第2章　形式语言的基本知识

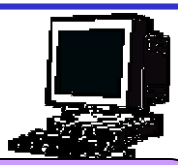
教学目标

1. 本章是编译原理课程的理论基础，要求掌握形式语言的基本术语和概念。
2. 掌握文法和语言的定义，文法的二义性与递归性的判断方法及句型的分析方法。
3. 熟练使用文法定义程序设计语言的单词和语法成分。
4. 对形式语言的理论有一个初步认识。

教学内容

- 2.1 字母表和符号串的基本概念
- 2.2 文法和语言的形式定义
- 2.3 句型的分析
- 2.4 文法和语言的分类
- 2.5 PL/0编译程序概述

2.1 字母表和符号串



符号就是字符，对吗？

e,for,while}

字母表：符号的非空有

C语言的字母表

A＝{a,b,…,0,1,…,9, +,－,×,_/, (,), =… if, else,for...}

符号：字母表中的元素　　例：　0，1

符号串：由字母表中的符号组成的任何有穷序列

例：0, 1，01，10，011，..

空符号串：无任何符号的符号串，用 ε 表示

符号串的前缀和后缀：

从符号串s的尾部删去若干个(包括0个)符号之后所余下的部分称为s的前缀

ε，0，01及011都是符号串011的前缀

ε，1，11及011都是符号串011的后缀

符号串集合：若集合A中的一切元素都是某字母表上的符号串，则称A为该字母表上的符号串集合。

例：∑={a，b，c}　A={ a,aa,ac}

符号串的运算

符号串的连接运算

符号串x和y的**连接**：是把y的符号写在x的符号之后得到的符号串xy

例如x=00，y=11，则xy=0011

对于任意一个符号串s，有εs= sε=s

符号串的幂运算

符号串自身连接n次得到的符号串s^n 定义为ss...ss ，包括n个s，称为符号串的**幂运算**

$s^0=\varepsilon$，$s^1=s$，$s^2=ss$，……

设s=01，则

$s^0=\varepsilon$

$s^1=01$

$s^2=0101$

……

符号串集合的乘积

设A、B为符号串集合，则A和B的乘积定义为：

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

例如，A＝{a,b}，B={c,d}

则AB= {ac, ad, bc, bd}

符号串集合的幂运算

有符号串集合A，定义$A^0=\{\varepsilon\}$，$A^1=A$，$A^2=AA$，$A^3=AAA$,……　$A^n=A^{n-1}A=AA^{n-1}$，$n>0$

例如，$A=\{0,1\}$，则

$A^0=\{\varepsilon\}$

$A^1=\{0,1\}$

$A^2=\{00,01,10,11\}$

$A^3=\{000,001,010,011,100,101,110,111\}$

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}

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

(A^1: x,y; A^2: xx,xy,yx,yy; A^3: xxx,xxy,xyx,xyy)

$A^* =$ {ε, x,y, xx,xy,yx,yy , xxx,xxy,xyx,xyy, ……}

(A^0: ε; A^1: x,y; A^2: xx,xy,yx,yy; A^3: xxx,xxy,xyx,xyy)

Σ的闭包Σ*

表示Σ上的一切符号串（包括ε）组成的集合

Σ的正闭包Σ^+

表示Σ上的除ε外的所有符号串组成的集合

$$\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup .....$$

$$\Sigma^+ = \Sigma^* - \{\varepsilon\} = \Sigma\Sigma^* = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup .....$$

例：Σ={a,b}

Σ^*={ε,a,b,aa,ab,ba,bb,aaa,aab,…}

Σ^+={a,b,aa,ab,ba,bb,aaa,aab,…}

★ 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为某语言的字母表

$$A = \{a,b,\ldots,0,1,\ldots,9, +, -, \times, \_/, (\ ,\ ), =\ldots \text{ if, else,for...}\}$$

B为单词集

$$B = \{\text{if, else,for,}\ldots\ldots,\langle\text{标识符}\rangle,\langle\text{常量}\rangle,\ldots\ldots\}$$

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$\subset C$

语言是由句子组成的集合，是由一组符号所构成的集合

- 字母表Σ上的一个语言是Σ上的一些符号串的集合
- 字母表Σ上的每个语言是Σ*的一个子集

例如：字母表

$\Sigma=\{a,b\}$,$\Sigma^*=\{\varepsilon,a,b,aa,ab,ba,bb,aaa,aab,\ldots\}$

集合$\{ab,aabb,aaabbb,\ldots,a^nb^n,\ldots\}$

或表示为$\{w|w\in\Sigma^*$且$w=a^nb^n,n\geq1\}$为字母表Σ上的一个语言

集合$\{a,aa,aaa,\ldots\}$或表示为$\{w|w\in\Sigma^*$且$w=a^n,n\geq1\}$ 为字母表Σ上的一个语言

2.2 文法和语言的形式定义

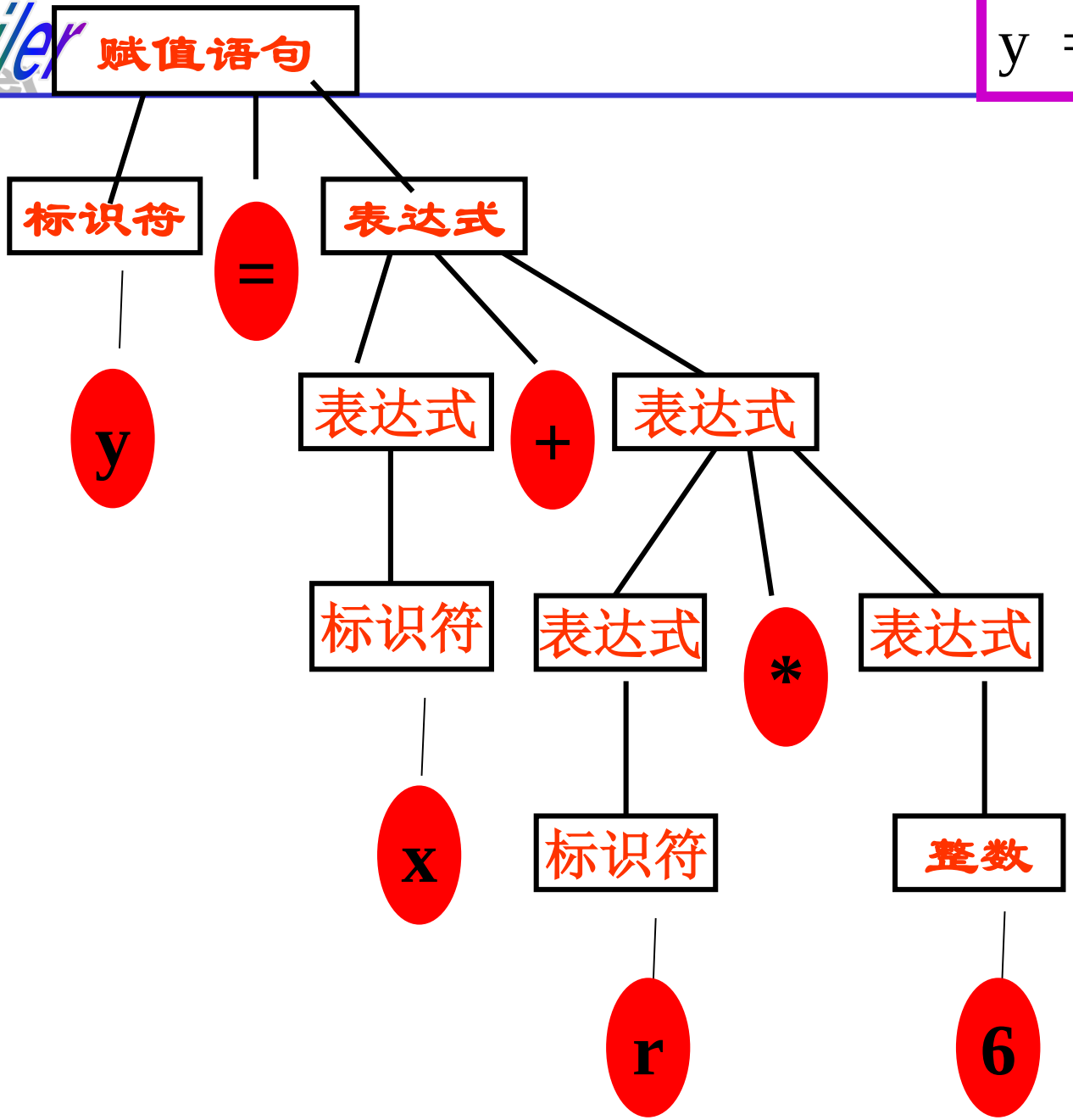
文法是对语言结构的定义与描述（或称为“语法”）

〈赋值语句〉::=〈标识符〉“=”〈表达式〉

〈表达式〉::=〈表达式〉“+”〈表达式〉 | 〈表达式〉“*”〈表达式〉

〈表达式〉::=“(”〈表达式〉“)” | 〈标识符〉 | 〈整数〉 | 〈实数〉

y = x + r * 6



文法的非形式定义

例：有一句子：“我是大学生”。这是一个在语法、语义上都正确定句子，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：只需逐一列举句子
- 无穷语言：使用文法定义句子的结构，用适当条数的规则把语言的全部句子描述出来。文法是以有穷的集合刻划无穷的集合的工具。

1. 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”。

文法的BNF表示

〈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〉

2. **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〉
〈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**（一般推导）

(2)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。

文法的形式定义

$V = V_N \cup V_T$
称为文法的字母表

文法G[S]=（V_N，V_T，P，S）

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

S：开始符号（识别符号）$S \in V_N$，S至少要在一条规则中作为左部出现

补：规则的定义

$\alpha ::= \beta$ 或 $\alpha \rightarrow \beta$

$\alpha \in V^+$且至少有一个非终结符，而$\beta \in (V_N \cup V_T)^*$

【例2.1】程序语言中只含+、*运算的算术表达式，用i表示变量或常数，其文法可以表示为：

描述了表达式的语法规则

$G=(V_T, V_N, S, P)$，其中：

V_N={表达式}

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

S=〈表达式〉

P={〈表达式〉→〈表达式〉+〈表达式〉

〈表达式〉→〈表达式〉*〈表达式〉

〈表达式〉→(〈表达式〉)

〈表达式〉→i

}

★ 几点说明：

产生式左边符号构成集合V_N，且 $S \in V_N$

V_N：代表程序的语法成份，如“<表达式>”，它不会出现在程序中

V_T：会出现在程序中，如i+i

给定一个文法，实际只需给出产生式集合，并指定识别符号

G[表达式]：

〈表达式〉→〈表达式〉+〈表达式〉

〈表达式〉→〈表达式〉*〈表达式〉

〈表达式〉→(〈表达式〉)

〈表达式〉→i

有些产生式具有相同的左部，可以和在一起

G[E]:

E→E+E|E*E|(E)|i

【例2.2】某种语言的标识符是以字母开头的字母数字串，如果用L表示〈字母〉，D表示〈数字〉，用S表示字母数字串

描述了标识符的词法规则

G[S]:
S→L|SL|SD
L→a|b|…|z
D→0|1|…|9

例： 无符号整数的文法：

描述了无符号整数的词法规则

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

★ 说明：

描述词法规则的文法

G[S]:
S→L|SL|SD
L→a|b|…|z
D→0|1|…|9

终结符集是输入字符集，它是构成单词的最基本元素

描述语法规则的文法

G[E]:

E→E+E|E*E|(E)|i

终结符集是经词法分析识别后的单词集，如变量i，运算符+、*和分界符(、)，它们被视为语法分析中最基本元素

文法的表示方法

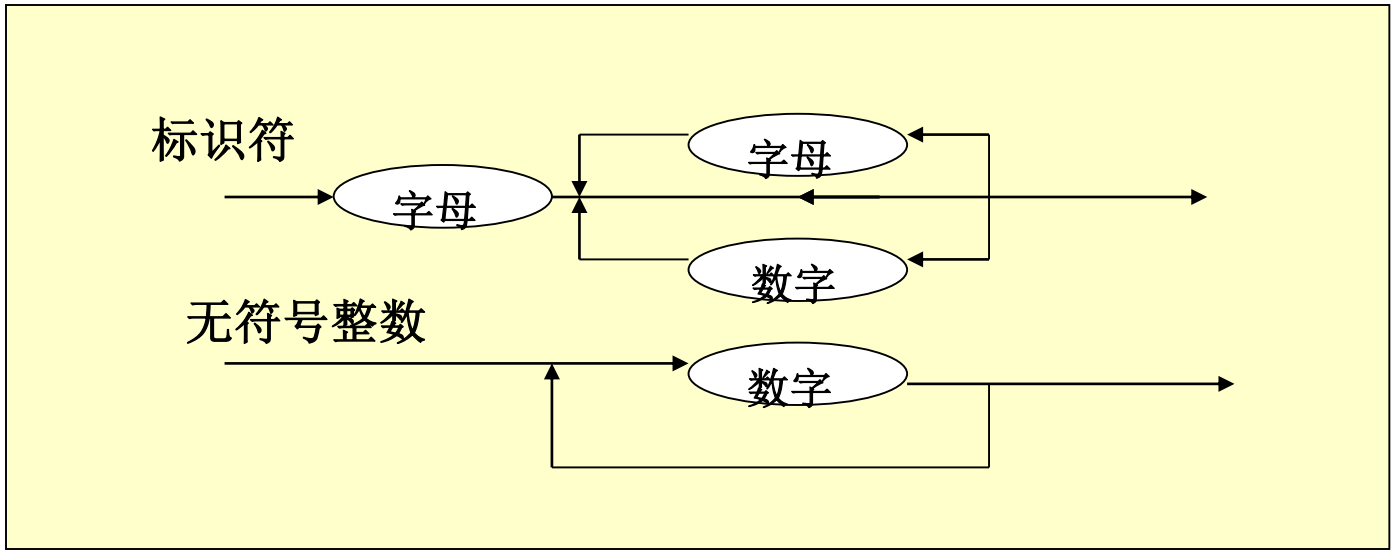
巴克斯瑙尔范式

1. BNF的元符号： <, >, ::=, |

分组

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

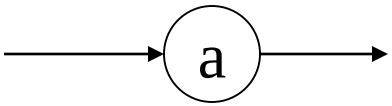
3.语法图



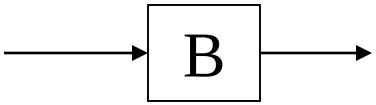
语法图

构造：规则右部的符号

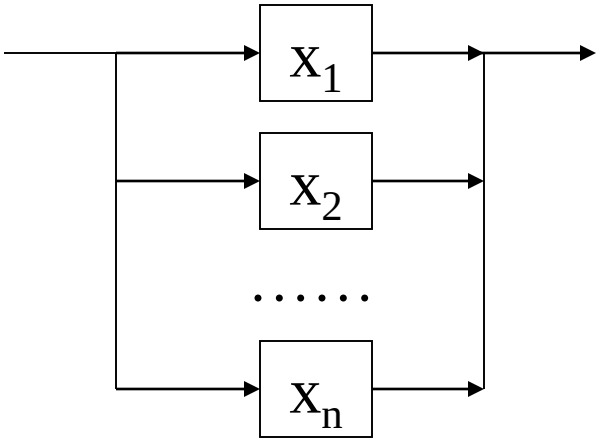
(1) a ∈Vt



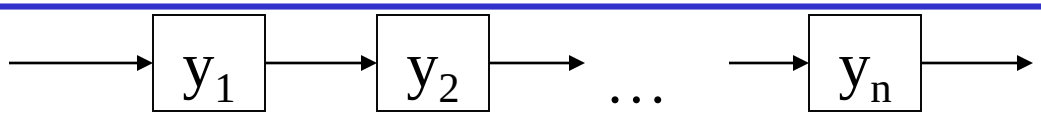
(2) B ∈Vn



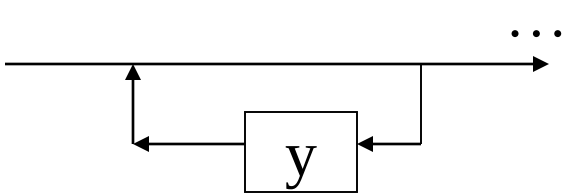
(3) 形如$U \rightarrow x_1 | x_2 | \cdots\cdots | x_n$



$X=y_1y_2\cdots\cdots y_n$



$X=\{y\}$

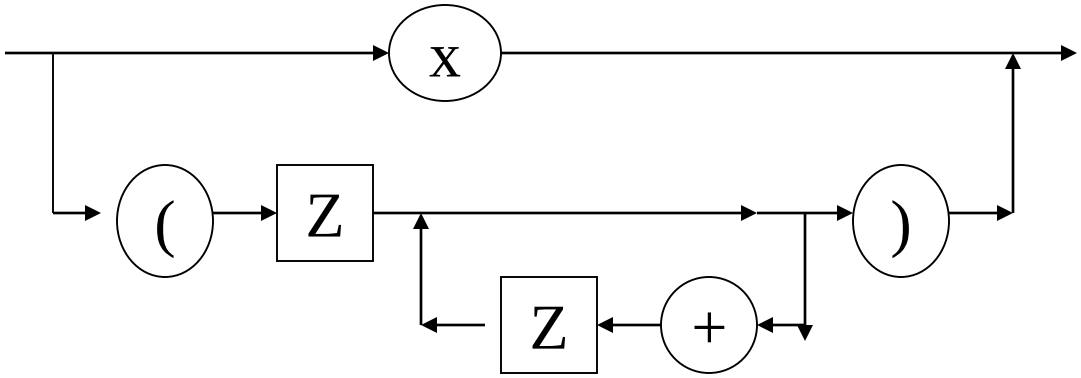


例： G[Z]：

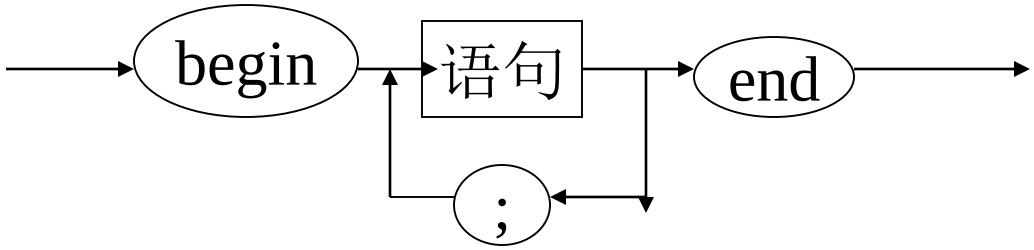
Z →x |(B)

B →ZC

C →{+Z}



例Pascal语言中的〈分程序〉



推导的形式定义

如果 $\alpha \rightarrow \beta$ 是文法$G=(V_T, V_N, S, P)$的一个产生式，γ，$\delta \in V^*$，有符号串x，y满足

$x=\gamma\ \alpha\ \delta$，$y=\gamma\ \beta\ \delta$，

则称y是x的**直接推导**，

也可以说，y**直接归约**到x，记作$x \Rightarrow y$。

直接推导：用产生式的右部替换产生式的左部

直接归约：用产生式的左部替换产生式的右部的过程

根据文法和推导的定义，可推出终结符号串

G[S]：
S→L|SL|SD
L→a|b|…|z
D→0|1|…|9

S=>SD，使用规则S→SD，其中 $\gamma=\delta=\varepsilon$；
SD=>LD，使用规则S→L，其中 $\gamma=\varepsilon$，δ=D；
LD=>aD，使用规则L→a，其中 $\gamma=\varepsilon$，δ=D；
aD=>a1，使用规则D→1，其中 γ=a，$\delta=\varepsilon$。

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 | 1 | 2 | 3 | …… | 9

<无符号整数> ==(1)==> <数字串> ==(2)==> <数字串><数字>

==(3)==> <数字><数字> ==(5)==> 1 <数字>

==(6)==> 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。

如果存在直接推导序列：

$$x \Rightarrow y0 \Rightarrow y1 \Rightarrow y2 \ldots yn=y$$

则称这个序列是一个从x至y的长度为n（n>0）的**推导**，或称y**归约**到x，记作$x \overset{+}{\Rightarrow} y$。

若有$x \overset{+}{\Rightarrow} y$，或x=y，则记作$x \overset{*}{\Rightarrow} y$。

例：$x \Rightarrow S \Rightarrow SD \Rightarrow LD \Rightarrow aD \Rightarrow a1=y$

记作$x \overset{+}{\Rightarrow} y$或记作$x \overset{*}{\Rightarrow} y$

G[S]:

S→L|SL|SD

L→a|b|…|z

D→0|1|…|9

语言的形式定义

句子是所有终结符号组成的句型

文法G[S]所产生的所有句子的集合

文法G[S]

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

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

(3) 语言：$L(G[S]) = \{\alpha \mid \alpha \in V_T^*, S \overset{+}{\Rightarrow} \alpha\}$;

G[E]:

E→E+E|E*E|(E)|i

句型：
E+E
E+E*E
E+E*i
E+i*i

句子：
i+i
i*i
i+i*i
(i+i)*i

形式语言理论可以证明以下两点：

（1）G →L（G）；

（2）L(G)→G1，G2，……，Gn；　多种文法

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

已知文法，求语言，通过推导

例： G[S]

S ::= aSb |ab

求其所产生的语言。

$L(G[S])=\{a^nb^n|n\geq 1\}$

若**S ::= aSb |** ε ，如何？

$L(G[S])=\{a^nb^n|n\geq 0\}$

课堂练习1： G[S]

S ::= bA

A ::= aA|a

$L(G[S])=\{ba^n|n\geq 1\}$

课堂练习2： G[S]

S ::= AB

A ::= aA|a

B ::= bB|b

$L(G[S])=\{a^mb^n|m,n\geq 1\}$

若n≥0，如何？

例：$\{ab^na|n\geq 1\}$,构造其文法

$G_1[S]$：S→aBa,
B→b|bB

$G_2[S]$：S→aBa,
B→b|Bb

G1[S]：S→aBa,
B→ ε |bB

G2[S]：S→aBa,
B→ ε |Bb

课堂练习3：$\{a^nb^nc^i|n\geq 1, i\geq 0\}$,
构造其文法

G[S]：S →AB
A →aAb|ab
B→cB| ε

定义. 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。

例：构造一个文法，使其描述的语言是无符号整数。

G[S]：

S→SD|D

D→0|1|2|3|4|5|6|7|8|9

G[〈无符号整数〉]

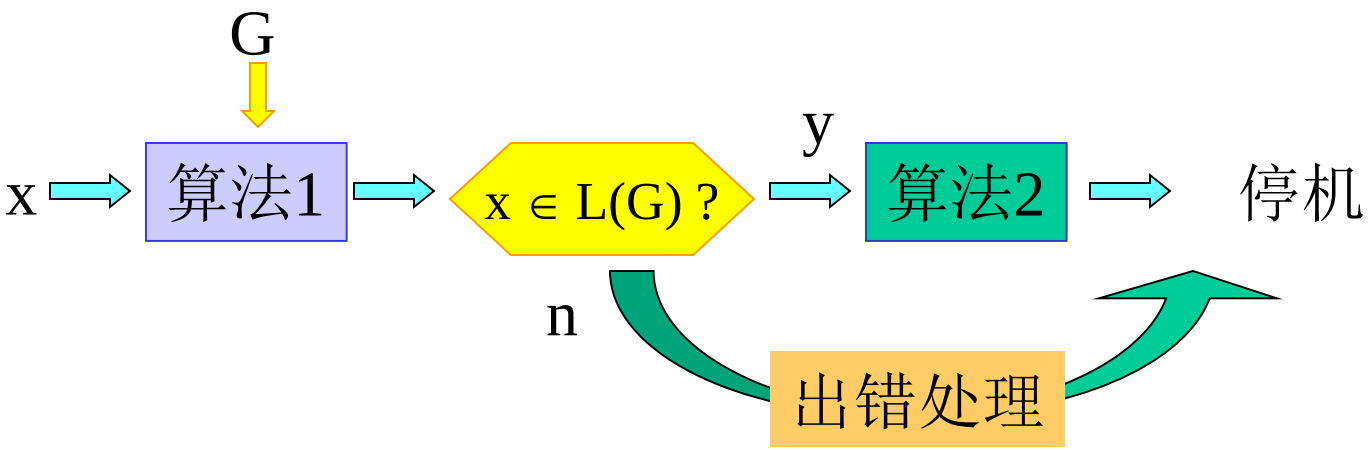
〈无符号整数〉 → 〈数字串〉

〈数字串〉 → 〈数字串〉〈数字〉 | 〈数字〉

〈数字〉 →0 | 1 | 2 | 3 | …… | 9

编译感兴趣的问题是：

- 给定x，G，求x ∈ L(G) ？



〈赋值语句〉::=〈标识符〉=〈表达式〉

递归文法

1. **递归规则**：规则右部有与左部相同的符号

左递归规则：A→A...

右递归规则：A→...A

自嵌入递归规则：A→...A...

2. **递归文法**：含有递归规则的文法，为**直接递归文法**

A→Ba
B→Ab

↓

间接递归文法

↓

G[S]:
S→L|SL|SD
L→a|b|...|z
D→0|1|...|9

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12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？

G[S]:
S→SD|D
D→0|1|2|3|4|5|6|7|8|9

G[E]：E→i+i|i*i|(i+i)*i

该文法所描述的语言为L(G)={i+i,i*i,(i+i)*i}，无法表示无穷的表达式语言

会造成死循环（后面将详细论述）

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

2.3 句型的分析

句型的分析： 是指识别输入的符号串是否为某一文法的句型(或句子)的过程

对于同一个句型或句子，可以通过不同的推导序列推导出来，这是因为在推导过程中所选择的非终结符的次序不同

G[E]：E→E+E|E*E|(E)|I

i+i*i的推导序列有哪些？

规范推导与规范归约

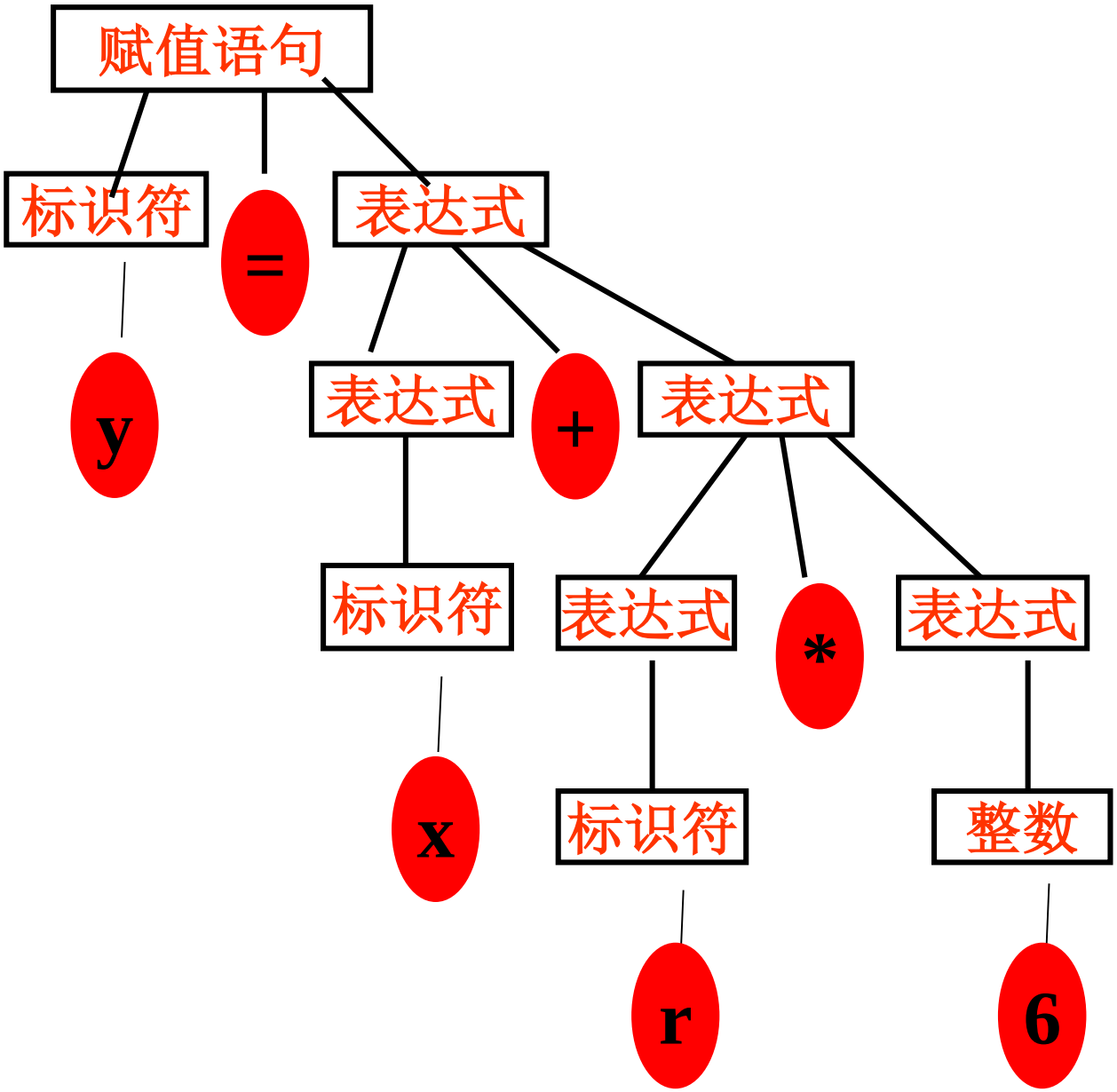
最左(右)推导指对于一个推导序列中的每一直接推导，被替换的总是当前符号串中的最左(右)非终结符号。最右推导也称为**规范推导**。

规范推导的逆过程，称为最左归约，也称为**规范归约**。

用最左推导所推导出的句型称为**最左句型**

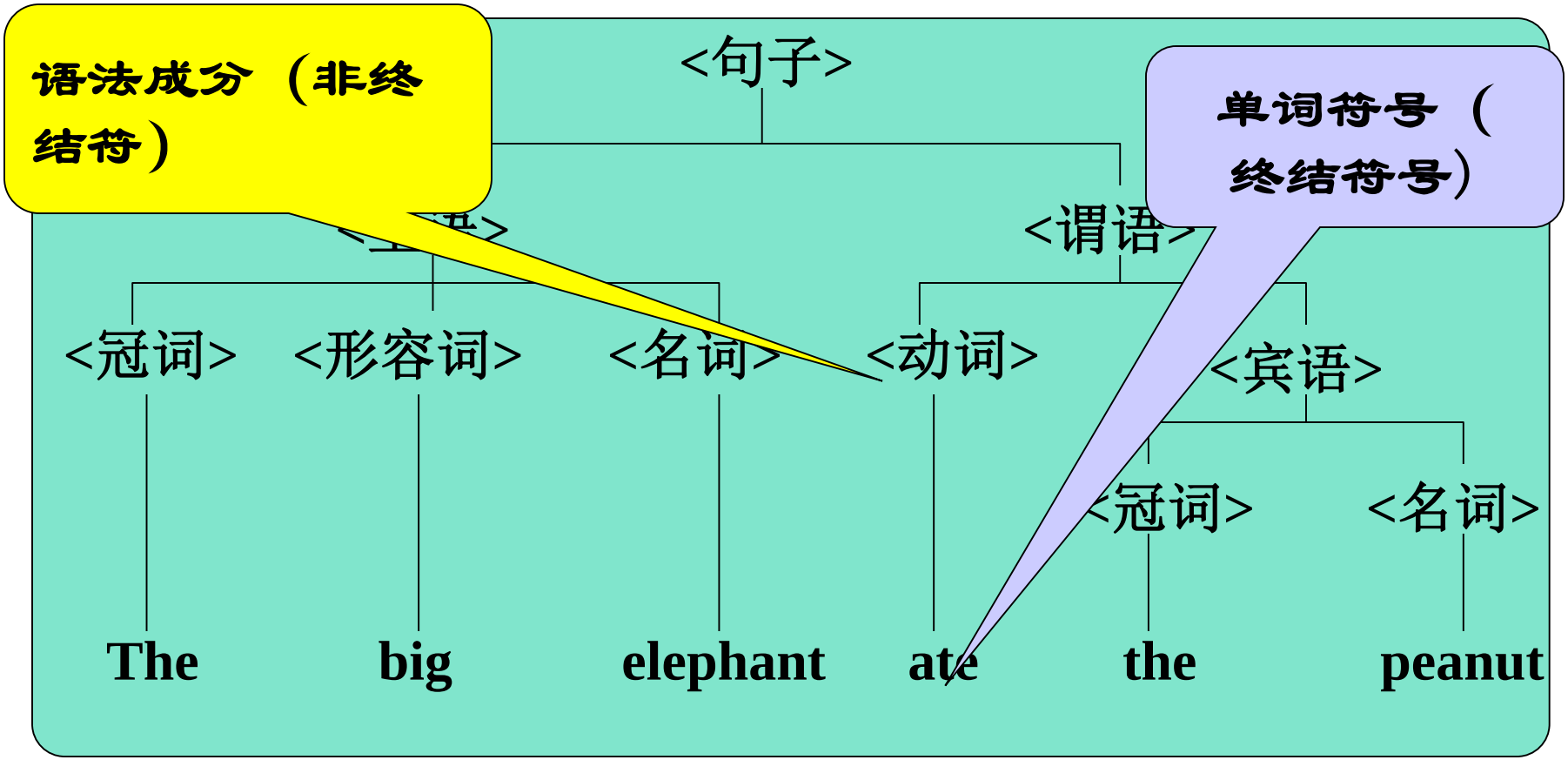
用最右推导所推导出的句型称为**最右句型**，通常称为**规范句型**

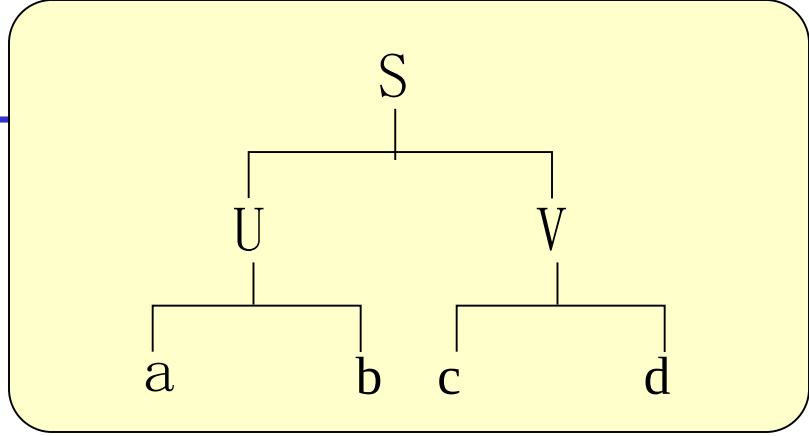
语法树与文法的二义性



语法分析的结果----语法树

1. 语法树：描述一个句子的语法结构





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

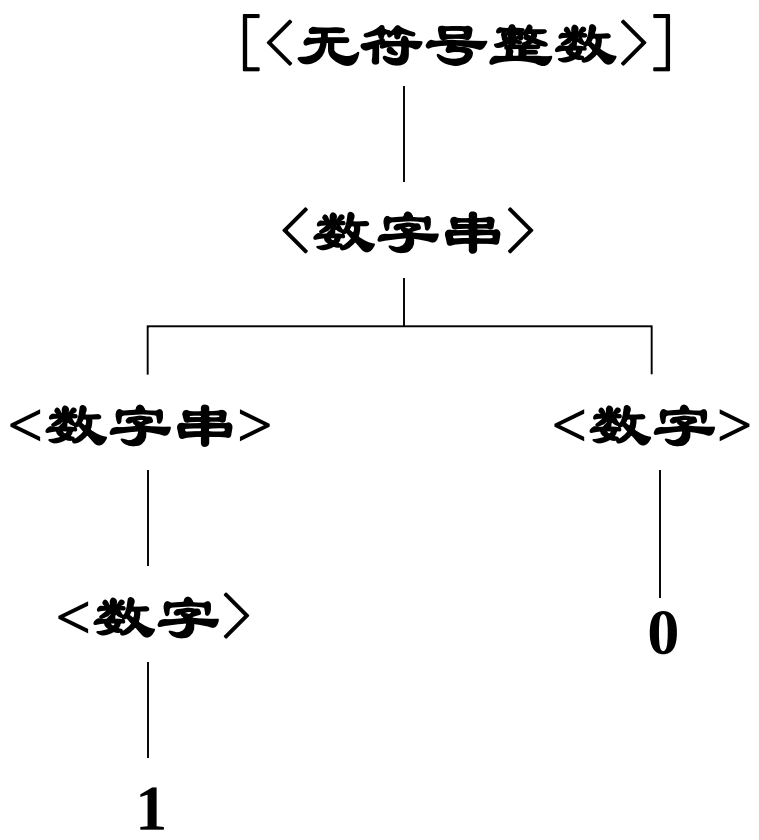
句型推导过程<==>句型语法树的生长过程

1 由推导构造语法树

从识别符号开始，逐步建立推导序列。

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



规范推导

〈无符号整数〉 ==> 〈数字串〉

==> 〈数字串〉〈数字〉

==> 〈数字串〉0

==> 〈数字〉0

==> 10

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，自右向左地逐步进行规约，建立推导序列。

规范规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

2. 文法的二义性

定义：若一个文法的某句子存在两个不同的最右（最左）推导，则该文法是二义性的，否则是无二义性的。

G[E]：

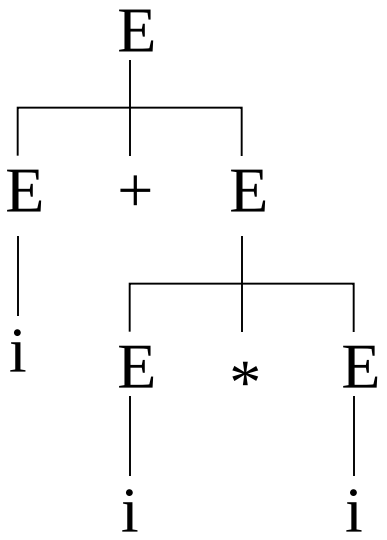
E→E+E|E*E|(E)|i

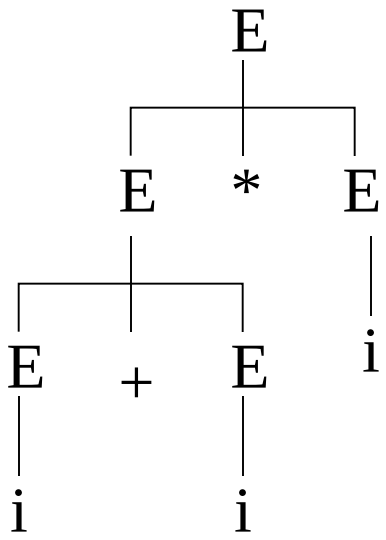
课堂练习：
对于句子i+i*i，给出两种不同的规范推导，并画出语法树

(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种不同的语法树





若文法是二义性的，则在编译时就会产生不确定性,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。

可以采用两种途径来解决文法的二义性问题

1. 不改变文法中原有的规则，仅加进一些语法的非形式规定。

   规定“*”运算优先级高于“+”运算，且服从左结合

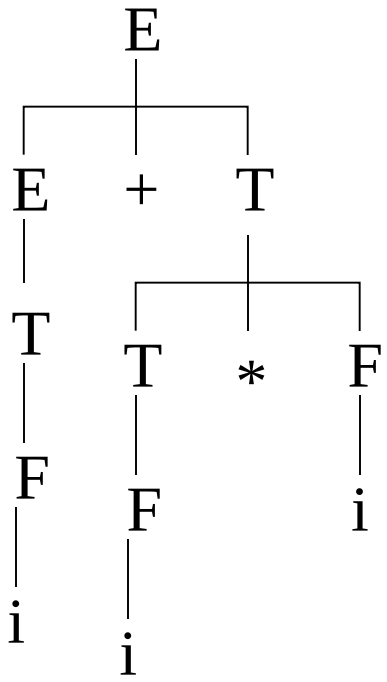
2. 改写文法，把排除二义性的规则合并到原有文法中

例:算术表达式的文法

G[E]:

E→E+E|E*E|(E)|i

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



§2.3 句型的分析

§2.3.1 规范推导和规范归约

1、**最左(右)推导**:在任一步推导V=>w中，都是对符号串V的最左(右)非终结符号进行替换，称最左（右）推导。

2、**规范推导**:即最右推导

3、**规范句型**：由规范推导所得的句型.

4、**规范归约**：规范推导的逆过程，称规范归约或最左归约。

例:G[〈标识符〉]

〈标识符〉→〈字母〉|〈标识符〉〈字母〉

|〈标识符〉〈数字〉

〈字母〉→a|b|…|z|A|…|Z

〈数字〉→0|1|2|…|9

问题:给出句子a4y的规范推导和规范归约.

给出句子a4y的最左推导.

请注意：规范推导和归范归约互为逆过程.

推导——自上而下的分析

规范推导

〈标识符〉

⇒〈标识符〉〈字母〉

⇒〈标识符〉y

⇒〈标识符〉〈数字〉y

⇒〈标识符〉4y

⇒〈字母〉4y

⇒a4y

问题：如何正确选择规则？

规范归约

a4y

〈≠〈字母〉4y

〈≠〈标识符〉4y

〈≠〈标识符〉〈数字〉y

〈≠〈标识符〉y

〈≠〈标识符〉〈字母〉

〈≠〈标识符〉

问题：如何准确选择可归约串？

归约——自下而上的分析

§2.3.2 短语、简单短语和句柄

——————语言句型中的几个概念

1、**短语**：文法G[Z], ω =xuy是一句型,x,y∈V^*

如有 $Z=^*>xUy$,且$U=^+>u$, $U\in V_n$, $u\in V^+$

称u是一个相对于非终结符号U句型ω的**短语**.

2、**简单短语**：文法G[Z], ω =xuy是一句型,

如有 $Z=^*>xUy$,且$U=>u$, $U\in V_n$, $u\in V^+$

称u是一个相对于非终结符号U句型ω的**简单短语**.

或：u是一个相对于非终结符号U句型ω**相对于产生式U->u的简单短语**

3、**句柄**：句型最左边的简单短语为该句型的**句柄**.

短语：文法G[Z], ω =xuy是一句型，

如有 $Z=^{*}>xUy$,且$U=^{+}>u$, $U\in V_n$, $u\in V^{+}$

称u是一个相对于非终结符号U句型ω的短语.

ω =xuy是一句型

结论：u是一个相对于非终结符号U句型ω的短语

u? U?

$U\in V_n$, $u\in V^{+}$,$x,y\in V^{*}$

$Z=^{*}>xUy$ $U=^{+}>u$

$Z=^{*}>xUy=^{+}>xuy$

简单短语：文法G[Z]， ω =xuy是一句型，

如有 Z=*>xUy，且U=>u， U∈Vn， u ∈V+

称u是一个相对于非终结符号U句型ω的简单短语

ω =xuy是一句型

结论：u是一个相对于非终结符号U句型ω的简单短语

u? U?

U∈Vn, u ∈V+ ，x,y ∈V*

Z=*>xUy U=>u

Z=*>xUy=> xuy

u是某产生式的右部

句柄：句型最左边的简单短语为该句型的句柄.

说明：

- 短语或简单短语必须是针对某一句型来说的，并且是该句型的一个子串.
- 短语或简单短语必须是相对某一非终结符号的.
- 两个条件缺一不可.
- 一句型可以有几个短语和简单短语.
- 一句型只有一个句柄(无二义性的文法)。
- 最左归约归约的是当前句型的句柄

例G[S]：

S→AB

A→Aa|bB

B→a|Sb

问题：给出句型baSb的短语、简单短语和句柄.

Sb是相对于B句型baSb的短语且为简单短语

a是相对于B句型baSb的短语且为简单短语.

ba是相对A句型baSb的短语.　　句柄为a.

(1) S=>AB =>bBB =>baB =>baSb　　且B=>Sb

U　　u　　u

(2) S=>AB =>ASb =>bBSb =>baSb　　且B=>a

(3) S=>AB =>ASb $=^{+}>$baSb　　且A$=^{+}>$ba

短语：文法G[Z], ω=xuy是一句型,

如有 Z=*>xUy,且U=+>u, U∈Vn, u ∈V+

称u是一个相对于非终结符号U句型ω的短语.

§2.3.3 语法树

1、语法树：一个句型或句子推导过程的图示法表示，形成一棵语法树.

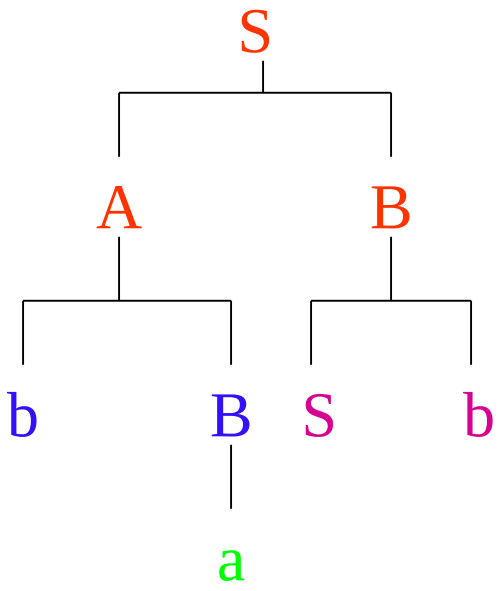
例G[S]：
S→AB
A→Aa|bB
B→a|Sb

句型baSb

最左推导

S=>AB
=>bBB
=>baB
=>baSb

语法树

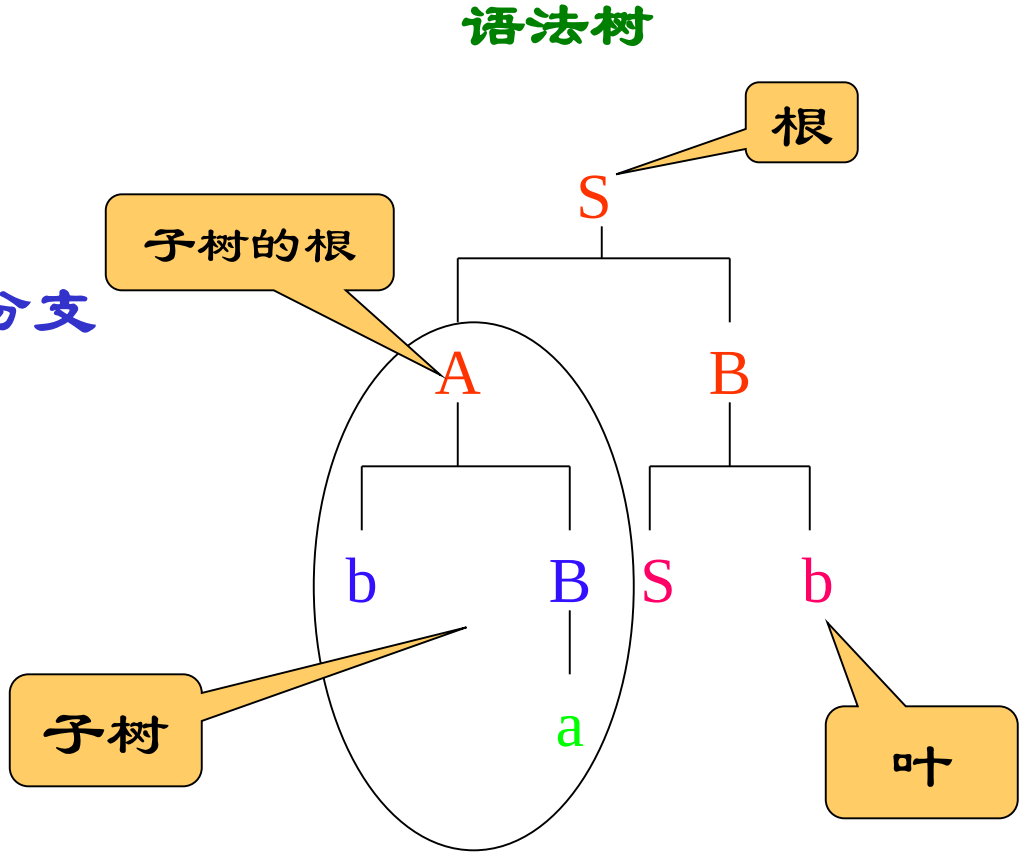


2、语法树与子树

根：开始符号

子树：某一非终结符号（子树的根）及其下面的分支

叶：树的末端结点



语法树的全部末端结点（自左向右）形成当前句型

§2.3.4 子树与短语、句柄

---通过树来寻找短语、简单短语、句柄

1、**短语**:子树的末端结点形成的符号串.

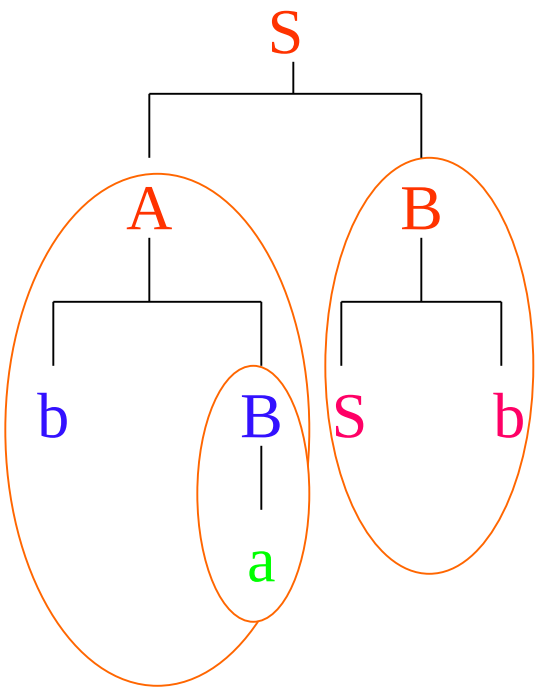
这个短语相对的句型:整个树的末端结点.

非终结符号:子树的根

2、**简单子树**:只有一层分支的子树

3、**简单短语**:简单子树的末端结点形成的符号串.

上例G[S]：　句型baSb的语法树



共有三棵子树，

三个短语：ba ，a，　Sb

简单短语：　a，　Sb

句柄：　a

这样的结论与短语定义义完全符合，为什么？

4、归约

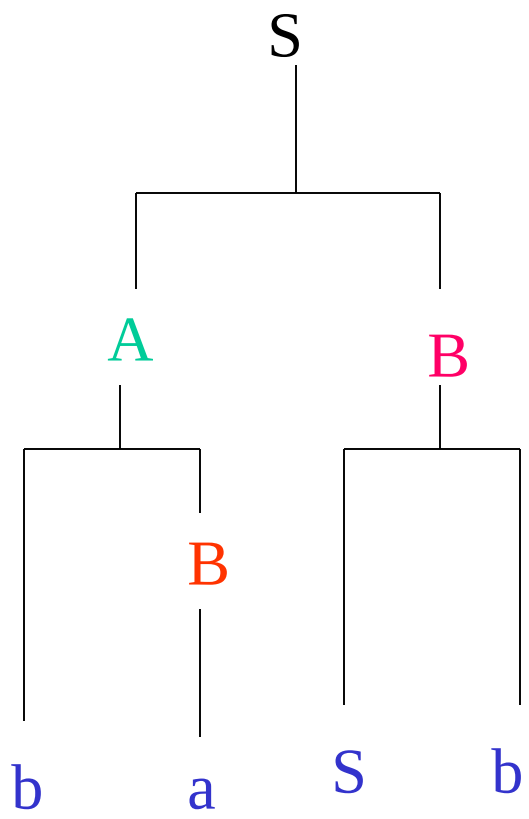
语法树由下向上生长，

通过规则替换到达开始符号的过程。

- 无二义性文法最左归约归约的是当前句型的句柄.
- 这个过程非常重要，因为源程序都是符号串形式的，这就需要把它归约为开始符号（程序）才算正确。
- 最左归约关键是找当前句型的句柄，这个问题，到语法分析时再着重讲解.

例G[S]：
S→AB
A→Aa|bB
B→a|Sb

句型baSb的归约过程.



归约过程

baSb

<≠bBSb

<≠ASb

<≠AB

<≠S

例G[S]：
S→AB
A→Aa|bB
B→a|Sb

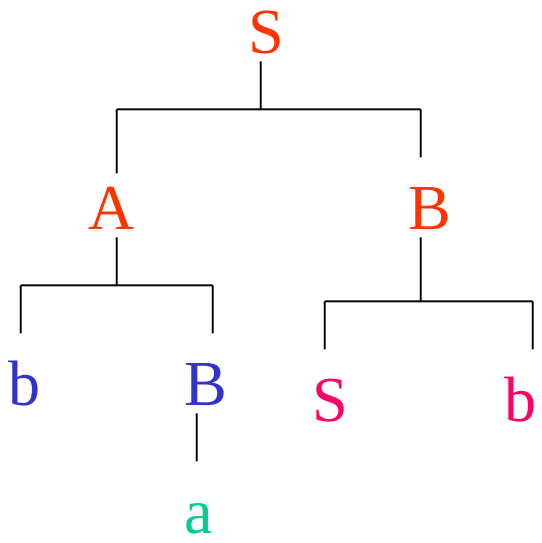
句型baSb

另一种推导的语法树

推导

S=>AB
=>ASb
=>bBSb
=>baSb

语法树



几个结论：

1、对每个语法树，至少存在一个推导过程

2、对于每个推导，都有一个相应的语法树（但不同的推导可能有相同的语法树）。

3、树的末端结点形成所要推导的句型。

但这个句型也可能对应两棵不同的语法树，这就是文法的二义性问题.

§2.3.5 文法的二义性

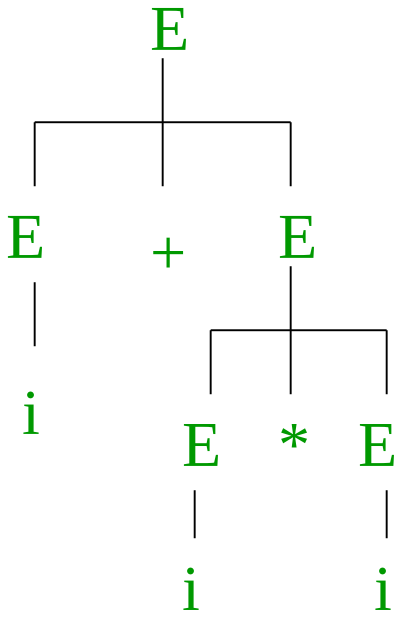
- 如果文法G的某一个句子存在两棵或两棵以上不同的语法树，则称句子是二义性的.
- 如果一文法含有二义性的句子，则称该文法是二义性的，否则该文法是无二义性的.

例G[E]:E→E+E|E*E|(E)|i

句子i+i*i 同是最左推导,对应两棵不同的语法树

最左推导1:

E=>E+E
=>i+E
=>i+E*E
=>i+i*E
=>i+i*i



例 G[E]:E→E+E|E*E|(E)|i

句子i+i*i 同是最左推导，对应两棵不同的语法树

最左推导2:

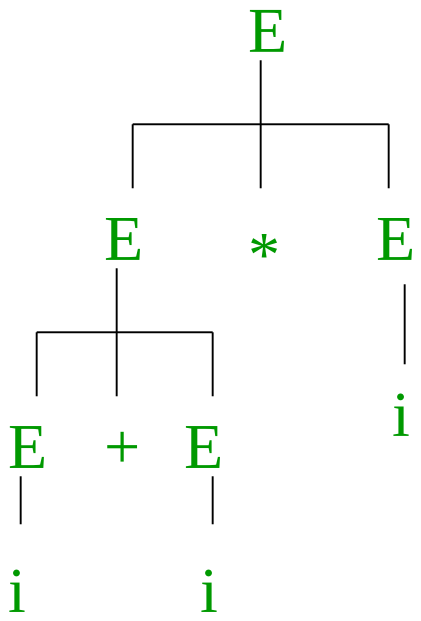
E=>E*E

=>E+E*E

=>i+E*E

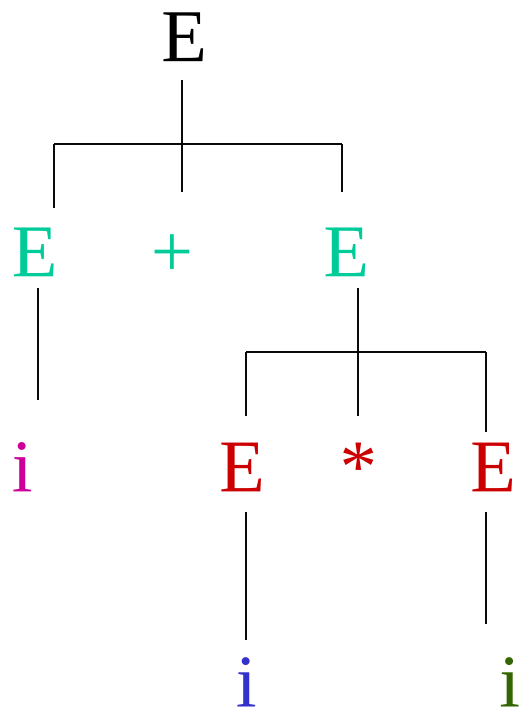
=>i+i*E

=>i+i*i



二义性的句子

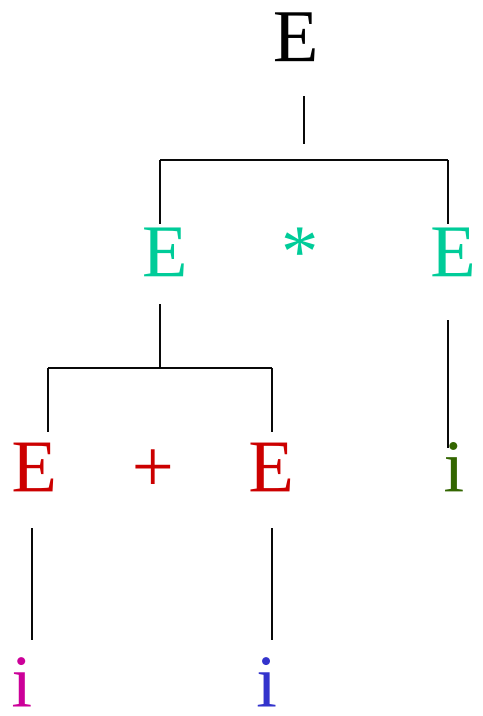
语法树1



句柄:i, i, i, E*E , E+E

表示:i+(i*i)

语法树2



句柄:i, i, E+E , i, E*E

表示:(i+i)*i

说明：

①文法的二义性：

某一句子有二个不同的最左（右）推导

或二个不同的最左（规范）归约

②文法的二义性是不可判定的：不存在一种算法，只能用一些简单条件来判定

③**特例**：若一文法G既含左递归又含右递归，则G必是二义性文法.（是经验）

例：G[E]:E→E+E|E*E|(E)|i

文法二义性的消除

1、定义规定或规则

二义性文法

G[E]:E→E+E|E*E|(E)|i

G[S]: S →if B then S else S
|if B then S

If A1 then if A2 then S1 else S2;

2、改写文法或制定规则

无二义性文法

G[E]:E→E+T|T
T→T*F|F
F→(E)|i

规定：else跟与它最近的尚未匹配的then匹配。

G3[S]:S → A | S-A

A → a | b | c

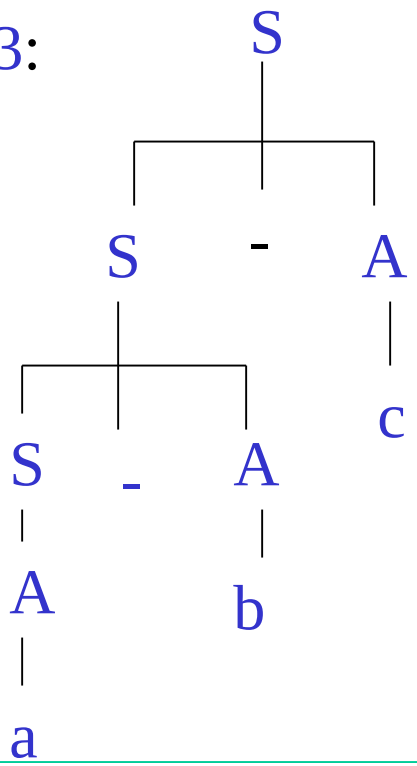
G4[S]:S → A | A-S

A → a | b | c

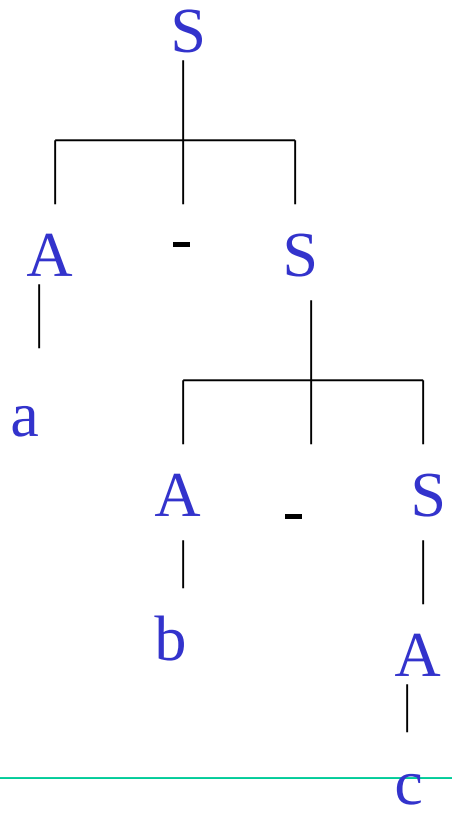
G3, G4等价文法

对句子 a-b-c语义不同

G3:



G4:



总结：

以上的分析中，忽略了两个问题

（1）自顶向下分析时：如有$V \rightarrow x_1|x_2...|x_n$选哪一产生式可一次推导成功？

（2）自底向上分析时，如何尽快找到当前句型的句柄？（进行归约）

这些问题将在语法分析时解决。

§2.4 文法的实用限制和其它表示方法

§2.4.1文法的实用限制

1、不含无用产生式

设G=（V_n，V_t，P，S）是一文法，G中的符号$x \in V_n \cup V_t$是有用的，则x 必满足

①存在α、$\beta \in V^*$，有$S =^* > \alpha x \beta$

②存在$\omega \in V_t^*$ 使$\alpha x \beta =^* > \omega$

称符号x是有用的，否则是无用的

G[S]:

S →aA |Bb

A →aA |c

B →bB

C →cC |d

无用产生式：产生式的左部或右部含有无用符号。

2、不含有害规则

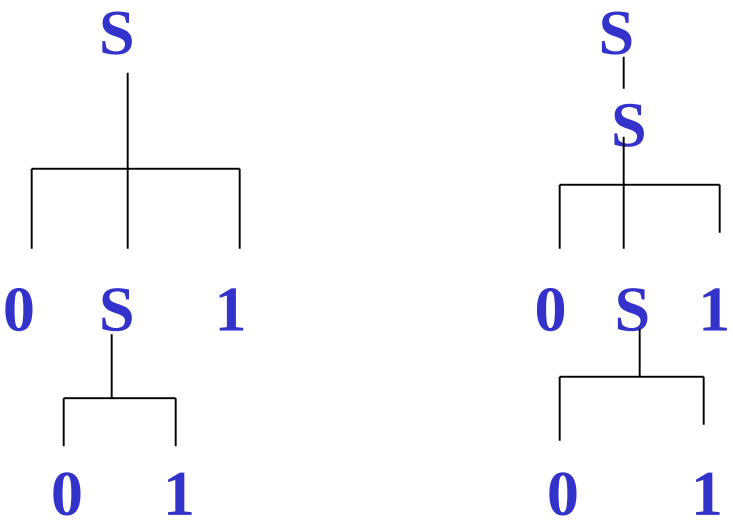
形如 U→ U 的规则 （原因①不必要②引起二义性）

例；$G_1[S]$: S→0S1|01　　G_1无二义性文法

$G_2[S]$: S→0S1|01|S　　G_2二义性文法

$L(G_1)=L(G_2)=\{0^n\ 1^n \mid n>=1\}$

G2文法句子0011 的两棵不同语法树.



§2.4.2 ε-产生式的消除

如某L(G)中不含ε，可消除G中的全部ε产生式；

如某L(G)中含ε，肯定不能消除G中的全部ε产生式；

消除步骤：

1. 算法2.3，找出G中满足$A=^{*}>\varepsilon$的所有A，构成集合W；

2. 算法2.4，若ε不属于L(G)，构造不含ε产生式的等价文法G′；

算法2.5，若ε属于L(G)，构造仅含$S^{①}\rightarrow\varepsilon$产生式的等价文法$G_1(S^{①})$；

算法2.3

设$G=(V_n, V_t, P, S)$

①作集合$W_1=\{A \mid A \rightarrow \varepsilon \in P\}$

②作集合序列$W_{k+1}= W_k \cup \{B \rightarrow \beta \in P$且$\beta \in W_k^+\}$

显然$W_k \subseteq W_{k+1}$ $(K>=1)$，由于V_n有限，故必存在某i，使得$W_i=W_{i+1}=\ldots$，令$W=W_i$，对每个$A \in W$，$A=^*> \varepsilon$

特别：当$S \in W$，则$\varepsilon \in L(G)$；否则，ε不属于$L(G)$.

算法2.4

设$G=(V_n, V_t, P, S)$，且ε不属于$L(G)$，则按下述算法构造

$G'=(V_n, V_t, P', S)$，使$L(G')=L(G)$；

①按算法2.3将V_n分为两个不相交的子集，W及V_n-W

②设$X \rightarrow X_1X_2...X_m$是P中的任一产生式，按下述规则将所有形

$Y \rightarrow Y_1Y_2...Y_m$的产生式放入P′中，对于一切$1<=i<=m$

（i）若X_i不属于W，即X_i属于$(V_n-W) \cup V_t$，则取$Y_i=X_i$；

（ii）若X_i属于W，则分别取Y_i为X_i和ε，即如果$Y_1Y_2...Y_m$中有j个符号属于W，则将有2^j个形如$Y \rightarrow Y_1Y_2...Y_m$的产生式放入P′中，但若所有的$X_i$均属于W，却不能把所有的$Y_i$都取为ε。

文法G′与G等价且不含ε产生式.

算法2.4 例题

判定条件：

S不能推出ε， 故ε不属于L(G)

文法G=({S,A,B,C}, {a,b,c}, P, S)

P:

S→aA

A →BC

B → bB

C →cC

B → ε

C →ε

解：

W={A,B,C} 算法2.3

算法2.4

由S→aA 得 S→aA及S→a 放入P′

由A →BC得A →BC及A →B及A →C放入P′

由B → bB得B → bB及B → b放入P′

由C →cC得C →cC及C →c放入P′

P′

S→aA S→a A→BC A→B A→C

B→bB B→b C→cC C→c

算法2.5

设$G=(V_n, V_t, P, S)$，且ε属于$L(G)$，S不出现在任何产生式的右部，执行算法2.4得$G'=(V_n, V_t, P', S)$，但$S\rightarrow\varepsilon$属于G'.

否则，按下述算法先构造$G'=(V_n^{①}, V_t, P', S^{①})$，再构造$G_1=(V_n^{①}, V_t, P^{①}, S^{①})$，使$L(G_1)=L(G)$；

①引入新的符号$S^{①}$（$S^{①}$不属于V），作为G′的开始符号，并令$V_n^{①}=V_n \cup \{S^{①}\}$；

②作产生式集$P'=P \cup \{S^{①}\rightarrow\alpha \mid S\rightarrow\alpha \in P\}$得到G′.

③对文法$G'=(V_n^{①}, V_t, P', S^{①})$，执行算法2.4消去P′中的全部$\varepsilon$产生式，并将$S^{①}\rightarrow\varepsilon$加入得到$P^{①}$得到文法$G_1=(V_n^{①}, V_t, P^{①}, S^{①})$，$L(G_1)=L(G)$；

算法2.5 例题

判定条件：

S能推出ε， 故ε属于L(G)

文法G=({S,A,B}, {a,b,c}, P, S)

求$G_1=(V_N^1, V_T, P^{①}, S^{①})$

P:

S →cS

S →AB

A→ aAb

B →Bb

A → ε

B →ε

解：

引入新符号$S^{①}$作产生式集：

$P'=P\cup\{S^{①} \rightarrow cS, S^{①} \rightarrow AB\}$

执行算法2.4，加入$S① \rightarrow \varepsilon$得$P^{①}$

$S^{①} \rightarrow cS$ $S^{①} \rightarrow c$ $S^{①} \rightarrow AB$ $S^{①} \rightarrow A$ $S^{①} \rightarrow B$

$S① \rightarrow \varepsilon$ $S \rightarrow cS$ $S \rightarrow c$ $S \rightarrow AB$ $S \rightarrow A$

$S \rightarrow B$ $A \rightarrow aBb$ $A \rightarrow ab$ $B \rightarrow Bb$ $B \rightarrow b$

$G_1=(V_N\cup\{S^{①}\}, V_T, P^{①}, S^{①})$

且$L(G) = L(G_1)$

§2.4.3 文法的其它表示方法

一、扩充的BNF表示

BNF:元符号 〈 , 〉 , ::=(→), |

扩充的BNF(EBNF): 〈 , 〉 , ::=(→), | ,

(,) , {, }, [,]

1、{ }

$\{t\}_n^m$ $t \in V^*$, 符号串t自重复n到m次.

$\{t\}$ $t \in V^*$, 符号串t自重复0到无穷次.

例:BNF: G[<无符号整数>] （含左递归）

<无符号整数>→<数字> | <无符号整数><数字>

<数字> →0 | 1 | 2 | 3 | …… | 9

扩充的BNF:

G[<无符号整数>]

<无符号整数>→<数字> { <数字>} （不含左递归）

<数字> →0 | 1 | 2 | 3 | …… | 9

2、[ ]

[t]，t ∈V*，t 符号串可有可无

例:BNF :S→if B then S

| if B then S else S

扩充的BNF:S→if B then S [else S]

3、( ) 规则中提取公因子

例:BNF: U→xy | xw | ..... | xz

扩充的BNF: U→x(y | w | ..... | z)

令:U′ → y | w | ..... | z

则文法为:U → x U′

U′ → y | w | ..... | z

例： BNF： G[〈标识符〉]

〈标识符〉→〈字母〉|〈标识符〉〈字母〉

|〈标识符〉〈数字〉

〈标〉→〈字〉|〈标〉(〈字〉|〈数〉)

〈字〉→a|b|...|z|A|...|Z

〈数〉→0|1|2|...|9

〈标〉=>〈标〉(〈字〉|〈数〉)

$=^{+}$>〈标〉(〈字〉|〈数〉)..... (〈字〉|〈数〉)

=>〈字〉(〈字〉|〈数〉)..... (〈字〉|〈数〉)

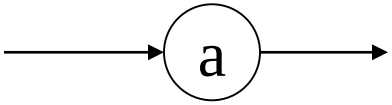
〈标〉 =>〈字〉{(〈字〉|〈数〉)}

〈标〉 =>〈字〉{〈字〉|〈数〉}

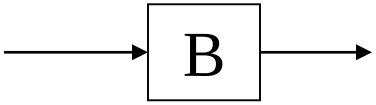
二、语法图

构造：规则右部的符号

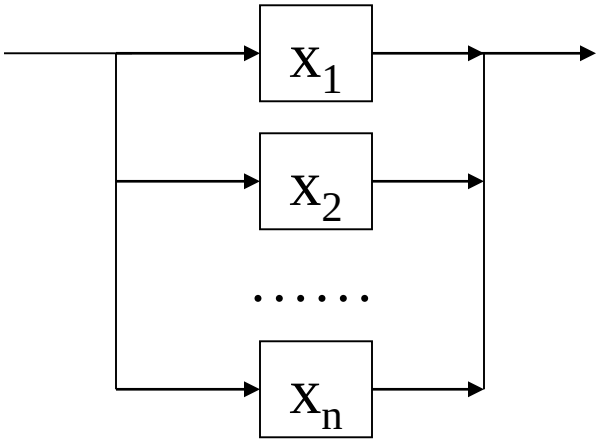
(1) a ∈Vt



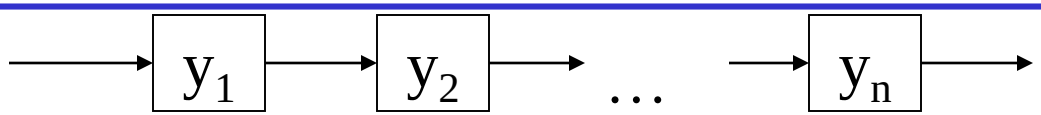
(2) B ∈Vn



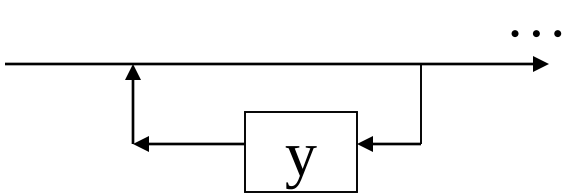
(3) 形如$U \rightarrow x_1 | x_2 | \cdots\cdots | x_n$



$X=y_1y_2\cdots\cdots y_n$



$X=\{y\}$

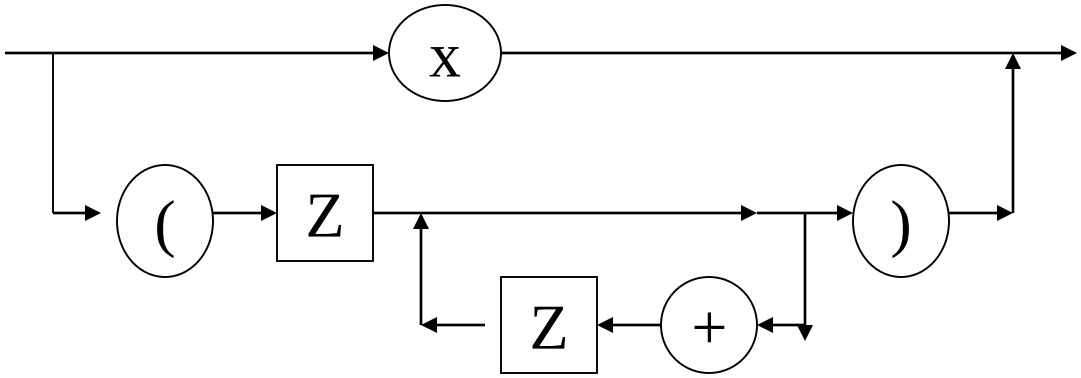


例： G[Z]：

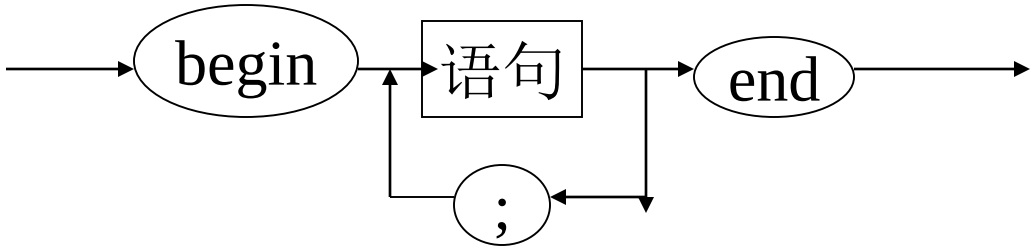
Z →x |(B)

B →ZC

C →{+Z}



例Pascal语言中的〈分程序〉



§2.5 文法和语言的Chomsky分类

文法是一个四元组G=（V_n，V_t，P，Z）

乔姆斯基根据文法中P的不同，将文法分为四类，每一种文法对应一种语言.

- 0型文法：文法G中规则呈

$\alpha \rightarrow \beta$　　$\alpha \in V^+$，$\beta \in V^*$

也称短语结构文法（Phrase Structure Grammar）

确定的语言为0型语言L_0

递归可枚举语言　〈-〉 图灵机

- 1型文法：文法G中规则呈

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 \qquad \alpha_1,\ \alpha_2 \in V^*,$$

$$A \in V_n,\ \beta \in V^+$$

也称上下文有关文法.(Context Sensitive Grammar)

确定的语言为1型语言L_1,也称上下文有关语言.

线性限界自动机

- 2型文法：文法G中规则呈

$$A \rightarrow \beta \qquad A \in V_n,\ \beta \in V^+$$

也称上下文无关文法.(Context Free Grammar)

非确定下推自动机识别

确定的语言为2型语言L_2或上下文无关语言.

在语法分析中用于描述语法类

➢ 3型文法：文法G中规则呈：

- $A \rightarrow aB$或$A \rightarrow a$　A、B $\in V_n$, $a \in V_t$,

  称G为右线形正则文法.

- $A \rightarrow Ba$或$A \rightarrow a$　A、B $\in V_n$, $a \in V_t$,

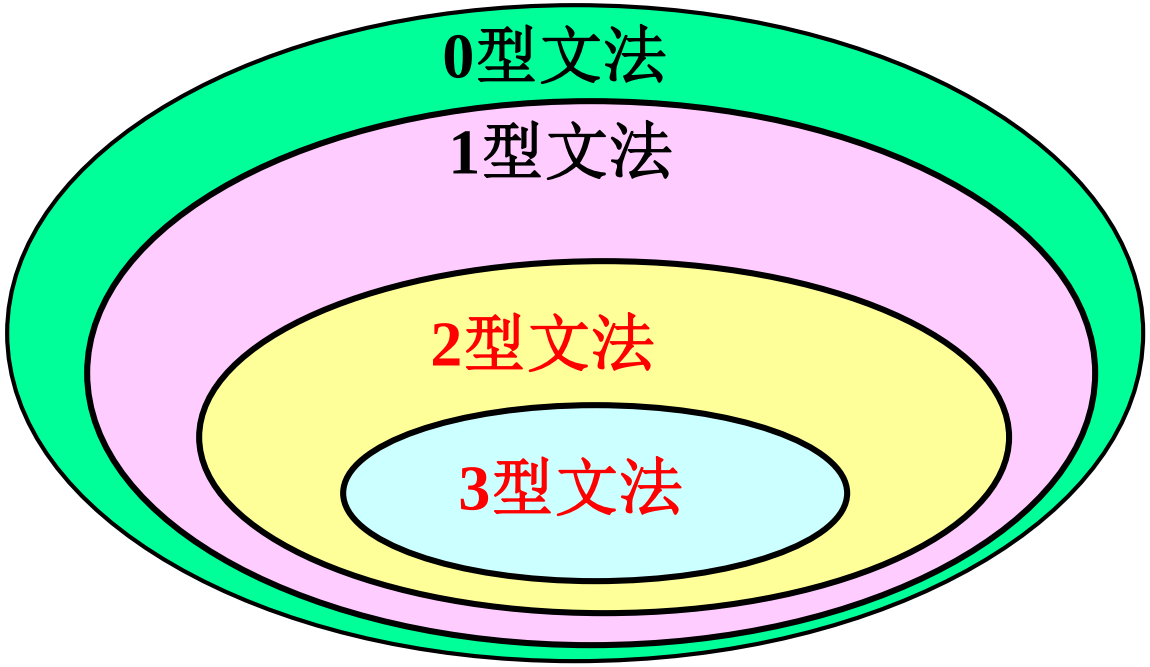
  称G为左线形正则文法.

  确定的语言为3型语言L_3或正则语言.

  有限自动机识别.

在词法分析中用于描述单词符号

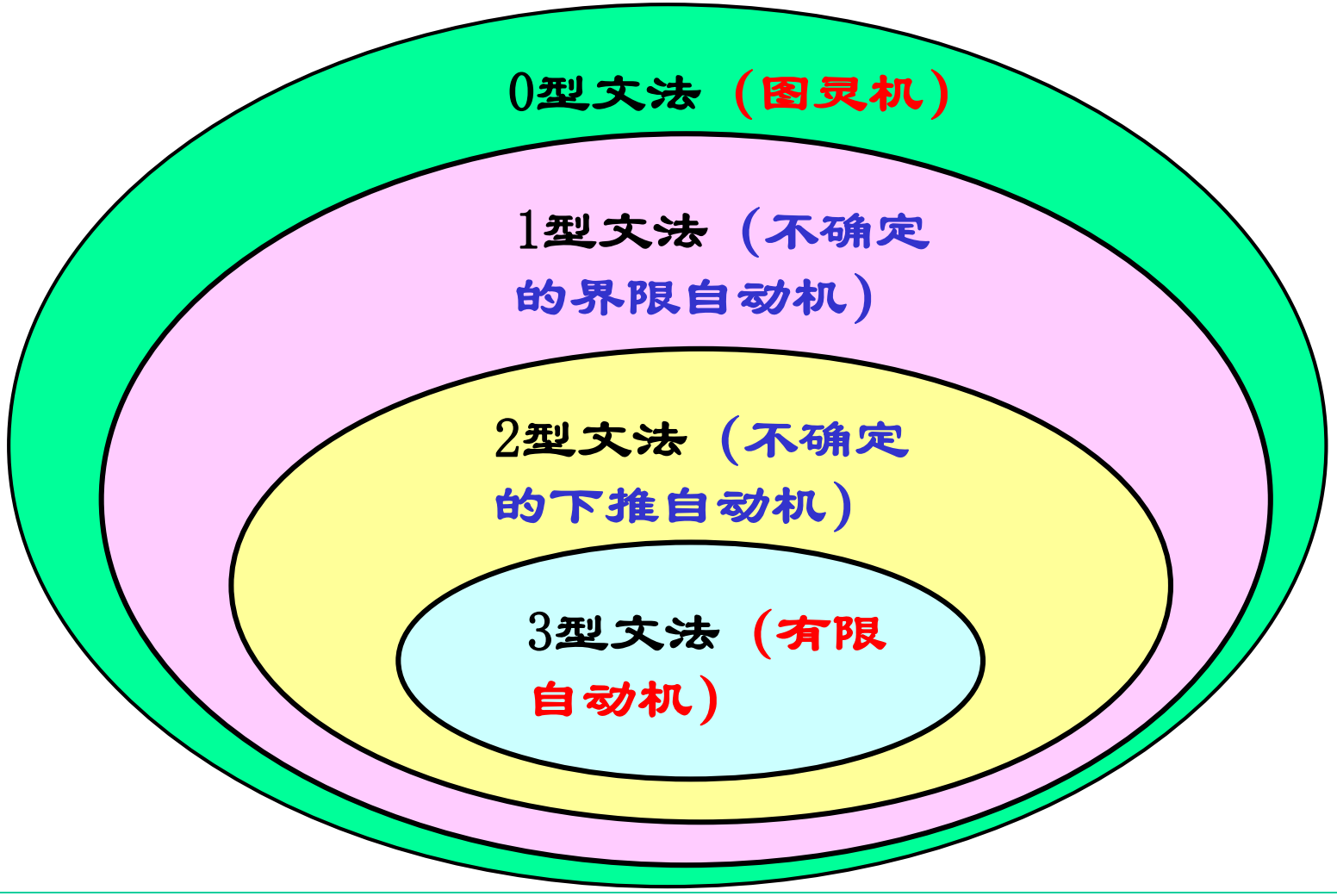
四种文法之间的逐级“包含”关系

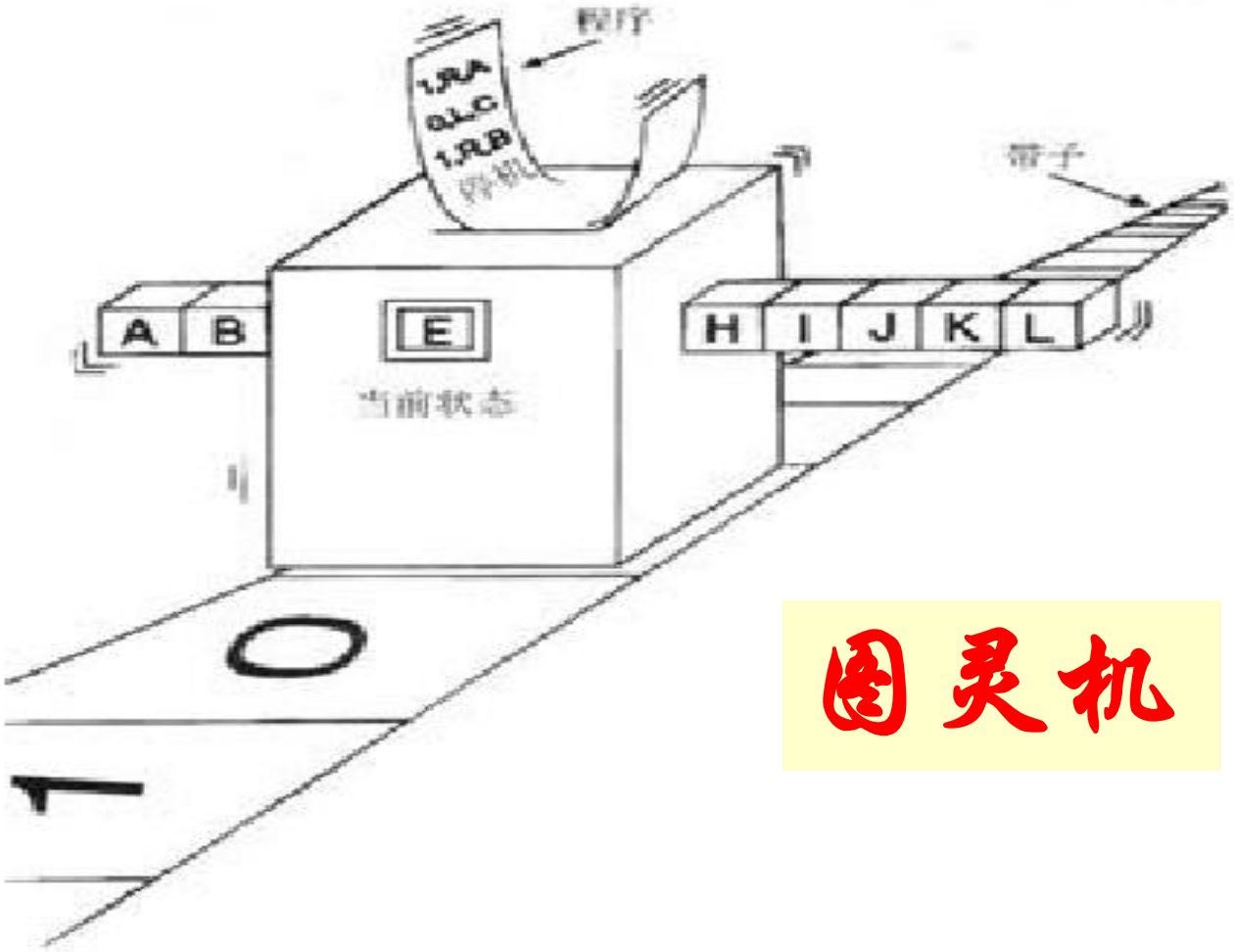


根据上述讨论 $L0 \supset L1 \supset L2 \supset L3$

0型文法可以产生L0、L1、L2、L3，但2型文法只能产生L2，不能产生L1。

形式语言与自动机





图灵机

说明：

①由于对规则的限制逐渐增加，

因此L_0、L_1、L_2、L_3的语言范围逐渐减小.

②一个文法是正则的，必然是上下文无关的 .

③本课主要讨论正则文法和上下文无关文法.

第二章 作业

P38

2-2 (1) (2)

2-3 (1) (2)

2-6

2-10

2-11 (2) (3)

课堂练习(一)

证明

S→aSb | Sb | b

为二义性文法

课堂练习(一)(答案)

证明

$S \rightarrow aSb \mid Sb \mid b$

为二义性文法

解：找一个句子，可以生成2棵语法树

如:aabbbb

课堂练习（二）

将下列文法改写成无二义性文法

S→SS | (S) | ()

课堂练习(二)（答案）

将下列文法改写成无二义性文法

S→SS | (S) | ()

解：

分析根源，再改写： S→SS

S→TS | T

T→(S) | ()

课堂练习（三）

写一个文法，使其语言L(G)是非零开头的正偶数集合

课堂练习（三）（答案）

写一个文法，使其语言L(G)是非零开头的正偶数集合

解：（写产生式集合）

S→XYZ | 2 | 4 | 6 | 8

X→1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Y→YX | Y0 | ε

Z→0 | 2 | 4 | 6 | 8

课堂练习（四）

写一个文法，使其语言L(G)是非零开头的正偶数集合

课堂练习（四）（答案）

写一个文法，使其语言L(G)是非零开头的正偶数集合

解：（写产生式集合）

S→XYZ | 2 | 4 | 6 | 8

X→1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Y→YX | Y0 | ε

Z→0 | 2 | 4 | 6 | 8