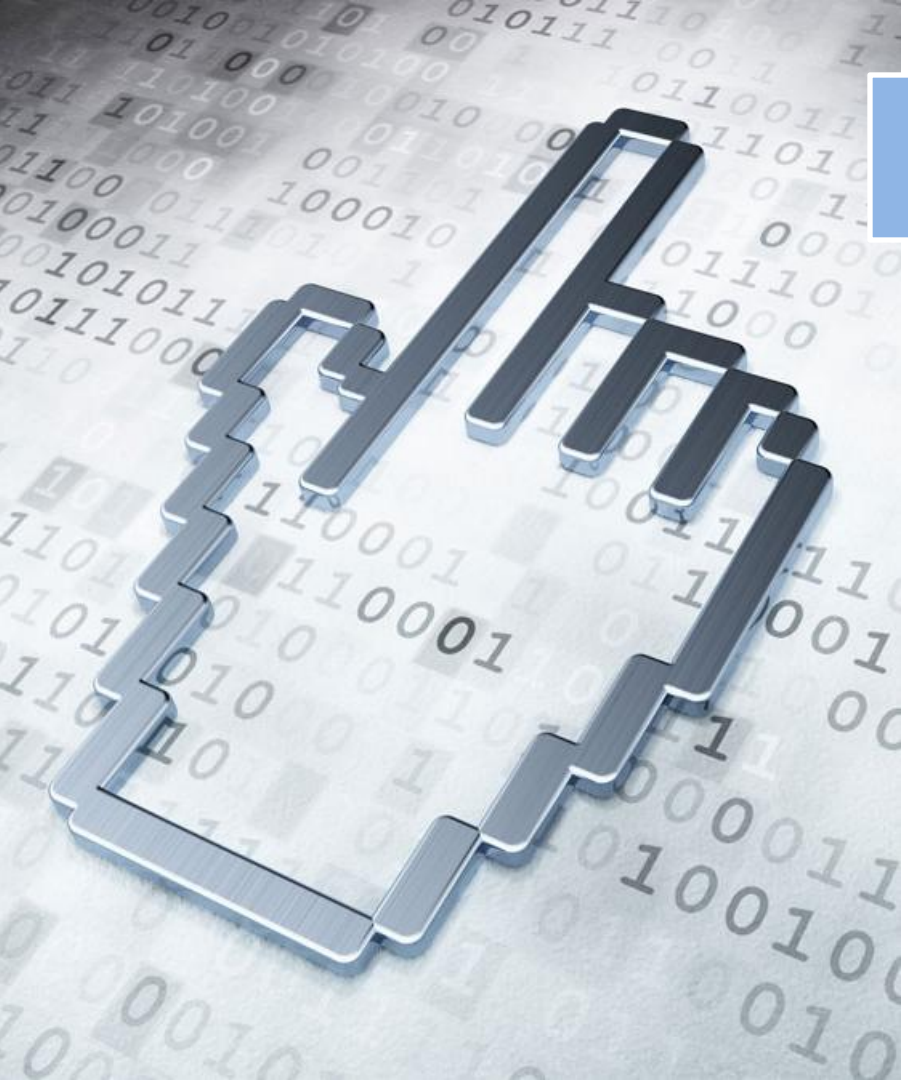
编译原理

第二章

语言及其文法

哈尔滨工业大学　陈鄞　单丽莉



本章内容

2.1 基本概念

- 串 (*String*)
  - 串是一个有穷符号（*symbol*）序列
    - 符号的典型例子：字母、数字、标点符号、…

例：
- 符号： *a*、*b*、*c*
- 串： *abcb*

2.1 基本概念

- 串 (*String*)
  - 串是一个有穷符号（*symbol*）序列
  - 串s的长度，通常记作|s|，是指s中符号的个数
    - 例: |*abcb*|=4
  - 空串（*empty string*）是长度为0的串，用 ε（*epsilon*）表示
    - |ε|= 0

串上的运算——连接

- 如果 *x*和*y*是串，那么*x*和*y*的连接(*concatenation*)是把*y*附加到*x*后面而形成的串，记作*xy*
  - 例如，如果 *x=dog*且 *y=house*，那么 *xy=doghouse*
  - 空串是连接运算的单位元(*identity*)，即，对于任何串*s*都有

设*y, z*是串，若 *x=yz*，则称*y*是*x*的前缀，*z*是*x*的后缀。

串上的运算——幂

- 串s的幂运算

串s的n次幂：将n个s连接起来

$$\begin{cases} s^0 = \varepsilon , \\ s^n = s^{n-1}s , n \geq 1 \end{cases}$$

- $s^1 = s^0 s = \varepsilon s = s$，$s^2 = ss$，$s^3 = sss$，…
- 例：如果 $s = ba$，

那么$s^1 = ba$，$s^2 = baba$， $s^3 = bababa$，…

字母表(*Alphabet*)

- 字母表∑是一个有穷符号集合

例：

- 二进制字母表：{ 0,1 }
- *ASCII*字符集
- *Unicode*字符集

字母表上的运算

- 字母表Σ_1和Σ_2的乘积(*product*)
  - $\Sigma_1\Sigma_2 = \{ab | a \in \Sigma_1, b \in \Sigma_2\}$

例： $\{0, 1\}\{a, b\} = \{0a, 0b, 1a, 1b\}$

字母表上的运算

- 字母表Σ_1和Σ_2的**乘积**(*product*)
- 字母表Σ的***n*次幂**(*power*)

$$\begin{cases} \Sigma^0 = \{ \varepsilon \} \\ \Sigma^n = \Sigma^{n-1} \Sigma , n \geq 1 \end{cases}$$

例： $\{0, 1\}^3 = \{0, 1\}\ \{0, 1\}\ \{0, 1\}$
$= \{000, 001, 010, 011, 100, 101, 110, 111\}$

字母表的*n*次幂：字母表上所有长度为*n*的串构成的集合

字母表上的运算

- 字母表Σ_1和Σ_2的**乘积**(*product*)
- 字母表Σ的***n*次幂**(*power*)
- 字母表Σ的**正闭包**(*positive closure*)
  - $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

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

字母表的正闭包：字母表上所有长度大于0的串构成的集合

字母表上的运算

- 字母表Σ_1和Σ_2的**乘积**(*product*)
- 字母表Σ的***n*次幂**(*power*)
- 字母表Σ的**正闭包**(*positive closure*)
- 字母表Σ的**克林闭包**(*Kleene closure*)
  - $\Sigma^* = \Sigma^0 \cup \Sigma^+ = \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$

例：$\{a, b, c, d\}^* = \{\varepsilon, a, b, c, d,$

aa, ab, ac, ad, ba, bb, bc, bd, …,
aaa, aab, aac, aad, aba, abb,
abc, …}

字母表的克林闭包：字母表上所有串（包括空串）构成的集合

语言的形式化定义

- 设∑是一个字母表，$L \subseteq \Sigma^*$，L称为字母表∑上的一个语言（*Language*），$x \in L$，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}

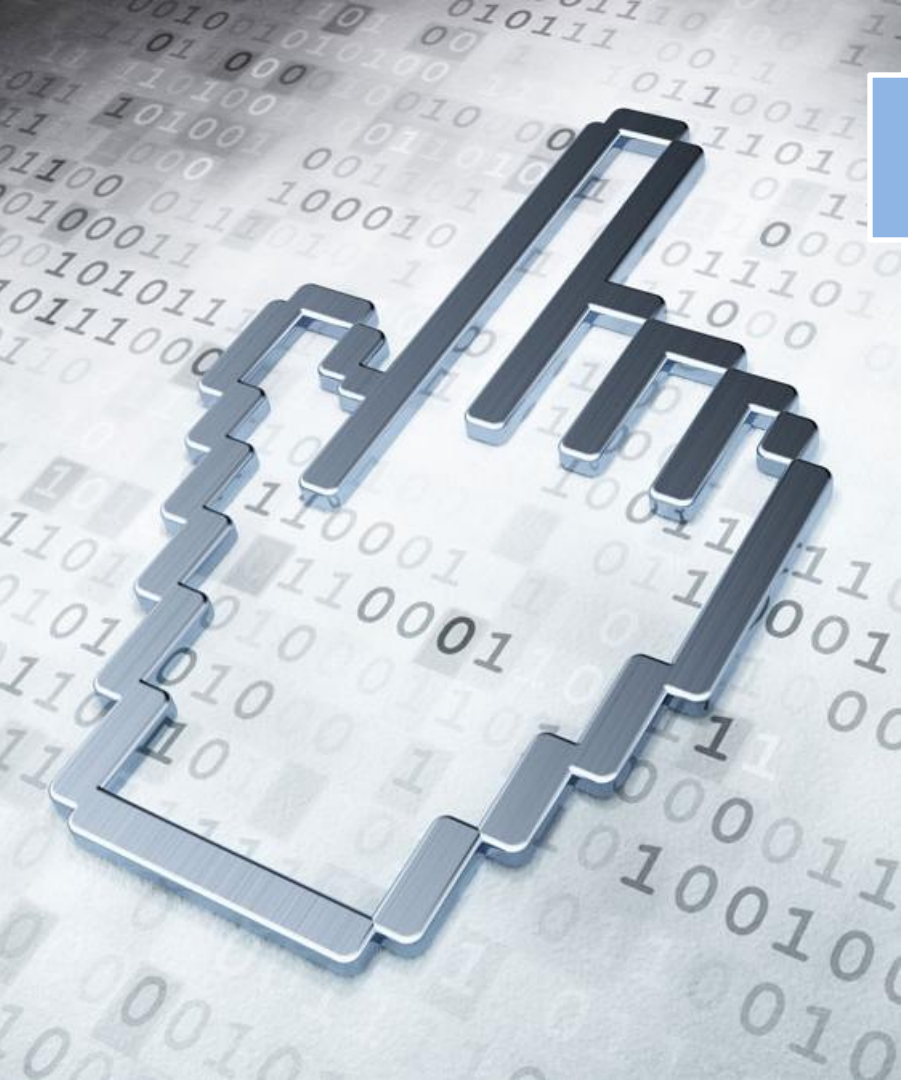
{0，1，00，11，01，10}

{00，11}*

{01，10}*

语言是某个字母表上任意个数的串的集合

语言可以包含有穷个串(句子)，也可以包含无穷多个串(句子)。



提纲

2.2 文法的定义

- 自然语言的例子——句子的构成规则
  - 句子　　名词短语　　动词短语
  - 名词短语　　形容词　　名词短语
  - 名词短语　　名词
  - 动词短语　　动词　　名词短语
  - 形容词　　*little* | *pretty*
  - 名词　　*boy* | *girl*
  - 名词　　*apple* | *pie*
  - 动词　　*eats* | *watches*

没有尖括号的部分表示语言的基本符号

尖括号括起来部分称为语法成分

文法的形式化定义——语言的有穷描述

$G = (V_T, V_N, P, S)$

- V_T：终结符集合

  终结符（*terminal symbol*）是文法所定义的语言的基本符号，有时也称为*token*

  - 例: V_T = { *apple, boy, eats, little, pretty, girl, pie, watches* }

文法的形式化定义

$G = (V_T, V_N, P, S)$

- V_T：终结符集合
- V_N：非终结符集合

  非终结符(*nonterminal*) 是用来表示语法成分的符号，也称为“语法变量”

  - 例: V_N = { 句子 ， 名词短语 ， 动词短语 ， 名词 ，<动词>, <形容词> }

文法的形式化定义

$G = (V_T, V_N, P, S)$

- V_T：终结符集合
- V_N：非终结符集合
- P ：产生式集合

$V_T \cap V_N = \Phi$ 都是有穷集合

$V_T \cup V_N$：称为G的文法符号集

产生式(*production*)描述了将终结符和非终结符组合成串的方法，即语法成分的构成方法。

产生式的一般形式： $\alpha \rightarrow \beta$ ， 读作：α定义为β

- $\alpha \in (V_T \cup V_N)^+$，且$\alpha$中至少包含一个$V_N$中的元素：称为产生式的头(*head*)或左部(*left side*)
- $\beta \in (V_T \cup V_N)^*$ ：称为产生式的体(*body*)或右部(*right side*)

文法的形式化定义

$G = (V_T, V_N, P, S)$

- V_T：终结符集合
- V_N：非终结符集合
- P ：产生式集合

产生式(*production*)描述了将终结符和非终结符组合成串的方法。

- 例： $P = \left\{ \begin{array}{lll} \text{句子} & \text{名词短语} & \text{动词短} \\ \text{名词短语} & \text{形容词} & \text{名词短语} \\ \cdots & & \end{array} \right.$ ，

文法的形式化定义

$G = (V_T, V_N, P, S)$

- V_T：终结符集合
- V_N：非终结符集合
- P ：产生式集合
- S ：开始符号

$S \in V_N$。开始符号(*start symbol*)表示的是该文法中最大的语法成分

- 例：$S =$ 句子

文法的形式化定义

$G = (V_T, V_N, P, S)$

- V_T：终结符集合
- V_N：非终结符集合
- P：产生式集合
- S：开始符号

约定

在不引起歧义的前提下，可以只用产生式集合表示文法G，第一个产生式的头就是开始符号。

例：$G = (\{id, +, *, (,)\}, \{E\}, P, E)$

$P = \{ E \to E + E,$
$E \to E * E,$
$E \to (E),$
$E \to id \}$

$G: E \to E + E$
$E \to E * E$
$E \to (E)$

产生式的简写

- 对一组有相同左部的α产生式

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

可以简记为：

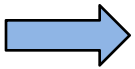
$$\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

读作：α定义为β_1，或者β_2，…，或者β_n。

β_1，β_2，…，β_n称为α的候选式(*Candidate*)

- 例

$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to$ id



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

符号约定

- 下述符号是终结符

(*a*) 字母表中排在前面的小写字母，如 *a*、*b*、*c*

(*b*) 运算符，如 +、*等

(*c*) 标点符号，如括号、逗号等

(*d*) 数字0、1、. . . 、9

(*e*) 粗体字符串，如id、if等

符号约定

- 下述符号是非终结符

  (*a*) 字母表中排在前面的大写字母，如*A*、*B*、 *C*

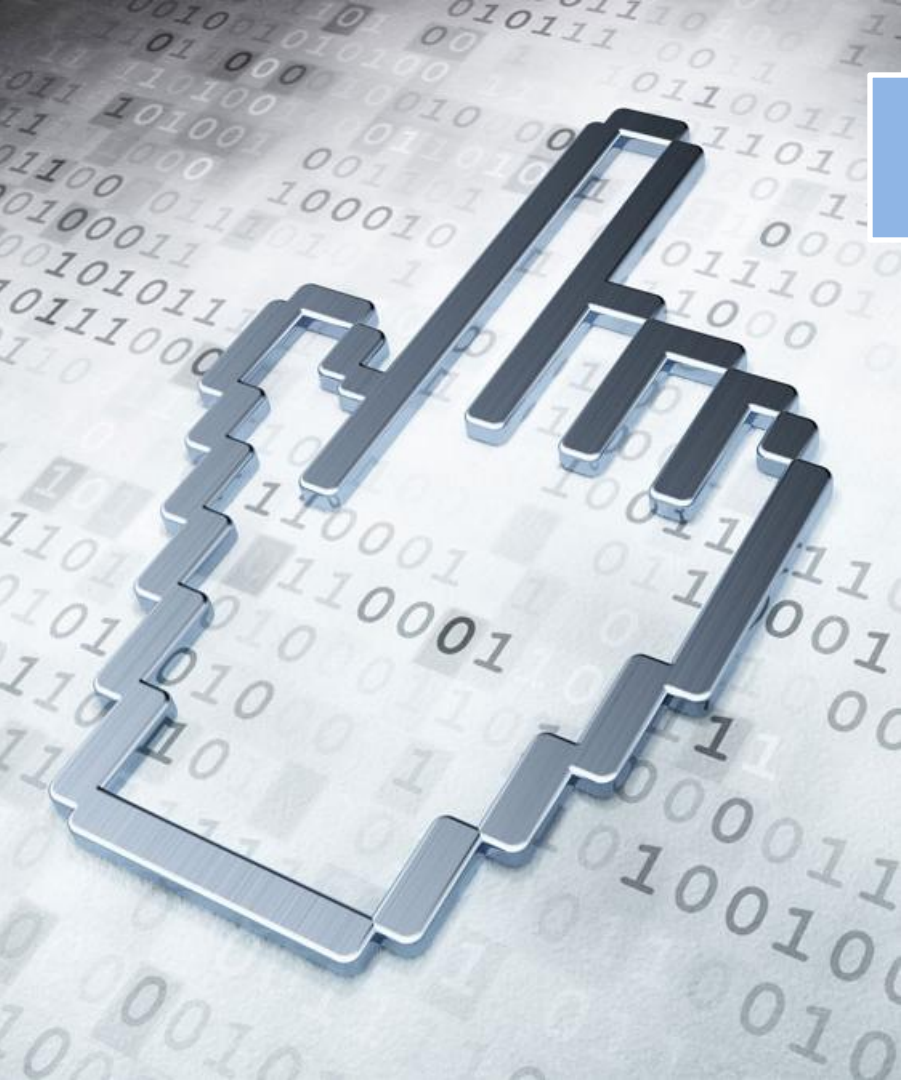
  (*b*) 字母*S*。通常表示开始符号

  (*c*) 小写、斜体的名字，如 *expr*、*stmt*等

  (*d*) 代表程序构造的大写字母。如*E*(表达式)、*T*(项)和*F*(因子)

符号约定

- 字母表中排在后面的大写字母（如*X*、*Y*、*Z*）
  表示文法符号（即终结符或非终结符）
- 字母表中排在后面的小写字母（主要是*u*、*v*、. . . 、*z*）
  表示终结符号串（包括空串）
- 小写希腊字母，如*α*、*β*、*γ*，表示文法符号串（包括空串）
- 除非特别说明，第一个产生式的左部就是开始符号



提纲

2.3 语言的定义

自然语言文法的例子：

① 句子　　名词短语　　动词短语

② 名词短语　　形容词　　名词短语

③ 名词短语　　名词

④ 动词短语　　动词

⑤ 形容词　　*little | pret*

⑥ 名词　　*boy | girl*

apple

有了文法，如何判定一个词串是否是满足文法规则的合法句子？

单词串：*little boy eats*

推导 (*Derivations*)和归约(*Reductions*)

- 推导:

给定文法$G=(V_T, V_N, P, S)$，如果$\alpha \rightarrow \beta \in P$，那么可以将符号串$\gamma\alpha\delta$中的$\alpha$替换为$\beta$，也就是说，将$\gamma\alpha\delta$ 重写(*rewrite*)为$\gamma\beta\delta$，记作 $\gamma\alpha\delta \quad \gamma\beta\delta$。此时，称文法中的符号串 $\gamma\alpha\delta$ 直接推导(*directly derive*)出 $\gamma\beta\delta$

  - 简而言之，就是用产生式的右部替换产生式的左部

推导 (*Derivations*)和归约(*Reductions*)

- 如果$\alpha_0 \Rightarrow \alpha_1$，$\alpha_1 \Rightarrow \alpha_2$，$\alpha_2 \Rightarrow \alpha_3$，…，$\alpha_{n-1} \Rightarrow \alpha_n$，则可以记作$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$，称符号串 α_0经过*n*步推导出α_n，可简记为$\alpha_0 \overset{n}{\Rightarrow} \alpha_n$
  - $\alpha \overset{+}{\Rightarrow} \alpha$
  - $\overset{*}{\Rightarrow}$ 表示“经过至少一步推导”
  - $\overset{0}{\Rightarrow}$ 表示“经过若干（包括0）步推导”

推导 (*Derivations*)和归约(*Reductions*)

例: *little boy eats apple*

文法：

① 句子 名词短语 动词短语

② 名词短语 形容词 名词短语

③ 名词短语 名词

④ 动词短语 动词 名词短语

⑤ 形容词 *little | pretty*

⑥ 名词 *boy | girl*

⑦ 名词 *apple | pie*

归约：归约是推导的逆过程

推导

归约

句子

名词短语 动词短语

形容词 名词短语 <动词短语

little 名词短语 <动词短语

little 名词 <动词短语

little boy <动词短语

little boy 动词 名词短语

little boy eats 名词短语

little boy eats 名词

little boy eats apple

最左推导和最右推导

- 最左推导(Left-most Derivation)
  - 每次推导都选择句型最左边的非终结符展开。

    ——与最右归约对应
- 最右推导(Right-most Derivation)
  - 每次推导都选择句型最右边的非终结符展开。

    ——与最左归约对应。

如何判断一个词串是否符合文法？

- 有了文法（语法规则），如何判定某一词串是否是满足文法规则的？
  - 推导（派生）-——从生成语言的角度
    - 从文法的开始符号开始，经过若干步推导，能够推导出该词串。
  - 归约 -——从识别语言的角度
    - 从该词串出发，经过若干步归约，能够归约出文法的开始符号。

句型和句子

- 如果 $S \overset{*}{\Rightarrow} \alpha$，$\alpha \in (V_T \cup V_N)^*$，则称$\alpha$是$G$的一个句型 (*sentential form*)
  - 一个句型中既可以包含终结符，又可以包含非终结符，也可能是空串
- 如果 $S \overset{*}{\Rightarrow} w$，$w \in V_T^*$，则称$w$是$G$的一个句子 (*sentence*)
  - 句子是不包含非终结符的句型

例

句子　　名词短语　　动词短语

形容词　　名词短语　<动词短语

little　名词短语　<动词短语

little　名词　<动词短语

little boy <动词短语

little boy　动词　　名词短语

little boy eats　名词短语

little boy eats　名词

句子 → *little boy eats apple*

句型

文法生成的语言

- 由文法G的开始符号S推导出的所有句子构成的集合称为文法G生成的语言，记为$L(G)$。即

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

思考：文法 $E \rightarrow E+E \mid E*E \mid (E) \mid \text{id}$ 生成的语言中包含多少个句子？

- 设有两文法G_1和G_2，如果$L(G_1) = L(G_2)$，则称G_1和G_2为等价文法。

例

- 文法G

① $S \rightarrow L \mid LT$

② $T \rightarrow L \mid D \mid TL \mid TD$

③ $L \rightarrow a \mid b \mid c \mid \ldots \mid z$

④ $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid 9$

课后练习:请写出无符号整数或浮点数的文法

$T \Rightarrow TL$

$\Rightarrow TDL$

$\Rightarrow TDDL$

$\Rightarrow TLDDL$

$\Rightarrow \ldots$

$\Rightarrow TD\ldots LDDL$

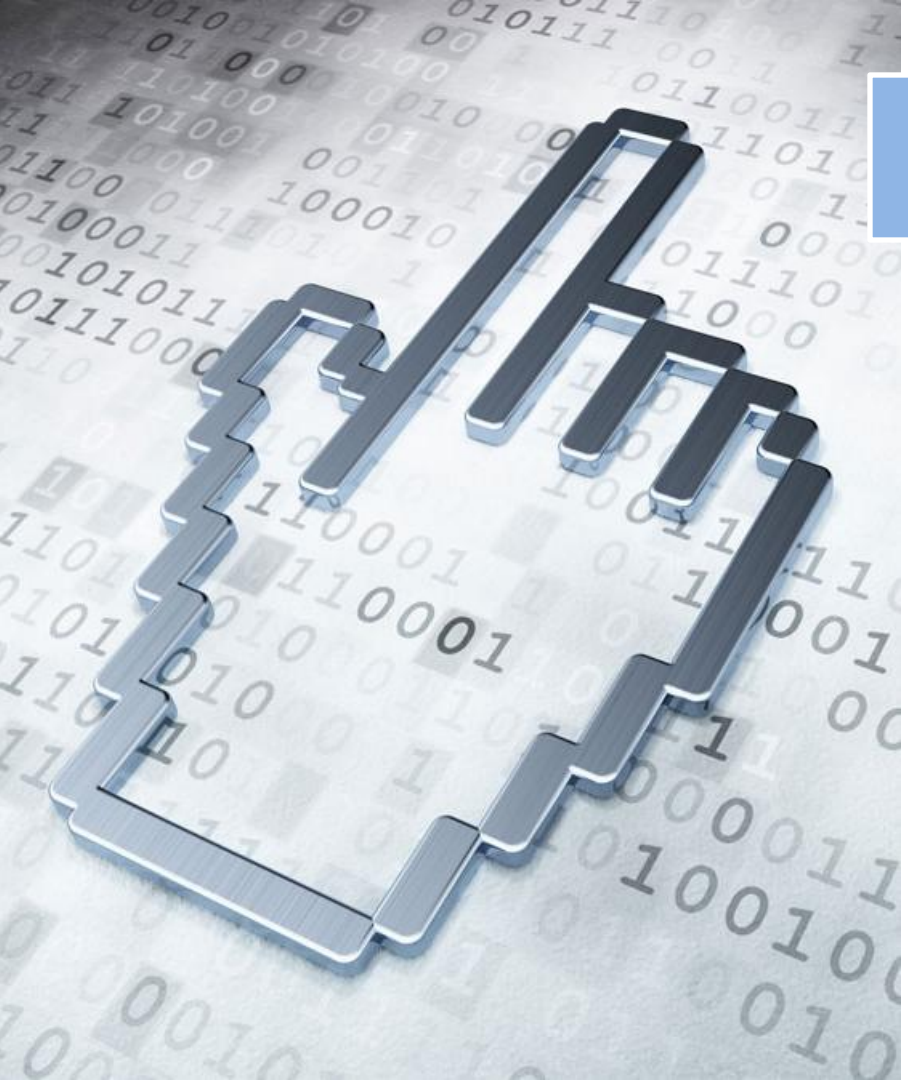
$\Rightarrow DD\ldots LDDL$

该文法识别的串：
由字母开头的字母数字串

语言上的运算

运算	定义和表示
L和M的并	$L \cup M = \{s \mid s$属于L或者s属于$M\}$
L和M的连接	$LM = \{st \mid s$属于L且t属于$M\}$
L的幂	$\begin{cases} L^0 = \{\varepsilon\} \\ L^n = L^{n-1}L, \ n \geqslant 1 \end{cases}$
L的Kleene闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

例：令L={a，b，…，z}，D={0，1，…，9}。则$L(L \cup D)^*$表示的语言是 字母开头的字母数字串的集合



提纲

2.4 文法的分类

- ***Chomsky*** **文法分类体系**
  - **0型文法 (*Type-0 Grammar*)**
  - **1型文法 (*Type-1 Grammar*)**
  - **2型文法 (*Type-2 Grammar*)**
  - **3型文法 (*Type-3 Grammar*)**

0型文法（*Type-0 Grammar*）

$$\alpha \rightarrow \beta$$

- 无限制文法(*Unrestricted Grammar*) /短语结构文法

  (*Phrase Structure Grammar, PSG*)

  - $\forall \alpha \rightarrow \beta \in P$ ， $\alpha \in (V_T \cup V_N)^+$ ，且α中至少包含1个非终结符， $\beta \in (V_T \cup V_N)^*$
- 0型语言
  - 由0型文法G生成的语言$L(G)$

1型文法（*Type-1 Grammar*）

$$\alpha \rightarrow \beta$$

- 上下文有关文法(*Context-Sensitive Grammar , CSG*)
  - $\forall \alpha \rightarrow \beta \in P$ ， $|\alpha| \leq |\beta|$
  - 产生式的一般形式： $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ ($\beta \neq \varepsilon$)
- 上下文有关语言（1型语言）
  - 由上下文有关文法 (1型文法) *G*生成的语言*L*(*G*)

可以包含 $S \rightarrow \varepsilon$，但此时*S*不能出现在任何产生式的右部

。

例 上下文有关语言的文法CSG

$L=\{a^n b^n c^n | n>0\}$的文法

$S \quad aBC|aSBC$

$CB \quad BC$

$aB \quad ab$

$bB \quad bb$

$bC \quad bc$

$cC \quad cc$

- “可以证明”不存在CFG G，使$L(G)=L$

2型文法（*Type-2 Grammar*）

$$\alpha \rightarrow \beta$$

- 上下文无关文法 (*Context-Free Grammar, CFG*)
  - $\forall \alpha \rightarrow \beta \in P$ ，$\alpha \in V_N$
  - 产生式的一般形式：$A \rightarrow \beta$ ，$A \in V_N$ ，$\beta \in (V_T \cup V_N)^*$

例：

$S \rightarrow L \mid LT$

$T \rightarrow L \mid D \mid TL \mid TD$

$L \rightarrow a \mid b \mid c \mid d \mid \ldots \mid z$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid 9$

2型文法（*Type-2 Grammar*）

$$\alpha \rightarrow \beta$$

- 上下文无关文法 (*Context-Free Grammar, CFG*)
  - $\forall \alpha \rightarrow \beta \in P$ ，$\alpha \in V_N$
  - 产生式的一般形式：$A \rightarrow \beta$ ，$A \in V_N$ ，$\beta \in (V_T \cup V_N)^*$
- 上下文无关语言（2型语言）
  - 由上下文无关文法 (2型文法) G生成的语言$L(G)$

通常使用上下文无关文法来描述程序设计语言的语法结构

3型文法（*Type-3 Grammar*）

$$\alpha \rightarrow \beta$$

- **正则文法** (*Regular Grammar, RG*) 设$A, B \in V_N$，$w \in V_T^+$
  - **右线性**(*Right Linear*)**文法**： $A \rightarrow wB$ 或 $A \rightarrow w$
  - **左线性**(*Left Linear*) **文法**： $A \rightarrow Bw$ 或 $A \rightarrow w$
  - 左线性文法和右线性文法等价，都称为正则文法

例（右线性文法）

① $S \rightarrow a \mid b \mid c \mid d$

② $S \rightarrow aT \mid bT \mid cT \mid dT$

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

④ $T \rightarrow aT \mid bT \mid cT \mid dT \mid 0T \mid 1T \mid 2T \mid 3T \mid 4T \mid 5T$

文法*G*（上下文无关文法）

① $S \rightarrow L \mid LT$

② $T \rightarrow L \mid D \mid TL \mid TD$

③ $L \rightarrow a \mid b \mid c \mid d$

④ $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5$

3型文法（*Type-3 Grammar*）

$$\alpha \rightarrow \beta$$

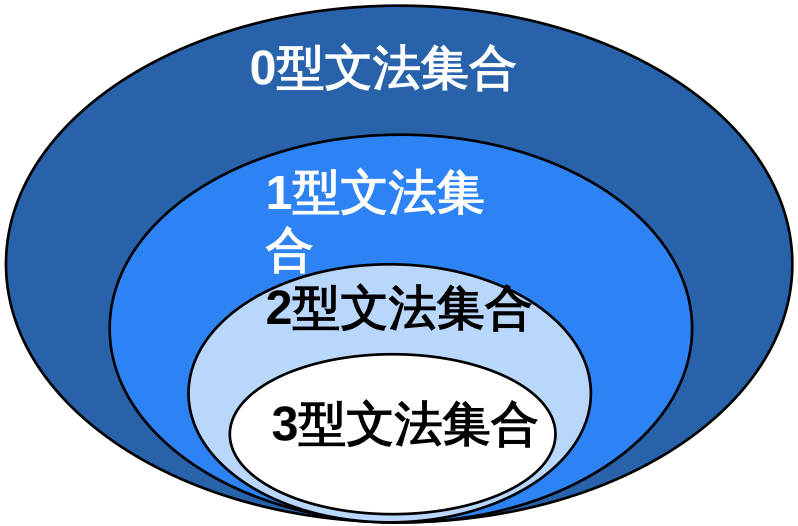
- 正则文法 (*Regular Grammar, RG* 设$A, B \in V_N$，$w \in V_T^+$
  - 右线性(*Right Linear*)文法： $A \rightarrow wB$ 或 $A \rightarrow w$
  - 左线性(*Left Linear*) 文法： $A \rightarrow Bw$ 或 $A \rightarrow w$
  - 左线性文法和右线性文法等价，都称为正则文法
- 正则语言（3型语言）
  - 由正则文法 (3型文法) *G*生成的语言*L*(*G*)

正则文法能描述程序设计语言的多数单词

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：



不同的文法可以定义相同的语言

使用上下文无关文法描述程序设计语言的语法

程序设计语言中CFG无法描述的语义规则

- 程序设计语言的有些语言规则不能用上下文无关文法描述

例2.9 $L_1=\{ wcw \mid w\in\{a,b\}^+ \}$

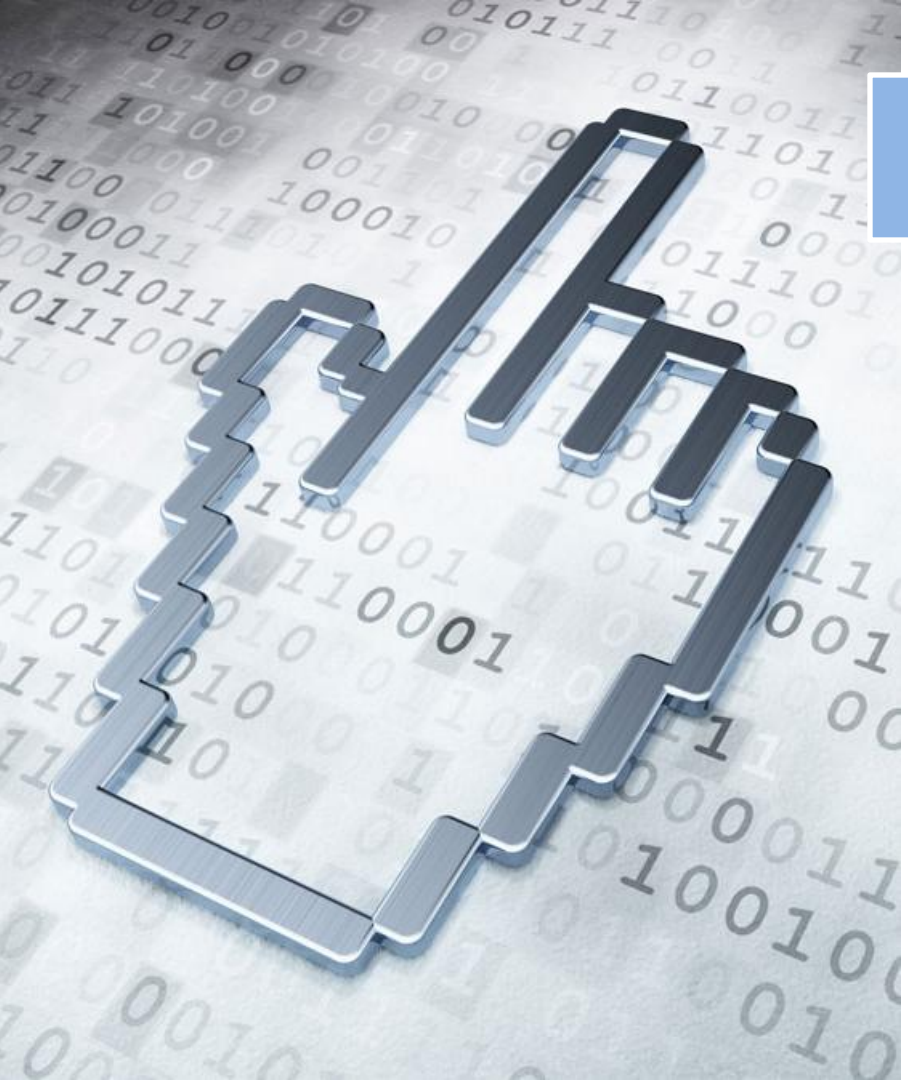
$aabcaab$就是L_1的一个句子

- 检查程序中标识符的声明应先于引用的抽象

例2.10 $L_2=\{a^n b^m c^n d^m \mid n, m\geq 0\}$

- 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

上下文无关文法无法描述的语义规则检查由语义分析阶段完成



提纲

CFG 的语法分析树

分析树是推导的图形化表示

最左推导：*E* -*E* -(*E*) -(*E*+*E*) -(id+*E*) -(id+id)

最右推导：*E* -*E* -(*E*) -(*E*+*E*) -(*E*+id) -(id+id)

文法：

① $E \rightarrow E+E$

② $E \rightarrow E*E$

③ $E \rightarrow -E$

④ $E \rightarrow (E)$

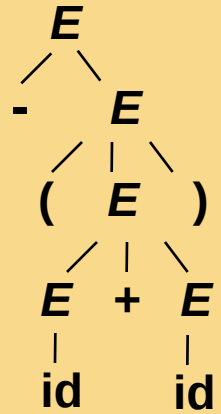
⑤ $E \rightarrow$ id

句型：

-

- 最左推导和最右推导最终的分析树是一样的
- 分析树忽略了推导的次序

分析树：



- 叶结点的标号既可以是非终结符，也可以是终结符。从左到右排列叶节点得到的符号串称为是这棵树的产出(*yield*)或边缘(*frontier*)

（句型的）短语

- 给定一个句型，其分析树中的每一棵子树的边缘称为该句型的一个短语(*phrase*)
  - 如果子树只有父子两代结点，那么这棵子树的边缘称为该句型的一个直接短语(*immediate phrase*)

文法：

① $E \rightarrow E + E$

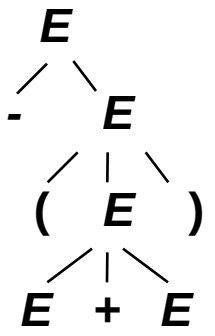
② $E \rightarrow E * E$

③ $E \rightarrow - E$

④ $E \rightarrow (E)$

⑤ $E \rightarrow$ id

分析树：



句型：

- - (E+E)

短语：

- - (E+E)
- (E+E)
- E+E

直接短语：

- E+E

句型的短语一定是句型的子串

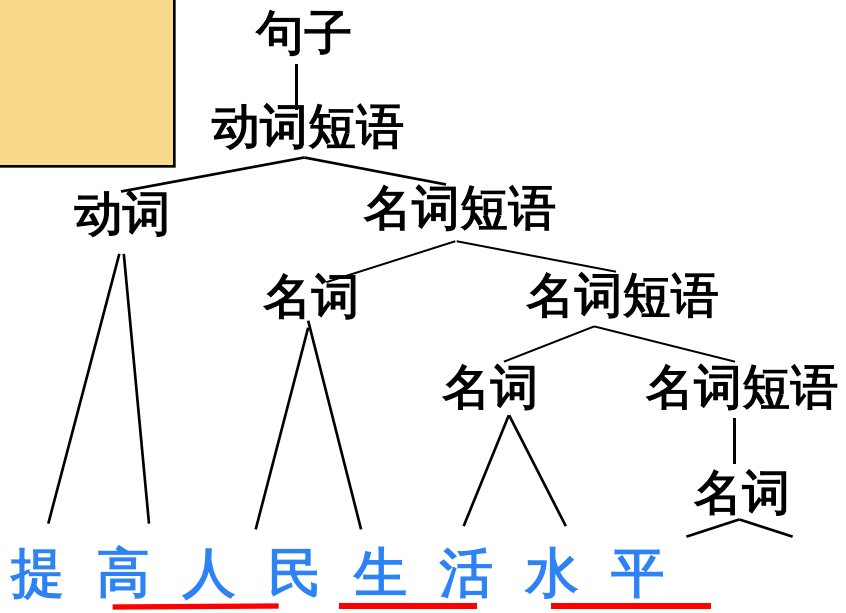
直接短语一定是某产生式的右部

例

文法：
① 句子　　动词短语
② 动词短语　　动词　　名词短语
③ 名词短语　　名词　　名词短语　|　名词
④ 动词　　提 高
⑤ 名词　　高 人|人 民|民 生|生 活|活 水|水 平

输入：提 高 人 民 生 活 水 平

但产生式的右部不总是给定句型的直接短语



二义性文法 (*Ambiguous Grammar*)

- 如果一个文法可以为某个句子生成多棵分析树，则称这个文法是二义性的

例：考虑表达式下面的文法 *G*[*E*],其产生式如下：

E　*E*+*E*　*E***E*　(*E*)　*a*

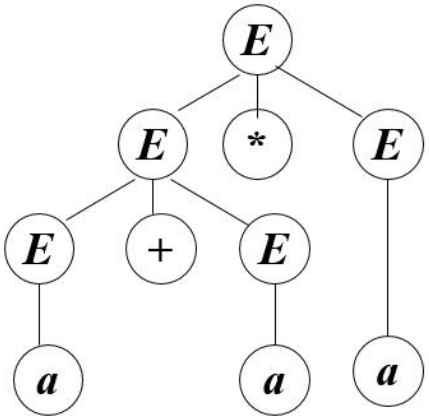
对于句子*a*+*a***a*, 有如下两个最左推导

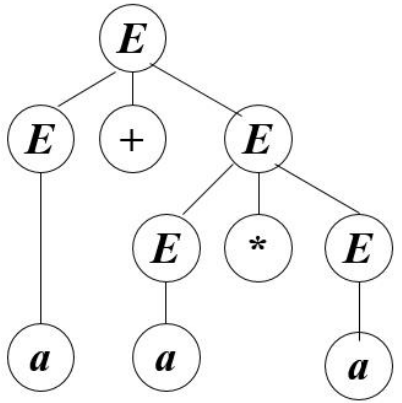
E　*E*+*E*　*a*+*E*　*a*+*E***E*

a+*a***E*　*a*+*a***a*

E　*E***E*　*E*+*E***E*　*a*+*E***E*

a+*a***E*　*a*+*a***a*

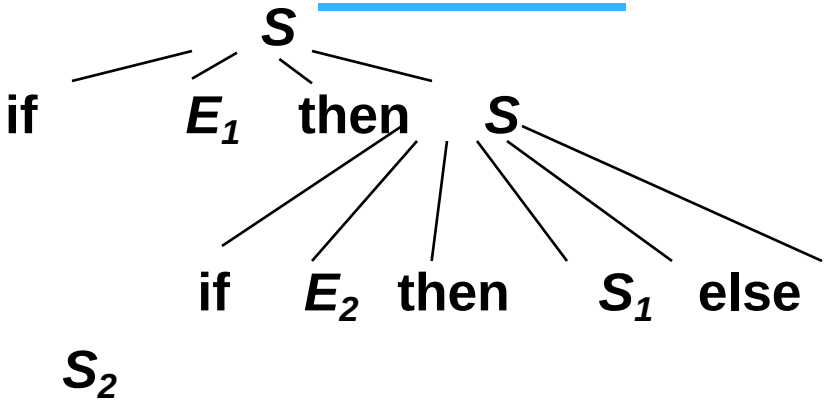


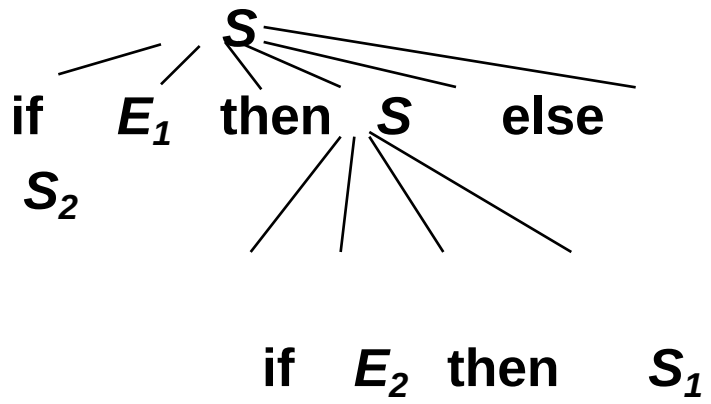


例

- 文法
  - *S*　　**if** *E* **then** *S*
    if *E* **then** *S* **else** *S*　　条件语句
    other　　其他语句
- 句型
  - **if** E_1 **then if** E_2 **then** S_1 **else** S_2

- 二义性文法的句子可能存在不同的语法分析树，对应不同的语义，因此语法分析时需要确定选择哪一种语法分析树，即消除二义性。





消除二义性的第一种方法

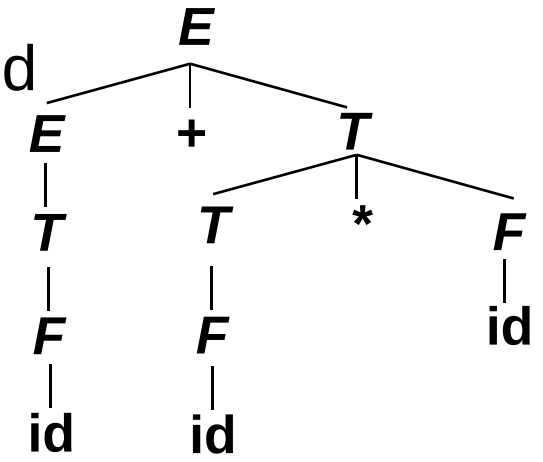
改造文法：描述算术运算的无二义性文法的产生式

(1) E → E+T　　(2) E → T

(3) T → T*F　　(4) T → F

(5) F → (E)　　(6) F → id

句子：id+id*id 的语法分析树



消除二义性的第一种方法

改造文法：描述if语句的无二义性文法的产生式

S *matched_s* *unmatched_s*

matched_s if *expr* then *matched_s* else *matched_s*
other

unmatched_s if *expr* then *S*

if *expr* then *matched_s* else
unmatched_s

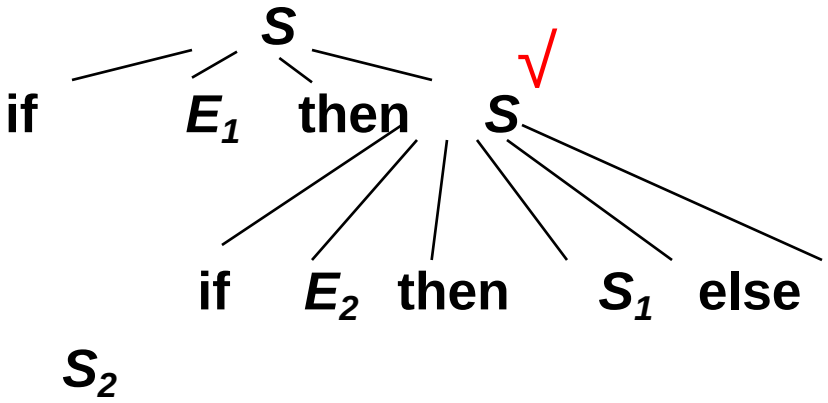
- 改造文法方法可能很复杂，且不存在通用的方法能将任意二义性方法转换为非二义性文法。
- 因此，为了方便处理，也可以使用二义性文法+消歧规则

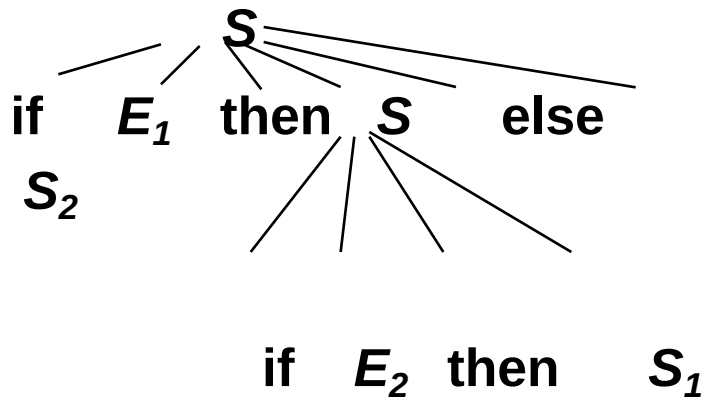
消除二义性的第二种方法

- 文法
  - *S* **if** *E* **then** *S*
    if *E* **then** *S* **else** *S*
    other
- 句型
  - **if** E_1 **then if** E_2 **then** S_1 **else** S_2

语法分析时加入消除二义性的规则

消歧规则：每个else和最近的尚未匹配的if匹配





无二义性文法的判定

- 对于任意一个上下文无关文法，不存在一个算法，判定它是否是无二义性的；但能给出一组充分条件，满足这组充分条件的文法是无二义性的
  - 确定性的上下文无关文法是无二义性的
  - LL(k)、LR(k)文法是无二义性的
  - ……

本章小结

- 基本概念
  - 字母表、字母表的正闭包和克林闭包、串、串的幂运算
- 文法的定义
  - 形式化定义，终结符、非终结符、文法符号、产生式、符号串
- 语言的定义：形式化定义，推导与归约、语言与文法的关系
- 文法的分类：四种类型的文法定义及关系，四种类型语言
- CFG的分析树
  - 句型分析树的构造、短语、直接短语、二义性文法

课程主要内容

1．绪论	（2学时）
2. 语言及其文法	（2学时）
3．词法分析	（4学时）
4．语法分析	（9学时）
5．语法制导翻译	（6学时）
6．中间代码生成	（7学时）
7．运行时的存贮组织	（3学时）
8．代码优化	（5学时）
9．代码生成	（2学时）

结束