第二章 文法和语言的概念和表示

- 预备知识 － 形式语言基础
- 文法和语言的定义
- 若干术语和重要概念
- 文法的表示：扩充的BNF范式和语法图
- 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：∑={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∑，定义：

（1） ε 是∑上的符号串；

（2）若x是∑上的符号串，且a∈ ∑，则ax或xa是∑上的符号串；

（3）y是∑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。

- 通常约定：
  - 用英文字母表开头的小写字母和字母表靠近末尾的大写字母来表示符号

    如：a, b, c, d, …, r 和 S, T, U, V, W, X, Y, Z
  - 用英文字母表靠近末尾的小写字母来表示符号串

    如： s, t, u, v, w, x, y, z
  - 用英文字母表开头的大写字母来表示符号串集合

    如：A, B, C, D, …, R

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≠yx，而εx＝xε

4. 符号串集合的乘积运算：令A、B为符号串集合，

定义 AB＝{ xy |x∈A, y∈B}

例：A＝{s, t}, B = {u,v}, AB= ？

{su，sv，tu，tv}

因为εx＝xε＝x，所以{ε}A=A {ε} =A

问题

{ε}A=A {ε} =A

{}A=A {} =?

$\phi A = A\phi = \phi$

5. 符号串集合的幂运算：有符号串集合A，定义

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

…… …… $A^n=A^{n-1}A=AA^{n-1}$ ，n>0

6. 符号串集合的闭包运算：设A是符号串集合，定义

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

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}

$A^+ = \{\underbrace{x,y}_{A^1},\ \underbrace{xx,xy,yx,yy}_{A^2},\ \underbrace{xxx,xxy,xyx,xyy,yxx,yxy,yyx,yyy}_{A^3},\ \ldots\ldots\}$

$A^* = \{\underbrace{\varepsilon}_{A^0},\ \underbrace{x,y}_{A^1},\ \underbrace{xx,xy,yx,yy}_{A^2},\ \underbrace{xxx,xxy,xyx,xyy,yxx,yxy,yyx,yyy}_{A^3},\ \ldots\ldots\}$

★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　　　（把字符看作符号）

A＝{a, b, ……z, 0, 1, ……, 9, +, －, ×, _, /, (,), =, ……}

B为单词集　　　（单词是符号串）

B ={begin, end, if, then, else, for, ……,<标识符>,<常量>, ……}

则$B \subset A^*$。

（把单词看作符号，句子便是符号串）

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

- 若把字符看作符号，则单词就是符号串，单词集合就是符号串的集合。
- 若把单词看作符号，则句子就是符号串，而所有句子的集合（即语言）就是符号串的集合。

习题：p29　3,4

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”（或“定义为...”）。

<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>
<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>

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

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〉
〈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〉

<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

上述推导可写成<句子> $\overset{+}{\Rightarrow}$ **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，类似的有最右推导(还有一般推导）。

(2) 从一组语法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。

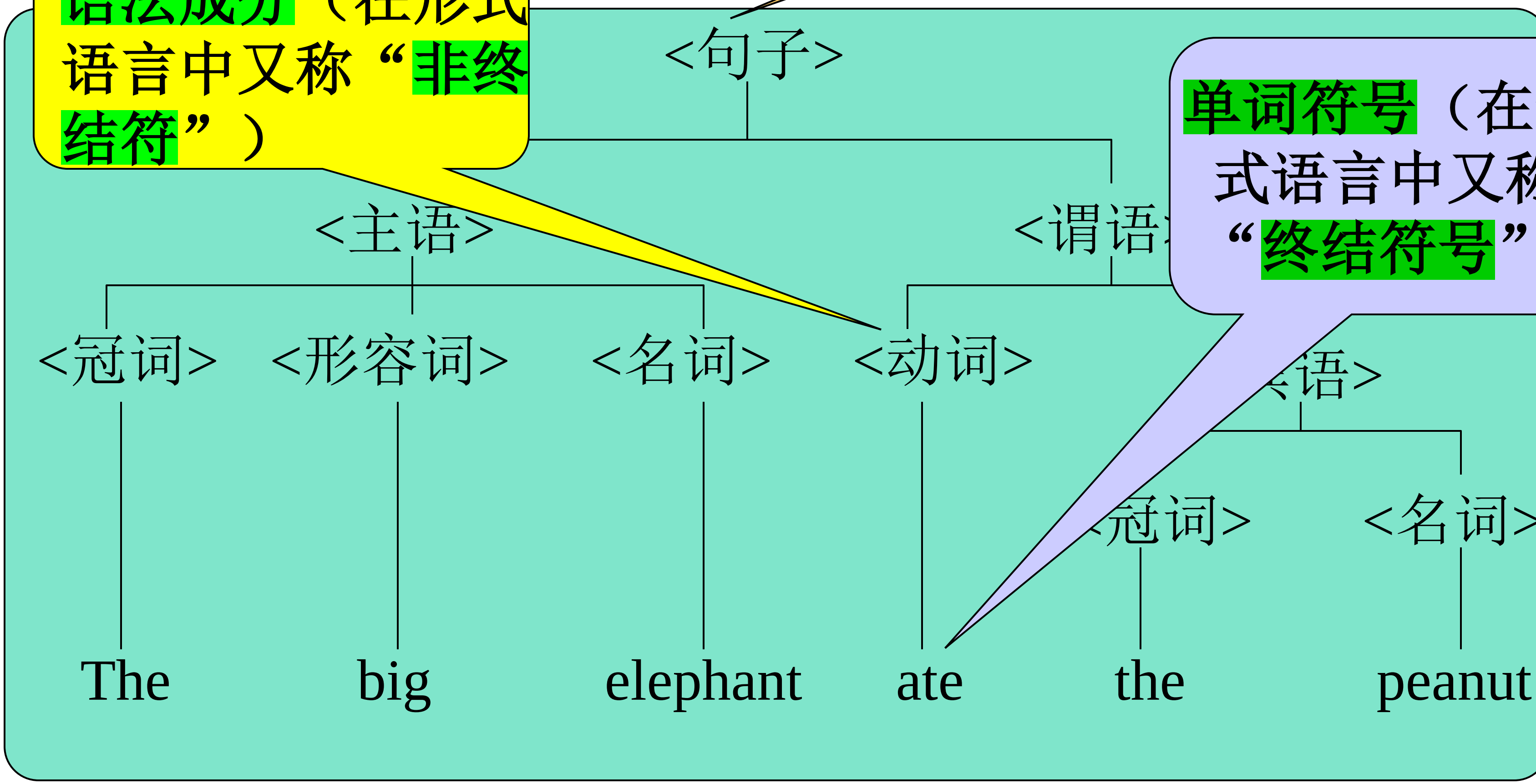
所谓文法是在形式上对句子结构的定义与描述，而未涉及语义问题。

4. 语法（推导）树：我们用语法（推导）树来描述一个句子的语法结构。

识别符号

语法成分（在形式语言中又称“非终结符”）

单词符号（在形式语言中又称“终结符号”）



2.3 文法和语言的形式定义

2.3.1文法的定义

$V = V_n \cup V_t$
称为文法的字汇表

定义1. 文法G=（Vn，Vt，P，Z）

Vn：非终结符号集

Vt：终结符号集

P：产生式或规则的集合

Z：开始符号（识别符号） $Z \in V_n$

规则：U ::= x
$U \in V_n$, $x \in V^*$

规则的定义：

规则是一个有序对(U, x), 通常写为:

$U ::= x$ 或 $U \rightarrow x$ ， $|U| = 1$ $|x| \geq 0$

例：无符号整数的文法：

G[<无符号整数>]=（Vn，Vt，P，Z）

Vn＝{<无符号整数>,<数字串>, <数字>}

Vt = {0, 1, 2, 3, …… 9}

P = {<无符号整数> → <数字串> ,

<数字串> → <数字串> <数字>,

<数字串> → <数字> ,

<数字> →0 ,

<数字> →1 ,

…………

<数字> →9 }

Z = <无符号整数>

★ 几点说明：

产生式左边符号构成集合Vn，且 Z ∈Vn

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> → <数字串>

<数字串> → <数字串> <数字> | <数字>

<数字> →0 | 1 | 2 | 3 | …… | 9

给定一个 文法，需给出产生式（规则）集合，并指定识别符号

例： G[<无符号整数>]:

<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

2.3.2 推导的形式定义

定义2：文法G：$v = xUy$，$w = xuy$,

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

若$U ::= u \in P$，则$v \underset{G}{\Rightarrow} w$。

若$x = y = \varepsilon$，有$U ::= u$，则$U \underset{G}{\Rightarrow} u$

根据文法和推导定义，可推出终结符号串，所谓通过文法能推出句子来。

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> $\underset{(1)}{\Longrightarrow}$ <数字串> $\underset{(2)}{\Longrightarrow}$ <数字串> <数字>

$\underset{(3)}{\Longrightarrow}$ <数字><数字> $\underset{(4)}{\Longrightarrow}$ 1 <数字>

$\underset{(5)}{\Longrightarrow}$ 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，$u_0, u_1, u_2, \ldots\ldots, u_n \in V^+$

if $v = u_0 \underset{G}{==>} u_1 \underset{G}{==>} u_2 \underset{G}{==>} \ldots\ldots \underset{G}{==>} u_n = w$

则 $v \overset{+}{\underset{G}{==>}} w$

例：<无符号整数>==> <数字串> ==> <数字串> <数字>

==> <数字><数字> ==>1 <数字>

==>1 0

即 <无符号整数> $\overset{+}{\underset{G}{==>}}$ 10

定义4：文法G，有v，w $\in V^+$

if $v \overset{+}{\underset{G}{\Rightarrow}} w$，或v=w，则 $v \overset{*}{\underset{G}{\Rightarrow}} w$

定义5：规范推导：有xUy ==> xuy，若 $y \in V_t^*$，则此推导为规范的，记为 xUy =|=> xuy

直观意义：规范推导＝最右推导

最右推导：若规则右端符号串中有两个以上的非终结符时，先推右边的。

最左推导：若规则右端符号串中有两个以上的非终结符时，先推左边的。

若有 $v = u_0$ =|=> u_1 =|=> u_2 =|=>……=|=> $u_n = w$，则 $v \overset{+}{=|=>} w$

2.3.3 语言的形式定义

文法G[Z]所产生的所有句子的集合

定义6：文法G[Z]

（1）句型：x是句型 $\Leftrightarrow$ $Z \overset{*}{\Rightarrow} x$，且 $x \in V^*$；

（2）句子：x是句子 $\Leftrightarrow$ $Z \overset{+}{\Rightarrow} x$，且 $x \in V_t^*$；

（3）语言：$L(G[Z]) = \{x \mid x \in V_t^*, Z \overset{+}{\Rightarrow} x\}$；

形式语言理论可以证明以下两点：

（1）$G \rightarrow L(G)$；

（2）$L(G) \rightarrow G1, G2, \ldots\ldots, Gn$；

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^na | $n \geq 1$}，构造其文法

$G_1[Z]$:

$Z \rightarrow aBa$,

$B \rightarrow b \mid bB$

$G_2[Z]$:

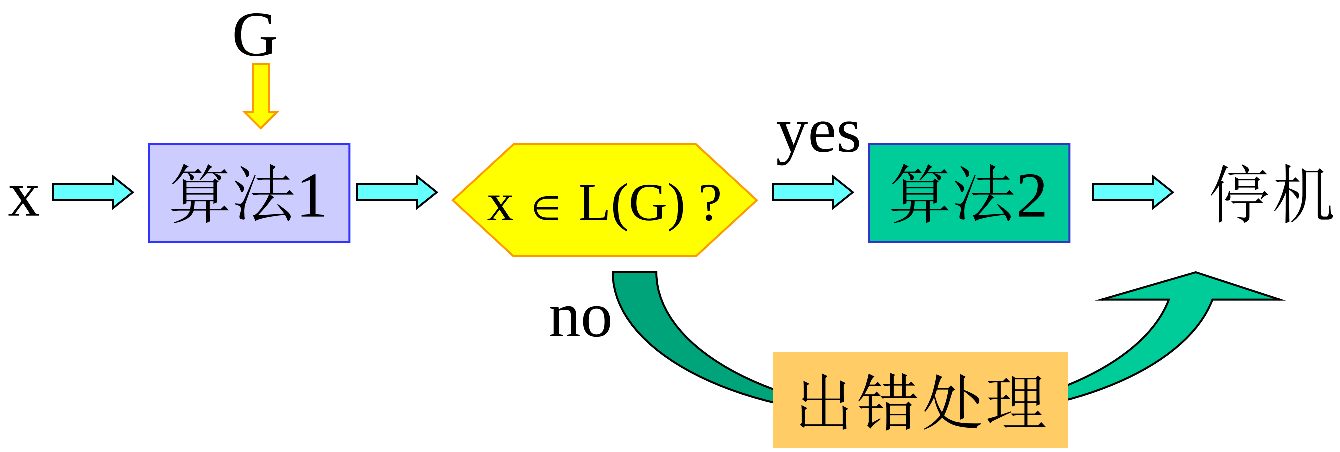
$Z \rightarrow aBa$,

$B \rightarrow b \mid Bb$

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:

- 给定句子 x 以及文法 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（非终结符）

对于 U::= xUy

若x= ε， 即U::= Uy， 左递归

若y= ε， 即U::= xU， 右递归

若x, y≠ ε， 即U::= xUy， 自嵌入递归

2.递归文法：文法G，存在U ∈Vn

if $U \overset{+}{\Rightarrow} \ldots U \ldots$, 则G为递归文法;

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

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

3. 递归文法的优点：可用有穷条规则，定义无穷语言

会造成死循环（后面将详细论述）

4. 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

例：对于前面给出的无符号整数的文法是左递归文法，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

例1：

G[<无符号整数>]

<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

L(G[<无符号整数>]) = Vt^+

Vt = {0,1,2,…,9}

例2：

G[S]: S::= aB | bB　　　L(G[S]) ={ aa, ab, ba, bb }

B::=a | b

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 $Z \overset{*}{\Longrightarrow} xUy$, 且$U \overset{+}{\Longrightarrow} u$, 则u是句型w相对于U的短语；

若 $Z \overset{*}{\Longrightarrow} xUy$, 且$U \Longrightarrow u$, 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

例：文法G[〈无符号整数〉], w = 〈数字串〉1

〈无符号整数〉 => <数字串> => <数字串><数字>
=> 〈数字串〉1

求：短语、 简单短语和句柄。

短语：<数字串>1，1 ； 简单短语：1；句柄：1

例: 文法G[<无符号整数>]， w = <数字串>1

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

x U y x U y x U y x u uyy

<无符号整数>=> <数字串> => <数字串> <数字> ==> <数字串>1

x U y U u

(1) <无符号整数> $\overset{*}{\Rightarrow}$ <无符号整数> <无符号整数>$\overset{+}{\Rightarrow}$ <数字串>1

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

习题2-3：p381-9

复习： 文法：G=（Vn，Vt，P，Z）

- 若有规则U ::= u，且 v＝x Uy，w＝xuy,

  则有推导 x Uy =>xuy ,即 v ⇒w

- 注意弄清 => $\stackrel{+}{=}>$ $\stackrel{*}{=}>$ $\models>$ $\stackrel{+}{\models}>$的概念

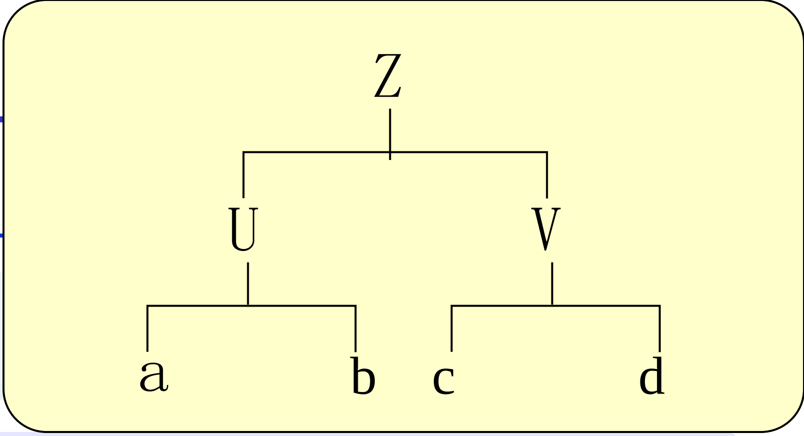
- 文法G对应的语言 L（G[Z]）={x| x∈V_t*, Z$\stackrel{+}{\Rightarrow}$ x };
- 递归U$\stackrel{+}{=}>$...U...
- 有句型w=xuy，若 Z$\stackrel{*}{=}>$ xUy, 且U$\stackrel{+}{=}>$u,

  则u是句型w相对于U的短语

- 简单短语和最左简单短语（句型）的概念

2.4 语法树与二义性文法

2.4.1 推导与语法（推导）树



（1）语法（推导）树：句子(句型）结构的图示表示法，
它是有向图，由结点和有向边组成。

结点：符号

根结点：　识别符号（非终结符）
中间结点：非终结符
叶结点：　终结符或非终结符

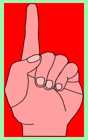
有向边：表示结点间的派生关系

(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，$Z \overset{*}{\underset{G}{==>}} w$

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w的语法树。



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

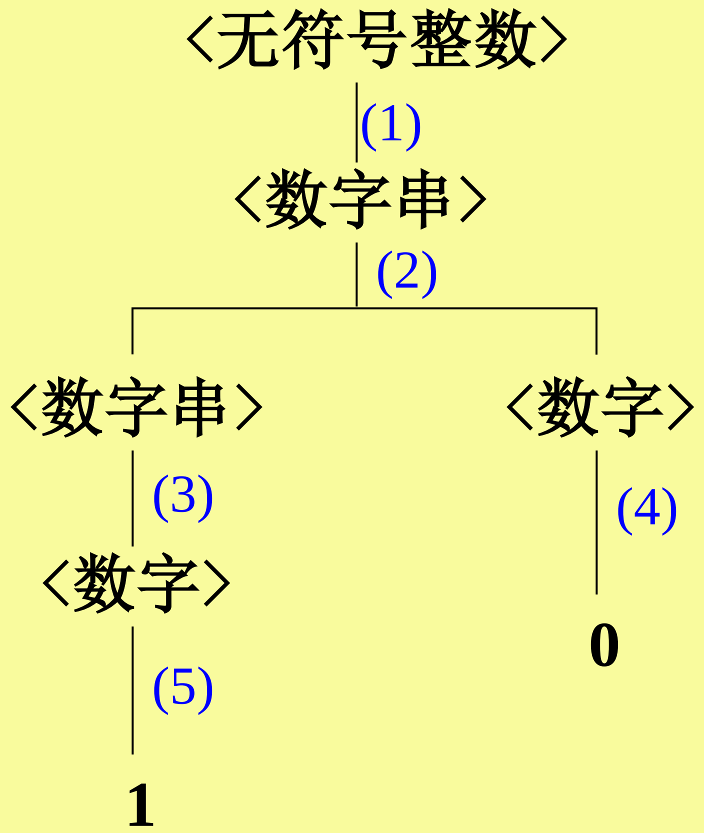
G[<无符号整数>]:

<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

一般推导:



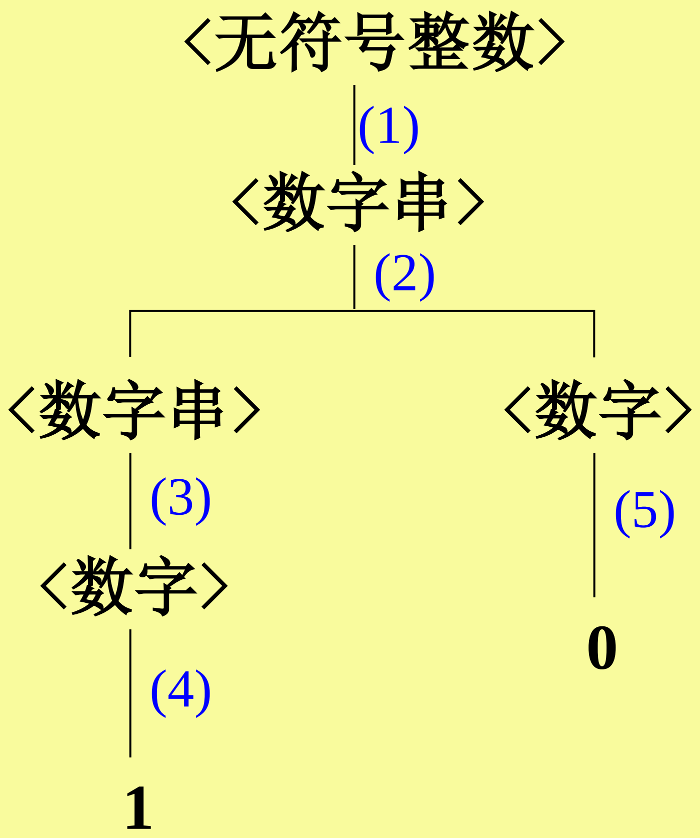
G[<无符号整数>]:

<无符号整数> → <数字串>

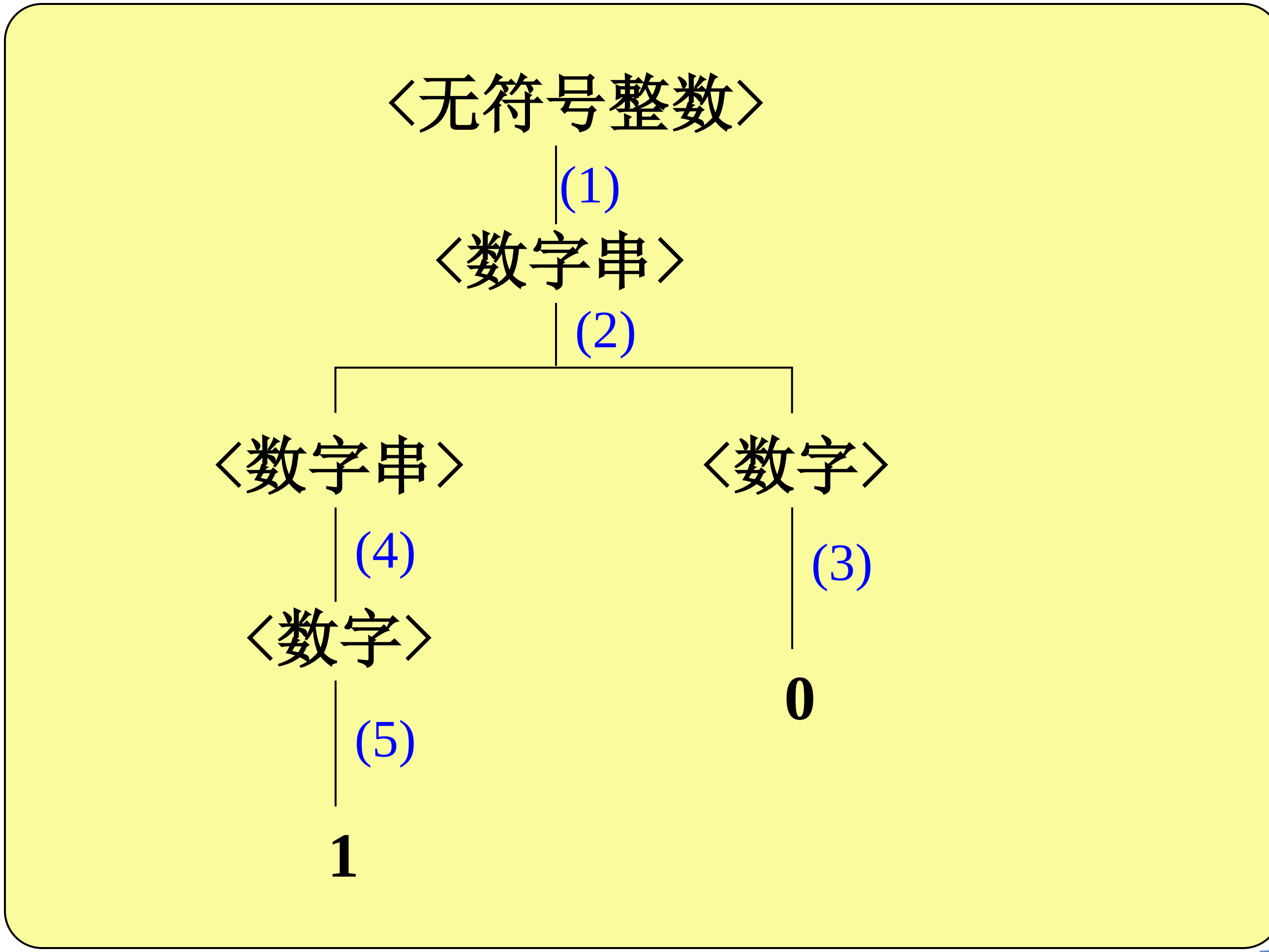
<数字串> → <数字串> <数字> | <数字>

<数字> →0 | 1 | 2 | 3 | …… | 9

最左推导：



最右推导



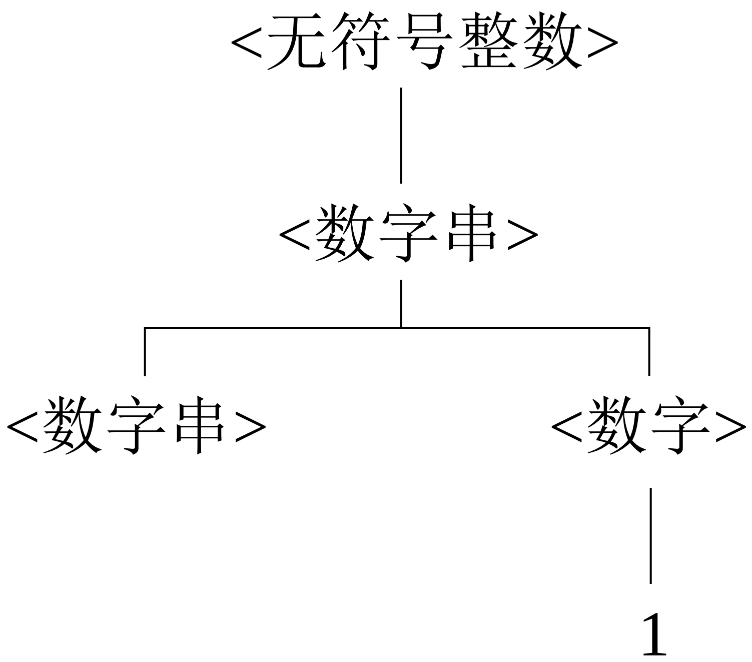
(3) 子树与短语

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。

例：G[<无符号整数>]　　句型　<数字串>1

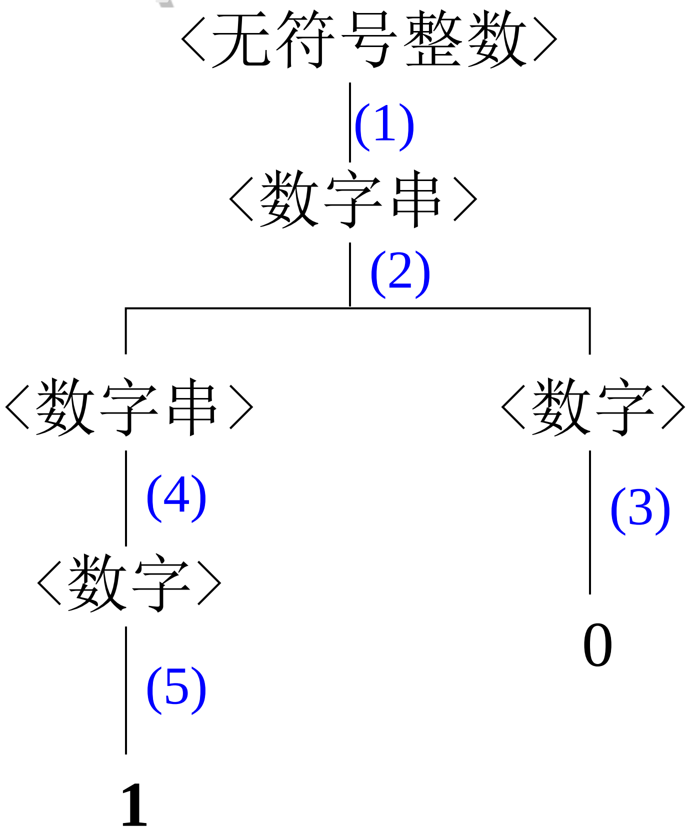


<无符号整数> ⇒ <数字串>

⇒ <数字串> <数字>

⇒ <数字串>1

短语：<数字串>1，1　　简单短语：1　　句柄：1



<无符号整数>=> <数字串>

=> <数字串> <数字>

=> <数字串>0

=> <数字>0

=> 10

句型	<数字串>	<数字串> <数字>	<数字串>0	<数字>0	10
短语	<数字串>	<数字串> <数字>	<数字串>0，0	<数字>0, <数字>,0	10, 1,0
简单短语	<数字串>	<数字串> <数字>	0	<数字>，0	1,0
句柄	<数字串>	<数字串> <数字>	0	<数字>	1

（4）树与推导

句型推导过程 <==> 该句型语法树的生长过程

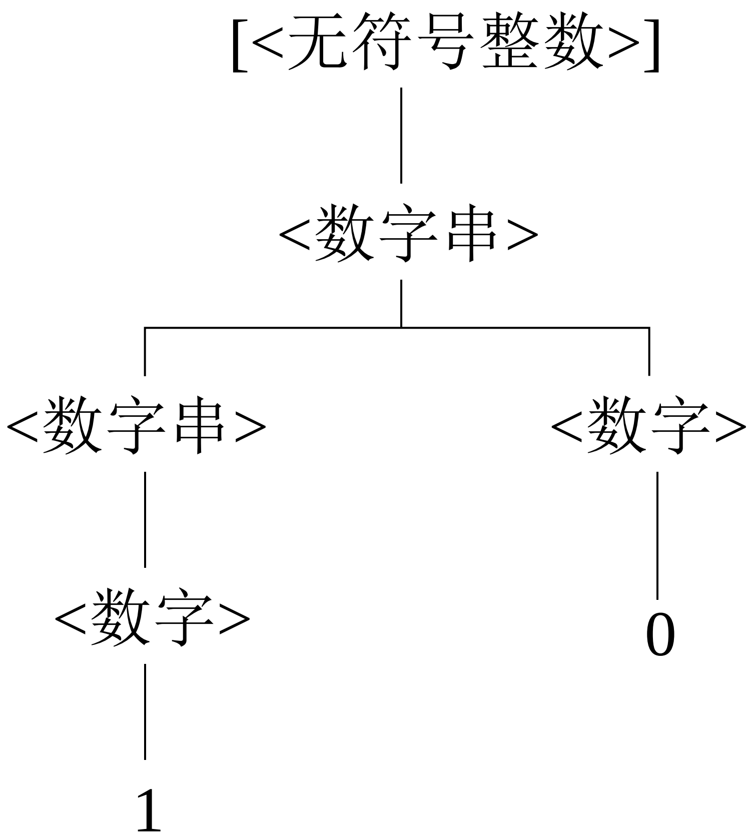
1 由推导构造语法树

从识别符号开始，自左向右建立推导序列。

↓

由根结点开始，自上而下建立语法树。

例：G[<无符号整数>]　　句型10



<无符号整数> =+> <数字串>

=+> <数字串> <数字>

=+> <数字串> 0

=+> <数字> 0

=+> 10

规范推导

2 由语法树构造推导

自下而上地修剪子树的某些末端结点（短语），直至把整棵树剪掉（留根），每剪一次对应一次归约。

↓

从句型开始，自右向左地逐步进行归约，建立推导序列。

通常我们每次都剪掉当前句型的句柄（最左简单短语）

即每次均进行规范归约

规范归约与规范推导互为逆过程

[<无符号整数>]

<无符号整数>

=+> <数字串>

=+> <数字串> <数字>

=+> <数字串> 0

=+> <数字> 0

=+> 10

定义12. 对句型中最左简单短语（句柄）进行的归约称为规范归约。

定义13. 通过规范推导或规范归约所得到的句型称为规范句型。

句型<数字><数字>不是文法的规范句型，因为：

<无符号整数>=|=><数字串>

=|=><数字串><数字>

==><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（或句型）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。

二义性文法举例：

G[E]:　　E ::= E+E | E*E | (E) | i

Vn={E}

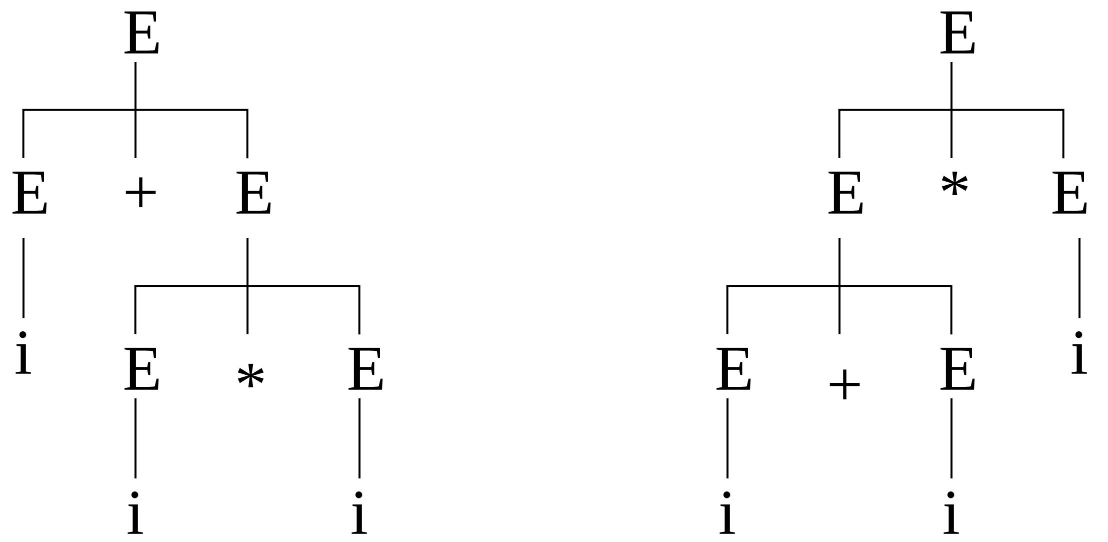
Vt={ +, * , (,) , i }

对于句子S＝i＋i ＊i ∈ L(G[E] ），存在不同的规范推导：

(1) E=|=>E+E=|=>E+E*E =|=>E+E*i =|=>E+i*i =|=> i＋i ＊i

(2) E=|=> E*E =|=> E*i =|=> E+E*i =|=> E+i*i =|=> i＋i * i

这两种不同的推导对应了两棵不同的语法树：

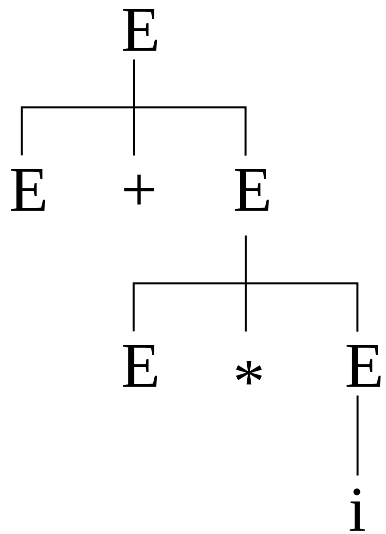


定义14.2 若一个文法的某句子存在两个不同的规范推导，则该文法是二义性的，否则是无二义性的。

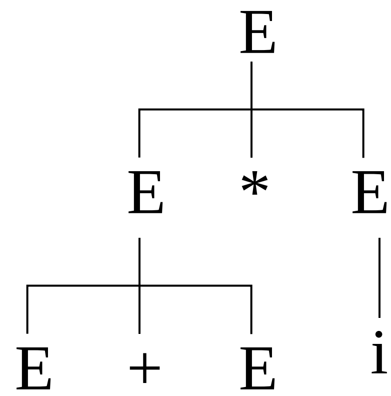
(1) E ⇒ E+E ⇒ E+E*E ⇒ E+E*i ⇒ E+i*i ⇒ i+i * i

(2) E ⇒ E*E ⇒ E*i ⇒ E+E*i ⇒ E+i*i ⇒ i+i * i

从自底向上的归约过程来看，上例中规范句型 **E+E*i** 是由**i+i * i**通过两步规范归约得到的，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，文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。

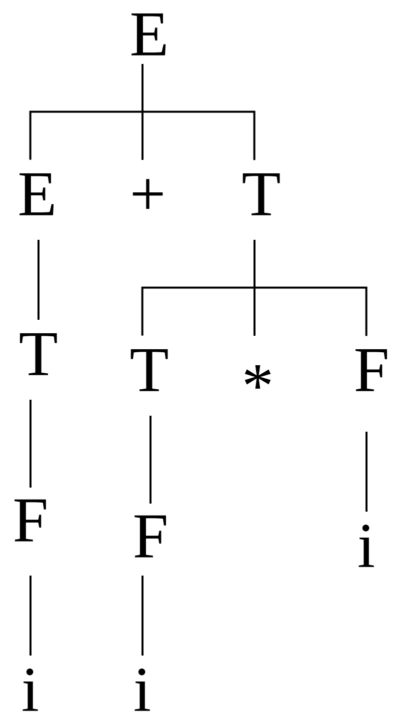
若文法是二义性的，则在编译时就会产生不确定性，遗憾的是在理论上已经证明：文法的二义性是不可判定的，即不可能构造出一个算法，通过有限步骤来判定任一文法是否有二义性。

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。

例:算术表达式的文法

E::= E+E | E*E | (E) | i

E::= E+T | T
T ::= T*F | F
F ::= (E) | i



句子： i+ i * i

E =|=> E+T =|=> E+T*F =|=> E+T*i

=|=> E+F*i =|=> E+i*i =|=> T+i*i

=|=> F+i*i =|=> i+i*i

无二义性的表达式文法：

E::= E+T | T
T ::= T*F | F
F ::= (E) | i

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: Pascal 条件语句的文法

<条件语句>::= If <布尔表达式>then<语句> |

If <布尔表达式> then <语句> else <语句>

<语句> ::= <条件语句> | <非条件语句> |…….

If B then If B then stmt else stmt

习题2-4：
P46-47 1,5,6,8,9

2.5 句子的分析

任务：给定G[Z]：S ∈ V_t*， 判定是否有 S ∈ L (G[Z]) ?

这是词法分析和语法分析所要做的工作，将在第三、四章中详细介绍。

2.6 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 | b,则有两棵语法树：

```
U          U
|          |
a          U
           |
           a
```

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（不可达符号）

（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（不活动符号）

例如给定G[Z]，若其中关于U的规则只有如下一条:

U::=xUy

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： G[<Z>] :

<Z> ::= <B> e

<A> ::= <A> e | e

<B> ::= <C> e | <A> f

<C> ::= <C> f

不活动

<D> ::= f

不可达

G'[<Z>] :

<Z> ::= <B> e

<A> ::= <A> e | e

<B> ::= <A> f

例2： G[S] :

S ::= ccc

S ::= Abccc

A ::= Ab

A ::= aBa

B ::= aBa

B ::= AD

D ::= Db

D ::= b

不活动

S ::=ccc

D ::= Db

D ::= b

不可达

G'[S] :

S ::= ccc

习题2-5： p53 3

2.7 文法的其它表示法

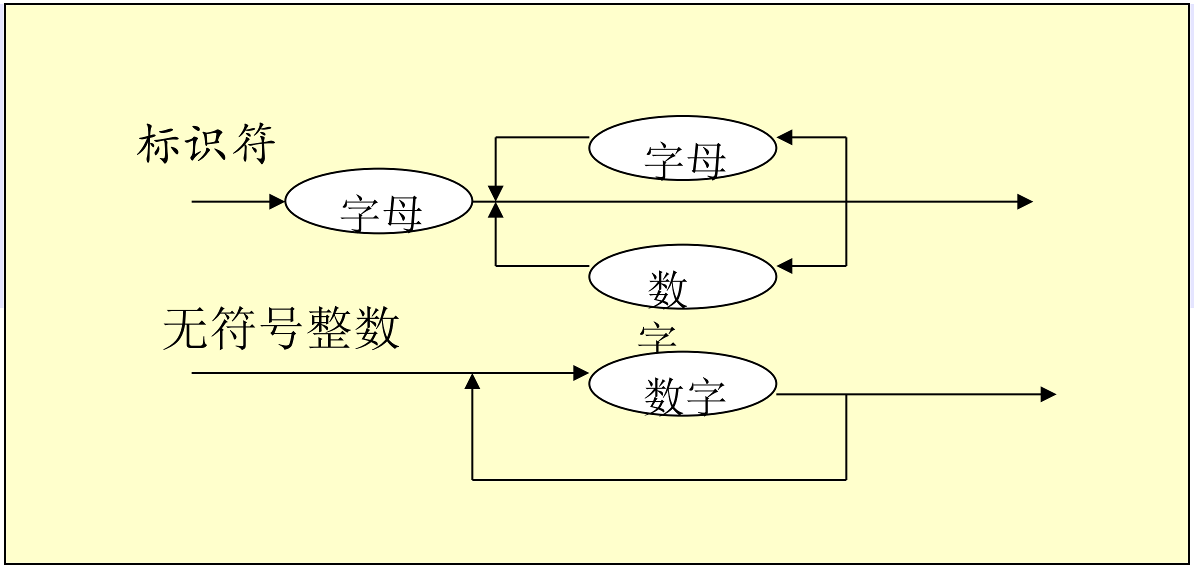
<标识符>::＝字母{字母|数字}

<无符号整数>::=数字{数字}

1、扩充的BNF表示

- **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。

文法定义：乔姆斯基将所有文法都定义为一个四元组：

G=（Vn，Vt，P，Z）

Vn：非终结符号集

Vt：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∈Vn

语言定义：　L（G[Z]）$=\{x \mid x \in V_t^*,\ Z \overset{+}{\Longrightarrow} x\}$

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（语法规则）施加不同的限制。

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

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∈Vn，

x、y、u∈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**

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∈Vn，

u∈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。(1型文法的规则中x，y均为 ε 时即为2型文法)

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：　$U ::= t$

或　$U ::= Wt$

其中 U、$W \in V_n$

$t \in V_t$

（右线性）

P：　$U ::= t$

或　$U ::= tW$

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：L3　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，L0 ⊃ L1 ⊃ L2 ⊃ L3
- 0型文法可以产生L0、L1、L2、L3，
- 但2型文法只能产生L2，L3不能产生L0，L1
- 3型文法只能产生L3

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：推导、规约、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。

消除不活动符号和不可达符号算法

10 Papers Every Programmer Should Read

1. On the criteria to be used in decomposing systems into modules – David Parnas
2. A Note On Distributed Computing – Jim Waldo, Geoff Wyant, Ann Wollrath, Sam Kendall
3. The Next 700 Programming Languages – P. J. Landin
4. Can Programming Be Liberated from the von Neumann Style? – John Backus
5. Reflections on Trusting Trust – Ken Thompson
6. Lisp: Good News, Bad News, How to Win Big – Richard Gabriel
7. An experimental evaluation of the assumption of independence in multiversion programming – John Knight and Nancy Leveson
8. Arguments and Results – James Noble
9. A Laboratory For Teaching Object-Oriented Thinking – Kent Beck, Ward Cunningham
10. Programming as an Experience: the inspiration for Self – David Ungar, Randall B. Smith