



第二章 文法和语言的概念和表示

- 2.1 预备知识 - 形式语言基础
- 2.2 文法和语言的定义
- 2.3 几个重要概念
- 2.4 文法的表示：扩充的BNF范式和语法图
- 2.5 文法和语言的分类



2.1 预备知识

一、字母表和符号串

字母表：符号的非空有限集 例： $\Sigma = \{a, b, c\}$

符号：字母表中的元素 例：a, b, c

符号串：符号的有穷序列 例：a, aa, ac, abc, ..

空符号串：无任何符号的符号串(ϵ)

符号串的形式定义

有字母表 Σ , 定义：

(1) ϵ 是 Σ 上的符号串；

(2) 若x是 Σ 上的符号串，且 $a \in \Sigma$ ，则 ax 或 xa 是 Σ 上的符号串；

(3) y是 Σ 上的符号串，iff (当且仅当) y可由(1)和(2)产生。

符号串集合：由符号串构成的集合。



二、符号串和符号串集合的运算

1. 符号串相等：若 x 、 y 是集合上的两个符号串，则 $x = y$ iff（当且仅当）组成 x 的每一个符号和组成 y 的每一个符号依次相等。

2. 符号串的长度： x 为符号串，其长度 $|x|$ 等于组成该符号串的符号个数。

例： $x = STV$ ， $|x|=3$ 。



3. 符号串的联接：若 x 、 y 是定义在 Σ 上的符号串，且 $x = XY$, $y = YX$, 则 x 和 y 的联接 $xy = XYYX$ 也是 Σ 上的符号串。

注意：一般 $xy \neq yx$, 而 $\epsilon x = x\epsilon$

4. 符号串集合的乘积运算：令 A 、 B 为符号串集合，定义

$$AB = \{ xy \mid x \in A, y \in B \}$$

例： $A = \{a,b\}$, $B = \{c,d\}$, $AB = ?$ {ac, ad, bc, bd}

因为 $\epsilon x = x\epsilon = x$, 所以 $\{\epsilon\}A = A$ $\{\epsilon\} = A$



5. 符号串集合的幂运算：有符号串集合A，定义

$$A^0 = \{\varepsilon\}, \quad A^1 = A, \quad A^2 = AA, \quad A^3 = AAA, \dots$$

$$\dots A^n = A^{n-1}A = AA^{n-1}, \quad n > 0$$

6. 符号串集合的闭包运算：设A是符号串集合，定义

$$A^+ = A^1 \cup A^2 \cup A^3 \cup \dots \cup A^n \cup \dots$$

称为集合A的正闭包。

$$A^* = A^0 \cup A^+$$

称为集合A的闭包。

例： $A = \{x, y\}$

$$A^+ = \{\underline{x, y}, \underline{\underline{xx, xy, yx, yy}}, \underline{\underline{\underline{xxx, xxy, xyx, xyy}}}, \dots\}$$

$$A^* = \{\underline{\varepsilon}, \underline{x, y}, \underline{\underline{xx, xy, yx, yy}}, \underline{\underline{\underline{xxx, xxy, xyx, xyy}}}, \dots\}$$



★ 为什么对符号、符号串、符号串集合以及它们的运算感兴趣？

若A为某语言的基本字符集

$$A = \{a, b, \dots, z, 0, 1, \dots, 9, +, -, \times, /, (,), =, \dots\}$$

B为单词集

$$B = \{\text{begin}, \text{end}, \text{if}, \text{then}, \text{else}, \text{for}, \dots, <\text{标识符}>, <\text{常量}>, \dots\}$$

则 $B \subset A^*$ 。

语言的句子是定义在B上的符号串。

若令C为句子集合，则 $C \subset B^*$ ， 程序 $\subset C$ 。



2.2 文法的非形式讨论

1. 什么是文法：文法是对语言结构的定义与描述。即从形式上用于描述和规定语言结构的称为“文法”（或称为“语法”）。

例：有一句子：“我是大学生”。这是一个在语法、语义上都正确的句子，该句子的结构（称为语法结构）是由它的语法决定的。在本例中它为“主谓结构”。

如何定义句子的合法性？

- 有穷语言
- 无穷语言



2. 语法规则：我们通过建立一组规则，来描述句子的语法结构。
规定用 “::=” 表示 “由……组成”。

<句子> ::= <主语><谓语>

<主语> ::= <代词> | <名词>

<代词> ::= 你 | 我 | 他

<名词> ::= 王民 | 大学生 | 工人 | 英语

<谓语> ::= <动词><直接宾语>

<动词> ::= 是 | 学习

<直接宾语> ::= <代词> | <名词>



3. 由规则推导句子：有了一组规则之后，可以按照一定的方式用它们去推导或产生句子。

推导方法：从一个要识别的符号开始推导，即用相应规则的右部来替代规则的左部，每次仅用一条规则去进行推导。

<句子> => <主语><谓语>

<主语><谓语> => <代词><谓语>

.....

这种推导一直进行下去，直到所有带<>的符号都由终结符号替代为止。



推导方法：从一个要识别的符号开始推导，即用相应规则的**右部**来替代规则的**左部**，每次仅用一条规则去进行推导。

<句子> => <主语><谓语>
=> <代词><谓语>
=> 我 <谓语>
=> 我 <动词><直接宾语>
=> 我 是 <直接宾语>
=> 我 是 <名词>
=> 我 是 大学生

<句子> ::= <主语><谓语>
<主语> ::= <代词> | <名词>
<代词> ::= 你 | 我 | 他
<名词> ::= 王民 | 大学生 | 工人 | 英语
<谓语> ::= <动词><直接宾语>
<动词> ::= 是 | 学习
<直接宾语> ::= <代词> | <名词>



例：有一英语句子： The big elephant ate the peanut.

<句子> ::= <主语><谓语>

<主语> ::= <冠词><形容词><名词>

<冠词> ::= the

<形容词> ::= big

<名词> ::= elephant

<谓语> ::= <动词><宾语>

<动词> ::= ate

<宾语> ::= <冠词><名词>

<名词> ::= peanut



<句子> => <主语><谓语>
=> <冠词><形容词><名词><谓语>
=> the <形容词><名词><谓语>
=> the big <名词><谓语>
=> the big elephant <谓语>
=> the big elephant <动词><宾语>
=> the big elephant ate <宾语>
=> the big elephant ate <冠词><名词>
=> the big elephant ate the <名词>
=> the big elephant ate the peanut

<句子> ::= <主语><谓语>
<主语> ::= <冠词><形容词><名词>
<冠词> ::= the
<形容词> ::= big
<名词> ::= elephant | peanut
<谓语> ::= <动词><宾语>
<动词> ::= ate
<宾语> ::= <冠词><名词>

最左推导！



<句子> => <主语> <谓语>
=> <主语> <动词> <宾语>
=> <主语> <动词> <冠词> <名词>
=> <主语> <动词> <冠词> peanut
=> <主语> <动词> the peanut
=> <主语> ate the peanut
=> <冠词> <形容词> <名词> ate the peanut
=> <冠词> <形容词> elephant ate the peanut
=> <冠词> big elephant ate the peanut
=> the big elephant ate the peanut

<句子> ::= <主语> <谓语>
<主语> ::= <冠词> <形容词> <名词>
<冠词> ::= the
<形容词> ::= big
<名词> ::= elephant | peanut
<谓语> ::= <动词> <宾语>
<动词> ::= ate
<宾语> ::= <冠词> <名词>

最右推导！



上述推导可写成<句子> $\stackrel{+}{\Rightarrow}$ the big elephant ate the peanut

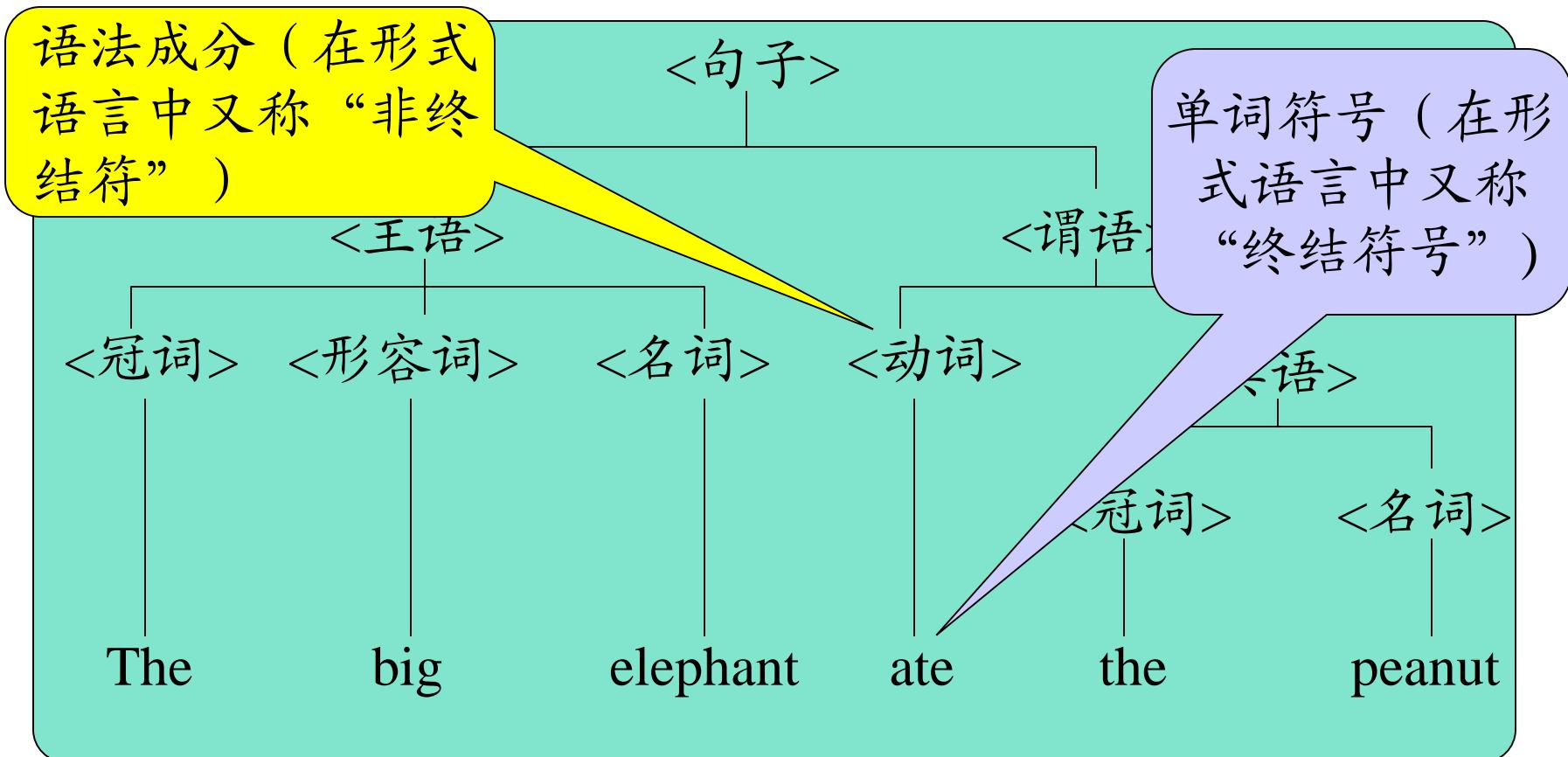
说明：

- (1) 有若干语法成分同时存在时，我们总是从最左的语法成分进行推导，这称之为**最左推导**。类似地还有**最右推导**（一般推导）。
- (2) 除了最左和最右推导，还可能存在其它形式的推导。
- (3) 从一组规则可推出不同的句子，如以上规则还可推出“大象吃象”、“大花生吃象”、“大花生吃花生”等句子，它们在语法上都正确，但在语义上都不正确。

所谓**文法**是在**形式上**对句子结构的定义与描述，而未涉及**语义**问题。



4. 语法树：我们用语法树来描述一个句子的语法结构。





2.3 文法和语言的形式定义

2.3.1 文法的定义

定义1. 文法 $G = (V_n, V_t, P, Z)$

V_n : 非终结符号集

V_t : 终结符号集

P : 产生式或规则的集合

Z : 开始符号 (识别符号) $Z \in V_n$

$$V = V_n \cup V_t$$

称为文法的字汇表

规则: $U ::= x$

$$U \in V_n, x \in V^*$$

补: 规则的定义

规则是一个有序对 (U, x) , 通常写为: $U ::= x$ 或 $U \rightarrow x$

$$|U| = 1 \quad |x| \geq 0$$



例：无符号整数的文法：

$G[<\text{无符号整数}>] = (Vn, Vt, P, Z)$

$Vn = \{<\text{无符号整数}>, <\text{数字串}>, <\text{数字}>\}$

$Vt = \{0, 1, 2, 3, \dots, 9\}$

$P = \{<\text{无符号整数}> \rightarrow <\text{数字串}>$

$<\text{数字串}> \rightarrow <\text{数字串}> <\text{数字}>$

$<\text{数字串}> \rightarrow <\text{数字}>$

$<\text{数字}> \rightarrow 0$

$<\text{数字}> \rightarrow 1$

.....

$<\text{数字}> \rightarrow 9 \}$

$Z = <\text{无符号整数}>$



★ 几点说明：

产生式左边符号构成集合 V_n , 且 $Z \in V_n$

有些产生式具有相同的左部, 可以合在一起。文法的BNF表示

例、<无符号整数> \rightarrow <数字串>

<数字串> \rightarrow <数字串><数字> | <数字>

<数字> \rightarrow 0 | 1 | 2 | 3 | | 9

给定一个文法, 实际只需给出产生式集合, 并指定识别符号。
(识别符号一般约定为第一条规则的左部符号)

例、 $G[<\text{无符号整数}>]$

<无符号整数> \rightarrow <数字串>

<数字串> \rightarrow <数字串><数字> | <数字>

<数字> \rightarrow 0 | 1 | 2 | 3 | | 9



2.3.2 推导的形式定义

定义2：文法G: $v = x \cup y, w = x u y,$

其中 $x, y \in V^*$, $U \in Vn$, $u \in V^*$,

若 $U ::= u \in P$, 则 $v \xrightarrow[G]{} w$.

若 $x = y = \varepsilon$, 有 $U ::= u$, 则 $U \xrightarrow[G]{} u$

例如: G[<无符号整数>]

- (1) <无符号整数> \rightarrow <数字串>
- (2) <数字串> \rightarrow <数字串> <数字>
- (3) <数字串> \rightarrow <数字>

- (4) <数字> \rightarrow 0
- (5) <数字> \rightarrow 1
-
- (13) <数字> \rightarrow 9



例如：G[<无符号整数>]

(1) <无符号整数> → <数字串>

(2) <数字串> → <数字串><数字>

(3) <数字串> → <数字>

(4) <数字> → 0

(5) <数字> → 1

.....

(13) <数字> → 9

<无符号整数> $\xrightarrow{(1)}$ <数字串> $\xrightarrow{(2)}$ <数字串><数字>
 $\xrightarrow{(3)}$ <数字><数字> $\xrightarrow{(4)}$ 1 <数字>
 $\xrightarrow{(5)}$ 1 0

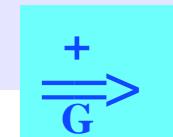
当符号串已没有非终结符号时，推导就必须终止。因为终结符不可能出现在规则左部，所以将在规则左部出现的符号称为非终结符号。



定义3：文法 G , $U_0, U_1, U_2, \dots, U_n \in V^+$

if $v = U_0 \xrightarrow{G} U_1 \xrightarrow{G} U_2 \xrightarrow{G} \dots \xrightarrow{G} U_n = w \quad (n > 0)$

则 $v \xrightarrow[G]{+} w$



例: $\langle \text{无符号整数} \rangle \Rightarrow \langle \text{数字串} \rangle \Rightarrow \langle \text{数字串} \rangle \langle \text{数字} \rangle$

$\Rightarrow \langle \text{数字} \rangle \langle \text{数字} \rangle \Rightarrow 1 \langle \text{数字} \rangle$

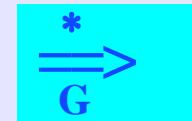
$\Rightarrow 1 0$

即 $\langle \text{无符号整数} \rangle \xrightarrow[G]{+} 10$

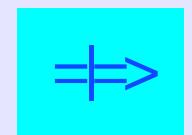


定义4：文法 G , $v, w \in V^+$

if $v \xrightarrow[G]{+} w$, 或 $v = w$, 则 $v \xrightarrow[G]{*} w$ 。



定义5：规范推导：有 $xUy \Rightarrow xuy$, if $y \in V_t^*$, 则此推导为规范的，记为 $xUy \xrightarrow{\text{def}} xuy$ 。



直观意义：规范推导 = 最右推导

最右推导：若符号串中有两个以上的非终结符时，先推右边的。

最左推导：若符号串中有两个以上的非终结符时，先推左边的。

若有 $v \xrightarrow{\text{def}} U_0 \xrightarrow{\text{def}} U_1 \xrightarrow{\text{def}} U_2 \xrightarrow{\text{def}} \dots \xrightarrow{\text{def}} U_n \xrightarrow{\text{def}} w$, 则 $v \xrightarrow[G]{+} w$ 。



2.3.3 语言的形式定义

定义6：文法 $G[Z]$

文法 $G[Z]$ 所产生的所有句子的集合

- (1) 句型： x 是句型 $\Leftrightarrow Z \xrightarrow{*} x$, 且 $x \in V^*$;
- (2) 句子： ~~x 是句子 $\Leftrightarrow Z \xrightarrow{*} x$, 且 $x \in V_t^*$~~ ;
- (3) 语言： $L(G[Z]) = \{x \mid x \in V_t^*, Z \xrightarrow{*} x\}$;

形式语言理论可以证明以下两点：

- (1) $G \rightarrow L(G)$;
- (2) $L(G) \rightarrow G_1, G_2, \dots, G_n$;

已知文法，求语言，通过推导；

已知语言，构造文法，无形式化方法，更多是凭经验。



例： $\{ab^n a \mid n \geq 1\}$ ，构造其文法

$$\begin{aligned} G_1[Z]: \quad Z &\rightarrow aBa \\ B &\rightarrow b | bB \end{aligned}$$

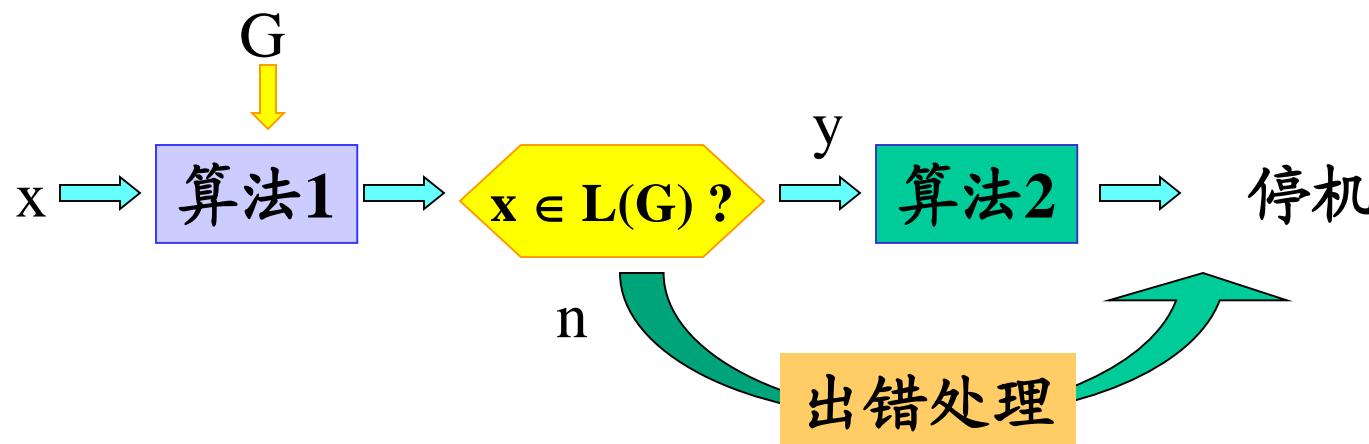
$$\begin{aligned} G_2[Z]: \quad Z &\rightarrow aBa \\ B &\rightarrow b | Bb \end{aligned}$$

定义7. G 和 G' 是两个不同的文法，若 $L(G) = L(G')$ ，
则 G 和 G' 为等价文法。



编译感兴趣的问题是：

- 给定 x, G , 求 $x \in L(G)$?





2.3.4 递归文法

1. 递归规则：规则右部有与左部相同的符号

对于 $U ::= xUy$

若 $x = \varepsilon$, 即 $U ::= Uy$, 左递归;

若 $y = \varepsilon$, 即 $U ::= xU$, 右递归。

2. 递归文法：文法 G , 存在 $U \in V^n$

if $U \stackrel{+}{\Rightarrow} \dots U \dots$, 则 G 为递归文法(自嵌入递归);

if $U \stackrel{+}{\Rightarrow} U \dots$, 则 G 为左递归文法;

if $U \stackrel{+}{\Rightarrow} \dots U$, 则 G 为右递归文法。



会造成死循环（后面将详细论述）

3. 左递归文法的缺点：不能用自顶向下的方法来进行语法分析

4. 递归文法的优点：可用有穷条规则，定义无穷语言

例：对于前面给出的无符号整数的文法是有递归文法，用13条规则就可以定义出所有的无符号整数。若不用递归文法，那将要用多少条规则呢？

<无符号整数> → <数字串>
<数字串> → <数字串> <数字> | <数字>
<数字> → 0 | 1 | 2 | 3 | | 9





2.3.5 句型的短语、简单短语和句柄

定义8. 给定文法 $G[Z]$, $w = xuy \in V^+$, 为该文法的句型,
若 $Z \xrightarrow{*} xUy$, 且 $U \xrightarrow{+} u$, 则 u 是句型 w 相对于 U 的短语;
若 $Z \xrightarrow{*} xUy$, 且 $U \xrightarrow{=} u$, 则 u 是句型 w 相对于 U 的简单短语。
其中 $U \in V_n$, $u \in V^+$, $x, y \in V^*$

直观理解：短语是前面句型中的某个非终结符所能推出的符号串。

任何句型本身一定是相对于识别符号 Z 的短语。



定义9. 任一句型的最左简单短语称为该句型的句柄。

给定句型找句柄的步骤：

短语 → 简单短语 → 句柄

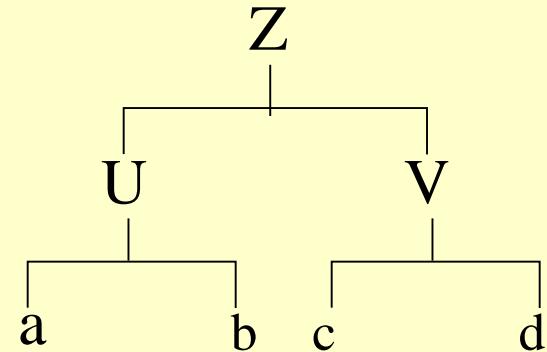


注意：短语、简单短语是相对于句型而言。一个句型可能有多个短语、简单短语，但句柄只能有一个。



2.4 语法树与二义性文法

2.4.1 推导与语法树



(1) 语法树：句子结构的图示表示法，它是一种有向图，由结点和有向边组成。

结点： 符号

根结点：识别符号

中间结点：非终结符

叶结点：终结符或非终结符

有向边： 表示结点间的派生关系。



(2) 句型的推导及语法树的生成（自顶向下）

给定 $G[Z]$, 句型 w :

可建立推导序列, $Z \xrightarrow[G]{*} w$

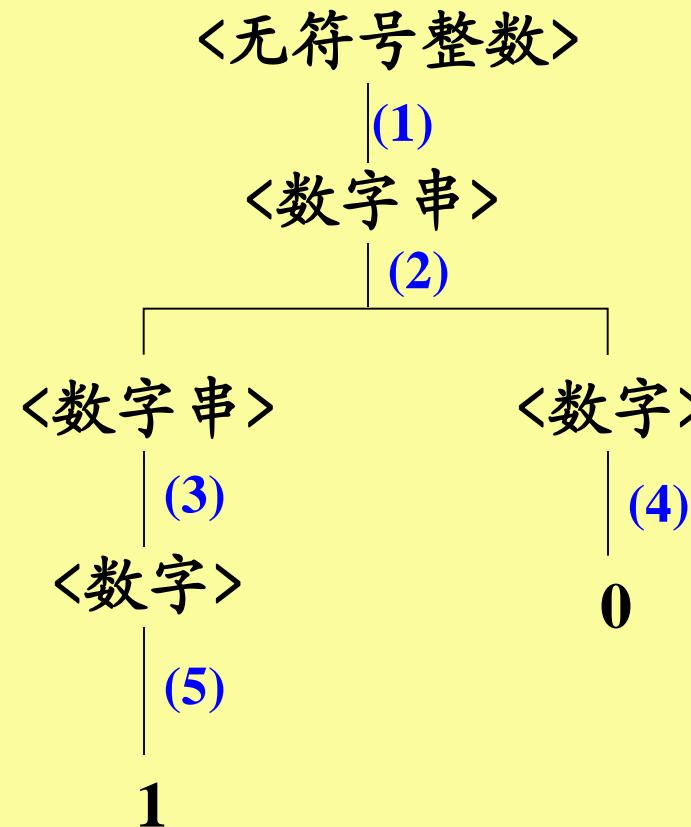
可建立语法树, 以 Z 为树根结点, 每步推导生成语法树的一枝, 最终可生成句型的语法树。



注意一个重要事实: 文法所能产生的句子, 可以用不同的推导原则(使用产生式顺序不同)将其推导出来。语法树的生成规律不同, 但最终生成的语法树形状完全相同。某些文法有此性质, 而某些文法不具有此性质。

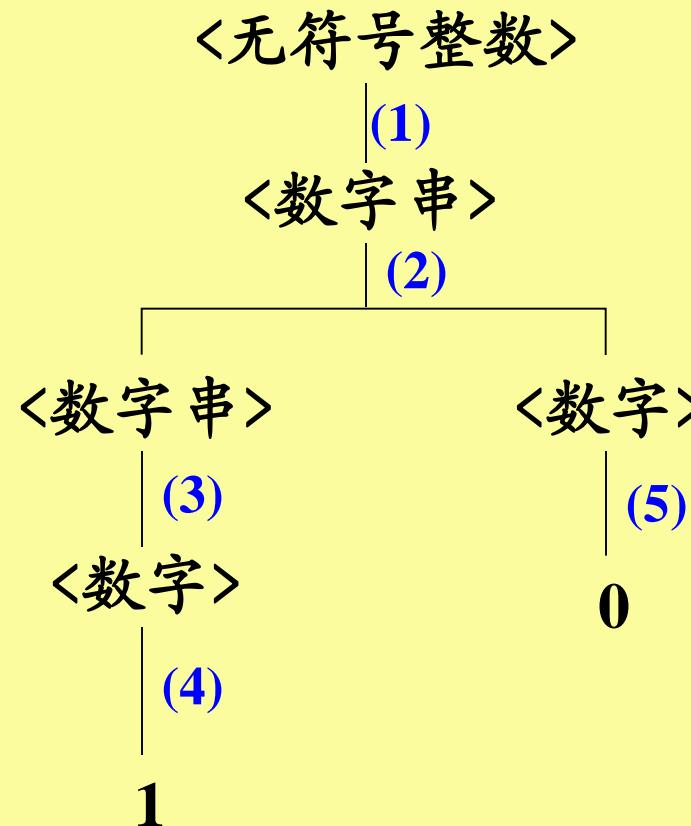


一般推导：



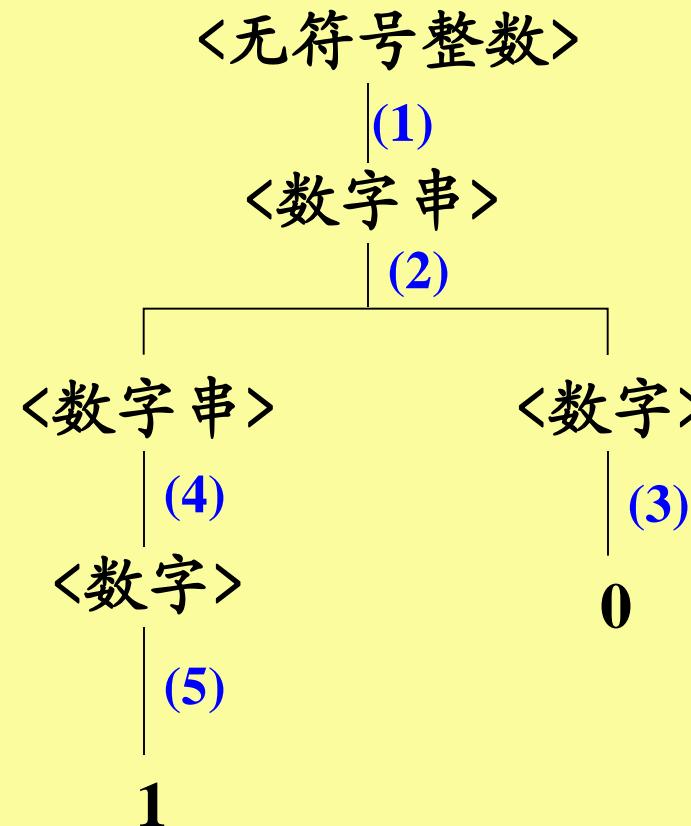


最左推导：





最右推导：





(3) 子树与短语

子树：语法树中的某个结点（子树的根）连同它向下派生的部分所组成。

定理

某子树的末端结点按自左向右顺序为句型中的符号串，则该符号串为该句型的相对于该子树根的短语。

只需画出句型的语法树，然后根据子树找短语→简单短语→句柄。



(4) 树与推导

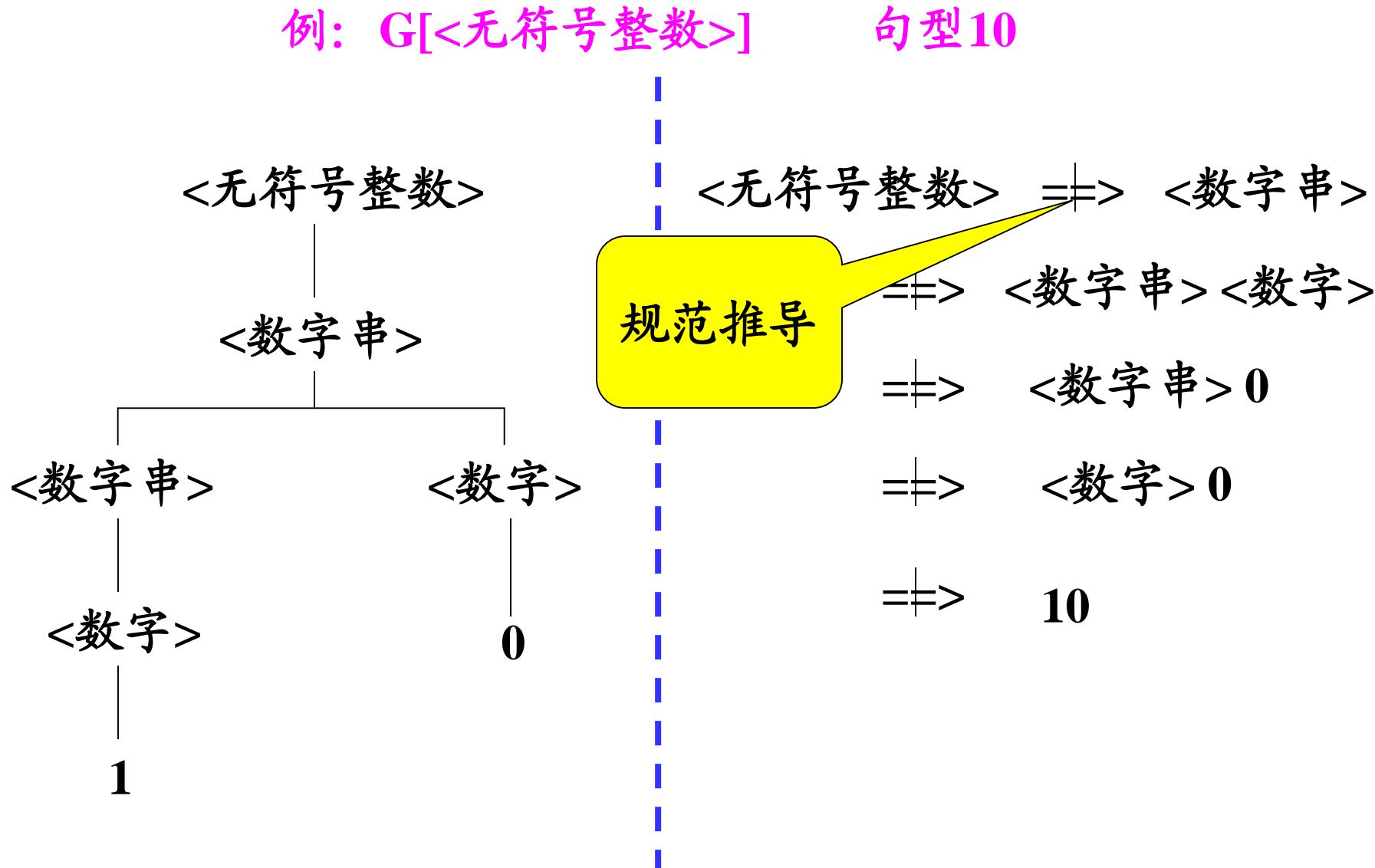
句型推导过程 \iff 句型语法树的生长过程

1 由推导构造语法树

从识别符号开始，自右向左建立推导序列。



由根结点开始，自上而下建立语法树。





2 由语法树构造推导

自下而上地修剪子树的末端结点，直至把整棵树剪掉（留根），每剪一次对应一次规约。



从句型开始，自左向右地逐步进行规约，建立推导序列。

定义12. 对句型中最左简单短语（句柄）进行的规约称为规范规约。



规范规约与规范推导互为逆过程

<无符号整数>

<无符号整数>

$\not\Rightarrow$ <数字串>

$\not\Rightarrow$ <数字串> <数字>

$\not\Rightarrow$ <数字串> 0

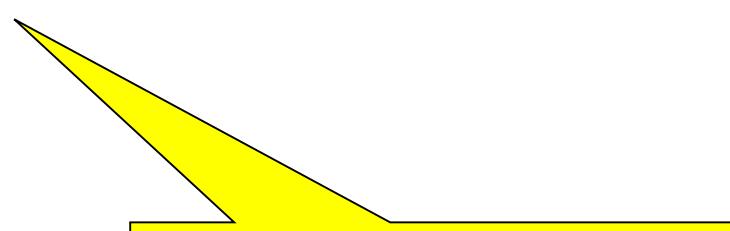
$\not\Rightarrow$ <数字> 0

$\not\Rightarrow$ 10



定义13.通过规范推导或规范规约所得到的句型称为规范句型。

<数字><数字>



不是规范推导！



2.4.2 文法的二义性

定义14.1 若对于一个文法的某一句子存在两棵不同的语法树，则该文法是二义性文法，否则是无二义性文法。

换而言之，无二义性文法的句子只有一棵语法树，尽管推导过程可以不同。

下面举一个二义性文法的例子：

$$G[E]: \quad E := E+E \mid E^*E \mid (E) \mid i$$

$$V_n = \{E\}$$

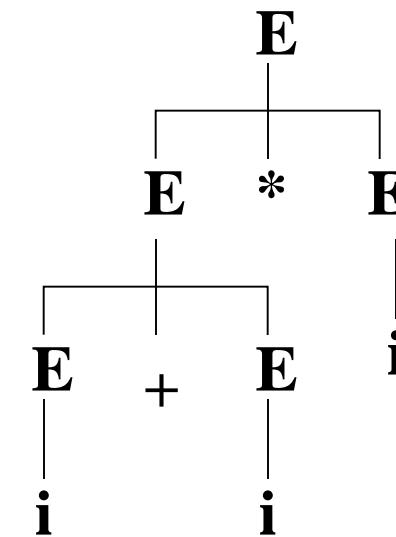
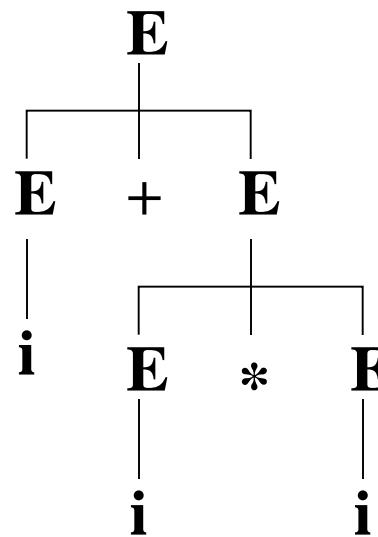
$$V_t = \{ +, *, (,), i \}$$



对于句子 $S = i + i * i \in L(G[E])$, 存在不同的规范推导:

- (1) $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * i \Rightarrow E + i * i \Rightarrow i + i * i$
- (2) $E \Rightarrow E * E \Rightarrow E * i \Rightarrow E + E * i \Rightarrow E + i * i \Rightarrow i + i * i$

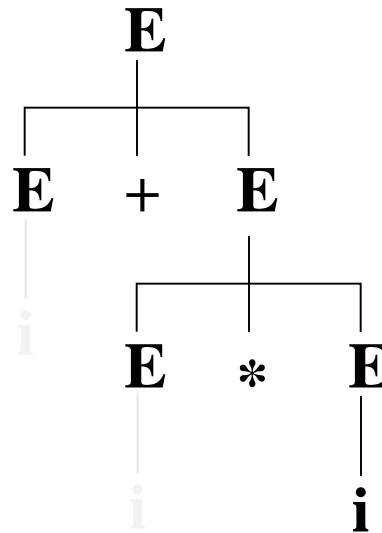
这两种不同的推导对应了两种不同的语法树



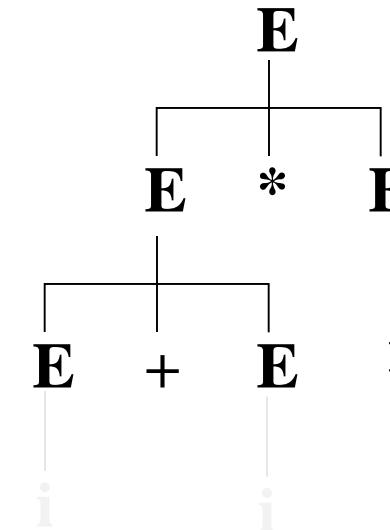


定义14.2 若一个文法的某句子存在两个不同的规范推导，则该文法是二义性的，否则是无二义性的。

以上是自顶向下来文法的二义性，我们还可以自底向上来看文法的二义性。上例中，规范句型 $E+E^*i$ 是由 $i+i * i$ 通过两步规范规约得到的，但对于同一个句型 $E+E^*i$ ，它有两个不同的句柄（对应上述两棵不同的语法树）： i 和 $E + E$ 。因此语法的二义性意味着句型的句柄不唯一。



句柄: i



句柄: $E + E$

定义14.3 若一个文法的某规范句型的句柄不唯一，则该文法是二义性的，否则是无二义性的。



若文法是二义性的，则在编译时就会产生不确定性。

遗憾的是在理论上已经证明：**文法的二义性是不可判定的**，即不可能构造出一个算法，通过有限步骤来判定任一文法是否有二义性。

现在的解决办法是：提出一些**限制条件**，称为无二义性的充分条件。当文法满足这些条件时，就可以判定文法是无二义性的。

由于无二义性文法比较简单，我们也可以采用另一种解决办法：即不改变二义性文法，而是确定一种**编译算法**，使该算法满足无二义性充分条件。



例：算术表达式的文法

- $E ::= E + E \mid E * E \mid (E) \mid i$

- $E ::= E + T \mid T$
- $T ::= T * F \mid F$
- $F ::= (E) \mid i$

例：Pascal 条件语句的文法

$<\text{条件语句}> ::= \text{If } <\text{布尔表达式}> \text{then} <\text{语句}> \mid$

$\text{If } <\text{布尔表达式}> \text{then } <\text{语句}> \text{else } <\text{语句}>$

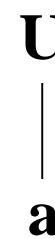
$<\text{语句}> ::= <\text{条件语句}> \mid <\text{非条件语句}> \mid \dots\dots$

$\text{If } B \text{ then If } B \text{ then stmt else stmt}$

2.5 有关文法的实用限制

若文法中有如 $U ::= U$ 的规则，则这就是有害规则，它会引起二义性。

例如存在 $U ::= U$, $U ::= a \mid b$, 则句子a有两棵语法树:





多余规则：

- (1) 在推导文法的所有句子中，始终用不到的规则。即该规则的左部非终结符不出现在任何句型中。
- (2) 在推导句子的过程中，一旦使用了该规则，将推不出任何终结符号串。即该规则中含有推不出任何终结符号串的非终结符。

例如给定 $G[Z]$ ，若其中关于 U 的规则只有如下一条：

$U ::= x \cup y$

该规则是多余规则。

若还有 $U ::= a$ ，则此规则
并非多余

若某文法中无有害规则或多余规则，则称该文法是压缩过的。



例：

① **S→Be**

② **S→Ec**

③ **A→Ae**

④ **A→e**

⑤ **A→A**

⑥ **B→Ce**

⑦ **B→Af**

⑧ **C→Cf**

⑨ **D→f**

① **S→Be**

② **B→Af**

③ **A→Ae**

④ **A→e**

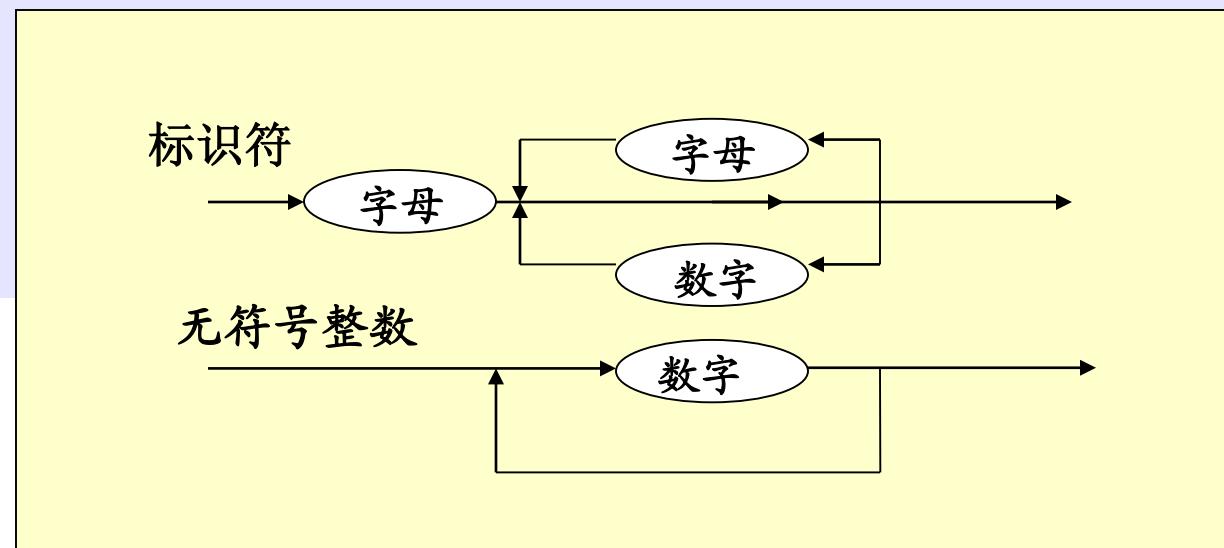


2.7 文法的其它表示法

1、扩充的BNF表示(Backus Normal Form)

- BNF的元符号: <, >, ::=, |
- 扩充的BNF的元符号: <, >, ::=, |, {, }, [,], ()

2、语法图





2.8 文法和语言分类

形式语言：用文法和自动机所描述的没有语义的语言。

语言定义： $L(G[Z]) = \{ x \mid x \in Vt^*, Z \xrightarrow{*} x \}$

文法定义：乔姆斯基将所有文法都定义为一个**四元组**：

$$G = (V_n, V_t, P, Z)$$

V_n : 非终结符号集

V_t : 终结符号集

P : 产生式或规则的集合

Z : 开始符号（识别符号） $Z \in V_n$



文法和语言分类：0型、1型、2型、3型

这几类文法的差别在于对产生式施加不同的限制。

0型：
 $P: u ::= v$
其中 $u \in V^+$, $v \in V^*$

0型文法称为短语结构文法。规则的左部和右部都可以是符号串，一个短语可以产生另一个短语。

0型语言：L0 这种语言可以用图灵机(Turing)接受。



1型： P: $xUy ::= xuy$

其中 $U \in V^n$,

$x, y, u \in V^*$

称为上下文敏感或上下文有关。也即只有在 x, y 这样的上下文中才能把 U 改写为 u 。

这类文法的任何产生式 $\alpha ::= \beta$ 均满足 $|\alpha| \leq |\beta|$ (其中 $|\alpha|$ 和 $|\beta|$ 分别为 α 和 β 的长度)；仅仅 $S ::= \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。也就是说，一般不允许把 u 替换成空串 ε 。

1型语言：L1 这种语言可以由一种线性界限自动机接受。



2型: P: $U ::= u$

其中 $U \in V^n$,

$u \in V^*$

称为上下文无关文法。也即把 U 改写为 u 时，不必考虑上下文。

注意：2型文法与BNF表示相等价。

2型语言: L2 这种语言可以由下推自动机接受。



3型文法:

(左线性)

$$P: U ::= T \\ \text{或 } U ::= wT$$

其中 $U, w \in V_n$
 $T \in V_t$

(右线性)

$$P: U ::= T \\ \text{或 } U ::= T w$$

其中 $U, w \in V_n$
 $T \in V_t$

3型文法称为正则文法。它是对2型文法进行进一步限制。

3型语言: L_3 又称正则语言、正则集合
这种语言可以由有穷自动机接受。



根据上述讨论, $L_0 \supset L_1 \supset L_2 \supset L_3$

0型文法可以产生 L_0 、 L_1 、 L_2 、 L_3 , 但2型文法只能产生 L_2 , 不能产生 L_1 。



小结

- 掌握符号串和符号串集合的运算、文法和语言的定义。
- 几个重要概念：递归、短语、简单短语和句柄、语法树、文法的二义性、文法的实用限制等。
- 掌握文法的表示：BNF、扩充的BNF范式、语法图。
- 了解文法和语言的分类。



本 章 作 业

- P29 第3、4题
- P38 第1、2、4、5、6、7题
- P46 第1、5、6、8、9
- P53 第2题

做 $i + i * i$ 或 $i + i + i$ 其中一个即可