第2章　文法和语言的基本知识

本章学习导读

形式语言理论是编译的重要理论基础。本章主要介绍编译理论中用到的有关形式语言理论的最基本概念。具体包括下面5个方面的内容：

（1）字母表和符号串

（2）文法和语言的形式定义

（3）短语、直接短语和句柄

（4）语法树和文法的二义性

（5）文法和语言的分类

2.1　概　　述

在第1章我们介绍过编译程序，它的功能是将高级语言所写的源程序翻译成与之等价的机器语言或汇编语言的目标程序。也就是说，我们所要构造的编译程序是针对某种程序设计语言的，编译程序要对它进行正确地翻译，首先要对程序设计语言本身进行精确地定义和描述。对程序设计语言的描述是从语法、语义和语用3个因素来考虑的。所谓语法是对语言结构的定义；语义是描述了语言的含义；语用则是从使用的角度去描述语言。

例如，对于赋值语句 s = 2 * 3.1416 * r * (r + h) 的非形式化的描述如下：

语法——赋值语句由一个变量、一个后随赋值号“=”及其后跟一个表达式构成。

语义——首先计算语句右部表达式的值，然后把所得结果送入左部变量中。

语用——赋值语句可用来计算和保存表达式的值。

这种非形式化的描述，不够清晰和准确，为了精确定义和描述程序设计语言，需采用形式化的方法。所谓形式化的方法，是用一整套带有严格规定的符号体系来描述问题的方法。这种方法正是著名的语言学家 Noam Chomsky 在1956年提出的形式语言理论中所研究的问题，也就是说，形式语言理论是编译的理论基础，因此在这一章将介绍编译理论中用到的有关形式语言的某些基本概念和知识。

2.2　字母表和符号串的基本概念

2.2.1　字母表和符号串

1. 字母表

字母表是元素的非空有穷集合。

例如，$\Sigma=\{a,b,c\}$。

根据字母表的定义，Σ 是字母表，它由 a，b，c 三个元素组成。

需要注意的是，字母表中至少包含一个元素。字母表中的元素，可以是字母、数字或其他符号。

例如，$\Sigma'=\{0,1\}$ 是一个字母表，由 0 和 1 两个元素组成。

不同的语言有不同的字母表，如英文的字母表是 26 个字母、数字和标点符号的集合，C 语言的字母表是由字母、数字和若干专用符号组成。任何语言的字母表指出了该语言中允许出现的一切符号。

2. 符号（字符）

字母表中的元素称为符号，或称为字符。

例如，前述例子中 a，b，c 是字母表 Σ 中的符号；0 和 1 是字母表 Σ' 中的符号。

3. 符号串（字）

符号的有穷序列称为符号串。

例如，设有字母表 $\Sigma=\{a,b,c\}$，则有符号串 a，b，ab，ba，cba，abc，…

符号串总是建立在某个特定字母表上的且只能由字母表上的有穷多个符号组成。需要指出的是，符号串中符号的顺序是很重要的，如 ab 和 ba 是字母表 Σ 上的两个不同的符号串。不包含任何符号的符号串，称为空符号串，用 ε 表示，即空符号串由 0 个符号组成，其长度 $|\varepsilon|=0$。

2.2.2 符号串的运算

1. 符号串的连接

设 x 和 y 是符号串，则串 xy 称为它们的连接，即 xy 是将 y 符号串写在 x 符号串之后得到的符号串。

例如，设 $x=abc$，$y=10a$，则 $xy=abc10a$，$yx=10aabc$。

注意：对任意一个符号串 x，我们有 $\varepsilon x=x\varepsilon=x$。

2. 集合的乘积

设 A 和 B 是符号串的集合，则 A 和 B 的乘积定义为

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

例如，设 $A=\{a,b\}$，$B=\{c,d\}$，则 $AB=\{ac,ad,bc,bd\}$。

集合的乘积是满足于 $x\in A$，$y\in B$ 的所有符号串 xy 所构成的集合。

由于对任意的符号串 x，总有 $\varepsilon x=x\varepsilon=x$，所以，对任意集合 A，有

$$\{\varepsilon\}A=A\{\varepsilon\}=A$$

特别需要指出的是，ε 是符号串，不是集合，而 $\{\varepsilon\}$ 表示由空符号串 ε 所组成的集合，但这样的集合不是空集合 $\varnothing=\{\}$。

3. 符号串的幂运算

设 x 是符号串，则 x 的幂运算定义为

$$x^0=\varepsilon$$
$$x^1=x$$
$$x^2=xx$$

$$\cdots$$

$$x^n = \underbrace{xx\cdots x}_{n个} = xx^{n-1} \quad (n>0)$$

例如，设 $x=abc$，则

$$x^0 = \varepsilon$$
$$x^1 = abc$$
$$x^2 = xx = abcabc$$
$$\cdots$$

4. 集合的幂运算

设 A 是符号串的集合，则集合 A 的幂运算定义为

$$A^0 = \{\varepsilon\}$$
$$A^1 = A$$
$$A^2 = AA$$
$$\cdots$$
$$A^n = \underbrace{AA\cdots A}_{n个} = AA^{n-1} \quad (n>0)$$

例如，设 $A=\{a,b\}$，则

$$A^0 = \{\varepsilon\}$$
$$A^1 = \{a,b\}$$
$$A^2 = AA = \{aa,ab,ba,bb\}$$
$$A^3 = AAA = A^2A = \{aaa,aab,aba,abb,baa,bab,bba,bbb\}$$
$$\cdots$$

5. 集合 A 的正闭包 A^+ 与闭包 A^*

设 A 是符号串的集合，则 A 的正闭包 A^+ 和 A 的闭包 A^* 定义为

$$A^+ = A^1 \cup A^2 \cup \cdots \cup A^n \cdots$$
$$A^* = A^0 \cup A^1 \cup A^2 \cup \cdots \cup A^n \cdots = \{\varepsilon\} \cup A^+$$

例如，设 $A=\{a,b\}$ 则

$$A^+ = \{a,b,aa,ab,ba,bb,aaa,aab,\cdots\}$$
$$A^* = \{\varepsilon,a,b,aa,ab,ba,bb,aaa,aab,\cdots\}$$

可见，集合 A 的正闭包表示 A 上元素 a，b 构成的所有符号串的集合，集合 A 的闭包比集合 A 的正闭包多含一个空符号串 ε。

2.3　文法和语言的形式定义

2.3.1　形式语言

序列的集合称为形式语言。具体地说，每个形式语言都是某个字母表上按某种规则构成的所有符号串的集合，反之，任何一个字母表上符号串的集合均可定义为一个形式语言。

对每个具体语言，都有语法和语义两个方面，形式语言是指不考虑语言的具体意义，即不考虑语义。例如，C 语言是其基本符号字母表上的符号串的集合，而每个 C 语言程序是基本符号的符号串。

对形式语言的描述有两种方法，一种方法是当语言为有穷集合时，用枚举方法来表示语言。例如，设有字母表 $A=\{a,b,c\}$，则

$$L_1=\{a,b,c\}$$
$$L_2=\{a,aa,ab,ac\}$$
$$L_3=\{c,cc\}$$

均表示字母表 A 上的一个形式语言。由于这3个语言均是有限符号串的集合，因此，可枚举出其全部句子来表示该语言。但并不是所有语言都是有穷集，例如，设字母表 $\Sigma=\{0,1\}$，则 $\Sigma^+=\Sigma^1\cup\Sigma^2\cup\Sigma^3\cup\cdots=\{0,1,00,10,11,01,000,100,\cdots\}$，它是0和1构成的所有可能的符号串的集合，对这种无穷集合的语言，无法用枚举法来描述，我们需要设计文法来描述无穷集合的语言。假设用符号 A 代表该无穷集合 Σ^+，则 A 中自然包含符号串0，1，00，10，…；或者说 A 由0，1，00，10等符号串组成。我们可以记为 $A\rightarrow0$，$A\rightarrow1$，$A\rightarrow00$，$A\rightarrow10$，…。但如此下去，又将产生一个无穷集合，仍然不便于计算机的存储和处理。为了将无穷转换成有穷表达，可以借鉴递归函数、数学归纳法的类似思想。我们考虑，无穷集合中的任何一个字符串，其结尾一定是0或1，而去掉这个结尾字符后，剩余的串仍由0和1构成，所以剩余串也必定在 A 中。这里注意，串的长度为2以上，则有递归表达 $A\rightarrow A0$，$A\rightarrow A1$；也可以理解为，将 A 中的所有串，按结尾的不同，分为以0结尾或以1结尾的两类。反过来，A 中的任意串，其尾部添加0或1，仍在 A 中。下面用 A 表示 Σ^+，用式子 $A\rightarrow0$ 表示符号串 $0\in A$ 或 A 生成符号串0，符号“→”读作“生成”或“由……组成”，则集合 A 可表示成

$$A\rightarrow0$$
$$A\rightarrow1$$
$$A\rightarrow A\ 0$$
$$A\rightarrow A\ 1$$

显然，由 A 生成的符号串属于 Σ^+，这就是所谓用文法描述语言，它描述了无穷集合的语言。

2.3.2 文法的形式定义

1. 规则

规则也称产生式，它是一个符号与一个符号串的有序对 (A,β)，通常写作

$$A\rightarrow\beta \qquad （或\ A::=\beta）$$

其中，A 是规则左部，它是一个符号；β 是规则右部，它是一个符号串；“→”和“::=”表示“定义为”或“生成”，意思是左部符号用右部的符号串定义或左部符号生成右部符号串。

例如，前述例中一组规则

$$A\rightarrow0$$
$$A\rightarrow1$$
$$A\rightarrow A\ 0$$
$$A\rightarrow A\ 1$$

描述的语言序列只可能是由0和1组成的符号串，即 $\Sigma^+=\{0,1,00,01,10,000,100,\cdots\}$。从例子可以看出，规则的作用是告诉我们如何用规则中的符号串生成语言中的序列，也就是说，一组规则规定了一个语言的语法结构。

规则中出现的符号分为两类，一类是终结符号，另一类是非终结符号。非终结符号是出现在规则左部能派生出符号或符号串的那些符号，即每个非终结符号表示一定符号串的集合，用大写字母表示或用尖括号把非终结符号括起来。例如，上例中的 A。终结符号是不属于非终结符号的那些符号，它是组成语言的基本符号，是一个语言的不可再分的基本符号，通常用小写字母表示。例如，上例中的 0 和 1。

2. 文法

文法是规则的非空有穷集合，通常表示成四元组 $G=(V_N,V_T,P,S)$。其中，

V_N 是规则中非终结符号的集合。

V_T 是规则中终结符号的集合。$V_N \cap V_T = \varnothing$。通常用 V 表示 $V_N \cup V_T$，称为文法 G 的字汇表。

P 是文法规则的集合。

S 是一个非终结符号，称为文法的开始符号或文法的识别符号，它至少要在一条规则中作为左部出现。由它开始，识别出我们所定义的语言。

由文法定义可知，文法是对语言结构的定义和描述，文法四大要素中，关键是规则的集合。

为了书写方便，对于若干个左部相同的规则，如

$$A \to \alpha_1$$
$$A \to \alpha_2$$
$$\cdots$$
$$A \to \alpha_n$$

将它们缩写为 $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$，其中每个 α_n 有时也称为 A 的一个候选式。我们约定：第一条规则的左部是识别符号；对文法 G 不用四元式显示表示，而只将规则写出。

下面举例说明给定语言 L 后，如何写出能正确描述此语言的文法 G。

【例 2.1】 设字母表 $\Sigma=\{a,b\}$，试设计一个文法，描述语言

$$L=\{a^{2n},b^{2n} \mid n \geqslant 1\}$$

分析　设计一个文法来描述一个语言，关键是设计一组规则生成语言中的符号串。因此，为设计该语言文法，必须分析这个语言是由怎样一些符号串组成的，即首先分析语言中符号串的结构特征：

当 $n=1$　　$L=\{aa,bb\}$

当 $n=2$　　$L=\{aaaa,bbbb\}$

当 $n=3$　　$L=\{aaaaaa,bbbbbb\}$

…

$L=\{aa,bb,aaaa,bbbb,aaaaaa,bbbbbb,\cdots\}$

即语言 L 是由偶数个 a、偶数个 b 这样的符号串组成的集合。因此，定义语言 L 的文法

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

其中，

$$V_N=\{A,B,D\},V_T=\{a,b\}$$
$$P=\{A \to aa \mid aaB \mid bb \mid bbD$$
$$B \to aa \mid aaB$$

$$D \rightarrow bb \mid bbD\}$$
$$S = A$$

那么，描述该语言的文法是否唯一呢？我们不难对语言 L 设计出文法 G'

$$G' = (\{A, B, D\}, \{a, b\}, P, A)$$

其中，P 为

$$A \rightarrow B \mid D$$
$$B \rightarrow aa \mid aBa$$
$$D \rightarrow bb \mid bDb$$

显然，G 不同于 G'。由此可见，对于一个给定的语言，描述该语言的文法不是唯一的。

G 和 G'是两个不同的文法，如果它们描述的语言相同，那么称 G 和 G'为等价文法。等价文法的存在，使我们能在不改变文法所确定的语言的前提下，为了某种目的而对文法进行改写。

对此例，我们提出下面这样一个问题：描述该语言的文法为什么不是 G''？

$$G'' = (\{A\}, \{a, b\}, P, A)$$

其中，P 为

$$A \rightarrow aa \mid bb \mid Aaa \mid Abb$$

对于文法 G''来说，它所产生的所有符号串都应该属于语言 L，但 G''产生的有些符号串，如 $aabb$，$bbaa$，…不属于语言 L，即设计的文法超出了所定义语言的范围。

【例 2.2】 试设计一个表示所有标识符的文法。

分析　题意是用文法定义标识符，必须确定 P 中规则。为了设计出一组规则，首先应搞清楚集合中符号串的结构特征。标识符的定义是字母或以字母开头的字母或数字串，其结构如图 2.1 所示。

字母	字母或数字串

图 2.1　标识符的结构

用 I 代表标识符，L 代表字母，D 代表数字，则定义标识符的文法为

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

其中，

$$V_N = \{I, L, D\}$$
$$V_T = \{a, b, c, \cdots, x, y, z, 0, 1, 2, \cdots, 9\}$$
$$P = \{I \rightarrow L \mid IL \mid ID$$
$$L \rightarrow a \mid b \mid c \mid \cdots \mid x \mid y \mid z$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \cdots \mid 9\}$$
$$S = I$$

若将定义标识符的文法设计成

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

其中，V_N，V_T，S 同上，

$$P = \{I \rightarrow L \mid ID$$
$$L \rightarrow a \mid b \mid c \mid \cdots \mid x \mid y \mid z$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \cdots \mid 9\}$$

该文法不能定义 ab，abc，…仅由字母串组成的标识符，缩小了所定义语言的范围。

【例 2.3】 用文法定义一个含 +、* 的算术表达式，定义用下述自然语言描述：变量 i 是一个表达式；若 E_1 和 E_2 是算术表达式，则 E_1+E_2，E_1*E_2，(E_1) 也是算术表达式。

分析　算术表达式的定义用自然语言描述，这是对算术表达式的非形式定义，题意是用文法来定义算术表达式，即是用形式化的方法定义表达式。定义算术表达式的文法为

$$G=(\{E\},\{i,+,*,(,)\},P,E)$$

其中，P 为

$$E\rightarrow i\mid E+E\mid E*E\mid (E)$$

【例 2.4】 设字母表 $\Sigma=\{a,b\}$，试设计一个文法，描述语言 $L=\{ab^na\mid n\geqslant 0\}$。

分析　该语言中符号串的结构特征是

当 $n=0$　　$L=\{aa\}$　　$(b^0=\varepsilon)$

当 $n=1$　　$L=\{aba\}$

当 $n=2$　　$L=\{abba\}$

…

$L=\{aa,aba,abba,\cdots\}$

所以，定义语言的文法为

$$G=(\{A,B\},\{a,b\},\{A\rightarrow aBa,B\rightarrow Bb\mid\varepsilon\},A)$$

2.3.3　语言的形式定义

当一个文法已知时，如何确定出该文法所定义的语言呢？为此，我们首先引进有关直接推导、推导等概念。

1. 直接推导

令 G 是一文法，我们从 xAy 直接推出 $x\alpha y$，即 $xAy\Rightarrow x\alpha y$，仅 $A\rightarrow\alpha$ 是 G 的一个规则且 $x,y\in(V_N\cup V_T)^*$。也就是说，从符号串 xAy 直接推导出 $x\alpha y$ 仅使用一次规则。

例如，设有文法 $G[S]$（符号 $G[S]$表示 S 为文法 G 的开始符号）：

$$G[S]=(\{S\},\{0,1\},P,S)$$

其中，P 为

$$S\rightarrow 0\ 1\mid 0\ S\ 1$$

有如下直接推导：

$S\Rightarrow 01$	使用规则 $S\rightarrow 01$，	此时 $x=\varepsilon$，　$y=\varepsilon$
$S\Rightarrow 0S1$	使用规则 $S\rightarrow 0S1$，	此时 $x=\varepsilon$，　$y=\varepsilon$
$0S1\Rightarrow 0011$	使用规则 $S\rightarrow 01$，	此时 $x=0$，　$y=1$
$00S11\Rightarrow 000S111$	使用规则 $S\rightarrow 0S1$，	此时 $x=00$，　$y=11$
$000S111\Rightarrow 00001111$	使用规则 $S\rightarrow 01$，	此时 $x=000$，$y=111$

注意推导和规则的区别：一是形式上的区别，推导用“⇒”表示，规则用“→”表示；二是对文法 G 中任何规则 $A\rightarrow\alpha$，有 $A\Rightarrow\alpha$，即推导的依据是规则。

2. 推导

如果存在一个直接推导序列：

$$\alpha_0\Rightarrow\alpha_1\Rightarrow\alpha_2\Rightarrow\cdots\Rightarrow\alpha_n$$

则称这个序列是一个从 α_0 至 α_n 的长度为 n 的推导，记为 $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$。即 $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$ 表示从 α_0 出发，经一步或若干步或使用若干次规则可推导出 α_n。

例如，设有文法 $G[E]$：

$$G[E] = (\{E,T,F\},\{i,+,*,(,)\},P,E)$$

其中，P 为

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (E) \mid i$$

对 $i+i*i$ 有如下直接推导序列

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T$$
$$\Rightarrow i+T*F \Rightarrow i+F*F \Rightarrow i+i*F \Rightarrow i+i*i$$

我们可记为 $E \overset{+}{\Rightarrow} i+i*i$。

3. 广义推导

$\alpha_0 \overset{*}{\Rightarrow} \alpha_n$ 表示从 α_0 出发，经 0 步或若干步，可推导出 α_n。也就是说，$\alpha_0 \overset{*}{\Rightarrow} \alpha_n$ 意味着，$\alpha_0 = \alpha_n$，或者 $\alpha_0 \overset{+}{\Rightarrow} \alpha_n$。

对上例，有

$$E \overset{*}{\Rightarrow} E$$
$$E \overset{*}{\Rightarrow} i+i*i$$

显然，直接推导的长度为 1，推导的长度大于等于 1，而广义推导的长度大于等于 0。

4. 句型和句子

设有文法 $G[S]$（S 是文法 G 的开始符号），如果 $S \overset{*}{\Rightarrow} x$，$x \in (V_N \cup V_T)^*$，则称符号串 x 为文法 $G[S]$的句型。$S \overset{*}{\Rightarrow} x$，$x \in V_T^*$，则称 x 是文法 $G[S]$的句子。

【例 2.5】 设有文法 $G[S]$：

$$S \rightarrow 01 \mid 0S1$$

有

$$S \overset{*}{\Rightarrow} 01$$
$$S \overset{*}{\Rightarrow} 0S1$$
$$S \overset{*}{\Rightarrow} 00S11$$
$$S \overset{*}{\Rightarrow} 000111$$

显然，符号串 01，0S1，00S11 和 000111 都是文法 $G[S]$的句型，而 01 和 000111 又是文法 $G[S]$的句子。

【例 2.6】 设有文法 $G[E]$：

$$E \rightarrow E+E \mid E*E \mid (E) \mid i$$

试证明符号串（$i*i+i$）是文法 $G[E]$的一个句子。

分析 只要证明符号串（$i*i+i$）对文法 G 存在一个推导，就可证明符号串（$i*i+i$）是文法 $G[E]$的一个句子。因为有

$$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (E*E+E) \Rightarrow (i*E+E) \Rightarrow (i*i+E) \Rightarrow (i*i+i)$$

即有 $E \overset{*}{\Rightarrow} (i*i+i)$，所以符号串$(i*i+i)$是文法 $G[E]$的一个句子。

5. 语言

文法 $G[S]$产生的所有句子的集合称为文法 G 所定义的语言，记为 $L(G[S])$：

$$L(G[S])=\{x \mid S\overset{+}{\Rightarrow}x \text{ 且 } x\in V_T^{*}\}$$

注意，这里用 $S\overset{+}{\Rightarrow}x$，而不是用 $S\overset{*}{\Rightarrow}x$。因为 S 是文法的开始符号，$S\in V_N$，若用 $S\overset{*}{\Rightarrow}x$，那么有可能有 $S=x$，这与 $x\in V_T^{*}$ 相矛盾。此时的 x 不可能是句子。此处，如果用 $S\overset{*}{\Rightarrow}x$，由于后面的条件是“且”的关系，也会自然在集合中去除掉开始符号，所以得到的最终集合与这里定义的 L 相同。

由语言定义可知：

（1）当文法给定，语言也就确定。

（2）$L(G)$是 V_T^{*} 的子集，即属于 V_T^{*} 的符号串 x 不一定属于 $L(G)$。

【例 2.7】 设有文法 $G[S]$：

$$S\to 01 \mid 0S1$$

求该文法所描述的语言。

分析　问题归结为由识别符号 S 出发，将推出一些什么样的句子，也就是说，$L(G[S])$是由一些什么样的符号串所组成的集合，找出其中的规律，用式子或自然语言描述出来。

此处应用第二个规则 $n-1$ 次，然后再应用第一个规则 1 次，有

$$S\Rightarrow 0S1\Rightarrow 00S11\Rightarrow 000S111\cdots\Rightarrow 0^{n-1}S1^{n-1}\Rightarrow 0^n1^n$$

即 $S\overset{+}{\Rightarrow}0^n1^n$，可见，此文法定义的语言为

$$L(G[S])=\{0^n1^n \mid n\geqslant 1\}$$

【例 2.8】 设有文法 $G[S]$：

$$S\to 0S \mid 1S \mid \varepsilon$$

求该文法所定义的语言。

由该文法所确定的语言为

$$L(G[S])=\{\varepsilon,0,1,00,01,10,11,\cdots\}=\{x \mid x\in\{0,1\}^{*}\}$$

【例 2.9】 设有文法 $G[A]$：

$$A\to yB,B\to xB \mid x$$

求该文法所定义的语言。

分析　从文法开始符号 A 出发可推导出以 y 开头后面跟一个或多个 x 结尾的符号串，所以该文法定义的语言为 $L(G[A])=\{yx^n \mid n\geqslant 1\}$。

由此可见，从已知文法确定语言的中心思想是：从文法的开始符号出发，反复连续地使用规则，对非终结符施行替换和展开，找出句子的规律，用式子或自然语言描述出来。

形式语言理论可以证明如下两点：

（1）给定一个文法，就能从结构上唯一地确定其语言，即 $G\to L(G)$。

（2）给定一种语言，能确定其文法，但这种文法不是唯一的，即 $L\to G_1$ 或 G_2 或…或 G_n。

对此我们不予证明，但已通过前面的举例说明了这两点。

2.3.4　规范推导和规范归约

从前面所讲的内容可以看到，文法和语言是密切相关的，文法所定义的任一句型和句

子，都可以根据文法推导出来，但同一个句型（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与所选择非终结符的次序无关。

例如，设有文法 $G[N_1]$：

$$N_1 \to N$$
$$N \to ND \mid D$$
$$D \to 0 \mid 1 \mid 2$$

该文法所定义的语言是由数字 0，1，2 组成的所有无符号整数。符号串 12 是该文法的一个句子，该句子可以通过下列 3 个不同的推导序列推导出来：

① $N_1 \Rightarrow N \Rightarrow ND \Rightarrow N2 \Rightarrow D2 \Rightarrow 12$

② $N_1 \Rightarrow N \Rightarrow ND \Rightarrow DD \Rightarrow 1D \Rightarrow 12$

③ $N_1 \Rightarrow N \Rightarrow ND \Rightarrow DD \Rightarrow D2 \Rightarrow 12$

为了使句型或句子能按一种确定的推导序列来产生，以便对句子的结构进行确定性的分析，通常我们只考虑两种特殊推导，即最右推导和最左推导，下面给出最左（最右）推导定义。

所谓最左（最右）推导，是指对于一个推导序列中的每一步直接推导 $\alpha \Rightarrow \beta$，都是对 α 中的最左（最右）非终结符进行替换。例如，在上面的 3 个推导序列中，①是最右推导；②是最左推导；③既不是最左推导，也不是最右推导。

最右推导也称为规范推导，用规范推导推导出的句型称为规范句型。

规范推导的逆过程，称为最左归约，也称为规范归约。

事实上，归约是与推导相对的概念，推导是把句型中的非终结符用规则的一个右部来替换的过程，而归约则是把句型中的某个子串用一个非终结符来替换的过程。

若用 $\dot{\Rightarrow}$ 表示归约，设 $A \to \alpha$ 是文法 G 中的一个规则，则有

$$xAy \Rightarrow x\alpha y$$
$$x\alpha y \dot{\Rightarrow} xAy$$

例如，文法 $G[N_1]$ 中有规范推导

$$N_1 \Rightarrow N \Rightarrow ND \Rightarrow N2 \Rightarrow D2 \Rightarrow 12$$

则有规范归约

$$12 \dot{\Rightarrow} D2 \dot{\Rightarrow} N2 \dot{\Rightarrow} ND \dot{\Rightarrow} N \dot{\Rightarrow} N_1$$

【例 2.10】 设有文法 $G[S]$：

$$S \to AB$$
$$A \to A0 \mid 1B$$
$$B \to 0 \mid S1$$

请给出句子 101001 的最左、最右推导。

分析　最右推导是指在推导过程中任何一步 $\alpha \Rightarrow \beta$（α 和 β 是句型），都是对 α 中的最右非终结符进行替换。句子 101001 的最右推导为

$$S \Rightarrow AB \Rightarrow AS1 \Rightarrow AAB1 \Rightarrow AA01 \Rightarrow A1B01 \Rightarrow A1001 \Rightarrow 1B1001 \Rightarrow 101001$$

最左推导是指在推导过程中任何一步 $\alpha \Rightarrow \beta$，都是对 α 中的最左非终结符进行替换。句子 101001 的最左推导为

$$S \Rightarrow AB \Rightarrow 1BB \Rightarrow 10B \Rightarrow 10S1 \Rightarrow 10AB1 \Rightarrow 101BB1 \Rightarrow 1010B1 \Rightarrow 101001$$

从例 2.10 中可以看出，在规范推导（最右推导）中，每步直接推导 $xAy \Rightarrow x\alpha y$ 中的符号串 y 只含终结符。

2.3.5　递归规则与文法的递归性

递归的概念在编译技术中是一个很重要的概念。

1. 递归规则

所谓递归规则，是指在规则的左部和右部具有相同非终结符的规则。

如果文法中有规则 $A \rightarrow A\cdots$ 称为规则左递归。

如果文法中有规则 $A \rightarrow \cdots A$ 称为规则右递归。

如果文法中有规则 $A \rightarrow \cdots A \cdots$ 称为规则递归。

2. 文法的递归性

文法的递归性是指对文法中任一非终结符，若能建立一个推导过程，在推导所得的符号串中又出现了该非终结符本身，则文法是递归性的，否则是无递归性的。

若文法中有推导 $A \overset{+}{\Rightarrow} A\cdots$ 称文法左递归。

若文法中有推导 $A \overset{+}{\Rightarrow} \cdots A$ 称文法右递归。

若文法中有推导 $A \overset{+}{\Rightarrow} \cdots A \cdots$ 称文法递归。

例如，文法中有如下规则：

$$U \rightarrow Vx$$
$$V \rightarrow Uy \mid z$$

这三条规则都不是递归规则，但有 $U \overset{+}{\Rightarrow} Uyx$，则该文法是左递归的。

在文法中使用递归规则，使得我们能用有限的规则去定义无穷集合的语言。

【例 2.11】 考虑文法 $G[A]$：

$$A \rightarrow aB \mid bB$$
$$B \rightarrow a \mid b$$

由于该文法无递归性，由它所描述的语言是有穷的。该文法描述的语言为 $L(G[A]) = \{aa, ab, ba, bb\}$。

【例 2.12】 考虑文法 $G[N_1]$：

$$N_1 \rightarrow N$$
$$N \rightarrow ND \mid D$$
$$D \rightarrow 0 \mid 1 \mid 2$$

该文法有直接左递归规则 $N \rightarrow ND$，则称该文法为左递归文法或称文法左递归，其定义的语言为 $\{0,1,2\}^+$。

由于文法中使用了递归规则，使得我们可用有限的规则去刻画无穷集合的语言。若不用递归规则来定义文法，要表示无穷集合的语言需要用无穷多条规则。例如，例 2.12 中若不用递归规则 $N \rightarrow ND$，则需要用 $N \rightarrow D \mid DD \mid DDD \mid \cdots$ 即无穷多条规则来定义由数字 0，1，2 组成的所有无符号整数。

也就是说，当一个语言是无穷集合时，则定义该语言的文法一定是递归的。

需要指出的是，程序设计语言都是无穷集合，因此描述它们的文法必定是递归的。

2.4 短语、直接短语和句柄

2.4.1 短语和直接短语

令 G 是一个文法，S 是文法的开始符号，假定 $\alpha\beta\delta$ 是文法 G 的一个句型，如果有

$$S\stackrel{*}{\Rightarrow}\alpha A\delta \text{ 且 } A\stackrel{+}{\Rightarrow}\beta$$

则称 β 是相对于非终结符 A 的句型 $\alpha\beta\delta$ 的短语。特别是如果有

$$S\stackrel{*}{\Rightarrow}\alpha A\delta \text{ 且 } A\Rightarrow\beta$$

则称 β 是直接短语。

注意体会短语这个概念的定义，仅有 $A\stackrel{+}{\Rightarrow}\beta$，不一定意味着 β 就是句型 $\alpha\beta\delta$ 的一个短语，因为，还需要有 $S\stackrel{*}{\Rightarrow}\alpha A\delta$ 这一个条件。

例如，考虑文法 $G[N_1]$：

$$N_1\to N$$
$$N\to ND|D$$
$$D\to 0|1|2$$

对句型 ND，尽管有 $N_1\stackrel{+}{\Rightarrow}N$，但 N 不是该句型的一个短语，因为不存在从文法的开始符号 N_1 到 N_1D 的推导。事实上，句型 ND 的短语是 ND 自身。

需要指出的是，短语和直接短语的区别在于第二个条件，直接短语中的第二个条件表示有文法规则 $A\to\beta$，因此，每个直接短语都是某规则右部。

2.4.2 句柄

一个句型的最左直接短语称为该句型的句柄。

句柄特征：

（1）它是直接短语，即某规则右部。

（2）它具有最左性。

注意，短语、直接短语和句柄都是针对某一句型的，特指句型中的哪些符号子串能构成短语和直接短语，离开具体的句型来谈短语、直接短语和句柄是无意义的。

【例2.13】 设有文法

$$G[S]=(\{S,A,B\},\{a,b\},P,S)$$

其中，P 为

$$S\to AB$$
$$A\to Aa|bB$$
$$B\to a|Sb$$

求句型 $baSb$ 的全部短语、直接短语和句柄。

分析 根据短语定义，可以从句型的推导过程中找出其全部短语、直接短语和句柄。

对文法，首先建立该句型的推导过程：

$S\Rightarrow AB\Rightarrow bBB\Rightarrow baB\Rightarrow baSb$ （最左推导）

$S\Rightarrow AB\Rightarrow ASb\Rightarrow bBSb\Rightarrow baSb$ （最右推导）

从这两个推导过程中，有

① $S\overset{*}{\Rightarrow}S$

$S\overset{+}{\Rightarrow}baSb$　　句型本身是（相对于非终结符 S）句型 $baSb$ 的短语。

② $S\overset{*}{\Rightarrow}baB$

$B\Rightarrow Sb$　　句型 $baSb$ 中的子串 Sb，是（相对于非终结符 B）句型 $baSb$ 的短语，且为直接短语。

③ $S\overset{*}{\Rightarrow}bBSb$

$B\Rightarrow a$　　句型 $baSb$ 中的子串 a，是（相对于非终结符 B）句型 $baSb$ 的短语，且为直接短语、句柄。

④ $S\overset{*}{\Rightarrow}ASb$

$A\overset{+}{\Rightarrow}ba$　　句型 $baSb$ 中的子串 ba，是（相对于非终结符 A）句型 $baSb$ 的短语。

对于此句型，再没有其他能产生新的短语的推导了。

可见，根据定义求句型的短语、直接短语和句柄比较麻烦、难求。下一节通过对语法树的介绍可以看到，用语法树来求句型的短语、直接短语和句柄是非常直观和简单的。

2.5　语法树与文法的二义性

2.5.1　推导和语法树

1. 语法树的生成

对句型的推导过程给出一种图形表示，这种表示称为语法树，也称推导树。设文法 $G=(V_N,V_T,P,S)$，对 G 的任何句型都能构造与之关联的、满足下列条件的一棵语法树。

（1）每个结点都有一个标记，此标记是 $V=V_N\cup V_T\cup\{\varepsilon\}$ 中的一个符号。

（2）树根的标记是文法的开始符号 S。

（3）若某一结点至少有一个分支结点，则该结点上的标记一定是非终结符。

（4）若 A 的结点有 k 个分支结点，其分支结点的标记分别为 A_1，A_2，…，A_k，则 $A\rightarrow A_1A_2\cdots A_k$ 一定是 G 的一条规则。

下面用一个例子来说明根据句型的推导构造语法树的过程。

例如，设有文法 $G[E]$：

$$E\rightarrow E+T\mid E-T\mid T$$

$$T\rightarrow T*F\mid T/F\mid F$$

$$F\rightarrow (E)\mid i$$

根据推导，画出句型$(T+i)*i-F$ 的语法树。

首先给出句型的推导过程（最右推导）：

$$\begin{aligned}E &\Rightarrow E-T\Rightarrow E-F\Rightarrow T-F\Rightarrow T*F-F\Rightarrow T*i-F\\&\Rightarrow F*i-F\Rightarrow (E)*i-F\Rightarrow (E+T)*i-F\\&\Rightarrow (E+F)*i-F\Rightarrow (E+i)*i-F\\&\Rightarrow (T+i)*i-F\end{aligned}$$

推导构造语法树的过程是：

以识别符号作为根结点，从它开始对每一步直接推导向下画一分支，分支结点的标记是直接推导中被替换的非终结符的名字，按此方法逐步向下，画出每一步直接推导对应的分支直到对该语法树再无分支可画出时，构造过程结束。构造句型（$T+i$）$*i-F$ 语法树的过程，见图2.2。

语法树中从左到右的末端结点构成了由该语法树所表示的那个推导出的符号串，如上例（$T+i$）$*i-F$。所谓末端结点，是指没有分支向下射出的结点，常称之为树叶。如果末端结点都是由终结符组成，则这些结点所组成的符号串为句子，否则为句型。

由上例可知，语法树的构造过程是从文法的开始符号出发，构造一个推导的过程，因为文法的每一个句型（句子）都存在一个推导，所以文法的每个句型（句子）都有一棵对应的语法树。

对句型$(T+i)*i-F$，还可给出最左推导：

$$
\begin{aligned}
E &\Rightarrow E-T \Rightarrow T-T \Rightarrow T*F-T \Rightarrow F*F-T \\
&\Rightarrow (E)*F-T \Rightarrow (E+T)*F-T \Rightarrow (T+T)*F-T \\
&\Rightarrow (T+F)*F-T \Rightarrow (T+i)*F-T \\
&\Rightarrow (T+i)*i-T \Rightarrow (T+i)*i-F
\end{aligned}
$$

不难看出，根据该推导得到的语法树，仍然是图2.2。可见对句型（$T+i$）$*i-F$ 的两种不同推导构造的语法树完全相同，也就是说，一棵语法树表示了一个句型的种种可能的（但未必是所有的）不同推导过程，包括最左（最右）推导。为方便找到句型短语和句柄，我们需引入子树和简单子树的概念。

2. 子树

语法树的子树是由某一非末端结点连同所有分支组成的部分。例如，图2.3是图2.2的子树。

3. 简单子树

语法树的简单子树是指只有单层分支的子树。例如，图2.4是图2.2的简单子树。

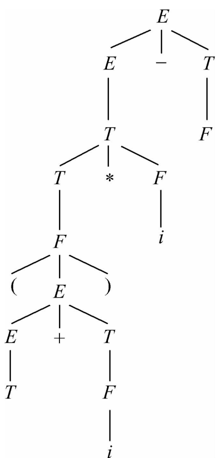

图2.2　语法树

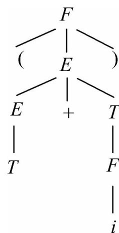

图2.3　子树

图2.4　简单子树

子树与短语的关系十分密切，根据子树的概念，句型的短语、直接短语和句柄的直观解释如下：

- 短语——子树的末端结点形成的符号串是相对于子树根的短语。
- 直接短语——简单子树的末端结点形成的符号串是相对于简单子树根的直接短语。
- 句柄——最左简单子树的末端结点形成的符号串是句柄。

例如，对前例文法 $G[E]$，用语法树求句型 $(T+i)*i-F$ 的短语、直接短语和句柄。

首先画出该句型的语法树，如图 2.2 所示，由语法树可知：

$(T+i)*i-F$ 为句型的相对于 E 的短语；

$(T+i)*i$ 为句型的相对于 T 的短语；

$(T+i)$ 为句型的相对于 F 的短语；

$T+i$ 为句型的相对于 E 的短语；

T 为句型的相对于 E 的短语，且为直接短语；

第一个 i 为句型的相对于 F 的短语，且为直接短语；

第二个 i 为句型的相对于 F 的短语，且为直接短语；

F 为句型的相对于 T 的短语，且为直接短语；

在 4 个直接短语中，T 为句柄。

【例 2.14】 对例 2.13 中的文法，可用语法树非常直观地求出句型 $baSb$ 的全部短语、直接短语和句柄。

分析　首先根据句型 $baSb$ 的推导过程画出对应的语法树，见图 2.5。

由语法树可知：

$baSb$ 为句型的相对于 S 的短语；

ba 为句型的相对于 A 的短语；

a 为句型的相对于 B 的短语，且为直接短语和句柄；

Sb 为句型的相对于 B 的短语，且为直接短语。

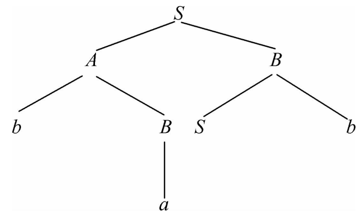

图 2.5　句型 $baSb$ 的语法树

2.5.2　文法的二义性

从前面的讨论可以看出，对于文法 G 中任一句型的推导序列，我们总能为它构造一棵语法树，这就是说，不同的推导序列，却对应着相同的语法树，那么文法的句型是否只对应唯一的一棵语法树呢？也就是说，它是否只有唯一的一个最左（最右）推导呢？回答是不尽然。

例如，设有文法 $G[E]$：

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

句子 $i*i+i$ 有两个不同的最左推导，对应两棵不同的语法树，如图 2.6 和图 2.7 所示。

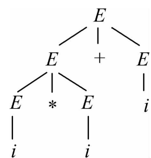

图 2.6　最左推导 1 的语法树

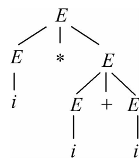

图 2.7　最左推导 2 的语法树

最左推导1 $E \Rightarrow E+E \Rightarrow E*E+E$ 最左推导2 $E \Rightarrow E*E \Rightarrow i*E$

$\Rightarrow i*E+E$ $\Rightarrow i*E+E$

$\Rightarrow i*i+E$ $\Rightarrow i*i+E$

$\Rightarrow i*i+i$ $\Rightarrow i*i+i$

图2.6的语法树先做乘法，而图2.7的语法树先做加法，到底先做何种运算呢？此处出现的这种现象称为文法的二义性。

如果一个文法存在某个句子对应两棵不同的语法树，则说这个文法是二义性的。或者说，若一个文法中存在某个句子，它有两个不同的最左（最右）推导，则这个文法是二义性的。

上述文法 $G[E]$ 就是一个二义性的文法。

显然，二义性的文法将给编译程序的执行带来问题。对于二义性文法的句子，当编译程序对它的结构进行语法分析时，就会产生两种甚至更多种不同的理解。由于语法结构上的不确定性，将必然会导致语义处理上的不确定性。例如在上例中，当编译程序分析句子 $i*i+i$ 时，是按图2.6去进行分析，还是按图2.7去进行分析？为使编译程序对每个语句的分析是唯一的，希望描述语言的文法是无二义性的。当然，对于二义性文法，我们可以利用文法的等价性来消除文法的二义性。

2.5.3 文法二义性的消除

（1）不改变文法中原有的语法规则，仅加进一些语法的非形式规定。

例如，对于上例文法 $G[E]$，不改变已有的4条规则，仅加进运算符的优先顺序和结合规则，即 * 优先于 +； +、* 服从左结合。这样，对于文法 $G[E]$ 中的句子 $i*i+i$ 只有唯一的一棵语法树（见图2.6），从而避免了文法的二义性。

（2）构造一个等价的无二义性文法，即把排除二义性的规则合并到原有文法中，改写原有的文法。

例如，对于上例文法 $G[E]$，将运算符的优先顺序和结合规则（ * 优先于 +； +、* 左结合）加到原有文法中，可构造出无二义性文法 $G'[E]$ 如下：

$$E \to E+T \mid T$$
$$T \to T*F \mid F$$
$$F \to (E) \mid i$$

则句子 $i*i+i$ 只有唯一一棵语法树，见图2.8。

可见，改写后的文法与原文法等价且为无二义性文法。此例告诉我们，对于由二义性文法描述的语言，有时可以找到等价的无二义性文法来描述它。

【例2.15】 定义某程序语言条件语句的文法 G 为

$S \to$ if b S
$\mid$ if b S else S
$\mid A$ （其他语句）

试证明该文法是二义性的，并消除它的二义性。

分析 该文法的句子 if b if b A else A 对应图2.9中两棵不同的语法树。所以该文法是二义性的。消除文法的二义性可采用下面两种方法。

（1）不改变已有规则，仅加进一项非形式的语法规定：else 与前面最接近的不带 else 的 if 相对应。这样，文法 G 的句子 if b if b A else A 只对应唯一的一棵语法树（见图 2.9（a）），由此消除了二义性。

（2）改写文法 G 为 G'：

$S \to S_1 | S_2$

$S_1 \to$ if b S_1 else $S_1 | A$

$S_2 \to$ if b $S |$ if b S_1 else S_2

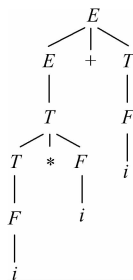

图 2.8　G'的句子 $i*i+i$ 的语法树

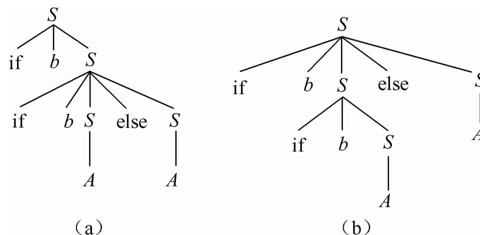

图 2.9　复合 if 语句的两棵语法树

这是因为，通过分析得知引起二义性的原因是 if-else 语句的 if 后可以是 if 型，因此改写文法时规定 if 和 else 之间只能是 if-else 语句或其他语句。这样，对改写后的文法，句子 if b if b A else A 只对应唯一的一棵语法树，如图 2.10 所示。

应该指出的是，文法的二义性和语言的二义性是两个不同的概念。通常我们只说文法的二义性，而不说语言的二义性，这是因为可能有两个不同的文法 G 和 G'，而且其中一个是二义性的，另一个是无二义性的，但却有 $L(G)=L(G')$，即这两个文法所产生的语言是相同的。而将一个语言说成是二义性的，是指对它不存在无二义性的文法，这样的语言称为先天二义性的语言，例如 $L=\{a^i b^j c^k | i=j$ 或 $j=k, i,j,k \geq 1\}$ 便是这种语言。

人们已经证明，不存在一个算法，它能在有限步骤内确切地判定任给的一个上下文无关文法是否为二义性文法，或它是否产生一个先天二义性的上下文无关语言。

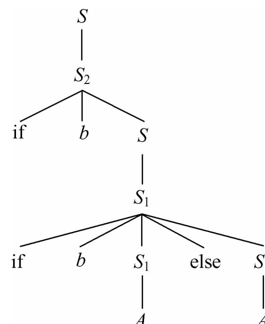

图 2.10　G'的复合 if 语句的语法树

2.6　文法和语言的分类

文法用于生成语言，不同的文法生成不同的语言。

著名的语言学家乔姆斯基（Chomsky）将文法和语言分为 4 大类，即 0 型、1 型、2 型和 3 型。划分的依据是对文法中的规则施加不同的限制。

1. 0 型文法（无限制文法）

若文法 $G=(V_N, V_T, P, S)$ 中的每条规则 $\alpha \to \beta$ 是这样一种结构：

$\alpha \in (V_N \cup V_T)^*$ 且至少含一个非终结符，而 $\beta \in (V_N \cup V_T)^*$，

则称 G 是 0 型文法。0 型文法描述的语言是 0 型语言。

由定义可见，α 和 β 均是文法的终结符和非终结符组成的符号串，且 β 可能为空，而 α 不等于空，即允许 $|\alpha|>|\beta|$。由于 0 型文法没有加任何限制条件，故又称为无限制性文法，相应的语言称为无限制性语言。

例如，有 0 型文法 $G=(V_N,V_T,P,S)$，其中

$$
\begin{aligned}
V_N &= \{A,B,S\} \\
V_T &= \{0,1\} \\
P &= \{S\to 0AB \\
&\quad 1B\to 0 \\
&\quad B\to SA|01 \\
&\quad A1\to SB1 \\
&\quad A0\to S0B\}
\end{aligned}
$$

其描述的 0 型语言为 $L_0(G[S])=\{\}$。

2. 1 型文法（上下文有关文法）

若文法 $G=(V_N,V_T,P,S)$ 中的每一条规则的形式为 $\alpha A\beta\to\alpha u\beta$，其中 $A\in V_N$，α，$\beta\in(V_N\cup V_T)^*$，$u\in(V_N\cup V_T)^+$，则称 G 是 1 型文法，1 型文法描述的语言是 1 型语言。

由定义可见，利用规则将 A 替换成 u 时，则必须考虑非终结符 A 只有在 α 和 β 这样的一个上下文环境中才可以把它替换为 u，并且不允许替换成空串，也就是 $|\alpha A\beta|\leqslant|\alpha u\beta|$，故又称 1 型文法为上下文有关文法，相应的语言又称为上下文有关语言。

例如，有 1 型文法 $G=(V_N,V_T,P,S)$，其中

$$
\begin{aligned}
V_N &= \{S,A,B\} \\
V_T &= \{a,b,c\} \\
P &= \{S\to aSAB|abB \\
&\quad BA\to BA' \\
&\quad BA'\to AA' \\
&\quad AA'\to AB \\
&\quad bA\to bb \\
&\quad bB\to bc \\
&\quad cB\to cc\}
\end{aligned}
$$

其描述的 1 型语言为 $L_1(G[S])=\{a^nb^nc^n|n\geqslant 1\}$。

3. 2 型文法（上下文无关文法）

若文法 $G=(V_N,V_T,P,S)$ 中的每一条规则的形式为 $A\to\beta$，其中 $A\in V_N$，$\beta\in(V_N\cup V_T)^*$，则称 G 是 2 型文法，2 型文法描述的语言是 2 型语言。

由定义可见，利用规则将 A 替换成 β 时，与 A 的上下文无关，即无需考虑 A 在上下文中出现的情况，故又称 2 型文法是上下文无关文法，其产生的语言又称为上下文无关语言。

通常定义程序设计语言的文法是上下文无关文法，因此，上下文无关文法及相应语言是我们主要研究的对象。

例如，有 2 型文法 $G=(V_N,V_T,P,S)$，其中

$$
\begin{aligned}
V_N &= \{S,A,B\} \\
V_T &= \{a,b\}
\end{aligned}
$$

$$P=\{S\to aB\mid bA$$
$$A\to a\mid aS\mid bAA$$
$$B\to b\mid bS\mid aBB\}$$

其描述的语言为 $L_2(G[S])=\{x\mid x\in\{a,b\}^+$ 且 x 中 a 和 b 的个数相同$\}$。

4. 3 型文法（正规文法）

若文法 $G=(V_N,V_T,P,S)$ 中的每一条规则的形式为 $A\to\alpha B$ 或 $A\to\alpha$，其中 A，$B\in V_N$，$\alpha\in V_T^*$，则称 G 是右线性文法。

若文法 $G=(V_N,V_T,P,S)$ 中的每一条规则的形式为 $A\to B\alpha$ 或 $A\to\alpha$，其中 $A,B\in V_N$，$\alpha\in V_T^*$，则称 G 是左线性文法。

右线性文法和左线性文法都称为 3 型文法或正规文法，3 型文法描述的语言称为 3 型语言或正规语言。

通常定义程序设计语言词法规则的文法是正规文法。

例如，用左线性正规文法和右线性正规文法定义标识符。用 I 代表标识符，l 代表任意一个字母，d 代表任意一个数字，则定义标识符的文法为

左线性文法	右线性文法
$P:I\to l\mid Il\mid Id$	$P:I\to l\mid lT$
	$T\to l\mid d\mid lT\mid dT$

例如，用左线性正规文法和右线性正规文法定义无符号整数。

用 N 代表无符号整数，d 代表任意一个数字，则定义的无符号整数文法为

左线性文法	右线性文法
$P:N\to d\mid Nd$	$P:N\to d\mid dN$

由上述 4 类文法的定义可知，从 0 型文法到 3 型文法，是逐渐增加对规则的限制条件而得到的，因此每一种正规文法都是上下文无关的文法，每一种上下文无关的文法都是上下文有关的文法，而每一种上下文有关的文法都是 0 型文法，而由它们所定义的语言类是依次缩小的，即有 $L_0\supset L_1\supset L_2\supset L_3$。

2.7　有关文法的实用限制和变换

文法是用来描述程序设计语言的，在实际应用中需要对文法加一些限制条件。对文法的实用限制有以下两点：

（1）文法中不能含有形如 $A\to A$ 的规则，这种规则称为有害规则。这样的规则对描述语言显然是没有必要的，并且它还会引起文法的二义性。所以在设计文法时，应该避免定义这样的规则。

（2）文法中不能有多余规则。所谓多余规则是指文法中出现以下两种规则的情况。一是某条规则 $A\to\alpha$ 的左部符号 A 不在所属文法的任何其他规则右部出现，即在推导文法的所有句子中始终都不可能用到的规则；二是对文法中的某个非终结符 A，无法从它推导出任何终结符号串来。

例如，设有文法 $G[S]$：

$$S\to Bd$$

$$A \to Ad \mid d$$
$$B \to Cd \mid Ae$$
$$C \to Ce$$
$$D \to e$$

在该文法中，因为非终结符 D 不在任何规则的右部出现，所以在句子的推导中始终不可能用到它，因此规则 $D \to e$ 为多余规则，应该删除。

又因为非终结符 C 推导不出终结符号串，所以规则 $C \to Ce$ 和规则 $B \to Cd$ 为多余规则，也应该删除。

删除多余规则后的文法变换为

$$S \to Bd$$
$$B \to Ae$$
$$A \to Ad \mid d$$

另外，即使有些规则的左部非终结符出现在了其他规则的右部，但如果在推导文法句子时永远用不到它，这样的规则也是多余规则。例如下面的文法 $G[S]$：

$$S \to Aa$$
$$A \to Aa \mid d$$
$$B \to a \mid Ca$$
$$C \to d \mid Bd$$

该文法中，虽然 B，C 都出现在了其他规则的右部，但由于在推导文法句子时，永远无法用到这两个非终结符，所以它们对应的规则 $B \to a \mid Ca$ 及 $C \to d \mid Bd$ 都是多余的，也应该删除。

若程序设计语言的文法含有多余规则，其中必定有错误存在，因此检查文法中是否含有多余规则是很重要的。

本章小结

本章重点介绍了语言的语法结构的形式描述、语法树以及文法的二义性，主要内容有：

1. 设计一个文法，定义一个已知的语言

（1）文法是一个四元组 $G=(V_N,V_T,P,S)$，文法四大要素中，关键是一组规则，它定义（或描述）了一个语言的结构。从文法定义可知，对于程序设计者来说，文法给出了语言的精确定义和描述；对于编译程序的开发者而言，文法是进行正确编译的准则；对于语言的使用者而言，文法是进行程序设计的依据。

（2）分析已知语言句子的结构特征，设计出相应的一组规则，但不唯一。

（3）设计的文法必须能定义已知的语言，不能扩大或缩小所定义语言的范围。

（4）若语言是无穷集合，设计该语言的文法一定是递归的。

2. 已知一个文法，确定该文法所定义的语言

（1）文法所定义的语言 $L(G[S])=\{x \mid S \stackrel{+}{\Rightarrow} x \text{ 且 } x \in V_T^{\,*}\}$。

（2）给定一个文法，可根据语言和推导定义推导出文法的句子，从而确定出该文法所定义的语言。