{下标变量} \rangle, \langle \text{运算符} \rangle \}$
- $T = \{ a, b, c, m[1], m[2], m[3], +, - \}$
- $P = \{ \langle \text{赋值语句} \rangle \rightarrow \langle \text{左部量} \rangle = \langle \text{右部表达式} \rangle, \langle \text{左部量} \rangle \rightarrow \langle \text{简单变量} \rangle, \langle \text{左部量} \rangle \rightarrow \langle \text{下标变量} \rangle, \langle \text{简单变量} \rangle \rightarrow a, \langle \text{简单变量} \rangle \rightarrow b, \langle \text{简单变量} \rangle \rightarrow c, \langle \text{下标变量} \rangle \rightarrow m[1], \langle \text{下标变量} \rangle \rightarrow m[2], \langle \text{下标变量} \rangle \rightarrow m[3], \langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle, \langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle, \langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle, \langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle, \langle \text{运算符} \rangle \rightarrow +, \langle \text{运算符} \rangle \rightarrow - \}$
- $S = \langle \text{赋值语句} \rangle$



例2.9 赋值语句的文法（续）

- 符号化之后:
- $V = \{$
 - <赋值语句> (表示为A)
 - <左部量> (表示为B)
 - <右部表达式> (表示为E)
 - <简单变量> (表示为C)
 - <下标变量> (表示为D)
 - <运算符> (表示为O)}



例2.9 赋值语句的文法（续）

- 符号化之后:

- $G = (\{A, B, E, C, D, O\}, \{a, b, c, m[1], m[2], m[3], +, -\}, P, A)$,

其中, $P = \{A \rightarrow B = E, B \rightarrow C, B \rightarrow D, C \rightarrow a, C \rightarrow b, C \rightarrow c, D \rightarrow m[1], D \rightarrow m[2], D \rightarrow m[3], E \rightarrow COC, E \rightarrow COD, E \rightarrow DOC, E \rightarrow DOD, O \rightarrow +, O \rightarrow -\}$



产生式的简写

- 对一组有相同左部的产生式

$$\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$$

可以简单地记为：

$$\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

读作： α 定义为或者 β_1 ，或者 β_2 ， \dots ，或者 β_n 。
并且称它们为 α 产生式。 $\beta_1, \beta_2, \dots, \beta_n$ 称为
候选式(Candidate)。

产生式的简写

- 对于一个文法 $G=(V, T, P, S)$, 可以只列出该文法的所有产生式, 且所列的**第一个产生式的左部**是该文法的**开始符号**。

赋值语句文法:

$G1: A \rightarrow B = E$

$B \rightarrow C \mid D$

$C \rightarrow a \mid b \mid c$

$D \rightarrow m[1] \mid m[2] \mid m[3]$

$E \rightarrow COC \mid COD \mid DOC \mid DOD$

$O \rightarrow + \mid -$

一般来讲, 大写字母表示语法变量, 小写字母表示终结符号



产生式的简写

**问题：有了定义句子的规则（文法），
如何判定某一句子是否属于某语言？**

句子的推导

---从产生语言的角度

<赋值语句>

\Rightarrow <左部量> = <右部表达式>

\Rightarrow <简单变量> = <右部表达式>

\Rightarrow a = <右部表达式>

\Rightarrow a = <简单变量> <运算符> <简单变量>

\Rightarrow a = a <运算符> <简单变量>

\Rightarrow a = a + <简单变量>

\Rightarrow a = a + a

推导可以
看作是用
产生式的
右部替换
左部

句子的归约

---从识别语言的角度

$a = a + a$

$\Leftarrow a = a + \langle \text{简单变量} \rangle$

$\Leftarrow a = a \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$

$\Leftarrow a = \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$

$\Leftarrow a = \langle \text{右部表达式} \rangle$

$\Leftarrow \langle \text{简单变量} \rangle = \langle \text{右部表达式} \rangle$

$\Leftarrow \langle \text{左部量} \rangle = \langle \text{右部表达式} \rangle$

$\Leftarrow \langle \text{赋值语句} \rangle$

归约可以看作
是用产生式的
左部替换右部

■ 派生与识别均根据语法规则

基于产生式的变换--推导或归约

- 定义2.17 设 $G=(V, T, P, S)$ 是一个文法, 如果 $\alpha \rightarrow \beta \in P$, $\gamma, \delta \in (V \cup T)^*$, 则称 $\gamma\alpha\delta$ 在 G 中**直接推导出** $\gamma\beta\delta$, 记作:

$\gamma\alpha\delta \xRightarrow{G} \gamma\beta\delta$ (意义明确时, 也可以省略箭头下的 G)

- 读作: $\gamma\alpha\delta$ 在文法 G 中**直接推导出** $\gamma\beta\delta$ 。
- 在不特别强调推导的直接性时, “直接推导”可以简称为推导(derivation)或派生。将 $\gamma\alpha\delta$ 中的 α 变成了 β , γ 和 δ 都没有变化, 所以又称将 α 推导成 β 。

基于产生式的变换--推导或归约

- 定义2.17 设 $G=(V, T, P, S)$ 是一个文法, 如果 $\alpha \rightarrow \beta \in P$, $\gamma, \delta \in (V \cup T)^*$, 则称 $\gamma\beta\delta$ 在 G 中**直接归约**出 $\gamma\alpha\delta$, 记作:
 $\gamma\beta\delta \xleftarrow{G} \gamma\alpha\delta$ (意义明确时, 也可以省略箭头下的 G)
- 读作: $\gamma\beta\delta$ 在文法 G 中**直接归约**成 $\gamma\alpha\delta$ 。
- 在不特别强调归约的直接性时, “直接归约”可以简称为**归约**(reduction)。将 $\gamma\beta\delta$ 中的 β 变成了 α , γ 和 δ 都没有变化, 所以又称将 β 归约成 α 。



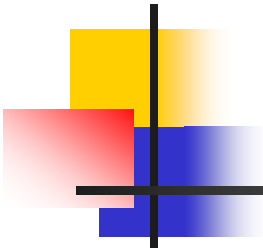
(多步)推导

- $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$

- 记为 $\alpha_0 \xRightarrow{n} \alpha_n$ (恰用 n 步推导)

- $\alpha_0 \xRightarrow{+} \alpha_n$ (至少一步推导)

- $\alpha_0 \xRightarrow{*} \alpha_n$ (若干步推导: 零步或多步)



(多步)归约

- $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$

- 记为 $\alpha_n \stackrel{n}{\Leftarrow} \alpha_0$ (恰用 n 步归约)

- $\alpha_n \stackrel{+}{\Leftarrow} \alpha_0$ (至少一步归约)

- $\alpha_0 \stackrel{*}{\Leftarrow} \alpha_n$ (若干步归约: 零步或多步)

文法 G 产生的语言

- 设 $G=(V, T, P, S)$ 是一个文法, 对于 $\forall A \in V$, 令

$$L(A) = \{x \mid A \xRightarrow{+} x \text{ \& } x \in T^*\}$$

$L(A)$ 就是语法变量 A 所代表的集合。

- 定义2.19 设有文法 $G=(V, T, P, S)$, 则称

$$L(G) = \{w \mid S \xRightarrow{*} w \text{ \& } w \in T^* \}$$

为文法 G 产生的**语言(language)**。

$\forall w \in L(G)$, w 称为 G 产生的一个**句子(sentence)**。

显然, 对于任意一个文法 G , G 产生的语言 $L(G)$ 就是该文法的**开始符号 S** 所对应的集合 $L(S)$ 。



文法 G 产生的语言(续)

$$L(G) = \{x \mid S \Rightarrow^* x \text{ and } x \in T^*\}$$

- 文法 G 可以派生出多少个句子?
- 文法 G 的作用——语言的有穷描述
 - 以有限的规则描述无限的语言现象
- 有限:
 - 产生式集合、终结符集合、非终结符集合
- 无限:
 - 可以导出无穷多个句子

推导/归约举例

$G1: A \rightarrow B=E$

$B \rightarrow C \mid D$

$C \rightarrow a \mid b \mid c$

$D \rightarrow m[1] \mid m[2] \mid m[3]$

$E \rightarrow COC \mid COD \mid DOC \mid DOD$

$O \rightarrow + \mid -$

证明 $a=b+m[1]$ 是文法 $G1$ 的句子

- | | |
|------------------------|---|
| $A \Rightarrow B=E$ | 有产生式 $A \rightarrow B=E$, 可以将 A 换成 $B=E$ |
| $\Rightarrow C=E$ | 有产生式 $B \rightarrow C$, 可以将 $B=E$ 中的 B 换成 C |
| $\Rightarrow a=E$ | 有产生式 $C \rightarrow a$, 可以将 $C=E$ 中的 C 换成 a |
| $\Rightarrow a=COD$ | 有产生式 $E \rightarrow COD$, 可以将 $a=E$ 中的 E 换成 COD |
| $\Rightarrow a=bOD$ | 有产生式 $C \rightarrow b$, 可以将 $a=COD$ 中的 C 换成 b |
| $\Rightarrow a=b+D$ | 有产生式 $O \rightarrow +$, 可以将 $a=bOD$ 的 O 换成 $+$ |
| $\Rightarrow a=b+m[1]$ | 有产生式 $D \rightarrow m[1]$, 可以将 $a=b+D$ 的 D 换成 $m[1]$ |



推导/归约举例

$G1: A \rightarrow B = E$

$B \rightarrow C \mid D$

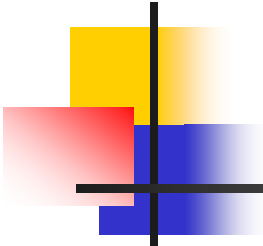
$C \rightarrow a \mid b \mid c$

$D \rightarrow m[1] \mid m[2] \mid m[3]$

$E \rightarrow COC \mid COD \mid DOC \mid DOD$

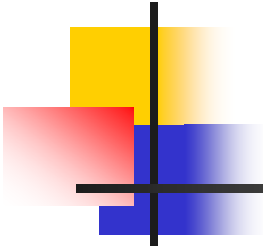
$O \rightarrow + \mid -$

归约操作: $m[1] = b + a$ (手动实现)



句型与句子

- **定义2.20** 设文法 $G=(V, T, P, S)$, 对于 $\forall \alpha \in (V \cup T)^*$, 如果 $S \xRightarrow{*} \alpha$, 则称 α 是 G 产生的一个**句型(sentential form)**
- 对于任意文法 $G=(V, T, P, S)$, G 产生的**句子 w 和句型 α 的区别** 在于句子满足 $w \in T^*$, 而句型满足 $\alpha \in (V \cup T)^*$



句型与句子

- 句子 w 是从 S 开始，在 G 中可以推导出来的终结符号行，它不含语法变量；
- 句型 α 是从 S 开始，在 G 中可以推导出来的符号行，它可能含有语法变量；
- 句子一定是句型，但句型不一定是句子



一些例子(1)

- $G = (\{S, A, B\}, \{0, 1\},$
 $\{S \rightarrow A \mid AB,$
 $A \rightarrow 0 \mid 0A,$
 $B \rightarrow 1 \mid 11\},$
 $S)$
- 给出该文法的推导过程的例子



一些例子(2)

- $G = (\{A\}, \{0, 1\},$
 $\{A \rightarrow 01,$
 $A \rightarrow 0A1\},$
 $A)$
- 给出该文法的推导过程的例子



2.4 文法的分类(Chomsky体系)

- **根据语言结构的复杂程度**
 - 涉及文法的复杂程度、分析方法的选择
 - 反映文法描述语言的能力
- **0型文法 (即：短语结构文法)**
- **1型文法 (即：上下文有关文法)**
- **2型文法 (即：上下文无关文法)**
- **3型文法 (即：正规文法)**



2.4 文法的分类(Chomsky体系)

- 如果 G 满足文法定义的要求, 则 G 是 **0 型文法** (短语结构文法PSG: Phrase Structure Grammar) 。
- $L(G)$ 为短语结构语言 (PSL) 。



1. 上下文有关文法(CSG)

- 如果对于 $\forall \alpha \rightarrow \beta \in P$, 均有 $|\beta| \geq |\alpha|$ 成立, 则称 G 为 **1型文法**
 - 即: 上下文有关文法 (CSG——Context Sensitive Grammar)
- $L(G)$ 为 1型/上下文有关语言(CSL)



2. 上下文无关文法(CFG)

- 如果对于 $\forall \alpha \rightarrow \beta \in P$, 均有 $|\beta| \geq |\alpha|$, 并且 $\alpha \in V^+$ 成立, 则称 G 为 **2型文法**
 - 即: 上下文无关文法 (CFG: Context Free Grammar)
- $L(G)$ 为 **2型/上下文无关语言 (CFL)**
 - CFG 能描述程序设计语言的 **多数语法成分**



3. 正规文法(RG)

- 设 $A, B \in V, \alpha \in T^+$
 - 右线性(Right Linear)文法: $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$
 - 左线性(Left Linear)文法: $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$
- 都是 **3 型文法**(正规文法 Regular Grammar - RG)
- $L(G)$ 为 3 型/正规集/正则集/正则语言 (RL)
 - 能描述程序设计语言的大多数单词



3. 正规文法(RG)

■ 左线性文法和右线性文法等价

- 文法等价：G1和G2等价，如果 $L(G1) = L(G2)$
- 左线性文法和右线性文法只是识别句子的方向不同
- $A \rightarrow w_1 B \quad B \rightarrow w_2$
- $A \rightarrow B w_2 \quad B \rightarrow w_1$



文法的分类

- **PSG: 短语结构文法, PSL: 短语结构语言**
 - **CSG: 上下文有关文法, CSL: 上下文有关语言**
 - **CFG: 上下文无关文法, CFL: 上下文无关语言**
 - **RG: 正规文法, RL: 正规语言**
-
- **$RG \subseteq CFG \subseteq CSG \subseteq PSG$**
 - **$RL \subseteq CFL \subseteq CSL \subseteq PSL$**



判断以下文法的类别

- $G1: S \rightarrow 0 \mid 1 \mid 00 \mid 11$ (正则文法)
- $G2: S \rightarrow A \mid B \mid AA \mid BB, A \rightarrow 0, B \rightarrow 1$ (上下文无关文法)
- $G3: S \rightarrow 0 \mid 1 \mid 0A \mid 1B, A \rightarrow 0, B \rightarrow 1$ (正则文法)
- $G4: S \rightarrow A \mid B \mid BC, A \rightarrow 0, B \rightarrow 1, C \rightarrow 21, C \rightarrow 11, C \rightarrow 2$
(上下文无关文法)
- $G5: S \rightarrow 0 \mid 0S$ (正则文法)
- $G6: S \rightarrow \varepsilon \mid 0S$ (短语结构文法)
- $G7: S \rightarrow \varepsilon \mid 00S111$ (短语结构文法)
- $G8: A \rightarrow aS \mid bS \mid cS \mid a \mid b \mid c$ (正则文法)



文法的分类

■ **G9: $S \rightarrow 0A \mid 1B \mid 2C \mid 0SA \mid 1SB \mid 2SC$**

$0A \rightarrow A0$ $1A \rightarrow A1$

$2A \rightarrow A2$ $0B \rightarrow B0$

$1B \rightarrow B1$ $2B \rightarrow B2$

$0C \rightarrow C0$ $1C \rightarrow C1$

$2C \rightarrow C2$

(上下文有关文法)

■ **G10: $S \rightarrow aT \mid bT \mid cT$**

$T \rightarrow \varepsilon \mid a \mid b \mid c \mid 0 \mid 1 \mid 2 \mid 3 \mid aT \mid bT \mid cT \mid 0T \mid 1T \mid 2T \mid 3T$

(短语结构文法)

文法的分类

- G6: $S \rightarrow \varepsilon \mid 0S$
- G7: $S \rightarrow \varepsilon \mid 00S111$
- G10: $S \rightarrow aT \mid bT \mid cT$

$A \rightarrow \varepsilon$ 型产生式的特殊处理

G6和G10是正则文法
G7是上下文无关文法

$T \rightarrow \varepsilon \mid a \mid b \mid c \mid 0 \mid 1 \mid 2 \mid 3 \mid aT \mid bT \mid cT \mid 0T \mid 1T \mid 2T \mid 3T$

$A \rightarrow \varepsilon$ 是一个特殊的产生式，它表示将A替换为空串

约定：RG、CFG、CSG可以含有形如 $A \rightarrow \varepsilon$ 的产生式



左线性文法和右线性文法

- 左线性文法和右线性文法只是识别句子的方向不同
- 构造语言 $\{0123456\}$ 的左线性文法和右线性文法



构造左线性文法

- 构造语言 $\{0123456\}$ 的左线性文法
- $G_1: S_1 \rightarrow A_1 6$
 - $A_1 \rightarrow B_1 5$
 - $B_1 \rightarrow C_1 4$
 - $C_1 \rightarrow D_1 3$
 - $D_1 \rightarrow E_1 2$
 - $E_1 \rightarrow F_1 1$
 - $F_1 \rightarrow 0$
- 利用该文法进行推导和归约的过程



构造右线性文法

- 构造语言 $\{0123456\}$ 的右线性文法
- $G_r: S_r \rightarrow 0A_r$
 $A_r \rightarrow 1B_r$
 $B_r \rightarrow 2C_r$
 $C_r \rightarrow 3D_r$
 $D_r \rightarrow 4E_r$
 $E_r \rightarrow 5F_r$
 $F_r \rightarrow 6$
- 利用该文法进行推导和归约的过程



Chomsky体系——总结

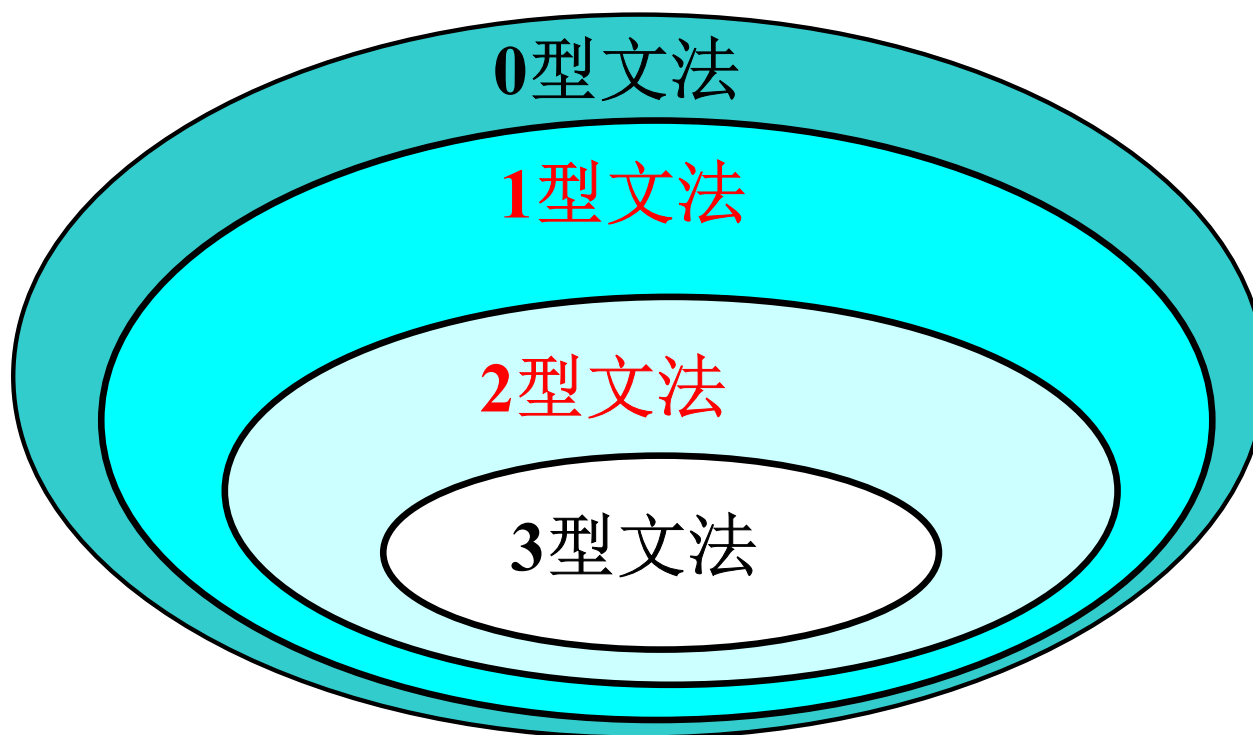
$G = (V, T, P, S)$ 是一个文法, $\alpha \rightarrow \beta \in P$

- * G 是0型文法, $L(G)$ 是0型语言;
- * $|\alpha| \leq |\beta|$: G 是1型文法, $L(G)$ 是1型语言(除 $S \rightarrow \varepsilon$);
- * $\alpha \in V$: G 是2型文法, $L(G)$ 是2型语言;
- * $A \rightarrow aB$ 或 $A \rightarrow a$: G 是右线性文法, $L(G)$ 是3型语言
 $A \rightarrow Ba$ 或 $A \rightarrow a$: G 是左线性文法, $L(G)$ 是3型语言

文法的类型

四种文法之间的关系是将产生式作进一步限制而定义的。

四种文法之间的逐级“包含”关系如下：



2.5 CFG的语法树

- **CFG: 上下文无关文法**

- 相对简单
- 描述能力也比较恰当
- 高级程序设计语言的绝大多数成分都可以用**CFG来描述**（可以看做问题1的解答）
- 语法成分是上下文无关的，即在其句型中总是存在这样的子串，该子串是由**某个变量**推导出来的（**可以看作问题2的部分解答，后面继续讨论**）

两个问题：

1. 为什么讨论CFG
2. 为什么讨论语法树

2.5 CFG的语法树

- 语法树的目的：希望**更清晰**地表示句型的文法结构以及推导和归约的过程

$G: A \rightarrow B=E$

$B \rightarrow C \mid D$

$C \rightarrow a \mid b \mid c$

$D \rightarrow m[1] \mid m[2] \mid m[3]$

$E \rightarrow COC \mid COD \mid DOC \mid DOD$

$O \rightarrow + \mid -$

$A \Rightarrow B=E$

$\Rightarrow C=E$

$\Rightarrow a=E$

$\Rightarrow a=COD$

$\Rightarrow a=bOD$

$\Rightarrow a=b+D$

$\Rightarrow a=b+m[1]$

如左侧的推导过程并不直观，图形格式的表达更清晰



2.5 CFG的语法树

- 再看上下文无关文法(CFG)的定义

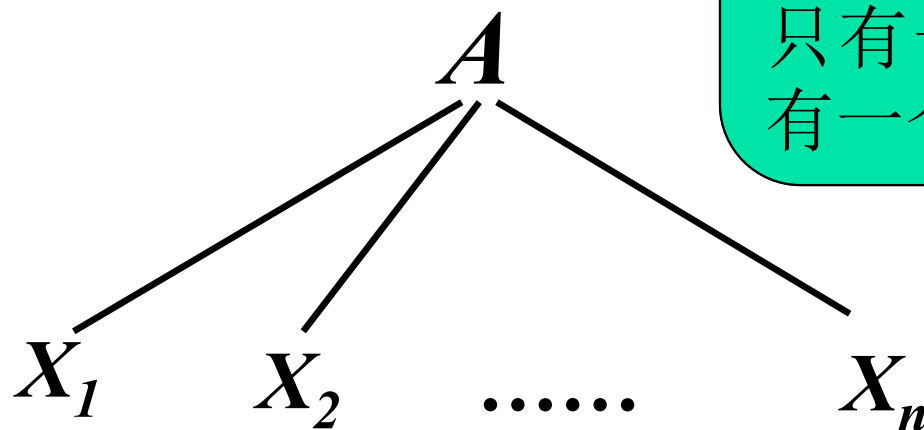
如果对于 $\forall \alpha \rightarrow \beta \in P$, 均有 $|\beta| \geq |\alpha|$, 并且 $\alpha \in V$ 成立, 则称 G 为 **2型文法**

- 即：上下文无关文法 (CFG: Context Free Grammar)

2.5 CFG的语法树

- $A \Rightarrow X_1 X_2 \dots X_n$

根据 CFG 的定义，
CFG 中每个产生式
 $A \rightarrow X_1 X_2 \dots X_n$ 的左侧
只有一个变量，右侧
有一个或多个变量

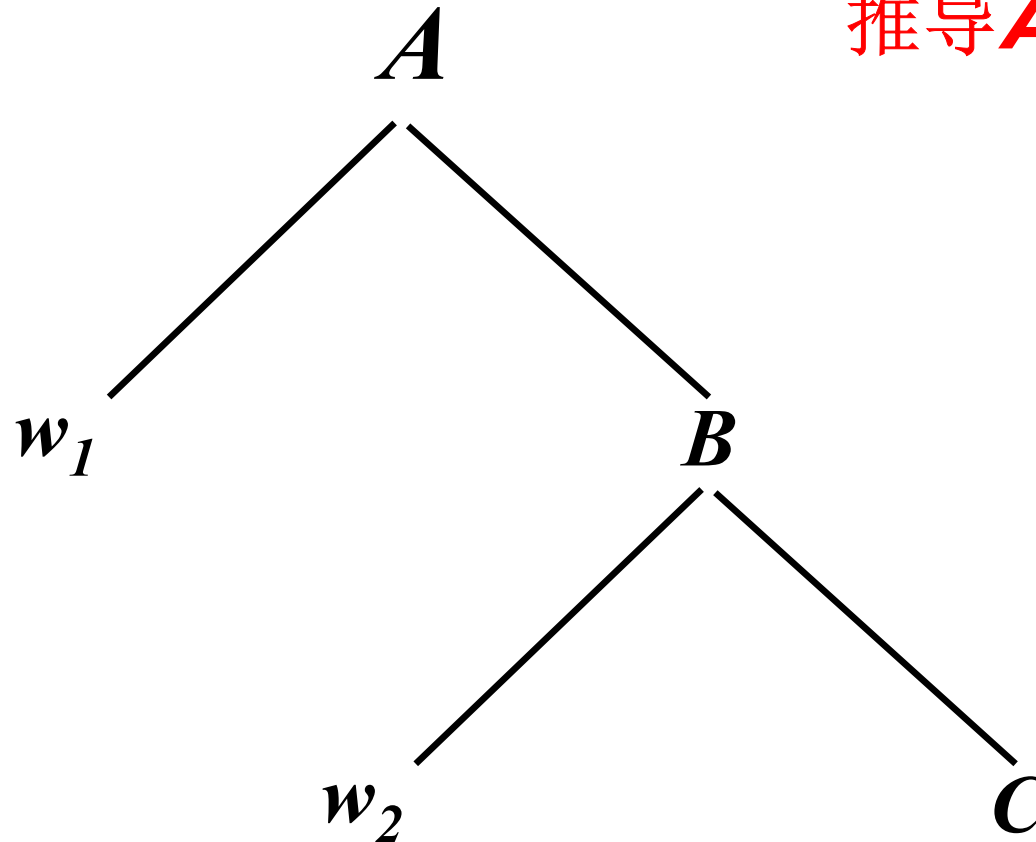


- 考虑上下文有关文法是否可以表示为语法树格式，例如 $AB \Rightarrow X_1 X_2 \dots X_n$

2.5 CFG的语法树

- $A \rightarrow w_1B, B \rightarrow w_2C$

推导 $A \Rightarrow w_1w_2C$





2.5 CFG的语法树

- 用**树**的形式表示**句型**的生成
 - 树根：开始符号
 - 中间结点：非终结符
 - 叶结点：终结符或者非终结符
- 每个推导对应一个中间结点及其儿子
- 又称为**分析树**(parse tree)、**推导树**(derivation tree)、**派生树**(derivation tree)

例子结构的表示

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$)

句子 **id + id * id** 的推导

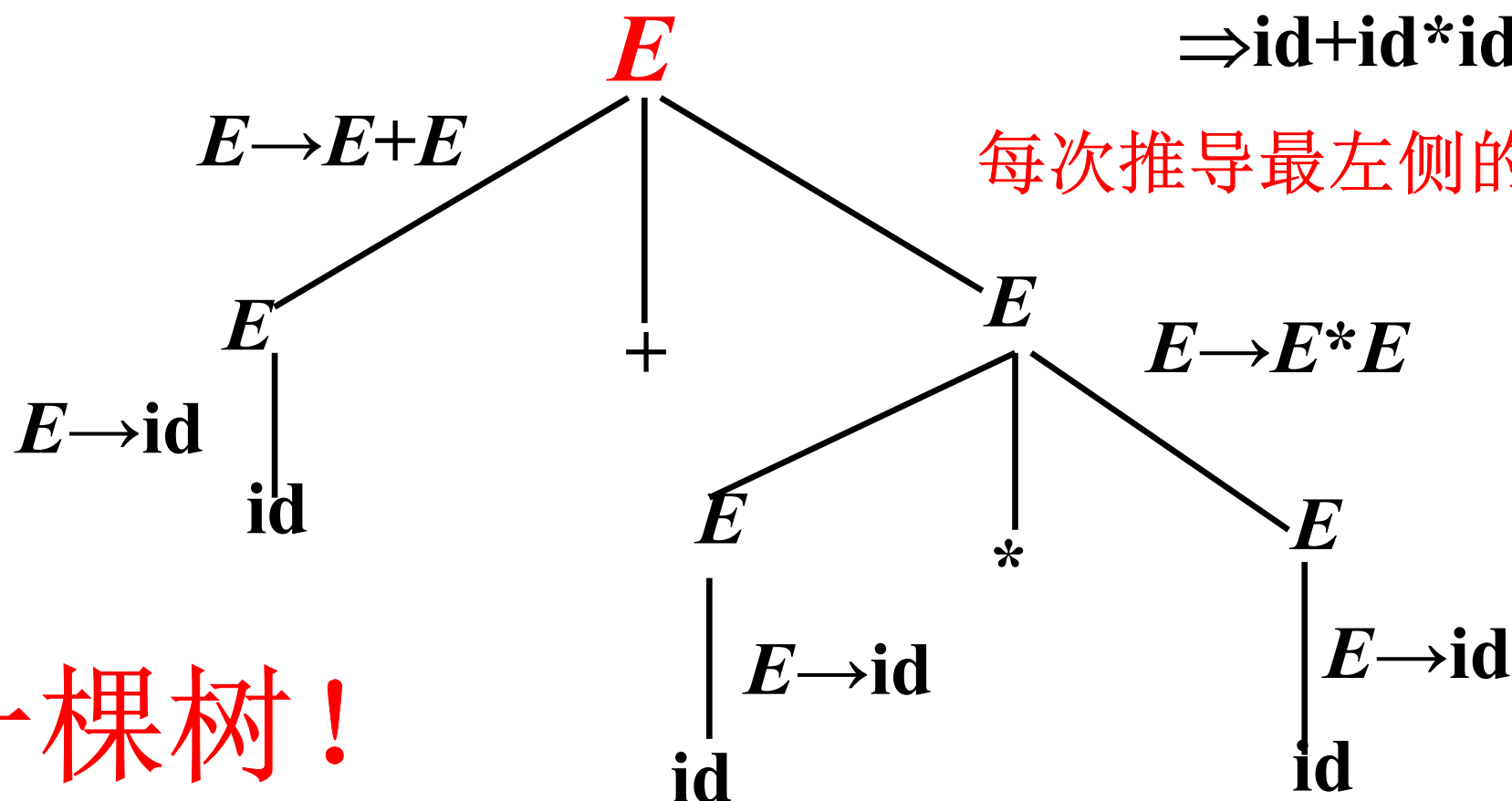
$$E \Rightarrow E + E$$

$$\Rightarrow \text{id} + E$$

$$\Rightarrow \text{id} + E * E$$

$$\Rightarrow \text{id} + \text{id} * E$$

$$\Rightarrow \text{id} + \text{id} * \text{id}$$



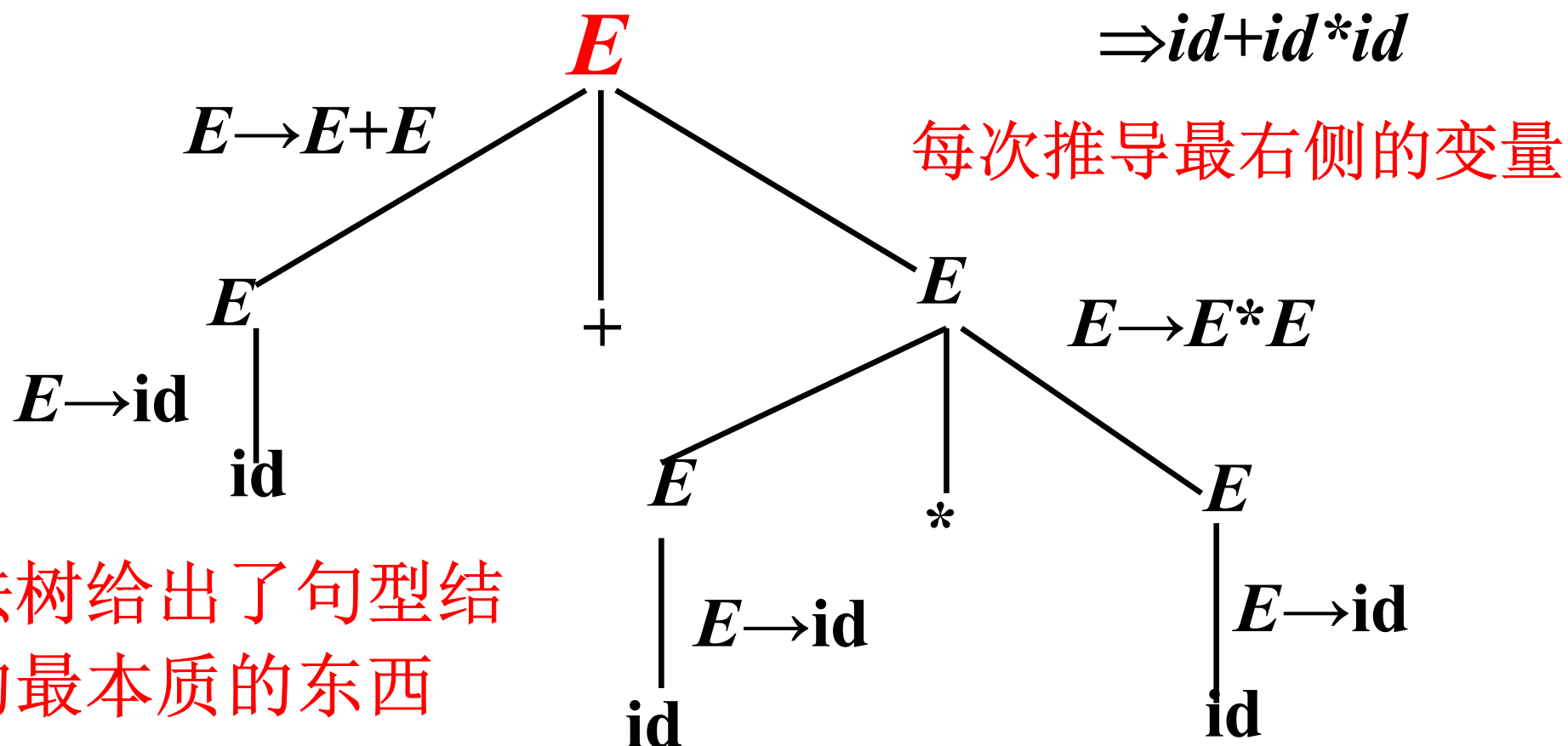
一棵树！

例 句子结构的表示

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid id$)

句子 **id + id * id** 的推导

$E \Rightarrow E + E$
 $\Rightarrow E + E * E$
 $\Rightarrow E + E * id$
 $\Rightarrow E + id * id$
 $\Rightarrow id + id * id$

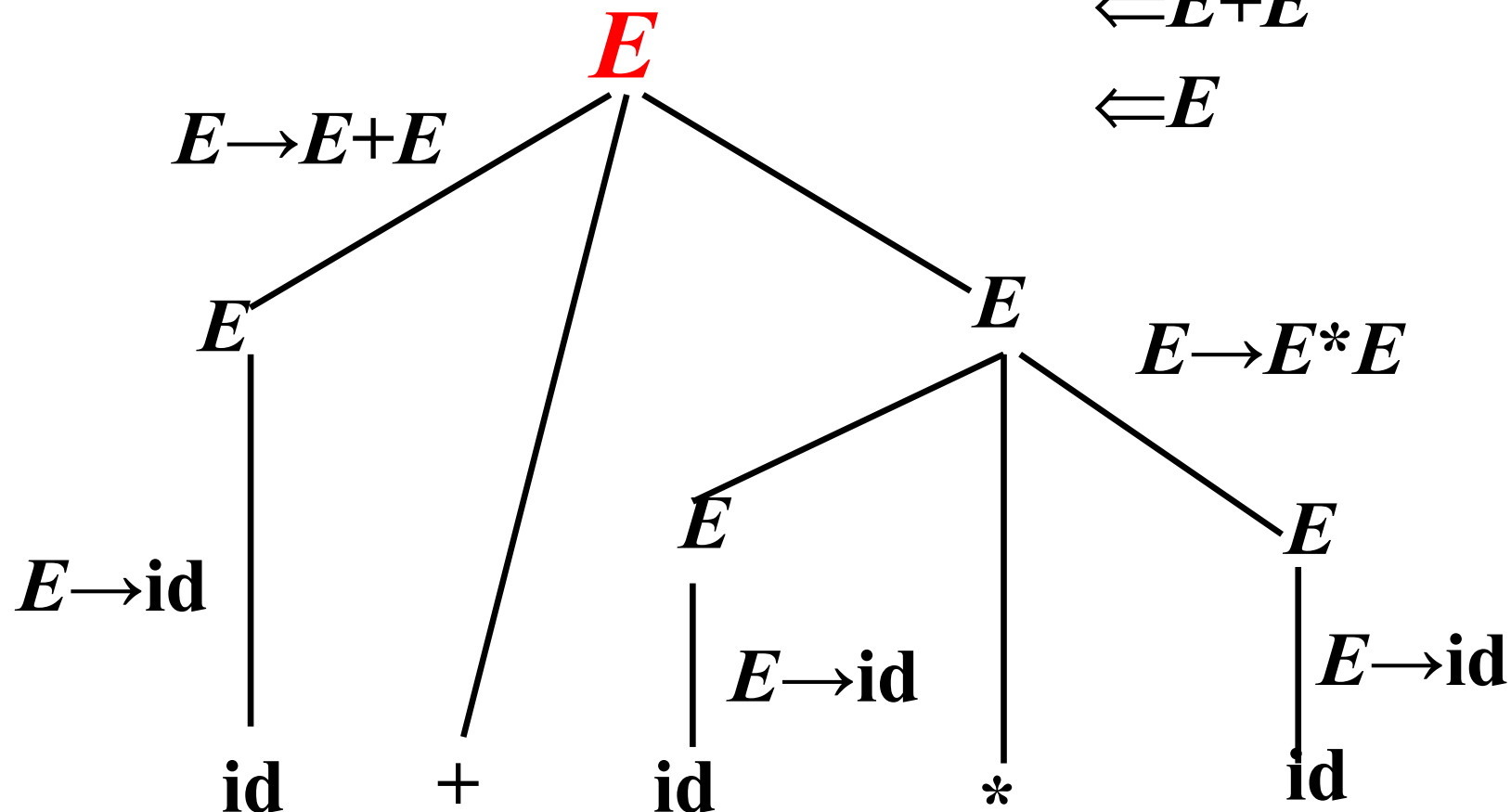


语法树给出了句型结构的最本质的东西

例 句子结构的表示

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid id$)

句子 **id + id * id** 的归约过程

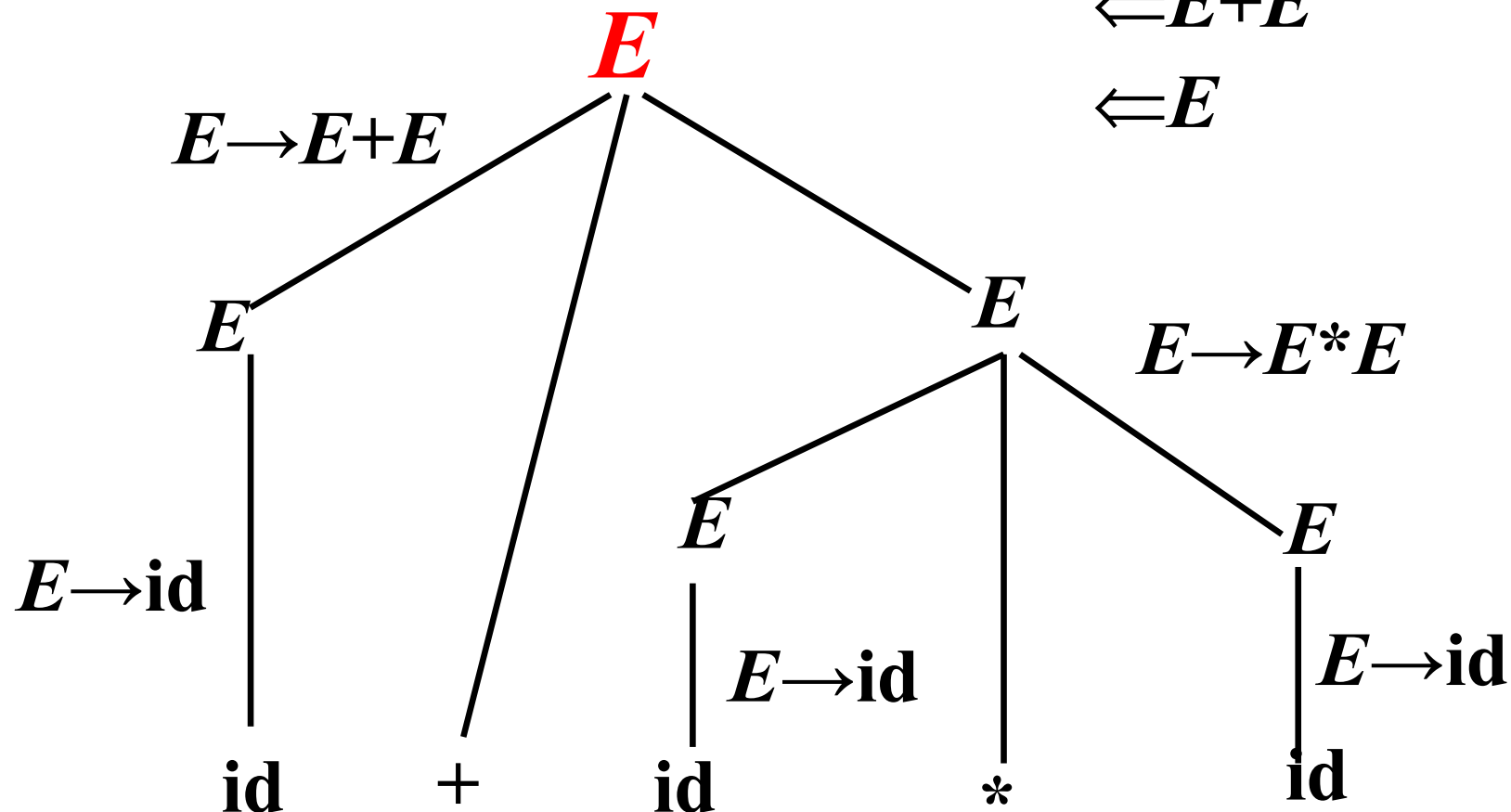


$id + id * id$
 $\Leftarrow E + id * id$
 $\Leftarrow E + E * id$
 $\Leftarrow E + E * E$
 $\Leftarrow E + E$
 $\Leftarrow E$

例 句子结构的表示

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid id$)

句子 **id + id * id** 的归约过程



$id + id * id$
 $\Leftarrow id + id * E$
 $\Leftarrow id + E * E$
 $\Leftarrow id + E$
 $\Leftarrow E + E$
 $\Leftarrow E$

短语(Phrase)

语法树用树的形式表示句型的生成，那么子树的结果是什么

■ **定义2.27** 设有CFG $G=(V, T, P, S)$, $\exists \alpha, \beta$, $\gamma \in (V \cup T)^*$, $S \xRightarrow{*} \gamma A \beta$, $A \xRightarrow{+} \alpha$, 则称 α 是句型 $\gamma \alpha \beta$ 的相对于变量 A 的**短语**(phrase);

如果此时有 $A \Rightarrow \alpha$, 则称 α 是句型 $\gamma \alpha \beta$ 的相对于变量 A 的**直接短语**(immediate phrase)。

短语可以看作是语法树中子树的结果。

短语和直接短语

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$)

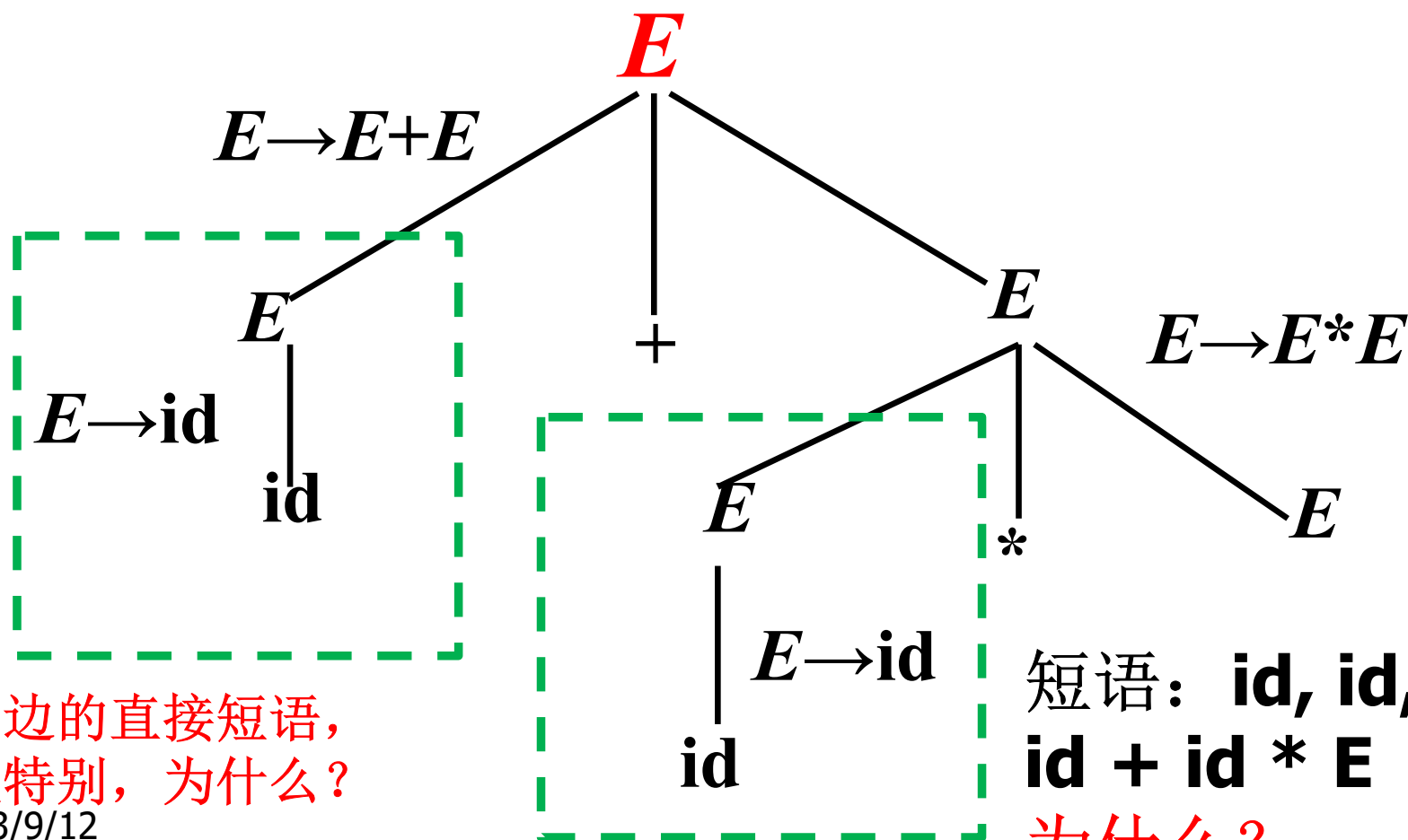
$$E \Rightarrow E + E$$

$$\Rightarrow \text{id} + E$$

$$\Rightarrow \text{id} + E * E$$

$$\Rightarrow \text{id} + \text{id} * E$$

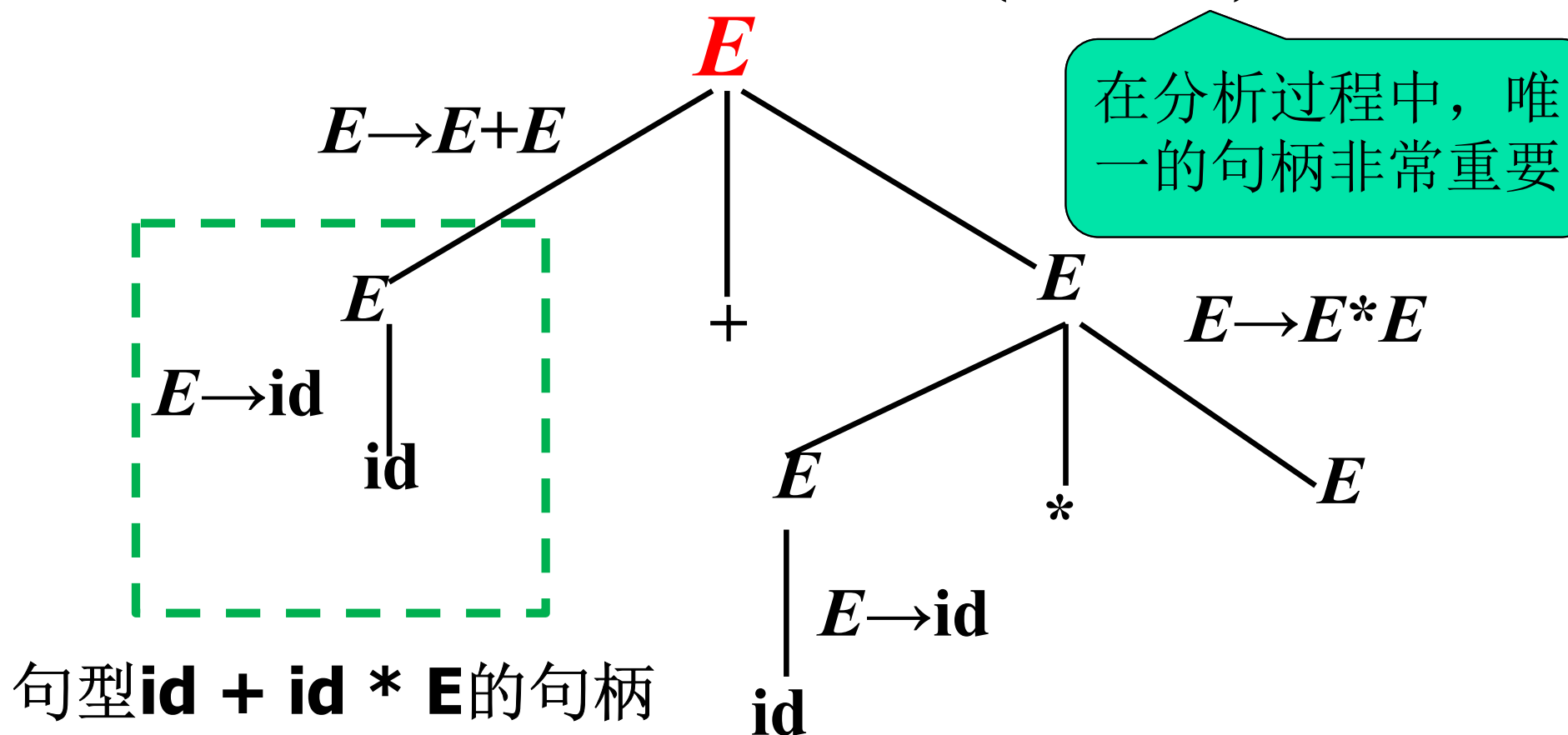
句型 $\text{id} + \text{id} * E$ 的短语和直接短语



2023/9/12

句柄(handle)

■ 定义2.28 设有CFG $G=(V, T, P, S)$, G 的句型的最左直接短语叫做句柄(handle)。





用子树解释短语，直接短语，句柄

- **短语**：一棵子树的所有叶子自左至右排列起来形成一个相对于子树根的短语。
- **直接短语**：仅有父子两代的一棵子树，它的所有叶子自左至右排列起来所形成的符号串。
- **句柄**：一个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列。

用子树解释短语，直接短语，句柄

- 例如，对表达式文法 $G[E]$ 和句子 $a_1 + a_2 * a_3$ ，挑选出推导过程中产生的句型中的短语，直接短语，句柄

表达式文法 $G[E]: E \rightarrow E + T \mid E - T \mid T$

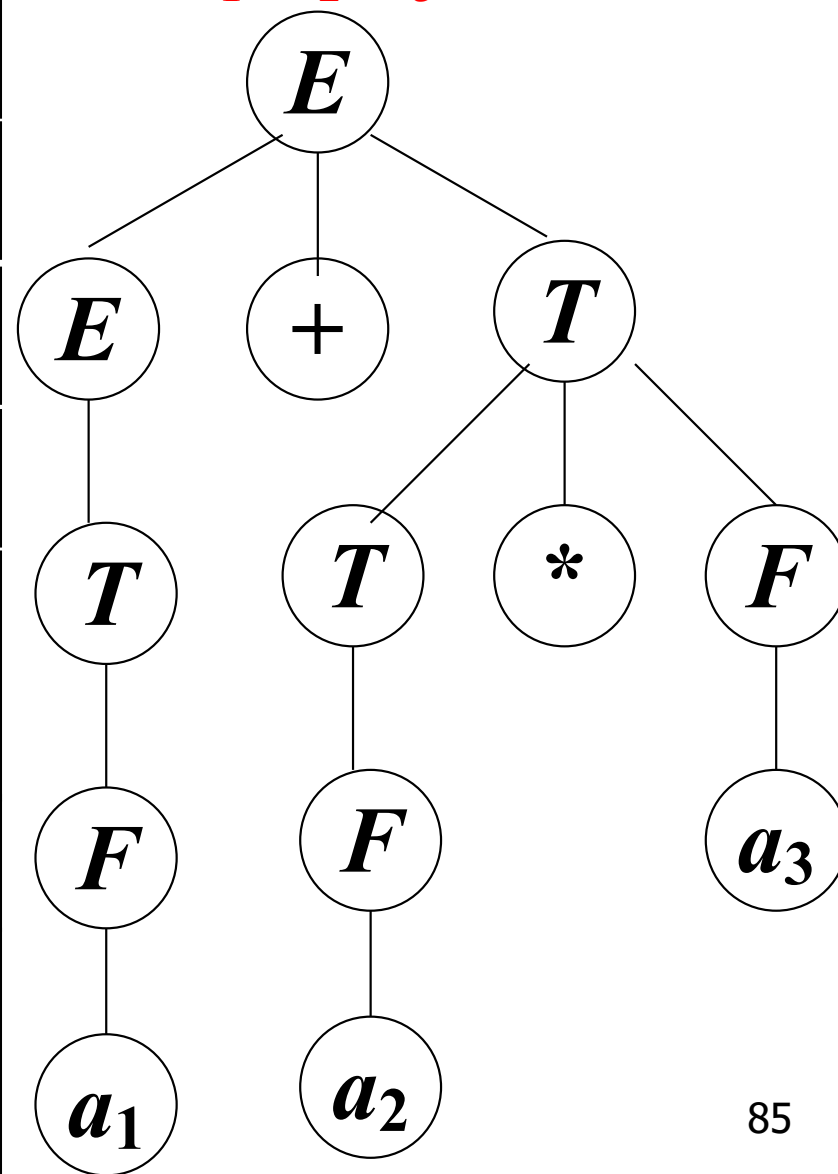
$T \rightarrow T * F \mid T / F \mid F$

$F \rightarrow a_1 \mid a_2 \mid a_3$

E	当前句型的短语
$\Rightarrow E+T$	$E+T$
$\Rightarrow T+T$	$T, T+T$
$\Rightarrow F+T$	$F, F+T$
$\Rightarrow a_1+T$	a_1, a_1+T
$\Rightarrow a_1+T^*F$	a_1, T^*F, a_1+T^*F
$\Rightarrow a_1+F^*F$	$a_1, F, F^*F,$ a_1+F^*F
$\Rightarrow a_1+a_2^*F$	$a_1, a_2, a_1+a_2^*F, a_2^*F$
$\Rightarrow a_1+a_2^*a_3$	$a_1, a_2, a_3, a_2^*a_3$ $a_1+a_2^*a_3$

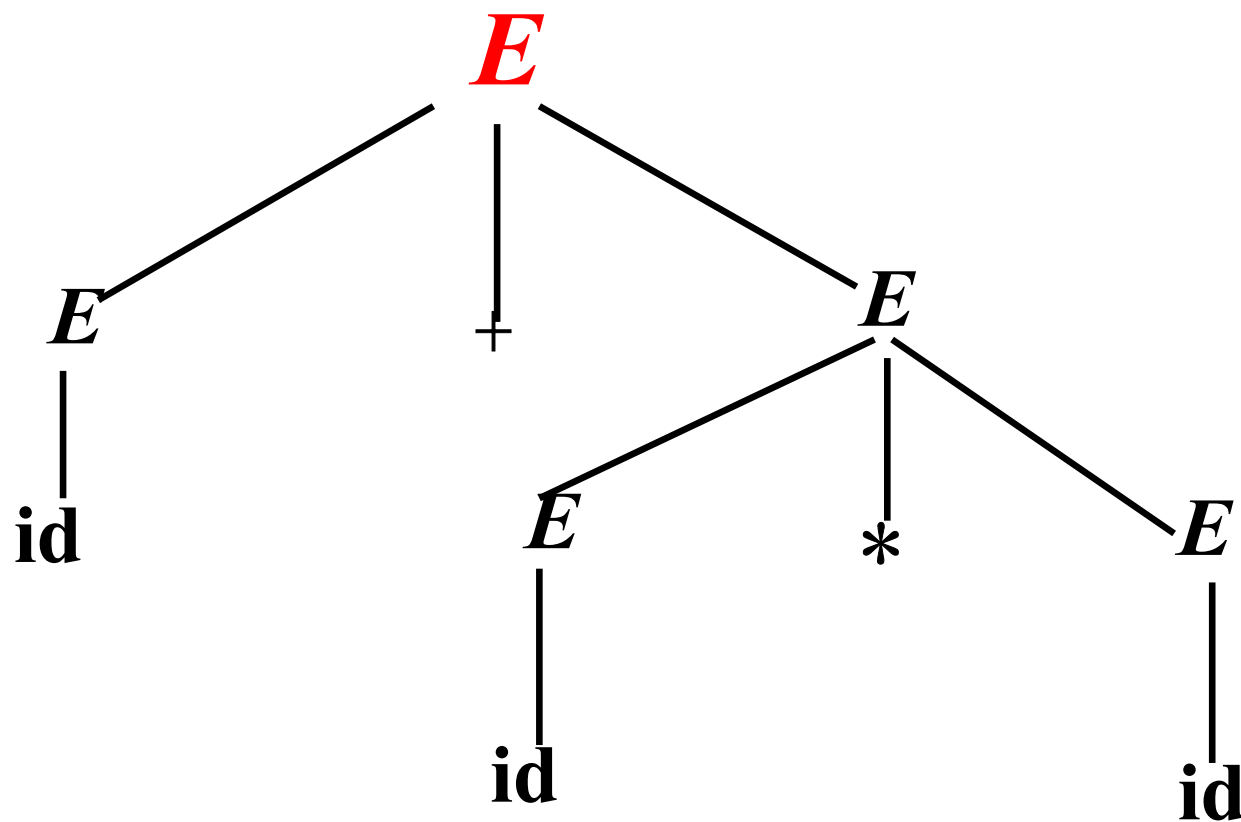
$G[E]: E \rightarrow E + T \mid E - T \mid T$
 $T \rightarrow T * F \mid T / F \mid F$
 $F \rightarrow a_1 \mid a_2 \mid a_3$

推导 $a_1+a_2^*a_3$



例 短语与分析树

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$)



id+id*id的语法树，它有多种推导方式

id+id*id的不同推导 $E \rightarrow E+E | E * E | (E) | id$

$E \Rightarrow E+E$
 $\Rightarrow id+E$
 $\Rightarrow id+E * E$
 $\Rightarrow id+id * E$
 $\Rightarrow id+id * id$

施于最左变量
(最左推导)

得到的句型称
为左句型

$E \Rightarrow E+E$
 $\Rightarrow E+E * E$
 $\Rightarrow E+E * id$
 $\Rightarrow E+id * id$
 $\Rightarrow id+id * id$

施于最右变量
(最右推导)

得到的句型称
为右句型

最左推导与最右推导

■最左推导(Left-most Derivation)

■每次推导都施加在句型的最左边的语法变量上。——与最右归约对应

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow \text{id}+E \\ &\Rightarrow \text{id}+E * E \\ &\Rightarrow \text{id}+\text{id} * E \\ &\Rightarrow \text{id}+\text{id} * \text{id} \end{aligned}$$

最右归约：总是对当前句型的最右直接短语进行归约
通过语法树来描述更加清晰

最左推导与最右推导

■最右推导(Right-most Derivation)

■每次推导都施加在句型最右边的语法变量上。——与最左归约（规范规约）对应

■对应的规范(Canonical)句型

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow E+E*E \\ &\Rightarrow E+E*id \\ &\Rightarrow E+id*id \\ &\Rightarrow id+id*id \end{aligned}$$

最左归约：总是对当前句型的最左直接短语（句柄）进行归约

最左推导与最右推导

■最右推导(Right-most Derivation)

■每次推导都施加在句型最右边的语法变量上。——与最左归约（规范规约）对应

■对应的规范(Canonical)句型

$$\begin{aligned} E &\Rightarrow E+E \\ &\Rightarrow E+E*E \\ &\Rightarrow E+E*\text{id} \\ &\Rightarrow E+\text{id}*\text{id} \\ &\Rightarrow \text{id}+\text{id}*\text{id} \end{aligned}$$

规范归约：

计算机系统的输入串一般都是自左至右顺序输入，所以对句型从左到右进行分析是很自然的

2.6 CFG的二义性

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$)

id + id * id的最左推导

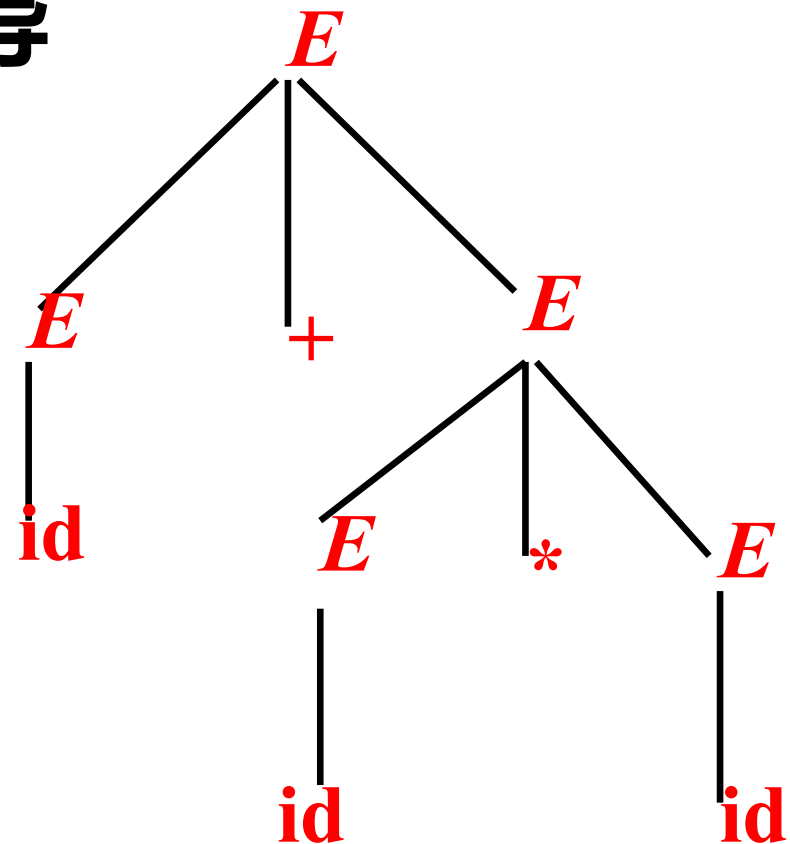
$E \Rightarrow E + E$

$\Rightarrow \text{id} + E$

$\Rightarrow \text{id} + E * E$

$\Rightarrow \text{id} + \text{id} * E$

$\Rightarrow \text{id} + \text{id} * \text{id}$



2.6 CFG的二义性

(文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$)

$\text{id} + \text{id} * \text{id}$ 的另一种最左推导

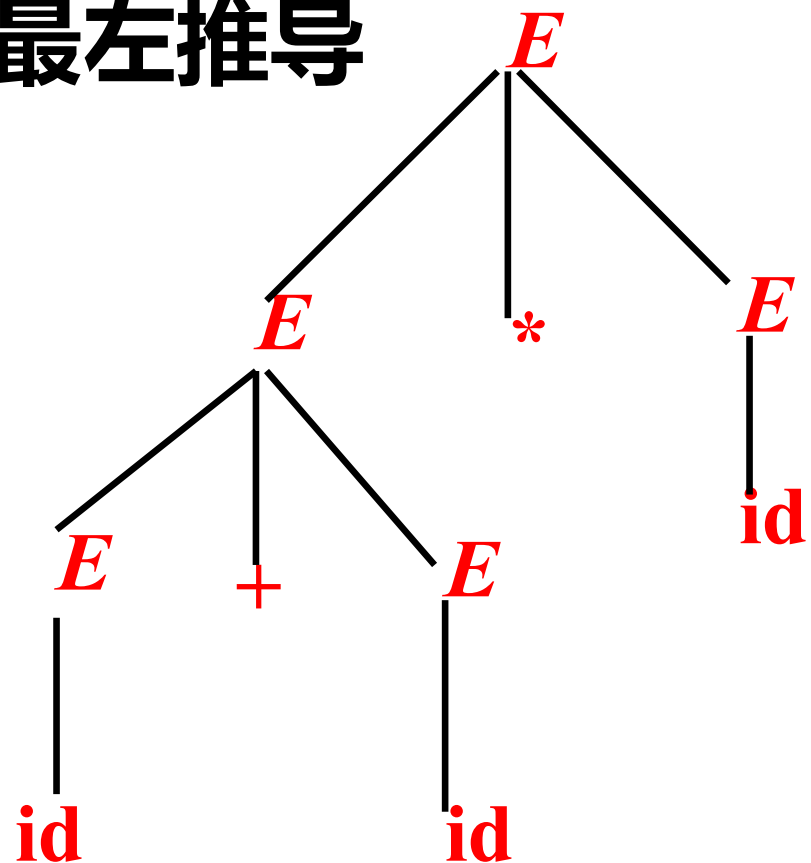
$$E \Rightarrow E * E$$

$$\Rightarrow E + E * E$$

$$\Rightarrow \text{id} + E * E$$

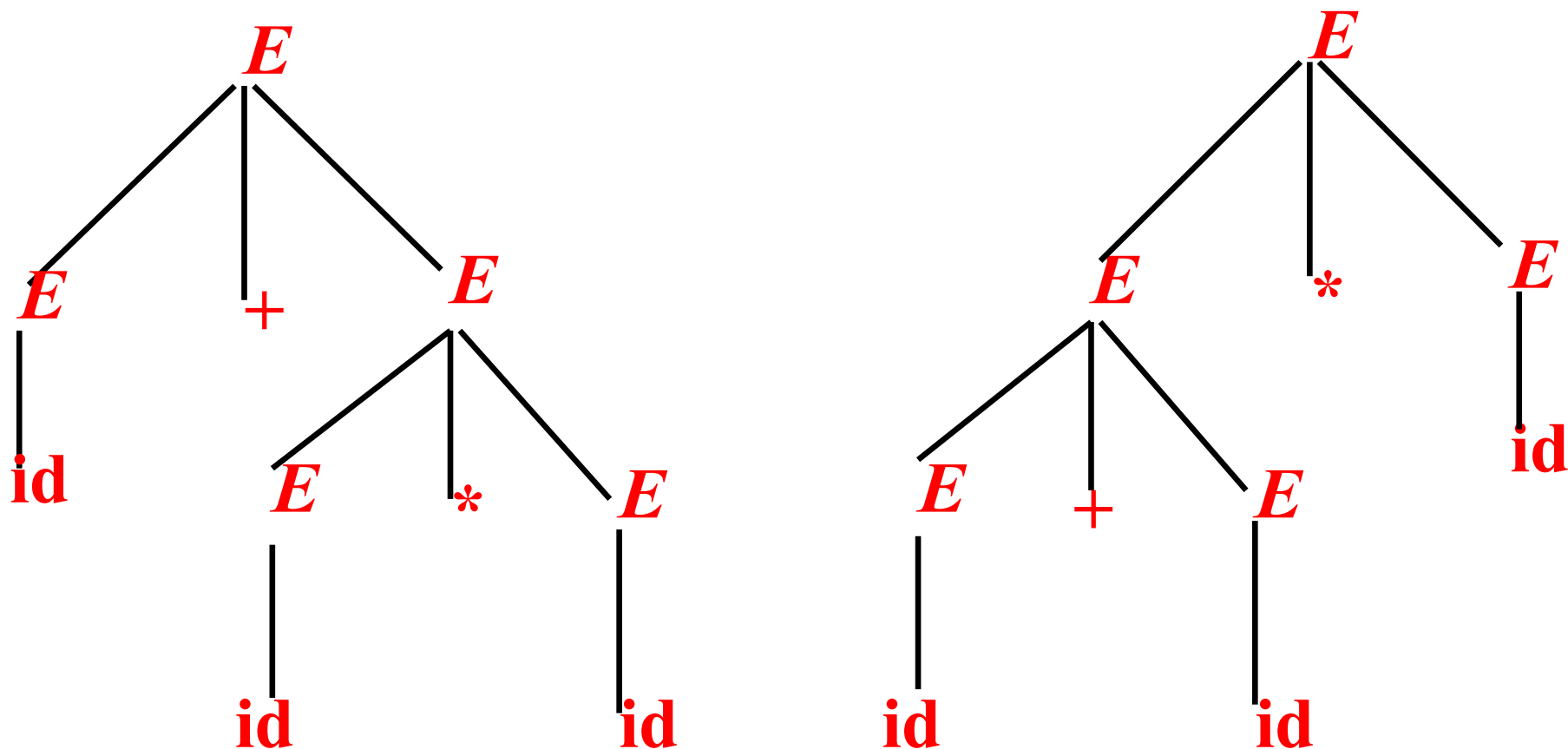
$$\Rightarrow \text{id} + \text{id} * E$$

$$\Rightarrow \text{id} + \text{id} * \text{id}$$



2.6 CFG的二义性

- 对同一句子存在两棵语法分析树
 - $\text{id} + \text{id} * \text{id}$ 的最左推导





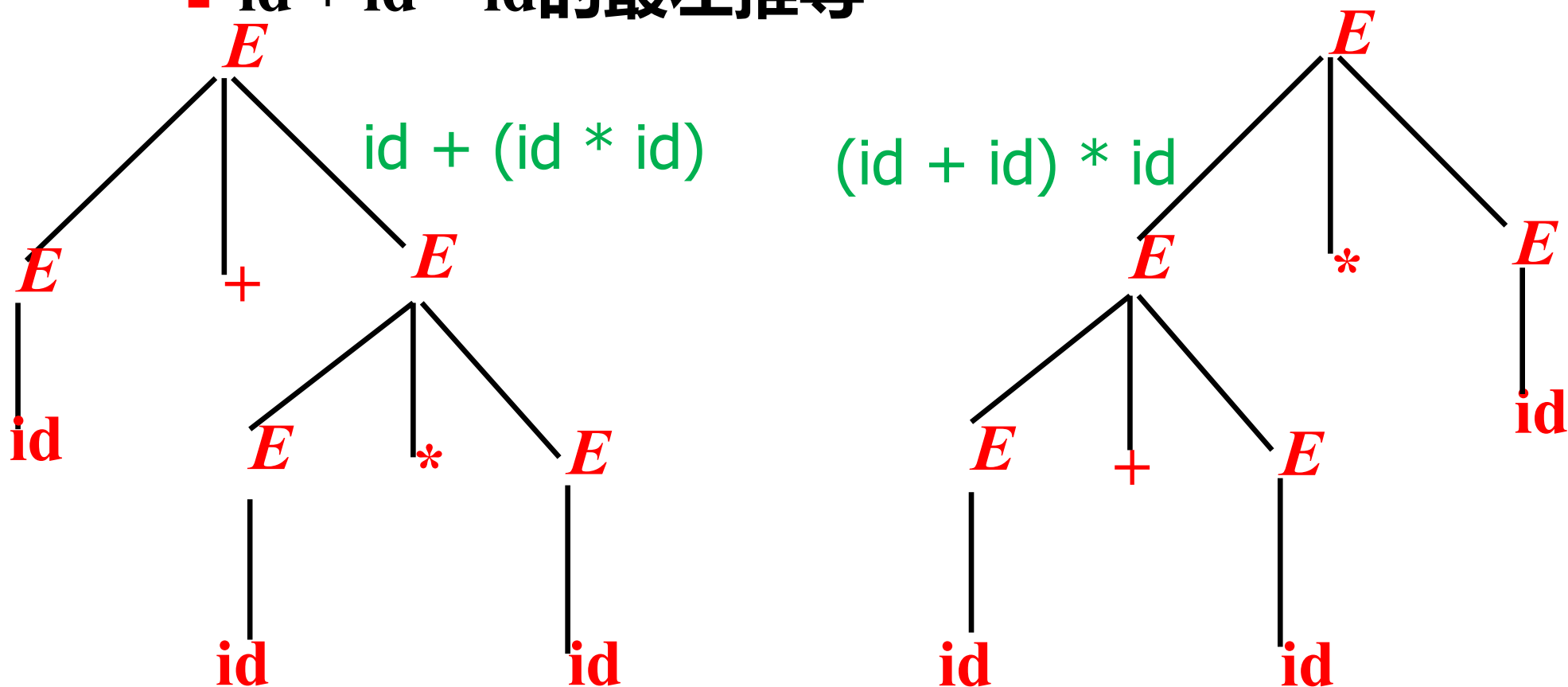
二义性的定义

- 如果一个文法的句子存在两棵分析树，那么该句子是二义性的。
- 如果一个文法包含二义性的句子，则称这个文法是二义性的；否则，该文法是无二义性的。

CFG的二义性

- 对同一句子存在两棵语法分析树

- id + id * id的最左推导



消除文法的二义性

■ (文法 $E \rightarrow E + E | E * E | (E) | id$)

■ $id + id * id$ 的最左推导

■ 找问题:

■ $+$ $*$ 给了相同的优先级

$$E \Rightarrow E + E$$

$$\Rightarrow id + E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$\Rightarrow id + id * id$$

$$E \Rightarrow E * E$$

$$\Rightarrow E + E * E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$\Rightarrow id + id * id$$

消除文法的二义性

■ (文法 $E \rightarrow E + E | E * E | (E) | id$)

■ $id + id * id$ 的最左推导

■ 找问题:

■ $+$ $*$ 给了相同的优先级

■ 解决方法:

■ 提高 $*$ 的优先级, 使得其高于 $+$ 的优先级

■ $E \rightarrow E + T | T$

■ $T \rightarrow T * F | T / F | F$

■ $F \rightarrow id | (E)$

消除文法的二义性

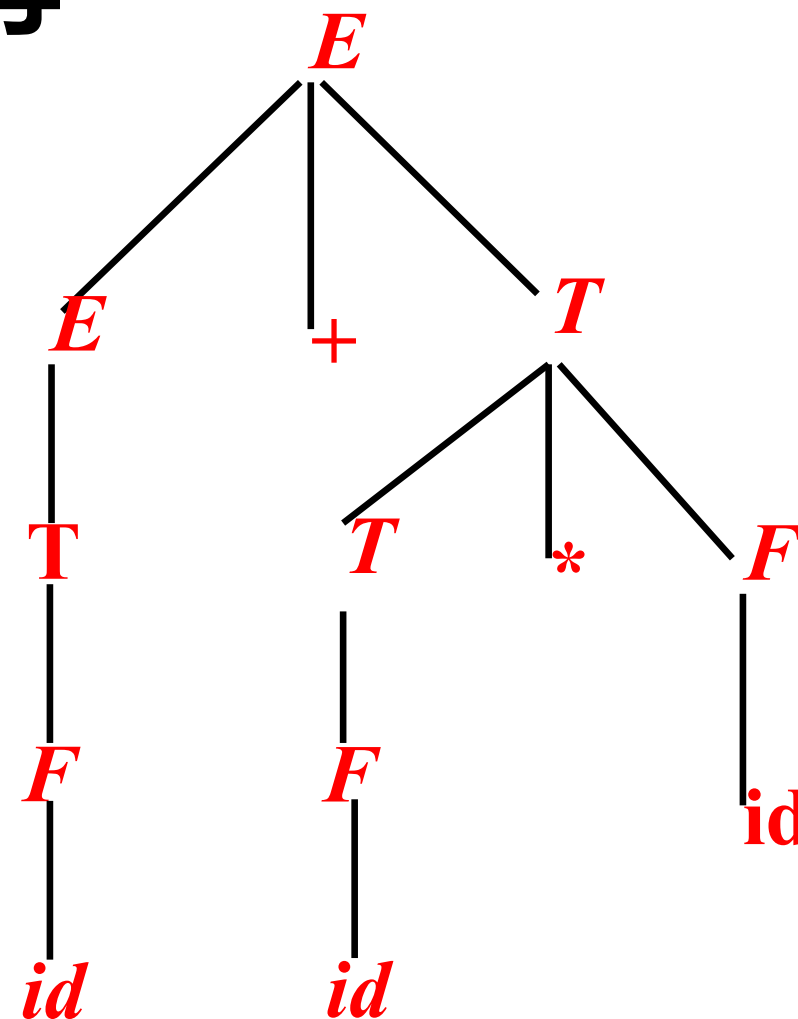
$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow \text{id} \mid (E)$$

■ $\text{id} + \text{id} * \text{id}$ 的最左推导

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow \text{id} + T \\ &\Rightarrow \text{id} + T * F \\ &\Rightarrow \text{id} + F * F \\ &\Rightarrow \text{id} + \text{id} * F \\ &\Rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$



描述if语句的无二义性文法的产生式

- 1. 对于条件语句，经常使用二义性文法描述它

- $S \rightarrow \text{if } \textit{expr} \text{ then } S$
 $\text{if } \textit{expr} \text{ then } S \text{ else } S$
 other

- 二义性的句子: $\text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } s_2$

$S \Rightarrow \text{if } \textit{expr} \text{ then } S$

$\Rightarrow \text{if } e_1 \text{ then } S$

$\Rightarrow \text{if } e_1 \text{ then if } \textit{expr} \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } s_2$

$S \Rightarrow \text{if } \textit{expr} \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } \textit{expr} \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } S \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } S$

$\Rightarrow \text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } s_2$

描述if语句的无二义性文法的产生式

下面是

$$\begin{aligned} S &\rightarrow matched_s \mid unmatched_s \\ matched_s &\rightarrow \text{if } expr \text{ then } matched_s \\ &\quad \text{else } matched_s \mid other \\ unmatched_s &\rightarrow \text{if } expr \text{ then } S \\ &\quad \mid \text{if } expr \text{ then } matched_s \\ &\quad \text{else } unmatched_s \end{aligned}$$

它显然比较复杂，因此：

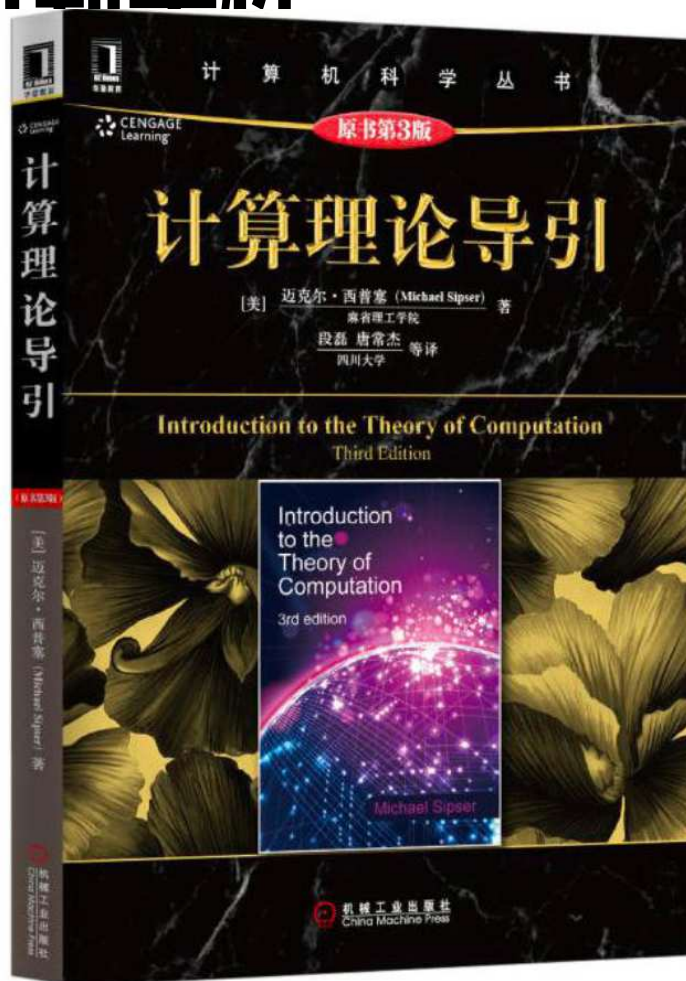
2. 在能驾驭的情况下，使用二义性文法。



3. 二义性问题是不可判定的

- 对于任意一个上下文无关文法，不存在一个算法，判定它是无二义性的
- 不过可以找到一些充分条件，满足这组充分条件的文法是无二义性的。

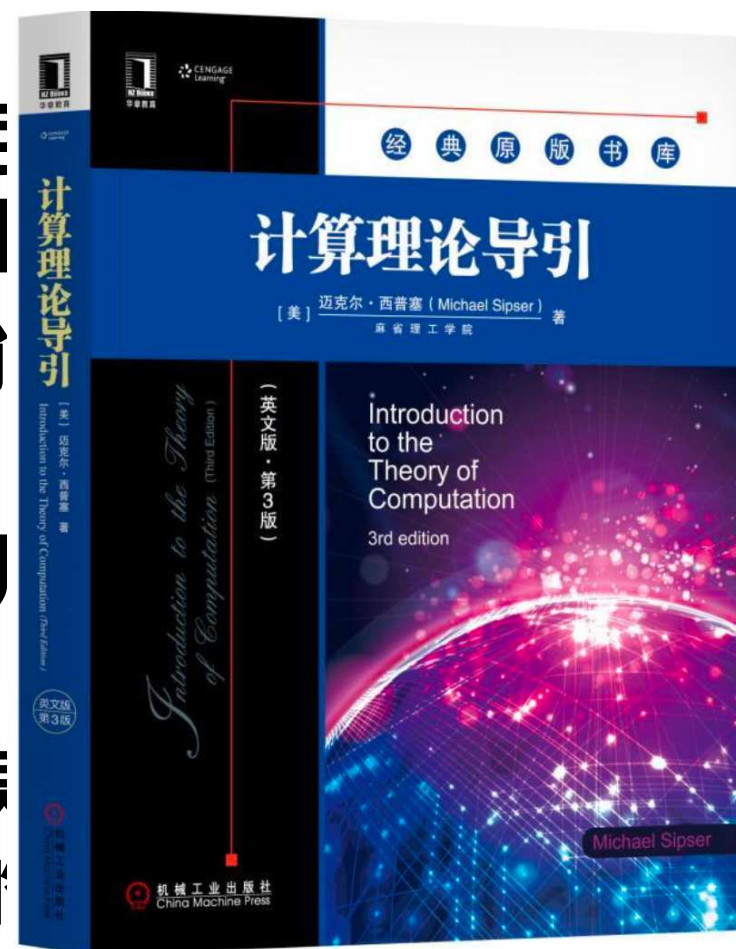
可判定性



容量的在
角的通用
几，可能

的图灵机

某一图灵
可判定的





2.7 本章小结

- 语言及其描述
- 文法的基本概念
- Chomsky体系
- CFG的语法树
- 文法的二义性