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定义。

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信息交流的工具。

什么是语言：

2.1 语言概述

- 信息交流的基础是什么?
  - 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”

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、背景知识、语气、二义性——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：严格

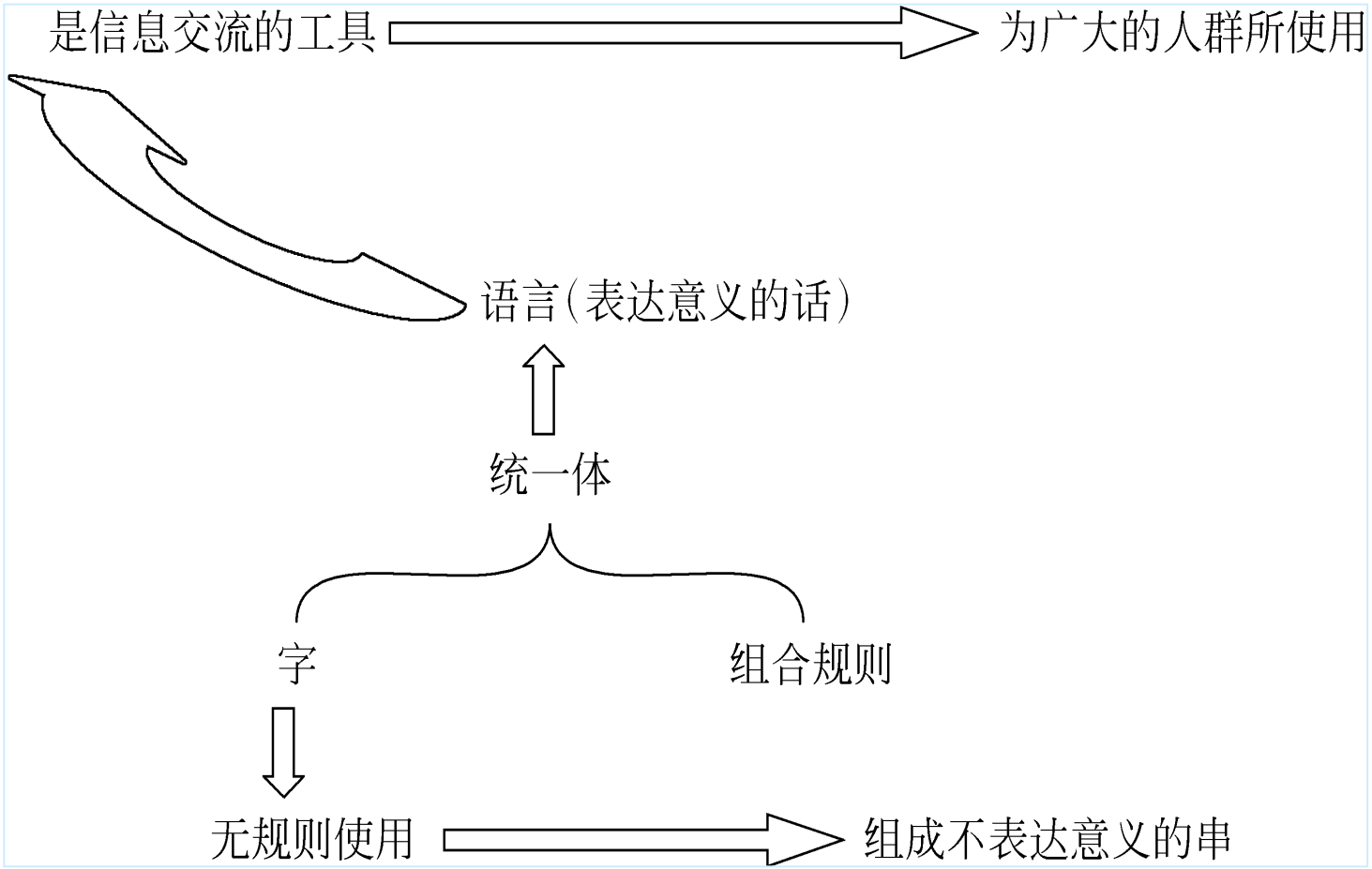
2.1 语言概述

- 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(Character)串
- 语言是字和组合字的规则
  - 例（自然语言：第译始二天课今开编上节）
  - 今天开始上第二节编译课

2.1 语言概述



语言是字及其组合规则的统一体

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：变量:=表达式
  - if 条件表达式 then 语句
  - while 条件表达式 do 语句
  - call 过程名(参数表)

2.1 语言概述

- 描述形式——文法
  - 语法——语句
    - 语句的组成规则
    - 描述方法：BNF范式、语法(描述)图
  - 词法——单词
    - 单词的组成规则
    - 描述方法：BNF范式、正规式

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。**
  - **1956年，通过抽象，他将语言形式地定义为是由一个字母表中的字母组成的一些串的集合。可以在字母表上按照一定的规则定义一个文法（Grammar），该文法所能产生的所有句子组成的集合就是该文法产生的语言。**

形式语言与自动机理论的产生与作用

- **克林（Kleene）在1951年到1956年间，从识别语言的角度研究语言，给出了语言的另一种描述。**
  - **克林是在研究神经细胞中，建立了自动机，他用这种自动机来识别语言：对于按照一定的规则构造的任一个自动机，该自动机就定义了一个语言，这个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克林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。

2.2 基本定义

- **定义2.1 字母表（Alphabet）∑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：**

(1) {*a*，*b*，*c*，*d*}

(2) {*a*，*b*，*c*，……，*z*}

(3) {0，1}

(4) ASCII字母表

2.2 基本定义

- **定义2.2 设$\sum_1$、$\sum_2$是两个字母表，$\sum_1$与$\sum_2$ 的乘积(Product)定义为$\sum_1\sum_2=\{ab|a\in\sum_1, b\in\sum_2\}$**
- **例:$\sum_1=\{0,1\}$, $\sum_2=\{a,b\}$, $\sum_1\sum_2=\{0a,0b,1a,1b\}$**
- **定义2.3 设$\sum$是一个字母表，$\sum$的n次幂(Power)递归地定义为:**
  - **(1) $\sum^0=\{\varepsilon\}$**
  - **(2) $\sum^n=\sum^{n-1}\sum$ $n\geq1$**
- **例: $\sum_1^3=$?**
- **{000,001,010,011,100,101,110,111}**

2.2 基本定义

- **定义2.4 设∑是一个字母表，∑的正闭包(Positive Closure)定义为：**
  - $\sum^{+}=\sum \cup \sum^{2} \cup \sum^{3} \cup \sum^{4} \cup \ldots\ldots$
- **∑的克林闭包(Kleene Closure)为：**
  - $\sum^{*}=\sum^{0} \cup \sum^{+}$
  - $=\sum^{0} \cup \sum \cup \sum^{2} \cup \sum^{3} \cup \ldots\ldots$

2.2 基本定义

- 例

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

$\{a,\ b,\ c,\ d\}^+ = ?$

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

2.2 基本定义

- 例

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

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

2.2 基本定义

- 定义2.5 设∑是一个字母表，$\forall x \in \sum^*$，x称为字母表∑上的一个**句子**（sentence），ε叫做∑上的一个**空句子**。
- 定义2.6 设∑是一个字母表，对任意的x，$y \in \sum^*$，$a \in \sum$，句子xay中的a叫做a在该句子中的一个**出现**(occurrence)。
- 定义2.7 设∑是一个字母表，$\forall x \in \sum^*$，句子x中字符出现的总个数叫做该字符串的**长度**(length)，记作$|x|$。

2.2 基本定义

- 定义2.8 设∑是一个字母表，$\forall x$，$y \in \sum^*$，x，y的**并置**(concatenation)是这样一个串，该串是由串x直接连接串y所组成的。记作xy。并置又叫做**连结**。
  - 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∑是一个字母表，对∀x，y，z∈∑*，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∑={a，b}上的句子$abaabb$的前缀、后缀、真前缀和真后缀如下：**
  - **前缀：ε，a，ab，aba，$abaa$，$abaab$，$abaabb$**
  - **真前缀：ε，a，ab，aba，$abaa$，$abaab$**
  - **后缀：ε，b，bb，abb，$aabb$，$baabb$，$abaabb$**
  - **真后缀：ε，b，bb，abb，$aabb$，$baabb$**

2.2 基本定义

- **定义2.10** 设$\sum$是一个字母表，对$\forall x$，y，z，w，$v \in \sum^*$，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$\sum$是一个字母表，对$\forall w$，x，y，$z \in \sum^*$，如果$w=xyz$，则称y是w的**子串**(substring)。
- 定义2.12 设$\sum$是一个字母表，对$\forall t$，u，v，w，x，y，$z \in \sum^*$，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，……，y_n是t和w的公共子串，且$|y_j|=\max\{|y_1|, |y_2|, \ldots\ldots, |y_n|\}$，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∑是一个字母表，$\forall L \subseteq \sum^*$，$L$称为字母表∑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}*

2.2 基本定义

- 2.14 设$\sum_1$，$\sum_2$是字母表，$L_1 \subseteq \sum_1^*$，$L_2 \subseteq \sum_2^*$，语言L_1与L_2的**乘积**(product)是字母表$\sum_1 \cup \sum_2$上的一个语言，该语言定义为：

$$L_1L_2=\{xy|x\in L_1，\ y\in L_2\}$$

2.2 基本定义

- **定义2.15 设∑是一个字母表，$\forall L \subseteq \sum^*$，$L$的$n$次幂(power)是一个语言，该语言定义为：**

  (1) 当n=0是，$L^n=\{\varepsilon\}$；

  (2) 当$n \geq 1$时，$L^n = L^{n-1}L$。

  L的正闭包(positive closure)L^+是一个语言，该语言定义为：

  $$L^+=L \cup L^2 \cup L^3 \cup L^4 \cup \ldots\ldots$$

  L的克林闭包(Kleene closure)L^*是一个语言，该语言定义为：

  $$L^* = L^0 \cup L \cup L^2 \cup L^3 \cup L^4 \cup \ldots\ldots$$

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

a = a+a

b = m[3]+b

m[1] = a+m[2]

句子的组成规则

- <赋值语句>→<左部量>=<右部表达式>
- <左部量>→<简单变量>
- <左部量>→ <下标变量>
- <简单变量>→a
- <简单变量>→b
- <简单变量>→ c
- <下标变量>→m[1]
- <下标变量>→m[2]
- <下标变量>→m[3]
- <右部表达式>→<简单变量><运算符><简单变量>
- <右部表达式>→<简单变量><运算符><下标变量>
- <右部表达式>→<下标变量><运算符><简单变量>
- <右部表达式>→<下标变量><运算符><下标变量>
- <运算符>→ +
- <运算符>→ -

问题：如何用符号来描述？即如何形式化？

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*=

{<赋值语句>，<左部量>，<右部表达式>，<简单变量>，<下标变量>，<运算符>}

终结符号集*T*=

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

语法规则集*P*=

{<赋值语句>→<左部量>=<右部表达式>，……}

开始符号*S*= <赋值语句>

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(Terminal)集
  - 语法变量（成分）——代表某个语言的各种子结构
- T：终结符(Variable)集
  - 语言的句子中出现的字符，$V \cap T = \varnothing$
- S：开始符号(Start Symbol)，$S \in V$
  - 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（定义式），读作：α定义为β。其中$\alpha \in (V \cup T)^+$，且α中至少有 V中元素的一个出现。$\beta \in (V \cup T)^*$。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={<赋值语句>，<左部量>，<右部表达式>，<简单变量>，<下标变量>，<运算符>}
- T={a，b，c，m[1]，m[2]，m[3]，+，-}
- P={<赋值语句>→<左部量>=<右部表达式>，
  <左部量>→<简单变量>, <左部量>→ <下标变量>,
  <简单变量>→a, <简单变量>→b, <简单变量>→ c,
  <下标变量>→m[1], <下标变量>→m[2]，
  <下标变量>→m[3],
  <右部表达式>→<简单变量> <运算符><简单变量>,
  <右部表达式>→<简单变量><运算符> <下标变量>,
  <右部表达式>→<下标变量><运算符><简单变量>，
  <右部表达式>→<下标变量><运算符><下标变量>，
  <运算符>→ +，<运算符>→ -}
- S=<赋值语句>

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

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

可以简单地记为：

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

读作：α定义为或者β_1，或者β_2，…，或者β_n。并且称它们为α产生式。β_1，β_2，…，β_n称为候选式(Candidate)。

- 对一个文法，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。

问题：有了定义句子的规则，如何判定某一句子是否属于某语言?

句子的派生(推导)-从产生语言的角度

<赋值语句>

⇒ <左部量> = <右部表达式>

⇒ <简单变量> = <右部表达式>

⇒ a = <右部表达式>

⇒ a = <简单变量><运算符><简单变量>

⇒ a = a <运算符><简单变量>

⇒ a = a + <简单变量>

⇒ a = a + a

句子的归约 ---从识别语言的角度

a = a + a

⇐ a = a + <简单变量>

⇐ a = a <运算符><简单变量>

⇐ a = <简单变量><运算符><简单变量>

⇐ a = <右部表达式>

⇐ <简单变量> = <右部表达式>

⇐ <左部量> = <右部表达式>

⇐ <赋值语句>

- 派生与识别均根据规则

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：

  $$\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta$$

- 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)，有时我们也称推导为**派生**。
- 与之相对应，也可以称$\gamma\beta\delta$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)。由于这里实际是将$\gamma\beta\delta$中的β变成了α，而γ和δ都没有变化，所以又称将β归约成α。

(多步)推导

- $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$
  - 记为 $\alpha_0 \overset{n}{\Rightarrow} \alpha_n$ (恰用n步)
  - $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$ (至少一步)
  - $\alpha_0 \overset{*}{\Rightarrow} \alpha_n$ (若干步: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

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

不难看出，$L(A)$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

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

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（L也可是有穷）

推导/归约举例

$A \to B=E$
$B \to C \mid D$
$C \to a \mid b \mid c$
$D \to m[1] \mid m[2] \mid m[3]$
$E \to COC \mid COD \mid DOC \mid DOD$
$O \to + \mid -$

$A \Rightarrow B=E$ 有产生式$A \to B=E$，所以可以将A换成$B=E$

$\Rightarrow C=E$ 有产生式$B \to C$，所以可以将$B=E$中的B换成C

$\Rightarrow a=E$ 有产生式$C \to a$，所以可以将$C=E$中的C换成a

$\Rightarrow a=COD$ 有产生式$E \to COD$，所以可以将a=E中的E换成COD

$\Rightarrow a=bOD$ 有产生式$C \to b$，所以可以将$a=COD$中的C换成b

$\Rightarrow a=b+D$ 有产生式$O \to +$，所以可以将$a=bOD$的O换成+

$\Rightarrow a=b+m[1]$ 有产生式$D \to m[1]$，所以可以将$a=b+D$的D换成m[1]

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$S\overset{*}{\Rightarrow}\alpha$，则称$\alpha$是$G$产生的一个句型(sentential form)，简称为**句型**
- 对于任意文法$G=(V，T，P，S)$，G产生的句子和句型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

2.4 文法的分类(Chomsky体系)

- 根据语言结构的复杂程度（形式语言）
  - 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
- 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PSL。

1. 上下文有关文法(CSG)

- 如果对于$\forall \alpha \to \beta \in P$，均有$|\beta| \geq |\alpha|$成立($S \to \varepsilon$除外)，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/敏感语言(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

例 标识符的文法

- $S \rightarrow L|LT$

  $T \rightarrow L|N|TL|TN$

  $L \rightarrow a|b|c|d$

  $N \rightarrow 0|1|2|3|4|5$

- $S \rightarrow a|b|c|d$

  $S \rightarrow aT|bT|cT|dT$

  $T \rightarrow a|b|c|d|0|1|2|3|4|5$

  $T \rightarrow aT|bT|cT|dT|0T$

  $N \rightarrow 1T|2T|3T|4T|5T$

3. 正规文法(RG)

- 设A、$B \in V$，$a \in T$或为ε
- 右线性(Right Linear)文法：$A \to aB$或$A \to a$
- 左线性(Left Linear)文法：$A \to Ba$或$A \to a$

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多数单词
  - 左、右线性文法不可混用

例 非CFL的文法

$L=\{a^n b^n c^n | n>0\}$的文法

$S \rightarrow aBC | aSBC$

$CB \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

- “可以证明”不存在CFG G,使$L(G)=L$

非上下文无关的语言结构

- 程序设计语言的有些语言结构不能用上下文无关文法描述
- 例2.9

$$L_1=\{wcw|w\in\{a,b\}^+\}$$

$aabcaab$就是L_1的一个句子

这个语言是检查程序中标识符的声明应先于引用的抽象

非上下文无关的语言结构

- 例2.10

$$L_2=\{a^n b^m c^n d^m | n,m\geq 0\}$$

- 它是检查过程声明的形参个数和过程引用的参数个数是否一致问题的抽象

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；

  ---其能力相当于图灵机

* $|\alpha|\leq|\beta|$:G是1型文法，$L(G)$是1型语言(除 $S\rightarrow\varepsilon$);

  ---其识别系统是线性界限自动机

* $\alpha\in V$：G是2型文法，$L(G)$是2型语言;

  ---其识别系统是不确定的下推自动机

* $A\rightarrow aB$或$A\rightarrow a$：G是右线性文法，$L(G)$是3型语言

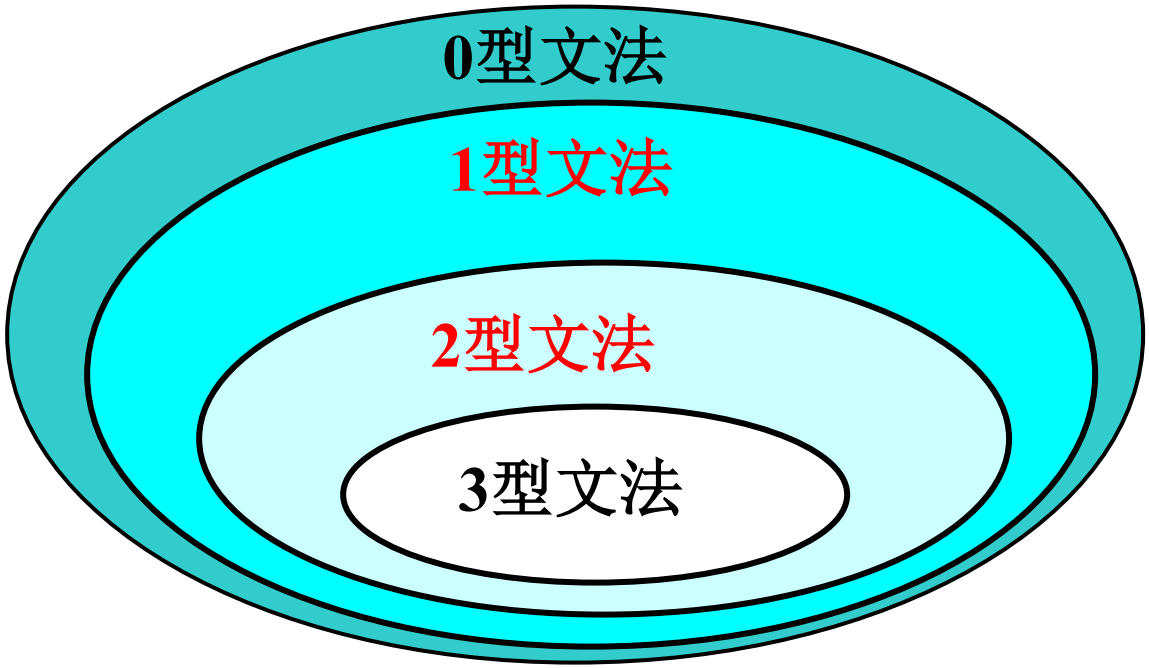
  $A\rightarrow Ba$或$A\rightarrow a$：G是左线性文法，$L(G)$是3型语言

  ---其识别系统是有穷自动机

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：



ＢＮＦ范式——Backus-Naur Form Backus-Normal Form

- $\alpha \rightarrow \beta$表示为$\alpha ::= \beta$
- 非终结符用“<”和“>”括起来
- 终结符：基本符号集
- 其他
  - $\beta(\alpha_1|\alpha_2\ldots|\alpha_n) \equiv \beta\alpha_1|\beta\alpha_2\ldots|\beta\alpha_n$
  - $[\alpha] \equiv \alpha|\ \varepsilon$
  - ……

ＢＮＦ范式——Backus-Naur Form
Backus-Normal Form

- 例 简单算术表达式(只写产生式)
  - <算术表达式>::=<简单表达式>+<简单表达式>
  - <简单表达式>::=<简单表达式>*<简单表达式>
  - <简单表达式>::=(<简单表达式>)
  - <简单表达式>::=id
- 即：<算术表达式>::=<简单表达式>+<简单表达式>|<简单表达式>*<简单表达式>|(<简单表达式>)|id
- 哪些是终结符？哪些是变量？

2.5 CFG的语法树

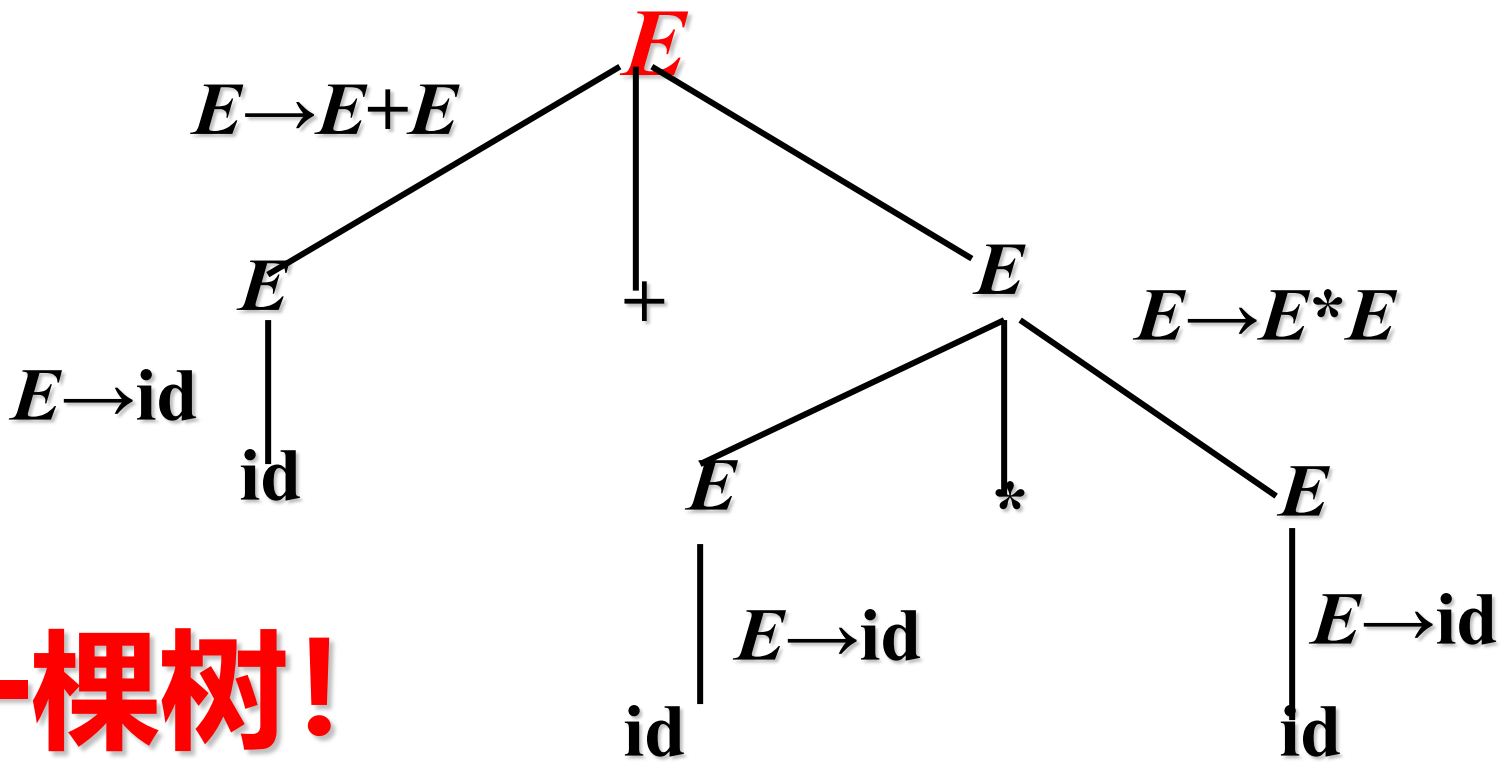
Parse Tree

- 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——一个二级子树-直接短语
- 又称为**分析树**(parse tree)、**推导树**(derivation tree)、**派生树**(derivation tree)

例 句子结构的表示

(文法$E \to E+E|E*E|(E)|\mathrm{id}$)

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



一棵树！

短语(Phrase)

- 定义2.27 设有CFG $G=(V, T, P, S)$，$\exists \alpha, \gamma, \beta \in (V \cup T)^*$，$S \overset{*}{\Rightarrow} \gamma A \beta$，$A \overset{+}{\Rightarrow} \alpha$，则称$\alpha$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)

在无意义冲突时，简称为句型$\gamma\alpha\beta$的直接短语。直接短语也叫做**简单短语**(simple phrase)。

- 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)。

例：(直接)短语

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid \mathrm{id}$

$E \Rightarrow E+T$

$\Rightarrow T+T$

$\Rightarrow F+T$

$\Rightarrow (E)+T$

$\Rightarrow (E+T)+T$

$\Rightarrow (E+T)+T$

$\Rightarrow (T+T)+T$

$\Rightarrow (F+T)+T$

$\Rightarrow (\mathrm{id}+T)+T$

$\Rightarrow (a+T*F)+T$

$\Rightarrow (a+F*F)+T$

$\Rightarrow (a+b*F)+T$

$\Rightarrow (a+b*c)+T$

$\Rightarrow (a+b*c)+F$

$\Rightarrow (a+b*c)+d$

句柄(Handle)：最左直接短语

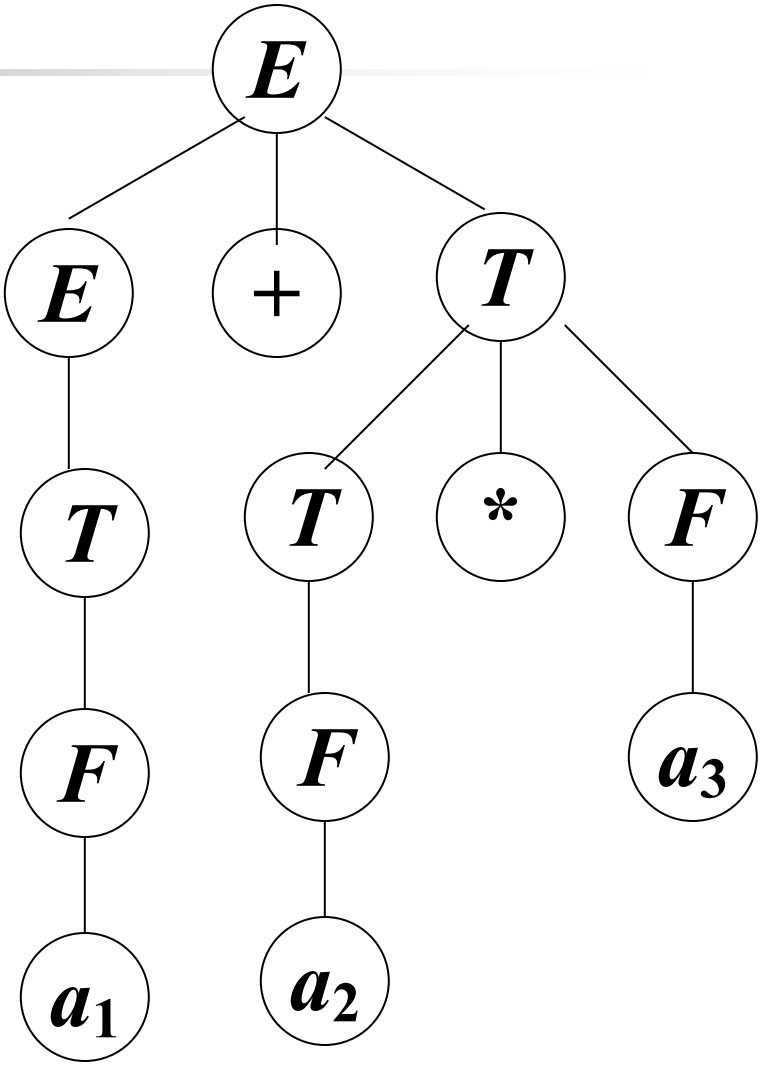
- $T \to T^* F \mid F$
- $E \to E+T \mid T$
- $F \to (E) \mid \mathrm{id}$

$E \Rightarrow E+T$
$\Rightarrow T+T$
$\Rightarrow F+T$
$\Rightarrow (E)+T$
$\Rightarrow (E+T)+T$
$\Rightarrow (E+T)+T$
$\Rightarrow (T+T)+T$
$\Rightarrow (F+T)+T$
$\Rightarrow (\mathrm{id}+T)+T$
$\Rightarrow (a+T*F)+T$
$\Rightarrow (a+F*F)+T$
$\Rightarrow (a+b*F)+T$
$\Rightarrow (a+b*c)+T$
$\Rightarrow (a+b*c)+F$
$\Rightarrow (a+b*c)+d$

用子树解释短语，直接短语，句柄

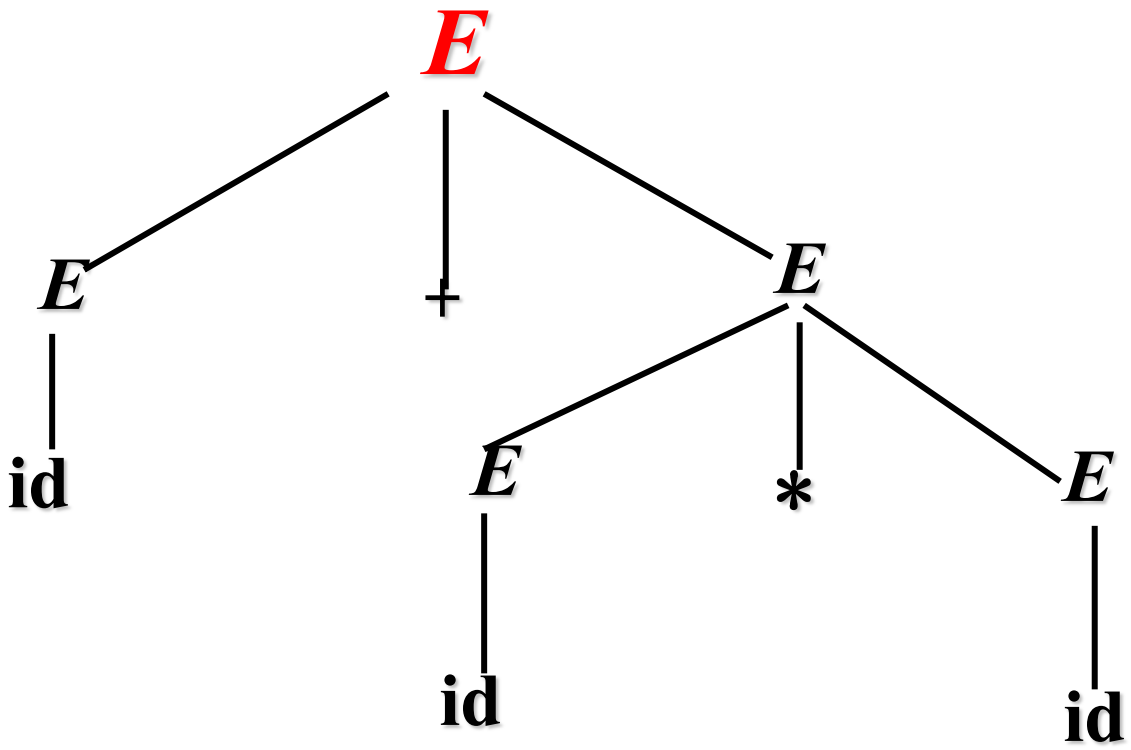
- **短语**：一棵子树的所有叶子自左至右排列起来形成一个相对于子树根的短语。
- **直接短语**：仅有父子两代的一棵子树，它的
- 所有叶子自左至右排列起来所形成的符号串。
- **句柄**：一个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列。
- 例如，对表达式文法$G[E]$和句子$a_1+a_2*a_3$，挑选出推导过程中产生的句型中的短语，直接短语，句柄

E	短语
$\Rightarrow E+T$	$E+T$
$\Rightarrow T+T$	$T, T+T$
$\Rightarrow F+T$	$F, F+T$
$\Rightarrow a_1+T$	$a_1, a_{1+}T$
$\Rightarrow a_1+T*F$	$a_1, T*F, a_{1+}T*F$
$\Rightarrow a_1+F*$	$a_1, F, F*F,$
$\Rightarrow a_1+a_2*F$	$a_1, a_2, a_{1+}a_2*F, a_2*F$
$\Rightarrow a_1+a_2*a_3$	a_1, a_2, a_3, a_2*a_3 $a_{1+}a_2*a_3$



例 短语与分析树

(文法$E \rightarrow E+E|E*E|(E)|\mathrm{id}$)



一棵子树的叶子！

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

$E \Rightarrow E*E$

$\Rightarrow E+E*E$

$\Rightarrow E+id*E$

$\Rightarrow id+id*E$

$\Rightarrow id+id*id$

不做限制

句型 (sentential Form)

(归约)

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

$E \Rightarrow E+E$

$\Rightarrow id+E$

$\Rightarrow id+E*E$

$\Rightarrow id+id*E$

$\Rightarrow id+id*id$

施于最左变量

左句型(left-~)

(最右归约)

$E \Rightarrow^{5} id+id*id$

$E \Rightarrow E+E$

$\Rightarrow E+E*E$

$\Rightarrow E+E*id$

$\Rightarrow E+id*id$

$\Rightarrow id+id*id$

施于最右变量

右句型/规范句型
(canonical ~)

(最左/规范归约)

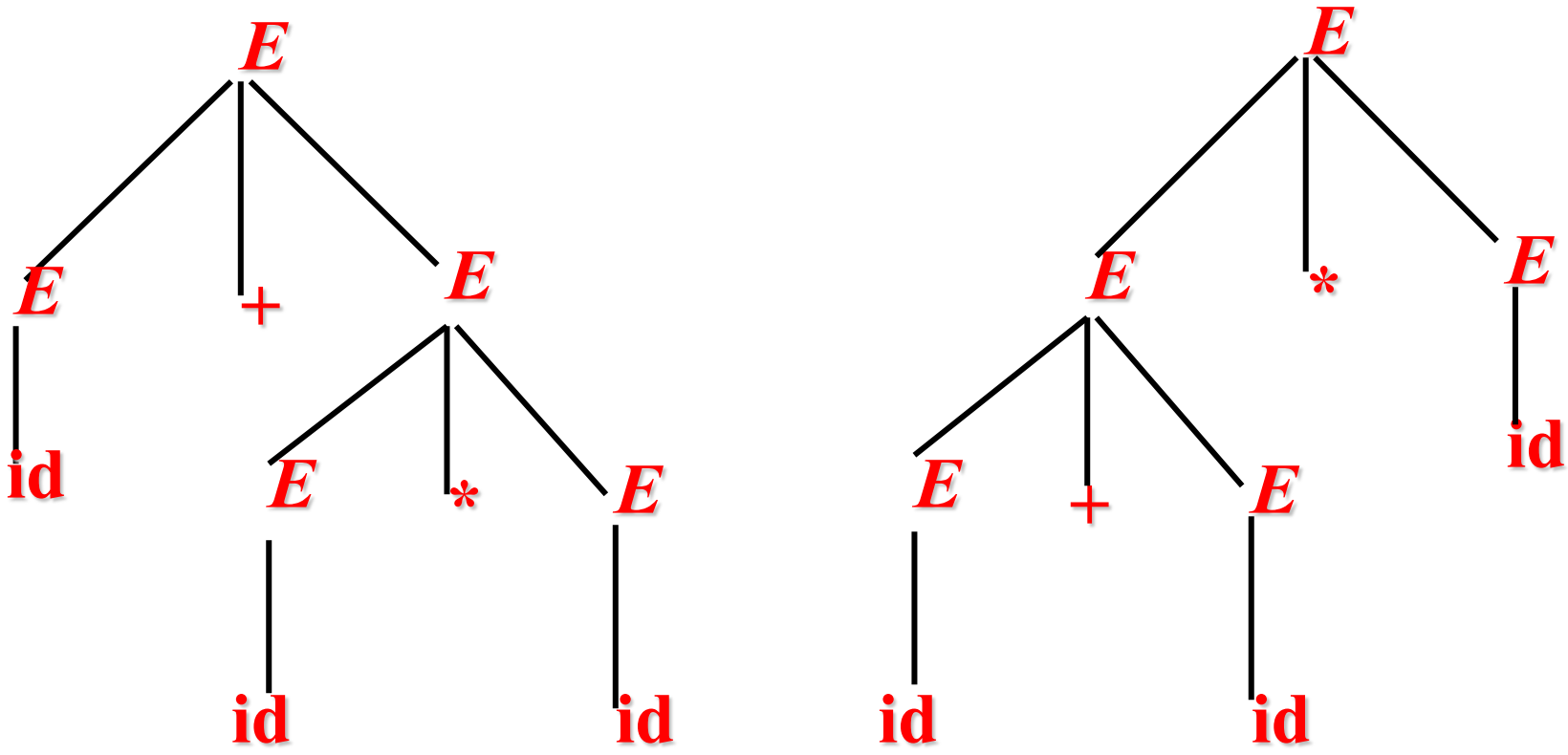
$E \Rightarrow^{+} id+id*id$

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
- 最右推导(Right-most Derivation)
  - 每次推导都施加在句型的最右边的语法变量上。——与最左归约（规范规约）对应的规范(Canonical)句型

2.6 CFG的二义性

- 对同一句子存在两棵语法分析树
  - 在理论上不可判定



文法的二义性

1. 描述一个句子的文法不是唯一的；
2. 对于一个句子的分析应是唯一的。

考虑表达式下面的文法 $G[E]$,其产生式如下：

$$E \to E+E \mid E*E \mid (E) \mid a$$

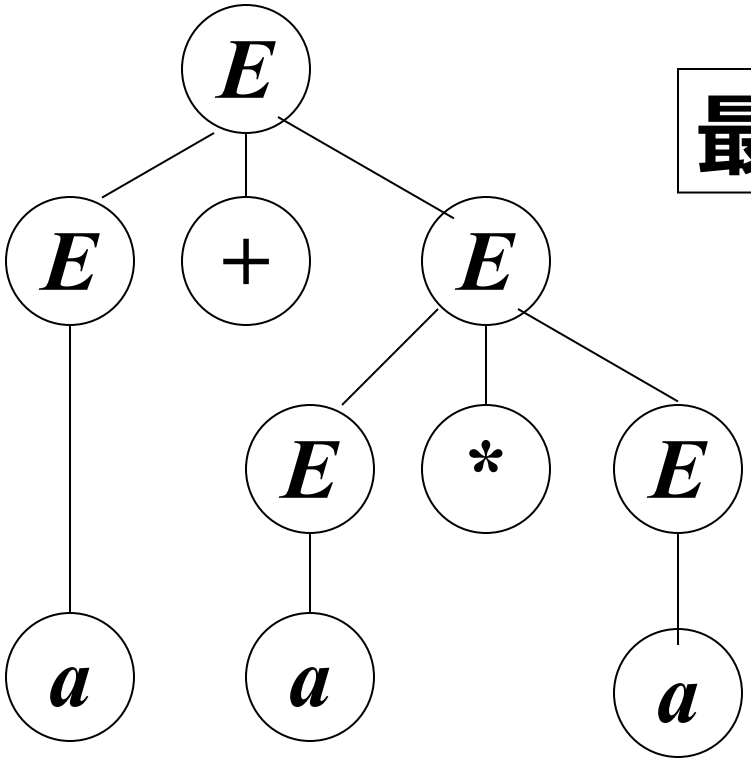
对于句子$a+a*a$, 有如下两个最左推导：

$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E \Rightarrow a+a*E \Rightarrow a+a*a$$

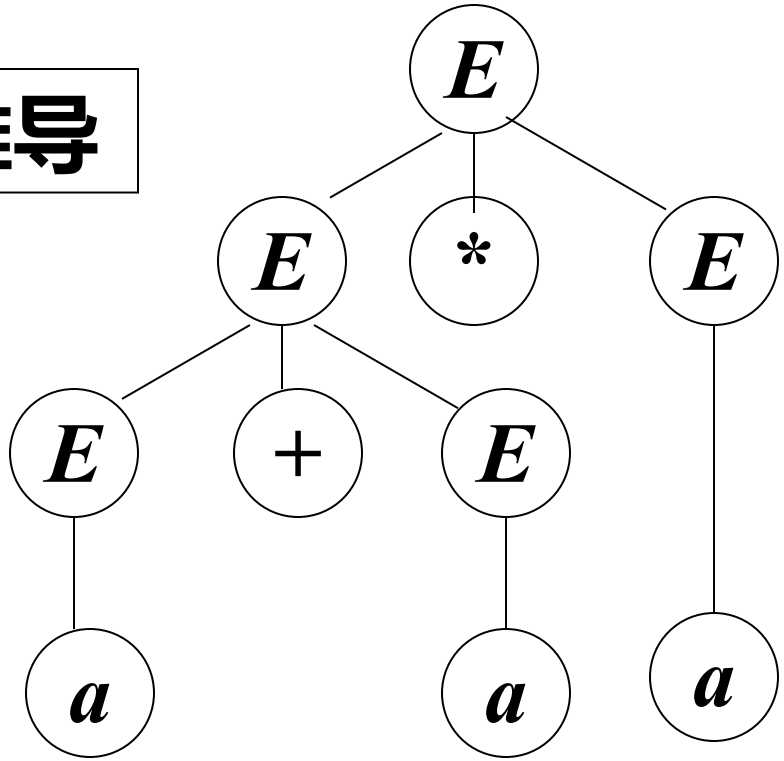
$$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow a+E*E \Rightarrow a+a*E \Rightarrow a+a*a$$

$E \Rightarrow E+E \Rightarrow a+E$
$\Rightarrow a+E*E \Rightarrow a+a*E$
$\Rightarrow a+a*a$

$E \Rightarrow E*E \Rightarrow E+E*E$
$\Rightarrow a+E*E \Rightarrow a+a*E$
$\Rightarrow a+a*a$

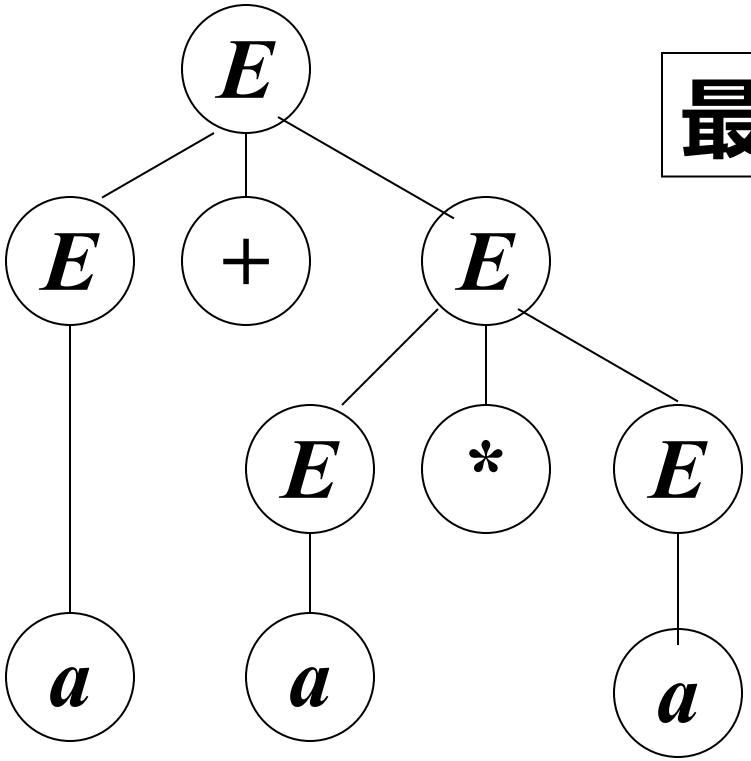


最左推导

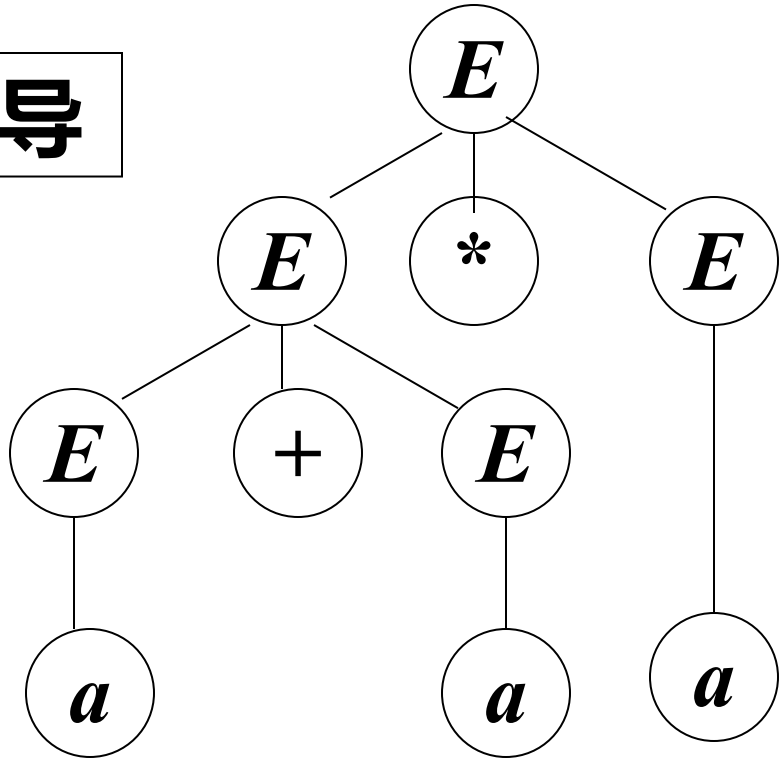


$E \Rightarrow E+E \Rightarrow E+E*E$
$\Rightarrow E+E*a \Rightarrow E+a*a$
$\Rightarrow a+a*a$

$E \Rightarrow E*E \Rightarrow E*a$
$\Rightarrow E+E*a \Rightarrow E+a*a$
$\Rightarrow a+a*a$



最右推导



二义性（ambiquity)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。几点说明:
- 1 . 一般来说，程序语言存在无二义性文法。对于条件语句，经常使用二义性文法描述它:
- $S \rightarrow$ if *expr* then S
  | if *expr* then S else S
  | other
- 二义性的句子: if e_1 then if e_2 then s_1 else s_2

描述if语句的无二义性文法的产生式

下面是无二义性文法

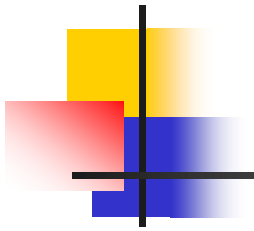
$S \rightarrow matched\_s \mid unmatched\_s$

$matched\_s \rightarrow$ if *expr* then $matched\_s$ else $matched\_s \mid$ other

$unmatched\_s \rightarrow$ if *expr* then S $\mid$ if *expr* then $matched\_s$ else $unmatched\_s$

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. 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；

4. 存在先天二义性语言。例如，

$$\{a^i b^i c^j \mid i, j \geq 1\} \cup \{a^i b^j c^j \mid i, j \geq 1\}$$

存在一个二义性的句子$a^k b^k c^k$。

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
- 语言的固有二义性