可计算性、计算复杂性与算法设计思路

吴哲辉　编著

中国石油大学出版社

可计算性、计算复杂性与算法设计思路

吴哲辉 编著

中国石油大学出版社

图书在版编目（CIP）数据

可计算性、计算复杂性与算法设计思路 / 吴哲辉编著.
东营：中国石油大学出版社，2009.9
ISBN 978-7-5636-2910-7

Ⅰ.可… Ⅱ.吴… Ⅲ.①可计算性－研究生－教材②计算复杂性－研究生－教材③电子计算机－算法设计－研究生－教材 Ⅳ.TP301

中国版本图书馆 CIP 数据核字（2009）第 164774 号

书　　名：可计算性、计算复杂性与算法设计思路
编　　著：吴哲辉

责任编辑：刘　静
封面设计：刘泽延

出 版 者：中国石油大学出版社（山东 东营，邮编 257061）
网　　址：http://www.uppbook.com.cn
电子邮箱：cbs2006@163.com
印 刷 者：青岛锦华信包装有限公司
发 行 者：中国石油大学出版社（电话 0546-8391810）
开　　本：185×260　印张：12　字数：229 千字
版　　次：2009 年 9 月第 1 版第 1 次印刷
定　　价：23.80 元

前　言

可计算性、计算复杂性和算法是关于计算的三个不同侧面的问题。这里所说的“计算”，不仅指数值计算。对问题的正确性判定是一种计算；信息加工也是一种计算；……总之，电子计算机所能做的各种工作，都可以归入计算的范畴。因此，可计算性、计算复杂性和算法，组成了计算机及科学理论的核心内容，是计算机科学理论的基础。

可计算性问题是20世纪30年代提出的。在此之前，人们（包括科学家）的意识中尚未有这方面的概念。哥德尔不完全性定理的提出，才使人们知道还存在着不可计算的问题。于是，用于衡量一个问题是否可计算的各种计算模型应运而生，最有代表性并得到公认的当属递归函数、λ-演算和图灵机。其中，图灵机不仅可以作为可计算性的衡量基准，而且也是现代电子计算机的理论模型，为现代电子计算机的诞生奠定了理论基础。可计算性理论也随着这些计算模型的被公认逐渐发展为一个学科分支。

类似于算法设计的一些思想方法，早在古希腊时代就有所显现。譬如，求两个数的最大公约数的欧几里德辗转相除法，今天来看也是一个高效的算法设计思想方法。然而，算法作为一个学科分支，是在现代电子计算机问世以后才形成的。因为对于许多复杂的计算问题来说，只有高速运行的电子计算机才能把算法设计的思想方法转化为求解结果。算法作为一个学科分支，不仅包含算法设计，而且包含算法分析。算法分析是指算法的正确性证明和算法的复杂性分析两方面的内容，而算法复杂性分析又可分为算法的时间复杂性分析和空间复杂性分析。

计算复杂性和算法复杂性是两个不同的概念，它们之间又有密切的联系。算法复杂性分析是指对一个（对某类问题）已设计出来的算法，分析其所需的时间量和空间量（它们都是输入量的函数）；而计算复杂性则是指求解该类问题所需的最低限度的时间量和空间量，同具体的算法无关。诚然，对于一类问题所设计出来的任何算法，其时间复杂性和空间复杂性都不可能低于该类问题的计算复杂性。NP-完全问题是计算复杂性理论中最重要的内容之一，这个问题直接联系着当今计算科学和计算机科学理论中最瞩目的一个理论问题：$NP \stackrel{?}{=} P$。

要把三个分支的内容在一本书中撰述，就只能写出其中最基本的部分。本书就是遵循这样一个宗旨写成的。而且，对于算法这一分支，书中只是对几种成熟的算法设计思路进行阐述，并通过具体实例对这些思路作一些注释。关于算法的复杂性分析，书中基本上没有涉及。一方面是篇幅的限制，另外也考虑到许多算法设计与分析的书中，相关的阐述都比较详细。

本书是根据作者为计算机软件与理论专业博士生讲授的一门课程的讲稿整理而写成的，绝大多数内容都来自所列的参考文献。此外，也写入一些作者的研究成果。山东科技大学吴振寰老师、张峰同学和王鹏伟同学为本书的录入和排版付出了艰辛的劳动，在此一并表示感谢。

由于水平所限，书中难免有错误和不当之处，恳请专家和读者指正。

吴哲辉

2009年3月

目 录

第1章　引　论

本书最核心的关键词是“计算”，书中所涉及的三方面内容都是围绕计算而展开的。说到计算，人们首先联想到的可能是数值计算，包括四则运算、初等函数计算以及由它们组合而产生的组合函数计算等。其实，计算的含义要广泛得多，逻辑运算、信息处理等也是计算，现代电子计算机所能做的各种工作都属于计算的范畴。

不过另一方面，我们又可以把所有计算问题都归结为以自然数集为定义域的函数计算来讨论。本章的内容就是为了说明这一点而组织安排的。

1.1　数论函数与数论谓词

函数是数学中的基础概念，函数值的计算也是最基本的计算问题。函数概念包括定义域X，值域Y和对应关系f三个要素，其中X和Y都是集合（可以是有限集或无限集），f是从集合X到集合Y的一个映射。记为

$$f:\ X\to Y$$

多元函数是函数概念的自然拓展，这时定义域是若干个集合的笛卡尔积，即

$$f:\ X_1\times X_2\times\cdots\times X_n\to Y$$

在数学基础理论的函数定义中，强调对定义域X（或$X_1\times X_2\times\cdots\times X_n$）中的每个元素$x$（或$(x_1,x_2,\cdots,x_n)$），值域$Y$中都有一个元素与之对应，即

$$\forall x\in X,\ \exists y\in Y:\ f(x)=y$$

或

$$\forall(x_1,x_2,\cdots,x_n)\in X_1\times X_2\times\cdots\times X_n,\ \exists y\in Y:\ f(x_1,x_2,\cdots,x_n)=y$$

在许多情况下，还要求y中的对应元素是唯一的，即

$$\forall x\in X,\ \exists!y\in Y:\ f(x)=y$$

或

$$\forall(x_1,x_2,\cdots,x_n)\in X_1\times X_2\times\cdots\times X_n,\ \exists!y\in Y:\ f(x_1,x_2,\cdots,x_n)=y$$

式中的符号“$\exists!$”表示“存在唯一的一个”。加上这个条件的函数称为单值函数，否则称为多值函数。

在讨论计算问题时，需要对函数的概念作进一步的拓展，或者说作一点修改。修改（拓展）之处在于不要求对定义域中的每个元素，值域中都必须有一个（或多个）元素与之对应。即对某些$x\in X$（或$(x_1,x_2,\cdots,x_n)\in X_1\times X_2\times\cdots\times X_n$），$f(x)$（或$f(x_1,x_2,\cdots,x_n)$）可以没有定义。这时记为$f(x)=\varnothing$（或$f(x_1,x_2,\cdots,x_n)=\varnothing$）。经过这种拓展的函数$f(x)$（或$f(x_1,x_2,\cdots,x_n)$）称为定义在集合$X$（或$X_1\times X_2\times\cdots\times X_n$）上的部分函数。

诚然，部分函数也可以分为（部分）单值函数和（部分）多值函数。我们约定，当没有加上“多值”的定语时，部分函数就是指（部分）单值函数。

数论函数是那些以自然数集 $\mathbf{N}=\{0,1,2,\cdots\}$ 或若干个 $\mathbf{N}$ 的笛卡尔积为定义域，并以 $\mathbf{N}\cup\{\varnothing\}$ 为值域的部分函数，其中，处处有定义的数论函数称为全函数。

定义 1.1 设 $\mathbf{N}=\{0,1,2,\cdots\}$ 为自然数集。称

$$f:\ \mathbf{N}\times\mathbf{N}\times\cdots\times\mathbf{N}\to\mathbf{N}\cup\{\varnothing\} \tag{1.1}$$

为一个数论函数 (number-theoretic function)。特别地，如果对于任意 $(x_1,x_2,\cdots,x_n)\in\mathbf{N}\times\mathbf{N}\times\cdots\times\mathbf{N}$，都存在 $y\in\mathbf{N}$，使得

$$f(x_1,x_2,\cdots,x_n)=y$$

则称 f 是一个全函数 (total function)。也就是说，全函数是如下的一个映射

$$f:\ \mathbf{N}\times\mathbf{N}\times\cdots\times\mathbf{N}\to\mathbf{N} \tag{1.2}$$

显然，(1.2) 式是 (1.1) 式的一种特殊情况，即全函数是一种处处有定义的数论函数。 □

例 1.1 设

$$f_i:\ \mathbf{N}\times\mathbf{N}\to\mathbf{N}\cup\{\varnothing\},\qquad i=1,2$$
$$f_3:\ \mathbf{N}\to\mathbf{N}\cup\{\varnothing\}$$

其中

$$f_1(x,y)=x+y$$
$$f_2(x,y)=x-y$$
$$f_3(x)=\sqrt{x}$$

那么 f_1、f_2 和 f_3 都是数论函数。其中，f_1 是一个全函数，因为对任意 $x,y\in\mathbf{N}$，$f_1(x,y)$ 都有定义（即 $f_1(x,y)\in\mathbf{N}$）；但 f_2 和 f_3 不是全函数，因为对 $x,y\in\mathbf{N}$，当 $x<y$ 时，f_2 没有定义（即 $f_2(x,y)=\varnothing$）；对 $x\in\mathbf{N}$，当 $x\notin\{0,1,4,9,\cdots,k^2,\cdots\}$ 时，f_3 没有定义（$f_3(x)=\varnothing$）。

定义 1.2 值域为 $\{0,1\}\cup\{\varnothing\}$ 的数论函数称为谓词函数，即谓词函数是一个映射

$$P:\ \mathbf{N}\times\mathbf{N}\times\cdots\times\mathbf{N}\to\{0,1\}\cup\{\varnothing\} \tag{1.3}$$

□

从定义 1.2 看出，谓词函数类是数论函数类的一个子类。由于谓词函数的取值只有 0 和 1 两个（否则便无定义），正好与逻辑值 {真,假} 相对应，因此 P 实际上起到了逻辑谓词的作用，这就是把 P 称作谓词函数的缘故。

例如，若用 1 和 0 分别表示真、假两个逻辑值，则有 $P(x^2-3x+2=0)$ 是一个谓词函数。

$$P(x^2-3x+2=0)=\begin{cases}1, & x=1\text{ 或 }2\\ 0, & \text{当 }x\in\mathbf{N}-\{1,2\}\end{cases}$$

1.2 字符串、语言和文法

字符串是现代计算机最基本的处理对象，现代电子计算机上所做的每一件工作都可以归结为字符串处理。对于一类特定的问题，字符串中出现的字符都是有特定范围的，这个特定范围

就是字母表。

1.2.1 字母表与字符串

由字符组成的一个非空有限集称为一个字母表 (alphabet)，通常用 Σ 表示。字母表是讨论字符串和字符串集合的出发点。当我们讨论一个字符串或一个字符串集合时，总是假定它们是由某个字母表上的一些字符组成的。

例如，$\Sigma=\{0,1\}$ 是一个字母表，一个二进制数就是这个字母表上的一个字符串；$\Sigma=\{\mathrm{a},\mathrm{b},\mathrm{c},\cdots,\mathrm{y},\mathrm{z}\}$ 也是一个字母表，它由 26 个英文小写字母组成；全体 ASCII 字符的集合也是一个字母表。

一方面，字母表中至少要有一个字符，空集不能作为字母表；另一方面，字母表中的字符个数必须是有限的，无限集不能作为字母表。

字符串通常简称为串 (string)。串是指某个字母表上的字符组成的有限序列。例如，01001 是字母表 $\Sigma=\{0,1\}$ 上的一个串，000 也是这个字母表上的一个串。

一个串中字符的位数称为这个串的长度。例如，01001 的长度等于 5，000 的长度等于 3。若用 x 表示一个串，那么 $|x|$ 表示这个串的长度。例如，若 $x=01001$，那么 $|x|=5$。

长度为 0 的串称为空串，用 ε 表示。换句话说，空串是不含任何字符的串。显然，空串可以看作任一个字母表上的串。

定义 1.3 *设 x 为一个串，若将 x 中的字符按逆向顺序重新排列，所得到的串称为 x 的逆* (reversal)，*记为* x^{R}。 □

例如，若 $x=\mathrm{abacd}$，则 $x^{\mathrm{R}}=\mathrm{dcaba}$。

显然，$|x^{\mathrm{R}}|=|x|$。空串的逆还是空串，即 $\varepsilon^{\mathrm{R}}=\varepsilon$。

定义 1.4 *设 x 和 y 是两个串，xy 称为 x 和 y 的连接* (concatenation)。*用符号“$\circ$”表示两个串的连接运算，即* $x\circ y=xy$。 □

例如，若 $x=\mathrm{aba}$，$y=\mathrm{cb}$，则 $x\circ y=xy=\mathrm{abacb}$。

显然，对任意一个串 x，都有 $x\circ\varepsilon=\varepsilon\circ x=x$。

定义 1.5 *设 x 和 y 都是串，$z=xy$。称 x 为 z 的前缀* (prefix)：*当 $x\neq z$（即 $y\neq\varepsilon$）时，称 x 为 z 的真前缀。称 y 为 z 的后缀* (suffix)：*当 $y\neq z$（即 $x\neq\varepsilon$）时，称 y 为 z 的真后缀。* □

例如，若 $z=\mathrm{abacb}$，那么 z 的前缀包括 ε、a、ab、aba、abac 和 abacb 6 个串，其中前 5 个串是 z 的真前缀；z 的后缀包括 ε、b、cb、acb、bacb 和 abacb 6 个串，其中前 5 个是 z 的真后缀。

定义 1.6 *设 x,y,z 都是串，$w=xyz$，则称 y 为 w 的子串* (substring)，*当 $y\neq w$ 时，称 y 为 w 的真子串。* □

显然，一个串 w 的前缀和后缀都是它的子串。此外，可能还有一些既不是 w 的前缀也不是 w 的后缀的子串。例如，设 $w=\mathrm{abacb}$，那么除了前面列出的 w 的前缀和后缀以外，w 的子串还有 ba、bac 和 ac 等。

设 Σ 为一个字母表，通常用 Σ^* 表示 Σ 上全体串（包括空串）组成的集合，而 Σ^+ 表示 Σ 上除空串以外的全体串的集合，即 $\Sigma^+=\Sigma^*-\{\varepsilon\}$。

下面讨论一下 Σ^* 上串的连接运算的一些性质。首先，连接运算在 Σ^* 上是封闭的，即 $\forall x, y \in \Sigma^*$，都有 $x \circ y \in \Sigma^*$。可见，$(\Sigma^*, \circ)$ 构成了一个代数系统。其次，连接运算满足结合律，即 $\forall x, y, z \in \Sigma^*$，$(x \circ y) \circ z = x \circ (y \circ z)$。此外，前面已经指出，$\forall x \in \Sigma^*$：$x \circ \varepsilon = \varepsilon \circ x = x$，即 ε 是这个代数系统的单位元，从而我们可以得到以下结论。

定理 1.1 设 Σ 为一个字母表，“$\circ$”表示 Σ^* 上的连接运算。那么 $(\Sigma^*, \circ)$ 构成一个单元半群（独异点）。 □

$(\Sigma^*, \circ)$ 不能构成一个群。这是因为对一个串 $x \in \Sigma^*$（$x \neq \varepsilon$ 时），不存在 x 关于“$\circ$”运算的逆元 y，使得 $x \circ y = y \circ x = \varepsilon$。注意，定义 1.3 中关于一个串 x 的逆串 x^{R} 同这里所说的关于“$\circ$”运算的逆元的概念是完全不同的。此外，“$\circ$”运算也不满足交换律，即一般地，$x \circ y \neq y \circ x$。

定义 1.7 设 Σ 为一个字母表，$x \in \Sigma^*$。x 的幂运算定义为

1) $x^0 = \varepsilon$ (1.4)

2) $x^n = x^{n-1} \circ x,\ n = 1, 2, \cdots$ (1.5)

□

当 $|x| = 1$ 时，从定义 1.7 就得到单个字符的幂：$\mathrm{a}^0 = \varepsilon$, $\mathrm{a}^1 = \mathrm{a}$, $\mathrm{a}^2 = \mathrm{aa}$, $\mathrm{a}^3 = \mathrm{aaa}$, $\cdots$, $\mathrm{a}^n = \mathrm{a}^{n-1}\mathrm{a}$, $\cdots$

定理 1.2 设 Σ 为一个字母表，m 和 n 为任意两个自然数，那么对任意 $x \in \Sigma^*$ 都有

1) $x^m \circ x^n = x^n \circ x^m = x^{m+n}$ (1.6)

2) $(x^m)^n = (x^n)^m = x^{mn}$ (1.7)

（从定义 1.7 出发，用数学归纳法容易证明本定理。） □

1.2.2 语言

定义 1.8 字母表 Σ 上满足特定条件的串的集合 L 称为 Σ 上的一个语言 (language)。 □

从定义 1.8 看出，语言是一种集合，这种集合中的元素都是字符串。若 L 是字母表 Σ 上的一个语言，则显然有 $L \subseteq \Sigma^*$。对任意一个字母表 Σ，Σ 上最简单的两个语言分别为 $\varnothing$ 和 $\{\varepsilon\}$，其中 $\varnothing$ 表示空集，即 $\varnothing$ 中不含任何元素；$\{\varepsilon\}$ 则是含有空串 ε 一个元素的集合，它不是空集。请注意两者的区别。此外，Σ^* 本身也是 Σ 上的一个语言。当我们以 Σ 上的字符串作为论域时，Σ^* 是一个全集。

例 1.2 全体二进制非负整数的集合 L_1 是字母表 $\Sigma_1 = \{0, 1\}$ 上的一个语言。如果从严格的意义来理解二进制非负整数，即要求二进制正整数的首位必须是 1，那么这个语言可以表示为

$$L_1 = \{ x \in \{0,1\}^* \mid x = 0 \text{ 或 } x \text{的首位为 } 1 \}$$

通常也从广义的角度来定义二进制数，即把每一个由 0 和 1 组成的串都看作一个二进制非负整数。这时，表示全体二进制非负整数的集合的语言就变成 $L_2 = \{0, 1\}^*$。

$L_3 = \{x \in \{0,1\}^* \mid |x| \leqslant 3\}$ 也是字母表 $\Sigma_1 = \{0, 1\}$ 上的一个语言。这个语言共有 15 个元素（字符串），即

$$L_3 = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$$

上面对 L_3 给出了两种表示法。前一种是解析表示法，通过表达式来给出语言 L_3 中的元素（字符串）应满足的条件；后一种表示法则列出了 L_3 中的全体元素。这两种都是常用的集合表示法。当然，要用第二种表示法来表示一个语言，其前提条件是这个语言是有限个字符串的集合，而且元素的个数不要太多。由于一个语言归根到底是一个集合（以字符串为元素的集合），因此集合的表示法以及集合的并、交、补等运算定义也适用于语言。

例 1.3 $L_4=\{\mathrm{a}^n\mathrm{b}^n \mid n\geqslant 1\}$ 是字母表 $\Sigma_2=\{\mathrm{a},\mathrm{b}\}$ 上的一个语言。它显然是一个无限集。因为对每个正整数 n，都有一个 $\mathrm{a}^n\mathrm{b}^n$ 是 L_4 中的元素，而且 n 的取值不同，L_4 中对应的元素也不相同。如果要用罗列的方法来给出语言 L_4 的表示，就只能加上省略号，即写成

$$L_4=\{\mathrm{ab},\ \mathrm{aabb},\ \mathrm{aaabbb},\cdots\}$$

$L_5=\{x\in\{\mathrm{a},\mathrm{b}\}^* \mid x=x^{\mathrm{R}}\}$ 也是字母表 $\Sigma_2=\{\mathrm{a},\mathrm{b}\}$ 上的一个语言。这个语言中的每个字符串 $x\in\{\mathrm{a},\mathrm{b}\}^*$ 都具有这样的性质：$x=x^{\mathrm{R}}$。我们称满足性质 $x=x^{\mathrm{R}}$ 的串 x 为回文 (palindrome)。这样 L_5 是字母表 $\{\mathrm{a},\mathrm{b}\}$ 上的全体回文的集合。

定义 1.9 设 L_1 为字母表 Σ_1 上的一个语言，L_2 是字母表 Σ_2 上的一个语言，那么

$$L_1\circ L_2=\{x\circ y \mid x\in L_1\wedge y\in L_2\} \tag{1.8}$$

称为语言 L_1 与 L_2 的连接。显然它是字母表 $\Sigma_1\cup\Sigma_2$ 上的一个语言。 □

定理 1.3 设 $L_i(i=1,2,3)$ 是字母表 Σ 上的语言，那么

$$L_1\circ(L_2\cup L_3)=(L_1\circ L_2)\cup(L_1\circ L_3) \tag{1.9}$$

$$(L_1\cup L_2)\circ L_3=(L_1\circ L_3)\cup(L_2\circ L_3) \tag{1.10}$$

(1.9) 式和 (1.10) 式中的“$\cup$”是指集合的并运算。

（从定义出发可以直接证明本定理。） □

定义 1.10 设 L 为字母表 Σ 上的一个语言，记

$$L^0=\{\varepsilon\} \tag{1.11}$$

$$L^n=L^{n-1}\circ L,\qquad n=1,2,\cdots \tag{1.12}$$

$$L^*=\bigcup_{i=0}^{\infty}L^i \tag{1.13}$$

$$L^+=\bigcup_{i=1}^{\infty}L^i \tag{1.14}$$

称 L^* 为语言 L 的 Kleene 闭包（或简称为闭包），L^+ 为 L 的正闭包。 □

当 $L=\{a\}$ 时，就得到

$$a^*=\bigcup_{i=0}^{\infty}a^i$$

$$a^+=\bigcup_{i=1}^{\infty}a^i$$

定理 1.4 设 L 是字母表 Σ 上的一个语言，则

$$L^+=L^*\circ L=L\circ L^* \tag{1.15}$$

$$L^*=L^+\cup\{\varepsilon\} \tag{1.16}$$

证明 (1.16) 式可以直接由定义 1.10 中的 (1.13) 式和 (1.14) 式得到，下面证明 (1.15) 式。

$$
\begin{aligned}
L^* \circ L &= \left(\bigcup_{i=0}^{\infty} L^i\right) \circ L \\
&= \bigcup_{i=0}^{\infty} (L^i \circ L) \\
&= \bigcup_{i=0}^{\infty} L^{i+1} \\
&= \bigcup_{i=1}^{\infty} L^i = L^+
\end{aligned}
$$

□

1.2.3 文法

文法（grammar）是语言的产生器。语言学家N. Chomsky提出可以在一个字母表上给出一组语言产生规则，根据这些规则推导出来的全体句子组成的集合就构成一个语言。这样的一组规则就称为文法，根据对规则的不同限制，可以对文法及其产生的语言进行分类，形成Chomsky文法体系。

定义 1.11 文法是一个四元组 $G=(V,T;P,S)$，其中 V 是变量的一个有限集(也称为非终极符集)；T 是终极符的一个有限集(也称为字母表)，$V\cap T=\varnothing$；S 是一个初始符($S\in V$)；P 是产生式的一个有限集。P 中的元素为 (α,β)（通常表示为 $\alpha\to\beta$），$\alpha,\beta\in(V\cup T)^*$，而且 α 中至少有一个字符是 V 的元素。 □

产生式集合 P 是一个文法的核心，它给出了产生语言的一组形式化规则。由于每个产生式 $\alpha\to\beta$ 的左边的串 α 中至少要含有变量集 V 的一个字符，因此又可写成

$$\alpha\in(V\cup T)^*\circ V\circ(V\cup T)^*$$

或把产生式集 P 写成笛卡尔积的一个子集，即

$$P\subseteq(V\cup T)^*\circ V\circ(V\cup T)^*\times(V\cup T)^*$$

通常用 A,B,C,D,S 等大写字母表示变量（即变量集 V 的元素）；用 a,b,c,d 等小写字母表示终极符（即 T 中的元素）；由终极符组成的串（终极符串）用 w,x,y,z 等表示；由变量和终极符组成的串（$(V\cup T)^*$ 中的元素）则用 $\alpha,\beta,\gamma,\delta$ 等希腊字母表示。

由文法 G 产生语言的过程是这样的：从初始符 S 开始，根据 G 中的一个产生式可以进行一步推导，如果经过若干步推导得到一个只含终极符的串，这个串就是 $L(G)$ 中的一个句子。所有能通过推导得到的句子组成的集合就是语言 $L(G)$。推导过程中出现的每一个串都称为 G 的一个句型。下面写出相关的形式定义。

定义 1.12 设 $G=(V,T;P,S)$ 为一个文法，那么

1) S 是 G 的一个句型；

2) 若 $\delta\alpha\gamma$ 是 G 的一个句型，$\alpha\to\beta$ 是 G 的一个产生式，那么 $\delta\beta\gamma$ 也是 G 的一个句型。从句型 $\delta\alpha\gamma$ 得到句型 $\delta\beta\gamma$ 的过程称为文法 G 的一步推导，记为

$$\delta\alpha\gamma\underset{G}{\Rightarrow}\delta\beta\gamma \quad 或 \quad \delta\alpha\gamma\Rightarrow\delta\beta\gamma$$

（在不会产生混淆的情况下，推导符“$\underset{G}{\Rightarrow}$”下方的字符 G 可以省略。）

3) 不含非终极符的句型是 $L(G)$ 的一个句子。 □

若 $\gamma_i \in (V \cup T)^*(i=1,2,\cdots,n)$，且 $\gamma_i \Rightarrow \gamma_{i+1}(i=1,2,\cdots,n-1)$是 G 的一步推导，即

$$\gamma_1 \Rightarrow \gamma_2,\ \gamma_2 \Rightarrow \gamma_3,\ \cdots,\ \gamma_{n-1} \Rightarrow \gamma_n$$

那么可以把 $n-1$ 步的推导过程记为 $\gamma_1 \overset{*}{\Rightarrow} \gamma_n$。特别地，当 $n=1$ 时，$\gamma_1 \overset{*}{\Rightarrow} \gamma_n$ 表示零步推导。

定义 1.13 设 $G=(V,T;P,S)$为一个文法，那么 G 所产生的语言为

$$L(G)=\left\{w \in T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\right\}$$ □

例 1.4 设有一个文法 $G_1=(V,T;P,S)$，其中 $V=\{S,A\}$，$T=\{0,1\}$

$$P:\quad S \to 0S,\quad S \to 0A,$$
$$A \to 1A,\quad A \to \varepsilon$$

那么，容易得到下面的一些推导过程

$$S \Rightarrow 0S \Rightarrow 00S \Rightarrow 000A \Rightarrow 0001A \Rightarrow 0001$$
$$S \Rightarrow 0A \Rightarrow 01A \Rightarrow 01$$
$$S \Rightarrow 0A \Rightarrow 0$$
$$S \Rightarrow 0S \Rightarrow 00A \Rightarrow 001A \Rightarrow 0011A \Rightarrow 00111A \Rightarrow 00111$$

可见，0001、01、0 和 00111 等都是 G_1 所产生的语言 $L(G_1)$ 中的句子。通过对这些推导的归纳，可以证明

$$L(G_1)=\left\{0^i1^j \mid i \geqslant 1, j \geqslant 0\right\}$$

Chomsky 文法体系把文法分为四种类型，它们是根据对定义 1.11 的文法定义加上不同的限制条件而得到的。由于文法分成四种类型，由文法所产生的语言也因此可以分成四大类。下面给出这四类文法和由此产生的四类语言的定义。

定义 1.14 设 $G=(V,T;P,S)$为一个文法。

1) 若 G 中的产生式如定义 1.11 所规定，不加任何其他限制，则称 G 为一个无限制文法 (unrestricted grammar)，或 0 型文法。这种文法产生的语言称为无限制语言（或 0 型语言）。

2) 若 G 中的产生式 $\alpha \to \beta$ 在定义 1.11 所规定的基础上加上限制条件：除 $S \to \varepsilon$ 外，都满足条件 $|\alpha| \leqslant |\beta|$，则称 G 为一个上下文有关文法 (context-sensitive grammar)，或 1 型文法。这种文法所产生的语言称为上下文有关语言（或 1 型语言）。

3) 若 G 中的产生式都有

$$A \to \beta,\qquad A \in V,\ \beta \in (V \cup T)^*$$

的形式，则称这种文法为上下文无关文法 (context-free grammar)，或 2 型文法。这种文法产生的语言称为上下文无关语言（或 2 型语言）。

4) 若 G 中的产生式都有

$$\begin{aligned} &A \to a, \\ &A \to aB\ (A \to Ba), \end{aligned} \qquad A,B \in V,\ a \in T$$

的形式，则称 G 为一个右（左）线性文法，所产生的语言称为右（左）线性语言。右线性文法和左线性文法统称为正规文法 (regular grammar)，或 3 型文法。右线性语言和左线性语言统称为正规语言（或 3 型语言）。 □

定理 1.5 正规语言类是上下文无关语言类的子类；上下文无关语言类是上下文有关语言

类的子类；上下文有关语言类是无限制语言类的子类。

证明 由定义 1.14 所给出的各类型文法的产生式形式易知本定理的结论。 □

定义 1.15 设 Σ 为一个字母表，Σ 上的正规表达式及其表示的正规集定义为

1) ϕ 是一个正规表达式，它表示空集 $\varnothing$；

2) ε 是一个正规表达式，它表示集合 $\{\varepsilon\}$；

3) $\forall a \in \Sigma$，a 是一个正规表达式，它表示集合 $\{a\}$；

4) 若 r 和 s 都是 Σ 上的正规表达式，它们分别表示集合 R 和 S，则 $r+s$、rs 和 r^* 也是 Σ 上的正规表达式，它们各自表示 $R \cup S$、$R \circ S$ 和 R^*。

正规表达式表示的集合称为正规集。 □

定理 1.6 L 是一个正规语言，当且仅当它可以由一个正规表达式所表示。 □

1.3 字符串的数值化

现代电子计算机对字符串的处理可以转化为数论函数的计算。这种转化是通过字符串的数值化而实现的。字符串的数值化是指把一个字符串转化为一个自然数，或者说用一个自然数来表示一个字符串。当然，这种表示应该是唯一的，而且是可逆的。所谓唯一的是指，两个不同的字符串，表示它们的自然数不相等（对同一种表示方法而言）；所谓可逆的是指，如果一个自然数是表示字符串的数（即满足相应的条件），那么存在一种方法（步骤）求出它所表示的字符串。字符串的数值化通常有两种，下面分别对它们进行介绍。

1.3.1 哥德尔配数法

哥德尔 (Gödel) 配数法的出发点是一张从小到大排列的素数表

$$P_1, P_2, \cdots, P_m$$

即 $P_1=2$, $P_2=3$, $P_3=5$ ……余类推。表中的素数个数 m 足够大，它不小于字符串处理过程中出现的每个字符串的长度。

假设某个问题的处理过程中出现的字符串都是字母表

$$\Sigma = \{X_1, X_2, \cdots, X_k\}$$

上的字符串。我们分别用正整数 $1, 2, \cdots, k$ 对 Σ 中的字符串进行编号，即 X_1 的编号为 1，X_2 的编号为 2，……余类推。

对任意一个字符串 $x \in \Sigma^*$，设 $x = a_1 a_2 \cdots a_n$，我们称

$$\text{Göd}(x) = P_1^{e_1} P_2^{e_2} \cdots P_n^{e_n} \tag{1.17}$$

为字符串 x 的 Gödel 数。其中 e_i 是字符 a_i 在 Σ 中的编号，即

$$\text{若 } a_i = X_j, \text{ 则 } e_i = j, \qquad i = 1, 2, \cdots, n, \quad j \in \{1, 2, \cdots, k\}$$

例 1.5 设 $\Sigma = \{\text{a}, \text{b}, \text{c}, \text{d}\}$。由于 $|\Sigma| = 4$，我们分别用 $1, 2, 3, 4$ 对 Σ 中的字母进行编号：a 的编号为 1，b 的编号为 2，c 的编号为 3，d 的编号为 4。根据这个编号，对任意 $x \in \Sigma^*$，都可以求出 x 的 Gödel 数。

例如：设 $x_1 = \text{daba}$，那么

$$\text{Göd}(x_1) = 2^4 \times 3^1 \times 5^2 \times 7^1 = 8\,400$$

设 $x_2 = \text{aaaaa}$，那么

$$\text{Göd}(x_2) = 2^1 \times 3^1 \times 5^1 \times 7^1 \times 11^1 = 2\,310$$

字符串的 Gödel 数的唯一性是显然的。即：若 $w_1 \neq w_2$，则 $\text{Göd}(w_1) \neq \text{Göd}(w_2)$。另一方面，根据整数的素因子唯一分解定理，两个不同的自然数对应的素因子（的幂）是不相同的，因此它们不可能表示同一个字符串。此外，若一个自然数是某个字符串的 Gödel 数，那么从该自然数求它所对应的字符串的过程也是显然的：只要对该自然数进行素因子分解，并把各个素因子从小到大排列，那么这些素因子的幂就指出了对应的字符串中各个位置的字符。例如，对自然数 42 000，由于

$$42\,000 = 2^4 \times 3^1 \times 5^3 \times 7^1$$

从对 Σ 中 a, b, c, d 各自的编号知，42 000 对应的字符串为 daca。

然而，并不是每一个自然数都可以作为字符串的 Gödel 数的。一个自然数 n 是某个字符串的 Gödel 数，必须满足下面的条件

$$\text{若 } P_i \mid n, \text{ 则 } \forall j < i : P_j \mid n$$

其中 P_i 和 P_j 分别表示素数表中的第 i 个素数和第 j 个素数。例如 189 不是字符串的 Gödel 数。因为

$$189 = 3^3 \times 7^1 = P_2^3 \cdot P_4^1 \tag{1.18}$$

这表明，如果用 189 来表示一个字符串，那么该字符串中（从左数起）的第 2 个字符应该是字母表中编号为 3 的那个字符，而字符串中的第 4 个字符应该是字母表中编号为 1 的那个字符。然而，字符串中第 1 个字符和第 3 个字符是什么呢？(1.18) 式中并没有给出相应的信息，其原因就在于 $P_4 \mid 189$ 和 $P_2 \mid 189$，但 $P_3 \nmid 189$ 且 $P_1 \nmid 189$。

1.3.2 多项式求值配数法

假设某个问题的处理过程中出现的字符串都是字母表

$$\Sigma = \{X_1, X_2, \cdots, X_k\}$$

上的串。我们分别用 $1, 2, \cdots, k$ 对 Σ 中的字符串依序进行编号。

对任意一个字符串 $w \in \Sigma^*$，设 $w = a_1, a_2, \cdots, a_l$，那么构造一个 $l-1$ 次多项式

$$P(x) = c_1 x^{l-1} + c_2 x^{l-2} + \cdots + c_{l-1} x + c_l$$

其中

$$c_i = j, \text{ 当且仅当 } a_i = X_j, \qquad i = 1, 2, \cdots, l, \quad j \in 1, 2, \cdots, k$$

并用 $x = k+1$ 代入上述多项式，便得到表示字符串 x 的自然数值

$$\text{Pol}(x) = P(x)|_{x=k+1} = c_1 (k+1)^{l-1} + c_2 (k+1)^{l-2} + \cdots + c_{l-1} (k+1) + c_l$$

例 1.6 设 $\Sigma = \{\text{a}, \text{b}, \text{c}, \text{d}\}$，我们用 1, 2, 3, 4 对 Σ 中的 4 个字符分别编号，即 a 的编号为 1，b 的编号为 2，c 的编号为 3，d 的编号为 4。基于这样一组编号，对任意 $w \in \Sigma^*$，都可以求出 $\text{Pol}(x)$ 的值。

例如，设 $w_1=\mathrm{abacd}$，由于 $|w_1|=5$，所以我们构造一个 4 次多项式

$$P(x)=c_1x^4+c_2x^3+c_3x^2+c_4x+c_5$$

从字符串 w_1 各个位置中的具体字符，易知 $c_1=1, c_2=2, c_3=1, c_4=3, c_5=4$，即

$$P(x)=x^4+2x^3+x^2+3x+4$$

用 $x=|\Sigma|+1=5$ 代入上述多项式，便得到

$$\mathrm{Pol}(w_1)=P(x)|_{x=5}=5^4+2\times5^3+5^2+3\times5+4=919$$

设 $w_2=\mathrm{bcdb}$，那么容易求出

$$\mathrm{Pol}(w_2)=2\times5^3+3\times5^2+4\times5+2=347$$

字符串的多项式值配数的唯一性是显然的。任意两个字符串 $w_1,w_2\in\Sigma^*$，如果 $w_1\neq w_2$，那么显然 $\mathrm{Pol}(w_1)\neq\mathrm{Pol}(w_2)$。另一方面，对任意给定的一个自然数 n，如果它是字母表 Σ 上的一个字符串 w 的多项式值配数，那么只要用 $|\Sigma|+1$ 作为除数，通过逐次做除法求余的运算，就可以从右到左逐个求出 w 的各位编号，从而也就求出了 w。

一个自然数可以作为某个字母表 Σ 上的字符串的多项式值配数，也是需要满足一定条件的。这个条件是，用 $|\Sigma|+1$ 作为除数，逐次做除法求余的过程中，出现的余数都不为零。或者换句话说，逐次做除法求余时，每次得到的商都不是 $|\Sigma|+1$ 的正整数倍数。

例 1.7 仍以 $\Sigma=\{\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d}\}$ 作为字母表，对 a, b, c, d 仍分别编号为 $1,2,3,4$。由于 $|\Sigma|=4$，所以对一个自然数 n，我们可以以 $|\Sigma|+1=5$ 作为除数，通过逐次做除法求余运算来判定 n 是否为 Σ 上某个字符串的多项式值配数。如果是，逐次做除法求余的过程也就是求出 n 所表示的字符串。例如，对于 $n_1=294$，用 5 作除数，逐次做除法求余运算的过程可以表示为

$$294=58\times5+4$$
$$58=11\times5+3$$
$$11=2\times5+1$$
$$2=0\times5+2$$

由于每一步除法得到的余数都不为 0，因此 294 是字母表 $\Sigma=\{\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d}\}$ 上的某个字符串的多项式值配数。这个字符串各位的字符编号从左到右分别为 $2,1,3,4$。因此，294 表示的字符串为 bacd。即 $\mathrm{Pol}(\mathrm{bacd})=294$。

对于 $n_2=528$，用 5 作除数逐次做除法求余的过程为

$$528=105\times5+3$$
$$105=21\times5+0$$
$$21=4\times5+1$$
$$4=0\times5+4$$

由于第二个式子中余数为 0，0 不是字母表 Σ 中某个字符的编号，因此 $n_2=528$ 不是 $\Sigma=\{\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d}\}$ 上的字符串的多项式值的配数。

1.3.3 字符串处理规约为数论函数计算

上面给出的两种字符串的数值化方法都用自然数来表示一个字母表中的各个字符串。不论

哪一种方法，只要字母表一旦确定，该字母表上的每一个字符串都对应着唯一的一个自然数。另一方面，并不是每个自然数都可以表示该字母表上的一个字符串，只有满足一定条件的自然数才能表示该字母表上的字符串。

通过这些字符串中的数值化方法，就可以把现代电子计算机对字符串的处理转化为数论函数的计算。假设用现代电子计算机求解某个问题时，输入字符串为 $x_1 \in \Sigma^*$，通过计算机的处理输出字符串为 $x_2 \in \Sigma^*$。那么这个计算机处理过程可以理解为数论函数的计算：当我们采用 Gödel 配数法对 Σ 上的字符串进行数值化时，上述处理过程可以规约为数论函数

$$f(\mathrm{Göd}(x_1)) = \mathrm{Göd}(x_2)$$

的计算；当我们用多项式求值配数法对 Σ 上的字符串进行数值化时，上述处理过程就可以规约为数论函数

$$f(\mathrm{Pol}(x_1)) = \mathrm{Pol}(x_2)$$

的计算。

由于用电子计算机做的各种工作都可以归结为字符串处理，因此用现代电子计算机求解各种问题都可以理解为数论函数的计算。

下面讨论一下逻辑公式的计算与判定问题。

一个一阶谓词逻辑公式由逻辑变元和谓词、逻辑运算符以及量词组成。对于一类特定问题，把相关的逻辑变元和谓词，以及逻辑运算符和量词的集合作为一个字母表 Σ，那么一个一阶谓词逻辑公式就是字母表 Σ 上的一个字符串，逻辑公式计算（推理）的过程中出现的逻辑公式仍然是这个字母表上的字符串。通过 Gödel 配数法或多项式求值配数法，可以把这些字符串转化为自然数。另一方面，逻辑公式的计算结果是逻辑值，即真或假。如果用 1 和 0 分别表示真、假两个逻辑值，那么一阶谓词逻辑公式的计算就规约为数论谓词

$$P: \{\mathrm{Göd}(x) \mid x \in \Sigma^*\} \to \{0,1\}$$

或

$$P: \{\mathrm{Pol}(x) \mid x \in \Sigma^*\} \to \{0,1\}$$

的计算。由于数论谓词是数论函数的一种特殊情形，我们也可以说逻辑公式的计算可以规约为数论函数的计算。

由于每一个判定问题都可以用一个一阶谓词逻辑公式来表述，因此可以把判定问题规约为数论函数的计算。

1.3.4 Cantor 编号

康托 (Cantor) 编号是用一个自然数来表示一个 n 元组（其中每一元也是一个自然数）的一种方法。当 $n = 2$ 时，Cantor 编号就称为 Cantor 对角线法则。在证明有理数集可以同自然数集建立一一对应关系（即有理数集是可列集）时，就使用过 Cantor 对角线法则。下面先叙述对二元组的 Cantor 编号。

对于二元组集合 $\mathbf{N} \times \mathbf{N} = \{(x,y) \mid x,y = 0,1,2,\cdots\}$ 中的每个元素 (x,y)，它的 Cantor 编号 $C(x,y)$ 根据对角线法则确定，如图 1.1 所示

元素（二元组）排列顺序根据虚线所示进行安排。首先由左上到右下选择虚线，每条虚线