高等教育计算机专业教材

可计算性与计算复杂性导引

张立昂　编著

北京大学出版社

高等教育计算机专业教材

可计算性与计算复杂性导引

张立昂 编著

北京大学出版社
北京

图书在版编目(CIP)数据

可计算性与计算复杂性/张立昂编著.-北京：
北京大学出版社，1997.1
ISBN 7-301-03229-3

Ⅰ.可… Ⅱ.张… Ⅲ.①电子计算机-可计算性-理论
②电子计算机-计算复杂性-理论 Ⅳ.TP301

书　　名：可计算性与计算复杂性导引
著作责任者：张立昂
责任编辑：杨锡林
标准书号：ISBN 7-301-03229-3/TP·0319
出 版 者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
电　　话：出版部　62752015　发行部　62559712　编辑部　62752032
排 印 者：中国科学院印刷厂
发 行 者：北京大学出版社
经 销 者：新华书店
　　850×1168 毫米　32 开本　9.75 印张　250 千字
　　1996 年 10 月第一版　1996 年 10 月第一次印刷
定　　价：15.00 元

内 容 简 介

本书是学习理论计算机科学基础的教材和参考书，内容包括三部分：可计算性、形式语言与自动机、计算复杂性。主要介绍几种计算模型及它们的等价性，函数、谓词和语言的可计算性等基本概念，形式语言及其对应的自动机模型，时间和空间复杂性，NP完全性等。

本书可作为计算机专业本科生和研究生的教材，也可作为从事计算机科学技术的研究和开发人员的参考书，还可作为对理论计算机科学感兴趣的读者的入门教材。

前　言

计算机科学技术日新月异，新东西层出不穷、旧东西迅速被淘汰。但是，作为一门科学，它有其自身的基础理论。这些思想精华长久的、甚至永恒的放射着光芒。这些理论在应用开发中好像是“无用的”。其实，对于每一位从事计算机科学技术的研究和开发的人来说，它们都是不可缺少的，就像能量守恒之类的物理定律对于每一位自然科学工作者和工程技术人员那样。北京大学计算机科学技术系开设了“理论计算机科学基础”这门课，就是希望能把这样一些最基本的知识介绍给学生。本书是在这门课的讲稿的基础上加工而成的。

本书的内容包括三部分：可计算性、形式语言与自动机、计算复杂性。这三个领域（更不用说整个理论计算机科学）的内容极其丰富并且在不断地发展。作为本科生一个学期的课程只能选择其中最基本的部分，使学生在这些方面有一个大的理论框架。本书主要取材于参考文献[1]—[4]。书中部分章节涉及到数理逻辑和图论中的一些问题，不熟悉这些内容的读者可查阅参考文献[6]、[7]。书末附有中英文名词索引和记号，并给出定义这些名词和记号的章节。

本书的出版得到北京大学出版社的热情支持，笔者在此表示衷心的感谢。在本书的出版和写作过程中得到董士海教授、袁崇义教授、王捍贫博士和黄雄的各种形式的帮助，对他们表示感谢。最后，笔者要特别感谢许卓群教授。作为主管教学工作的系领导，从这门课的开设到本书的出版许卓群教授给予了一贯的积极支持和指导。

张立昂

1996年春于北大燕北园

目　录

第一章　程序设计语言 $\mathscr{L}$ 和可计算函数

1.1　预 备 知 识

本书设想读者熟悉离散数学，掌握数理逻辑、集合论、图论中的基本概念、术语和符号（参阅参考文献[6]、[7]）. 这一节仅对本书中某些术语和符号的特殊用法作一说明.

在本书中通常只使用自然数. 如无特殊声明，“数”均指自然数. 自然数集合记作 $N=\{0,1,2,\cdots\}$.

设集合 S 和 T，$S\times T$ 的元素 (a,b) 称作**有序对**，又称作**有序二元组**或**二元组**. $S\times T$ 的子集称作 S 到 T 的**二元关系**. S 到 S 的二元关系，即 $S\times S$ 的子集，称作 S 上的二元关系.

设 R 是 S 到 T 的二元关系，R 的定义域

$$\mathrm{dom}R=\{a\mid\exists b\quad (a,b)\in R\}.$$

R 的值域

$$\mathrm{ran}R=\{b\mid\exists a\quad (a,b)\in R\}.$$

设 $A\subseteq S$，A 在 R 下的**象**

$$R(A)=\{b\mid\exists a\quad (a\in A\wedge (a,b)\in R)\}.$$

特别地，设 $a\in A$，把 $\{a\}$ 在 R 下的象简称作 a 在 R 下的象，并记作 $R(a)$，即

$$R(a)=\{b\mid(a,b)\in R\}.$$

设 f 是 S 到 T 的二元关系，如果对每一个 $a\in S$，$f(a)=\varnothing$ 或 $\{b\}$，则称 f 是 S 到 T 的**部分函数**，或 S 上的部分函数. 部分函数也可简称为函数. 若 $f(a)=\{b\}$，则称 $f(a)$ 有定义，b 是 f 在 a 点的函数值并记作 $f(a)=b$. 若 $f(a)=\varnothing$，则称 $f(a)$ 无定义并记作 $f(a)\uparrow$. 当 $f(a)$ 有定义时，可记作 $f(a)\downarrow$. 如果对每一个 $a\in S$ 都

有 $f(a)\downarrow$，即 $\mathrm{dom}f=S$，则称 f 是 S 上的**全函数**. 此时可记作 $f:S\to T$. 空集$\varnothing$本身是任何集合上的部分函数，称作**空函数**. 空函数处处无定义.

设 f 是笛卡儿积 $S_1\times S_2\times\cdots\times S_n$ 上的部分函数，把 $f((a_1,a_2,\cdots,a_n))$记作 $f(a_1,a_2,\cdots,a_n)$. 集合 S^n 上的部分函数称作 S 上的 **n 元部分函数**. 当需要表明 n 元时，常用 $f(x_1,x_2,\cdots,x_n)$代替 f.

N^n 到 N 的部分函数称作 n 元部分**数论函数**. 作为数论函数，$2x$ 是全函数，而 $x/2$，$x-y$，$\sqrt{x}$ 都只是部分函数，不是全函数. 在这里 $3/2$，$4-6$，$\sqrt{5}$ 都没有定义.

字母表是一个非空有穷集合. 设 A 是一个字母表，A 中元素的有穷序列 $w=(a_1,a_2,\cdots,a_m)$称作 A 上的**字符串或字**. 今后总把它记作 $w=a_1a_2\cdots a_m$. 字符串 w 的长度(即 w 中的符号个数)记作 $|w|$. 用 ε 表示空串，它不含任何符号，是唯一的长度为 0 的字符串. A 上字符串的全体记作 A^*. 设 $u,v\in A^*$，把 v 连接在 u 的后面得到的字符串记作 uv. 例如，$u=ab$，$v=ba$，则 $uv=abba$，$vu=baab$.

设 $u\in A^*$，规定

$$u^0=\varepsilon$$

$$u^{n+1}=u^nu,\quad n\in N.$$

显然，当 $n>0$ 时，u^n 等于 n 个 u 连接在一起.

$(A^*)^n$ 到 A^* 的部分函数称作 A 上的 n 元部分**字函数**.

1.2 程序设计语言 $\mathscr{S}$

考虑 N 上的计算. 先通过几个例子直观地说明程序设计语言 $\mathscr{S}$.

语言 $\mathscr{S}$ 使用三种变量：输入变量 $X_1,X_2,\cdots$，输出变量 Y 和中间变量 $Z_1,Z_2,\cdots$. 变量可以取任何数作为它的值. 语言还要使

用标号 $A_1, A_2, \cdots$. 约定：当下标为 1 时，可以略去. 例如，X_1 和 X 表示同一个变量. 另外，虽然在语言的严格定义中规定只能使用上述变量和标号，但在今后书写程序时也常使用其他字母表示中间变量和标号，以方便阅读.

语言 $\mathscr{S}$ 有三种类型的语句：

(1) 增量语句 $V \leftarrow V+1$，变量 V 的值加 1.

(2) 减量语句 $V \leftarrow V-1$，若变量 V 的当前值为 0，则 V 的值保持不变；否则 V 的值减 1.

(3) 条件转移语句 IF $V \neq 0$ GOTO L，若变量 V 的值不等于 0，则下一步执行带标号 L 的指令(转向标号 L)；否则顺序执行下一条指令.

开始执行程序时，中间变量和输出变量的值都为 0. 从第一条指令开始，一条一条地顺序执行，除非遇到条件转移语句. 当程序没有指令可执行时，计算结束. 此时 Y 的值为程序的输出值.

[**例 1.1**]

$[A]$ $X \leftarrow X-1$
$Y \leftarrow Y+1$
IF $X \neq 0$ GOTO A

这里 A 是第一条指令的标号. 不难看出，这个程序计算函数

$$f(x)=\begin{cases} x & \text{若 } x>0 \\ 1 & \text{否则} \end{cases}$$

这里有一个特殊的点 $x=0$. 如果我们希望把 X 的值复制给 Y，即计算 $f(x)=x$，则需要对 $x=0$ 的情况作特殊处理，可以修改程序如下.

[**例 1.2**]

$[A]$ IF $X \neq 0$ GOTO B
$Z_1 \leftarrow Z_1+1$
IF $Z_1 \neq 0$ GOTO E
$[B]$ $X \leftarrow X-1$
$Y \leftarrow Y+1$
$Z_2 \leftarrow Z_2+1$

IF $Z_2 \neq 0$ GOTO A

在这个程序中，执行

$Z_1 \leftarrow Z_1 + 1$

IF $Z_1 \neq 0$ GOTO E

的结果总是转向标号 E. 这相当于一条“无条件转向语句”

GOTO E

但是，在语言 $\mathscr{S}$ 中没有这样的语句. 我们把它作为这段程序的缩写，称作**宏指令**. 对应的这段程序称作这条宏指令的**宏展开**. 使用宏指令可以使程序的书写大为精简. 当然，在必要的时候，可以用宏展开代替宏指令得到详细的 $\mathscr{S}$ 程序.

程序的最后两条也可以缩写成宏指令 GOTO A. 利用宏指令改写程序如下：

$[A]$ IF $X \neq 0$ GOTO B
GOTO E
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
GOTO A

这个程序把 X 的值赋给 Y，但是当计算结束时 X 的值为 0，失去了计算开始时的值. 在把一个变量的值赋给另一个变量时，通常要求在赋值结束时保持前者的值不变. 为此，引入一个中间变量 Z，在把 X 的值赋给 Y 的同时也赋给 Z，在给 Y 的赋值完成后再把 Z 的值赋给 X. 程序在下例中给出.

[例 1.3]

$[A]$ IF $X \neq 0$ GOTO B
GOTO C
$[B]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
$Z \leftarrow Z + 1$
GOTO A
$[C]$ IF $Z \neq 0$ GOTO D

GOTO E

$[D]$ $Z \leftarrow Z-1$

$X \leftarrow X+1$

GOTO C

[**例 1.4**] $V \leftarrow V'$的宏展开.

宏指令 $V \leftarrow V'$的含义是把 V'的值赋给 V，而保持 V'的值不变. 例 1.3 中的程序把 X 的值赋给 Y，并且 X 的值在计算结束时与计算开始时相同. 这个程序已经基本上实现了这条宏指令的要求. 但是，一个宏展开和一个独立使用的程序是有区别的. 例 1.3 中的程序在执行开始时，Y 的值自动为 0. 而在开始执行宏指令 $V \leftarrow V'$时，变量 V 很可能在前面已经使用过，从而它的值不一定为 0. 因此，为了保证赋值的正确性，必须在宏展开的开头将 V 的值重新置 0. 按照习惯，把它写成

$$V \leftarrow 0$$

当然，这也是一条宏指令. 它的宏展开是

$[L]$ $V \leftarrow V-1$

IF $V \neq 0$ GOTO L

现将 $V \leftarrow V'$的宏展开列表如下，这里使用了多条宏指令.

$V \leftarrow 0$

$[A]$ IF $V' \neq 0$ GOTO B

GOTO C

$[B]$ $V' \leftarrow V'-1$

$V \leftarrow V+1$

$Z \leftarrow Z+1$

GOTO A

$[C]$ IF $Z \neq 0$ GOTO D

GOTO E

$[D]$ $Z \leftarrow Z-1$

$V' \leftarrow V'+1$

GOTO C

前面几个例子计算的函数都是全函数，下面举一个计算部分函数的例子.

[**例 1.5**]　　[A]　IF $X\neq 0$ GOTO B

$Z\leftarrow Z+1$

IF $Z\neq 0$ GOTO A

[B]　$X\leftarrow X-1$

$Y\leftarrow Y+1$

IF $X\neq 0$ GOTO B

它计算的函数是

$$f(x)=\begin{cases}x & \text{若 } x>0\\ \uparrow & \text{否则}\end{cases}$$

若程序执行开始时 X 的值为 0，则程序无休止地执行下去，永不停止.

现在给出程序设计语言 $\mathscr{S}$ 的严格描述.

1. 变量

输入变量 $X_1, X_2, \cdots$

输出变量 Y

中间变量 $Z_1, Z_2, \cdots$.

2. 标号 $A_1, A_2, \cdots$

正如前面所说的那样，下标 1 常常省去. 语言 $\mathscr{S}$ 严格地规定上述变量和标号，但在书写程序时通常可以任意地使用其他英文大写字母.

3. 语句

增量语句　$V\leftarrow V+1$

减量语句　$V\leftarrow V-1$

空语句　　$V\leftarrow V$

条件转移语句　IF $V\neq 0$ GOTO L

其中 V 是任一变量，L 是任一标号. 空语句不做任何运算，类似

FORTRAN 中的 CONTINUE，它对语言的计算能力没有影响，引入空语句是由于理论上的需要，这要在第三章才能看到.

4. 指令

一条**指令**是一个语句(称作无标号指令)或[L]后面跟一个语句，其中 L 是任一标号，称作该指令的标号，也称该指令带标号 L.

5. 程序

一个**程序**是一张指令表，即有穷的指令序列. 程序的指令数称作程序的长度. 长度为 0 的程序称作**空程序**. 空程序不包含任何指令.

6. 状态

设 σ 是形如等式 $V=m$ 的有穷集合，其中 V 是一个变量，m 是一个数. 如果：(1) 对于每一个变量 V，σ 中至多含有一个等式 $V=m$，(2) 若在程序 $\mathscr{P}$ 中出现变量 V，则 σ 中含有等式 $V=m$，那么称 σ 是程序 $\mathscr{P}$ 的一个**状态**.

例如，例 1.1 的程序中有变量 X 和 Y. 对于这个程序，

$$\{X=5, Y=3\}$$

是一个状态.

$$\{X_1=5, X_2=4, Y=3\}$$

$$\{X=5, Z=6, Y=3\}$$

也是它的状态. 根据定义，当 V 不出现在程序中时，允许状态中包含关于 V 的等式.

$$\{X=5, X=6, Y=3\}$$

不是一个状态，它包含 2 个关于 X 的等式.

$$\{X=5\}$$

也不是这个程序的状态，它缺少关于 Y 的等式.

状态描述程序在执行的某一步各个变量的值. 对于程序中的变量 V，在 σ 中有唯一的等式 $V=m$，表示 V 的当前值等于 m. 此时也称在状态 σ 中 V 的值等于 m. 规定：若 σ 中不含关于 V 的等式(因而程序中不出现 V)，则变量 V 的值自动取 0.

7. 快相

程序的一个**快相**或**瞬时描述**是一个有序对(i,σ)，其中σ是程序的状态，$1\leqslant i\leqslant q+1$，$q$是程序的长度. 快相$(i,\sigma)$表示程序的当前状态为$\sigma$，即将执行第$i$条指令. 当$i=q+1$时，表示计算结束. $(q+1,\sigma)$称作程序的**终点快相**.

除输入变量外，所有变量的值为0的状态称作**初始状态**. 若σ是初始状态，则称$(1,\sigma)$是**初始快相**.

8. 后继

设(i,σ)是程序$\mathscr{P}$的非终点快相，定义它的**后继**(j,τ)如下：

情况1：$\mathscr{P}$的第i条指令是$V\leftarrow V+1$(不带标号或带标号，下同)且σ包含等式$V=m$，则$j=i+1$，而τ由把σ中的$V=m$替换成$V=m+1$得到.

情况2：$\mathscr{P}$的第i条指令是$V\leftarrow V-1$且σ包含等式$V=m$，则$j=i+1$，并且当$m>0$时，把σ中的$V=m$替换成$V=m-1$得到τ；当$m=0$时，$\tau=\sigma$.

情况3：$\mathscr{P}$的第i条指令是$V\leftarrow V$，则$j=i+1$且$\tau=\sigma$.

情况4：$\mathscr{P}$的第i条指令是IF $V\neq 0$ GOTO L且σ包含等式$V=m$，则$\tau=\sigma$，并且当$m=0$时，$j=i+1$；当$m>0$时，若$\mathscr{P}$中有带标号L的指令，则j是$\mathscr{P}$中带标号L的指令的最小序号，即第j条指令是$\mathscr{P}$中带标号L的第一条指令；若$\mathscr{P}$中没有带标号L的指令，则$j=q+1$，q是程序$\mathscr{P}$的长度.

通过后继给出了语句的严格解释. 我们没有限制只能有一条带标号L的指令. 当程序中有多条指令以L为标号时，由IF $V\neq 0$ GOTO L只能转到这些指令中的第一条. 从而，下述程序和例1.1中的程序实际上是一样的，添加在第2条和第3条指令前的标号在计算中不起作用.

$[A]$ $X\leftarrow X-1$

$[A]$ $Y\leftarrow Y+1$

$[A]$ IF $X\neq 0$ GOTO A

9. 计算

设 $s_1, s_2, \cdots$ 是程序 $\mathscr{P}$ 的快相序列，序列的长为 k（对于无穷序列，$k=\infty$）. 如果：(1) s_1 是初始快相；(2) 对于每一个 i $(1 \leqslant i < k)$，s_{i+1} 是 s_i 的后继；(3) 当 $k<\infty$ 时，s_k 是终点快相，则称该序列是 $\mathscr{P}$ 的一个**计算**.

例如，下述 2 个序列都是例 1.5 中程序的计算，

(1) $(1,\{X=1, Z=0, Y=0\})$
$(4,\{X=1, Z=0, Y=0\})$
$(5,\{X=0, Z=0, Y=0\})$
$(6,\{X=0, Z=0, Y=1\})$
$(7,\{X=0, Z=0, Y=1\})$

(2) $(1,\{X=0, Z=0, Y=0\})$
$(2,\{X=0, Z=0, Y=0\})$
$(3,\{X=0, Z=1, Y=0\})$
$(1,\{X=0, Z=1, Y=0\})$
$(2,\{X=0, Z=1, Y=0\})$
$(3,\{X=0, Z=2, Y=0\})$
$(1,\{X=0, Z=2, Y=0\})$
$\vdots$

序列(2)不断地重复执行第 1,2,3 条指令，计算永不休止.

1.3 可计算函数

本章所说的函数均指数论函数.

设 $\mathscr{P}$ 是一个 $\mathscr{S}$ 程序，n 是一个正整数. $\mathscr{P}$ 计算的 n 元部分函数记作 $\varphi_{\mathscr{P}}^{(n)}(x_1, \cdots, x_n)$，规定如下：

对于任给的 n 个数 $x_1, x_2, \cdots, x_n$，构造初始状态 σ，它由下述等式组成：

$$X_1 = x_1,\ X_2 = x_2,\ \cdots,\ X_n = x_n,\ Y = 0$$

以及对于 $\mathscr{P}$ 中其余的变量 V，均有 $V=0$. 记初始快相 $s_1=(1,\sigma)$，有两种可能：

(1) 如果从 s_1 开始的计算是有穷序列 $s_1,s_2,\cdots,s_k$，其中 s_k 是终点快相，则 $\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n)$ 等于 Y 在 s_k 中的值；

(2) 如果从 s_1 开始的计算是一个无穷序列 $s_1, s_2, \cdots$，则 $\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n)\uparrow$.

前面在例 1.1 和例 1.5 中已经给出程序所计算的一元函数.

在上述定义中，对每一个正整数 n，程序 $\mathscr{P}$ 计算一个 n 元部分函数. n 可以等于、也可以大于或小于 $\mathscr{P}$ 中的自变量个数 m. 当 $n>m$ 时，多出的自变量 $x_{m+1},\cdots,x_n$ 不起作用；当 $n<m$ 时，多出的输入变量 $X_{n+1},\cdots,X_m$ 的初始值为 0，即在初始状态中的值为 0. 例如，设 $\mathscr{P}$ 有 2 个自变量，且 $\psi_{\mathscr{P}}^{(2)}(x_1,x_2)=x_1+x_2$，那么 $\psi_{\mathscr{P}}^{(1)}(x)=x+0=x$，$\psi_{\mathscr{P}}^{(3)}(x_1,x_2,x_3)=x_1+x_2$.

定义 1.1 设 $f(x_1,x_2,\cdots,x_n)$ 是一个部分函数，如果存在程序 $\mathscr{P}$ 计算 f，即对任意的 $x_1,x_2,\cdots,x_n$ 有

$$f(x_1,x_2,\cdots,x_n)=\psi_{\mathscr{P}}^{(n)}(x_1,x_2,\cdots,x_n),$$

则称 f 是**部分可计算的**.

如果一个函数既是部分可计算的，又是全函数，则称这个函数是**可计算的**.

定义中的等式的涵义是，等号两边都有定义且值相等，或者两边都没有定义.

[**例 1.6**] 证明 $f(x)=k$ 是可计算的，其中 k 是一个固定的常数.

证：计算 $f(x)=k$ 的程序很简单，由 k 条 $Y\leftarrow Y+1$ 组成：

$$\left.\begin{array}{l} Y\leftarrow Y+1 \\ Y\leftarrow Y+1 \\ \vdots \\ Y\leftarrow Y+1 \end{array}\right\} k\text{ 条}$$

当 $k=0$ 时，这是空程序. 空程序计算零函数 $\mathrm{n}(x)=0$.

不难证明，$x+y$，xy，$x\dot{-}y$ 都是可计算的，其中

$$x\dot{-}y=\begin{cases}x-y & \text{若 } x\geqslant y\\ 0 & \text{否则.}\end{cases}$$

$x-y$ 是部分可计算的. 注意：$x\dot{-}y$ 是全函数，而 $x-y$ 只是部分函数.

可计算函数的概念同样适用于谓词，只需把谓词看作取值 0 或 1 的全函数. 我们把真值等同于 1，假值等同于 0. 如果谓词 $P(x_1,x_2,\cdots,x_n)$ 作为一个全函数是可计算的，则称它是**可计算谓词**.

[**例 1.7**] 证明 $x\neq 0$ 是可计算谓词.

证：只需给出计算

$$P(x)=\begin{cases}1 & \text{若 } x\neq 0\\ 0 & \text{否则}\end{cases}$$

的程序. 程序如下：

IF $X\neq 0$ GOTO A
GOTO E
[A] $Y\leftarrow Y+1$

不难证明 $x>0$，$x\leqslant y$，$x<y$，$x=y$ 等都是可计算的.

虽然语言 $\mathscr{S}$ 很简单，但是我们将会看到它的计算能力是非常强的.

1.4 宏 指 令

在 1.2 节中为了简化程序的书写，把一段常用的程序缩写成一条宏指令. 实际上，对任何一个部分可计算函数和一个可计算谓词都可以构造一条对应的宏指令. 本节考虑两种形式的宏指令：

(1) 一般形式的赋值语句：

$$W\leftarrow f(V_1,V_2,\cdots,V_n).$$

(2) 一般形式的条件转移语句：

$$\text{IF } P(V_1, V_2, \cdots, V_n) \text{ GOTO } L,$$

其中 $f(x_1, x_2, \cdots, x_n)$是部分可计算函数，$P(x_1, x_2, \cdots, x_n)$是可计算谓词.

设程序 $\mathscr{P}$ 计算 $f(x_1, x_2, \cdots, x_n)$. 不失一般性，总可以假设 $\mathscr{P}$ 使用输入变量 $X_1, X_2, \cdots, X_n$，输出变量 Y，中间变量 $Z_1, Z_2, \cdots, Z_k$，标号 $A_1, A_2, \cdots, A_t$，并且程序唯一的“出口”是执行最后一条指令后停止计算. 记

$$\mathscr{P} = \mathscr{P}(Y, X_1, \cdots, X_n, Z_1, \cdots, Z_k; A_1, \cdots, A_t).$$

对于正整数 m，记

$$\mathscr{P}_m = \mathscr{P}(Z_m, Z_{m+1}, \cdots, Z_{m+n}, Z_{m+n+1}, \cdots, Z_{m+n+k}; A_m, \cdots, A_{m+t-1}),$$

$\mathscr{P}_m$ 是对 $\mathscr{P}$ 中的变量和标号做相应地替换后得到的程序.

$W \leftarrow f(V_1, V_2, \cdots, V_n)$的宏展开为：

$$Z_m \leftarrow 0$$
$$Z_{m+1} \leftarrow V_1$$
$$\vdots$$
$$Z_{m+n} \leftarrow V_n$$
$$Z_{m+n+1} \leftarrow 0$$
$$\vdots$$
$$Z_{m+n+k} \leftarrow 0$$
$$\mathscr{P}_m$$
$$W \leftarrow Z_m$$

m 应取得足够大，使得宏展开中的变量和标号不出现在使用这条宏指令的主程序中. 对 Z_m 以及 $Z_{m+n+1}, \cdots, Z_{m+n+k}$置 0 是必要的，因为在程序执行过程中可能要多次执行这段程序. 对第 2 次及其以后的执行，这些置 0 是必不可少的. 这些置 0 和对 $Z_{m+1}, \cdots, Z_{m+n}$ 的赋值在这里都是以宏指令的形式给出的. 当然，对它们的宏展开也要做类似的处理.

$f(x_1,x_2,\cdots,x_n)$是部分可计算函数. 当 $f(V_1,V_2,\cdots,V_n)$有定义时,宏展开计算结束后,已将 $f(V_1,V_2,\cdots,V_n)$的值赋给 W,并进入主程序继续计算;当 $f(V_1,V_2,\cdots,V_n)$无定义时,宏展开中 $\mathscr{P}_m$ 的计算永不停止,从而整个计算也永不停止.

例如,下面有两个程序

$\mathscr{P}_1$:	$\mathscr{P}_2$:
$Z\leftarrow X_1+X_2$	$Z\leftarrow X_1-X_2$
$Y\leftarrow Z-X_2$	$Y\leftarrow Z+X_2$

它们计算的二元函数是

$$\psi^{(2)}_{\mathscr{P}_1}(x_1,x_2)=x_1$$

$$\psi^{(2)}_{\mathscr{P}_2}(x_1,x_2)=\begin{cases}x_1 & \text{若 } x_1\geqslant x_2\\ \uparrow & \text{否则}\end{cases}$$

两者是不同的.

IF $P(V_1,V_2,\cdots,V_n)$ GOTO L 的宏展开是

$$W\leftarrow P(V_1,V_2,\cdots,V_n)$$
$$\text{IF } W\neq 0 \text{ GOTO } L$$

例如,下述宏指令是合法的,

$$\text{IF } V=0 \text{ GOTO } L$$
$$\text{IF } V_1=V_2 \text{ GOTO } L$$

习　题

1. 写一个不使用宏指令的 $\mathscr{S}$ 程序计算函数 $f(x)=5x$.

2. 写出计算下述函数的 $\mathscr{S}$ 程序(允许使用宏指令):

(1) $f(x)=\lfloor x/2\rfloor$,这里$\lfloor x\rfloor$等于不超过 x 的最大整数.

(2) 当 x 为偶数时 $f(x)=1$,当 x 为奇数时 $f(x)$没有定义.

3. 给出下述程序 $\mathscr{P}$ 计算的函数 $\psi^{(1)}_{\mathscr{P}}(x)$:

(1)

$$[A]\quad X\leftarrow X+1$$
$$X\leftarrow X-1$$

(2)

```
        IF X≠0 GOTO A
[A]     X←X−1
        IF X=0 GOTO A
        X←X−1
        IF X≠0 GOTO A
```

(3) 空程序.

4. 证明下述函数是部分可计算的：

(1) x_1+x_2. (2) x_1-x_2. (3) x_1x_2. (4) 空函数.

5. 证明下述谓词是可计算的：

(1) $x\geqslant a$，其中 a 是一个正整数.

(2) $x_1\leqslant x_2$.

(3) $x_1=x_2$.

6. 试写出例 1.2 中的程序从 $X=2$ 的初始快相开始的计算.

第二章　原始递归函数

2.1　原始递归函数

2.1.1　合成

设 f 是 k 元部分函数，$g_1,g_2,\cdots,g_k$ 是 k 个 n 元部分函数. 令

$$h(x_1,\cdots,x_n)=f(g_1(x_1,\cdots,x_n),\cdots,g_k(x_1,\cdots,x_n)),$$

称函数 h 是由 f 和 $g_1,\cdots,g_k$ **合成**得到的.

显然，$h(x_1,\cdots,x_n)$有定义当且仅当 $z_1=g_1(x_1,\cdots,x_n),\cdots,z_k=g_k(x_1,\cdots,x_n)$和 $f(z_1,\cdots,z_k)$都有定义. 如果 f 和 $g_1,\cdots,g_k$ 都是全函数，则 h 也是全函数. 下面证明(部分)可计算函数合成得到的函数也是(部分)可计算的.

定理 2.1　如果 h 是由(部分)可计算函数 f 和 $g_1,\cdots,g_k$ 合成得到的，则 h 也是(部分)可计算函数.

证：计算 h 的程序如下

$$\begin{aligned}&Z_1\leftarrow g_1(X_1,\cdots,X_n)\\&\vdots\\&Z_k\leftarrow g_k(X_1,\cdots,X_n)\\&Y\leftarrow f(Z_1,\cdots,Z_k)\end{aligned}$$

由于 f 和 $g_1,\cdots,g_k$ 是(部分)可计算的，这些宏指令都是合法的.

□

2.1.2　原始递归

设 g 是 2 元全函数，k 是一个常数. 函数 h 由下述等式给出

$$\begin{aligned}&h(0)=k,\\&h(t+1)=g(t,h(t)),\end{aligned}\qquad(2.1)$$

称 h 是由 g 经过**原始递归**运算得到的.

设 f 和 g 分别是 n 元和 $n+2$ 元全函数，$n+1$ 元函数 h 由下述等式给出

$$\begin{aligned} &h(x_1,\cdots,x_n,0)=f(x_1,\cdots,x_n),\\ &h(x_1,\cdots,x_n,t+1)=g(t,h(x_1,\cdots,x_n),x_1,\cdots,x_n), \end{aligned} \tag{2.2}$$

称 h 是由 f 和 g 经过原始递归运算得的.

下面证明，可计算函数经过原始递归运算得到的函数也是可计算的.

定理 2.2 设 h 由(2.2)式给出，如果 f 和 g 都是可计算的，则 h 是可计算的.

证：计算 $h(x_1,\cdots,x_{n+1})$ 的程序如下：

$$\begin{aligned} &Y\leftarrow f(X_1,\cdots,X_n)\\ [A]\quad &\text{IF } X_{n+1}=0 \text{ GOTO } E\\ &Y\leftarrow g(Z,Y,X_1,\cdots,X_n)\\ &Z\leftarrow Z+1\\ &X_{n+1}\leftarrow X_{n+1}-1\\ &\text{GOTO } A \end{aligned}$$

□

(2.1)式可以看作(2.2)式当 $n=0$ 时的特殊情况，此时 f 为某个常数. 例 1.6 已经证明常数 k 是可计算的，故有：

推论 2.3 设 h 由(2.1)式给出，如果 g 是可计算的，则 h 也是可计算的.

2.1.3 原始递归函数

初始函数包括下述函数：

后继函数 $s(x)=x+1$，

零函数 $n(x)=0$，

投影函数 $u_i^n(x_1,\cdots,x_n)=x_i$，　$1\leqslant i\leqslant n$.

定义 2.1 由初始函数经过有限次合成和原始递归得到的函数称作**原始递归函数**.

定理 2.4 由原始递归函数经过合成或原始递归得到的函数仍是原始递归函数.

证:由定义立即得到. □

定理 2.5 每一个原始递归函数都是可计算的.

证:根据定理 2.1, 2.2 和推论 2.3,只需证明初始函数都是可计算的.这是很容易的.$s(x)$用程序

$$X \leftarrow X + 1$$
$$Y \leftarrow X$$

计算,$n(x)$用空程序计算,$u_i^n(x)$用程序

$$Y \leftarrow X_i$$

计算. □

在本章的最后一节将要给出一个非原始递归的可计算函数,从而说明原始递归函数类是可计算函数类的真子集.

2.1.4 常用原始递归函数

下面给出一些常用的原始递归函数.当然这些函数也都是可计算的,从而使我们对语言 $\mathscr{S}$ 的计算能力有进一步的了解.

1. 常数 k

$\underbrace{s(\cdots(s}_{k\text{个}}(n(x))\cdots)=k$,故 $h(x)=k$ 是原始递归函数.

2. x

$u_1^1(x)=x$,它本身就是一个初始函数.

3. $x+y$

记 $h(x,y)=x+y$,可由下式得到

$$h(x,0) = x,$$
$$h(x,y+1) = h(x,y) + 1.$$

即,$x+y$ 可由 $f(x)=x$ 和 $g(y,z,x)=z+1$ 经过原始递归得到,故 $x+y$ 是原始递归函数.下面一般只给出所需要的表达形式,而不再作详细的解释.

4. $x \cdot y$

$x \cdot y$ 可由下式原始递归得到

$$x \cdot 0 = 0,$$
$$x \cdot (y+1) = x \cdot y + x.$$

5. $x!$

$x!$ 可由下式原始递归得到

$$0! = 1,$$
$$(x+1)! = x! \cdot (x+1).$$

6. x^y

原始递归等式为

$$x^0 = 1,$$
$$x^{y+1} = x^y \cdot x.$$

这里规定 $0^0=1$. 由于 x^y 是原始递归函数，故对于任何常数 a，x^a 和 a^y 都是原始递归函数，它们分别由 x^y 与 $y=a$ 和 $x=a$ 合成得到.

7. $p(x)$

前驱函数

$$p(x) = \begin{cases} x-1 & \text{若 } x > 0 \\ 0 & \text{若 } x = 0. \end{cases}$$

它可由下述原始递归等式得到

$$p(0) = 0,$$
$$p(x+1) = x.$$

8. $x \dot{-} y$

原始递归等式为

$$x \dot{-} 0 = x,$$
$$x \dot{-} (y+1) = p(x \dot{-} y).$$

9. $|x-y|$

$|x-y|$ 与通常的理解是一样的，即

$$|x-y| = \begin{cases} x-y & \text{若 } x \geqslant y \\ y-x & \text{否则.} \end{cases}$$

它可表示成

$$|x-y| = (x \dot{-} y) + (y \dot{-} x).$$

10. $\alpha(x)$

$\alpha(x)$的定义为

$$\alpha(x) = \begin{cases} 1 & \text{若 } x = 0 \\ 0 & \text{否则.} \end{cases}$$

我们有

$$\alpha(x) = 1 \dot{-} x.$$

实际上,$\alpha(x)$相当于谓词 $x=0$.

2.2 原始递归谓词

正如前面说过的那样,谓词就是一个取值 0 或 1 的全函数,因此原始递归函数的概念同样适用于谓词. 即,如果一个谓词作为 0 或 1 的全函数是原始递归的,则称作**原始递归谓词**. 下面接着给出常用的原始递归谓词.

1. $x=y$

谓词 $x=y$ 可以看作函数

$$e(x,y) = \begin{cases} 1 & \text{若 } x = y \\ 0 & \text{否则.} \end{cases}$$

它可表示为

$$e(x,y) = \alpha(|x-y|),$$

故 $x=y$ 是原始递归谓词.

2. $x \leqslant y$

它可表示为:$\alpha(x \dot{-} y)$.

定理 2.6 如果 P,Q 是原始递归谓词(可计算谓词),则$\neg P$,$P \vee Q$ 和 $P \wedge Q$ 也是原始递归谓词(可计算谓词).

证：由下述 3 个表达式，立即得到所需的结论：

$$\neg P(x_1,\cdots,x_n) = \alpha(P(x_1,\cdots,x_n)),$$

$$P(x_1,\cdots,x_n) \wedge Q(x_1,\cdots,x_n)$$
$$= P(x_1,\cdots,x_n) \cdot Q(x_1,\cdots,x_n),$$

$$P(x_1,\cdots,x_n) \vee Q(x_1,\cdots,x_n)$$
$$= \neg (\neg P(x_1,\cdots,x_n) \wedge \neg Q(x_1,\cdots,x_n)). \qquad \square$$

由定理 2.6 可得到下述两个原始递归谓词.

3. $x<y$

因为 $x<y \Leftrightarrow \neg (y \leqslant x)$.

4. $x \neq y$

因为 $x \neq y \Leftrightarrow \neg (x=y)$.

由上面的几个原始递归谓词，对于任意的常数 a，谓词 $x=a$，$x \leqslant a$，$x<a$ 和 $x \neq a$ 都是原始递归的.

定理 2.7 如果 g 和 h 是 n 元原始递归(可计算)函数，P 是 n 元原始递归(可计算)谓词，令

$$f(x_1,\cdots,x_n) = \begin{cases} g(x_1,\cdots,x_n) & \text{若 } P(x_1,\cdots,x_n) \\ h(x_1,\cdots,x_n) & \text{否则,} \end{cases}$$

则 f 也是原始递归(可计算)函数.

证：由下式，结论显然成立.

$$f(x_1,\cdots,x_n) = g(x_1,\cdots,x_n) \cdot P(x_1,\cdots,x_n)$$
$$+ h(x_1,\cdots,x_n) \cdot \alpha(P(x_1,\cdots,x_n)). \qquad \square$$

实际上还可以有下述结论：如果 g 和 h 是 n 元部分可计算函数，P 是 n 元可计算谓词，则 f 是 n 元部分可计算函数. 因此，当 g 和 h 是 n 元部分可计算函数，P 是 n 元可计算谓词时，在 $\mathscr{S}$ 程序中可以使用下述形式的“条件语句”：

IF $P(V_1,\cdots,V_n)$ THEN $W \leftarrow g(V_1,\cdots,V_n)$
ELSE $W \leftarrow h(V_1,\cdots,V_n)$.

2.3 迭代运算、有界量词和极小化

2.3.1 迭代运算

定理 2.8 设 $f(x_1,\cdots,x_n,x_{n+1})$是原始递归(可计算)函数，则

$$g(x_1,\cdots,x_n,y)=\sum_{t=0}^{y}f(x_1,\cdots,x_n,t)$$

和

$$h(x_1,\cdots,x_n,y)=\prod_{t=0}^{y}f(x_1,\cdots,x_n,t)$$

也是原始递归(可计算)函数.

证：由下述递归等式得证定理，

$$g(x_1,\cdots,x_n,0)=f(x_1,\cdots,x_n,0),$$

$$g(x_1,\cdots,x_n,y+1)=g(x_1,\cdots,x_n,y)+f(x_1,\cdots,x_n,y+1),$$

和

$$h(x_1,\cdots,x_n,0)=f(x_1,\cdots,x_n,0),$$

$$h(x_1,\cdots,x_n,y+1)=h(x_1,\cdots,x_n,y)\cdot f(x_1,\cdots,x_n,y+1)$$

□

有时需要考虑从 1(而不是从 0)开始的求和或求积，即考虑

$$\sum_{t=1}^{y}f(x_1,\cdots,x_n,t)\quad 和\quad \prod_{t=1}^{y}f(x_1,\cdots,x_n,t).$$

只要将上述定理证明中的 2 个初值改为

$$g(x_1,\cdots,x_n,0)=0$$

和

$$h(x_1,\cdots,x_n,0)=1,$$

就可得到下述结论：

推论 2.9 设 $f(x_1,\cdots,x_n,x_{n+1})$是原始递归(可计算)函数，则

$$g(x_1,\cdots,x_n,y)=\sum_{t=1}^{y}f(x_1,\cdots,x_n,t)$$

和

$$h(x_1,\cdots,x_n,y)=\prod_{t=1}^{y}f(x_1,\cdots,x_n,t)$$

也是原始递归(可计算)函数.

这里我们约定:空和($\sum_{t=1}^{0}$)等于 0,空积($\prod_{t=1}^{0}$)等于 1.

2.3.2 有界量词

定理 2.10 设 $P(x_1,\cdots,x_n,y)$是原始递归(可计算)谓词,则

$$(\forall\, t)_{\leqslant y}\,P(x_1,\cdots,x_n,t)$$

和

$$(\exists\, t)_{\leqslant y}\,P(x_1,\cdots,x_n,t)$$

也是原始递归(可计算)谓词.

证:注意到

$$(\forall\, t)_{\leqslant y}\,P(x_1,\cdots,x_n,t)\Leftrightarrow\prod_{t=0}^{y}P(x_1,\cdots,x_n,t)=1,$$

$$(\exists\, t)_{\leqslant y}\,P(x_1,\cdots,x_n,t)\Leftrightarrow\sum_{t=0}^{y}P(x_1,\cdots,x_n,t)\neq 0,$$

根据定理 2.8 以及 $x=1,x\neq 0$ 都是原始递归谓词,得证结论成立. □

有时需要使用量词$(\forall\, t)_{<y}$或$(\exists\, t)_{<y}$. 由于

$$(\forall\, t)_{<y}P(x_1,\cdots,x_n,t)\Leftrightarrow(\forall\, t)_{\leqslant y}[t=y\vee P(x_1,\cdots,x_n,t)],$$

$$(\exists\, t)_{<y}P(x_1,\cdots,x_n,t)\Leftrightarrow(\exists\, t)_{\leqslant y}[t\neq y\wedge P(x_1,\cdots,x_n,t)],$$

由上述定理立即得到:

推论 2.11 设 $P(x_1,\cdots,x_n,y)$是原始递归(可计算)谓词,则:

$$(\forall\, t)_{<y}P(x_1,\cdots,x_n,t)$$

和

$$(\exists\, t)_{<y}P(x_1,\cdots,x_n,t)$$

也是原始递归(可计算)谓词.

下面是两个常用的原始递归谓词.

1. $y|x$

$y|x$ 表示 y 可以整除 x,它是原始递归的,因为

$$y|x \Leftrightarrow (\exists\ t)_{\leqslant x}(y \cdot t = x).$$

这里约定:0 整除 0.

2. Prime(x)

Prime(x)表示 x 是素数. 它也是原始递归的,因为

$$\text{Prime}(x) \Leftrightarrow x > 1 \wedge (\forall\ t)_{<x}[t = 1 \vee \neg (t|x)].$$

2.3.3 极小化

设 $P(x_1,\cdots,x_n,t)$是一个谓词,考虑函数

$$g(x_1,\cdots,x_n,y) = \sum_{u=0}^{y}\prod_{t=0}^{u}\alpha(P(x_1,\cdots,x_n,t)).$$

设 $t_0 \leqslant y$ 是使 $P(x_1,\cdots,x_n,t)$为真的 t 的最小值,即

$$P(x_1,\cdots,x_n,t_0) = 1$$

并且对所有的 $t<t_0$,有

$$P(x_1,\cdots,x_n,t) = 0.$$

于是,

$$\prod_{t=0}^{u}\alpha(P(x_1,\cdots,x_n,t)) = \begin{cases} 1 & \text{若 } u < t_0 \\ 0 & \text{若 } u \geqslant t_0 \end{cases}$$

从而,

$$g(x_1,\cdots,x_n,y) = t_0$$

即,当存在 $t \leqslant y$ 使得 $P(x_1,\cdots,x_n,t)$为真时,函数 $g(x_1,\cdots,x_n,y)$ 的值等于使 $P(x_1,\cdots,x_n,t)$为真的 t 的最小值.

定义 2.2 设 $P(x_1,\cdots,x_n,t)$是一个谓词,定义

$$\min_{t \leqslant y} P(x_1,\cdots,x_n,t)$$

$$= \begin{cases} g(x_1,\cdots,x_n,y) & \text{若}(\exists\, t)_{\leqslant y}P(x_1,\cdots,x_n,t) \\ 0 & \text{否则}^{①} \end{cases}$$

称运算"$\min_{t\leqslant y}$"为**有界极小化**，y 是极小化的上界.

根据定理 2.7，2.8 和 2.10，我们有

定理 2.11 设 $P(x_1,\cdots,x_n,t)$是原始递归(可计算)谓词，则

$$f(x_1,\cdots,x_n,y) = \min_{t\leqslant y}P(x_1,\cdots,x_n,t)$$

是原始递归(可计算)函数.

下面继续给出三个常用的原始递归函数.

1. $\lfloor x/y\rfloor$

$\lfloor x/y\rfloor$是 x 除以 y 的整数部分. 由等式

$$\lfloor x/y\rfloor = \min_{t\leqslant x}\{(t+1)\cdot y > x\},$$

得证它是原始递归的. 这里取$\lfloor x/0\rfloor=0$.

2. $R(x,y)$

$R(x,y)$等于 x 除以 y 的余数. 它的原始递归性由下式可得

$$R(x,y) = x \dot{-} (\lfloor x/y\rfloor \cdot y).$$

根据这个式子，我们取 $R(x,0)=x$.

3. p_n

定义 $p_0=0$；当 $n>0$ 时，p_n 是第 n 个素数(按从小到大的顺序). 例如，$p_0=0, p_1=2, p_2=3, p_3=5, p_4=7, p_5=11,\cdots$. p_n 是 n 的函数.

引理 2.12 $p_{n+1}\leqslant (p_n)!+1$

证：对每一个 $i(1\leqslant i\leqslant n)$

$$\frac{(p_n)!+1}{p_i} = K + \frac{1}{p_i},$$

其中 K 是一个整数，故$(p_n)!+1$ 不被 $p_1,p_2,\cdots,p_n$ 整除. 因此，$(p_n)!+1$ 或者本身是一个素数，或者被一个大于 p_n 的素数整除. 总之，必存在大于 p_n 且小于等于$(p_n)!+1$ 的素数. 所以，$p_{n+1}\leqslant$

① 在有的书中把这个值规定为 $y+1$.

$(p_n)!+1$. □

由引理 2.12，有下述递归等式

$$p_0 = 0,$$

$$p_{n+1} = \min_{t \leqslant (p_n)!+1}\{\mathrm{Prime}(t) \wedge t > p_n\}.$$

为了说明第二个等式右端是一个原始递归函数. 令

$$h(y,z) = \min_{t \leqslant z}\{\mathrm{Prime}(t) \wedge t > y\},$$

由定理 2.11，$h(y,z)$是原始递归的. 上述递归等式可表示为

$$p_0 = 0,$$

$$p_{n+1} = h(p_n,(p_n)!+1),$$

得证 p_n 是原始递归的.

下面给出没有上界限制的极小化，并作初步讨论.

定义 2.3 设 $P(x_1,\cdots,x_n,t)$是一个谓词，如果存在 t 使 $P(x_1,\cdots,x_n,t)$为真，则$\min\limits_t P(x_1,\cdots,x_n,t)$等于使 $P(x_1,\cdots,x_n,t)$为真的 t 的最小值；否则$\min\limits_t P(x_1,\cdots,x_n,t)$没有定义. 运算“$\min\limits_t$”称作**极小化**.

$f(x_1,\cdots,x_n)=\min\limits_t P(x_1,\cdots,x_n,t)$是一个 n 元部分函数，称 f 是由谓词 P 经过极小化运算得到的.

设 $g(x_1,\cdots,x_n,t)$是一个 $n+1$ 元全函数，若

$$f(x_1,\cdots,x_n) = \min_t\{g(x_1,\cdots,x_n,t) = 0\},$$

则称 f 是由函数 g 经过极小化运算得到的.

定义 2.4 由初始函数经过有限次合成、原始递归和极小化运算得到的函数称作**部分递归函数**. 部分递归的全函数称作**递归函数**.

如果一个谓词作为取值 1(真值)或 0(假值)的全函数是递归的，则称这个谓词是**递归谓词**.

如，$x-y=\min\limits_t(|x-(y+t)|=0)$是一个部分递归函数. 由定义，原始递归函数都是递归函数，原始递归谓词都是递归谓词.

定理 2.13 设 $P(x_1,\cdots,x_n,t)$是可计算谓词，则$\min_t P(x_1,\cdots,x_n,t)$是部分可计算的.

证：下述程序计算这个函数：

$[A]$ IF $P(X_1,\cdots,X_n,Y)$ GOTO E

$Y\leftarrow Y+1$

GOTO A □

推论 2.14 设 $g(x_1,\cdots,x_n,t)$是可计算函数，则$\min_t\{g(x_1,\cdots,x_n,t)=0\}$是部分可计算的.

定理 2.15 部分递归函数是部分可计算函数.

证：因为初始函数都是可计算的，根据定理 2.1，2.2 以及推论 2.3，2.4，所以部分递归函数是部分可计算的. □

推论 2.16 递归函数是可计算函数，递归谓词是可计算谓词.

事实上，定理 2.15 和推论 2.16 的逆也成立，即部分递归函数、递归函数、递归谓词分别就是部分可计算函数、可计算函数、可计算谓词. 暂且将它们放下，待到第七章再讨论.

2.4 配对函数和 Gödel 数

配对函数和Gödel数是对数偶和有穷数列的一种编码方法.

2.4.1 配对函数

令

$$\langle x,y\rangle = 2^x(2y+1) \dot{-} 1,$$

$\langle x,y\rangle$称作配对函数. 它是一个原始递归函数. 由于 $2^x(2y+1)\neq 0$，减号上的点 · 可省去，写成

$$\langle x,y\rangle = 2^x(2y+1) - 1.$$

例如，$\langle 2,3\rangle = 2^2(2\times 3+1)-1=27$.

反之，任给一个数 z，存在唯一的一对数 x 和 y 使

$$\langle x,y\rangle = z.$$

x 是 $z+1$ 含有的因子 2 的个数，即使 $2^t \mid (z+1)$ 的 t 的最大值. $(z+1)/2^x$ 必为奇数，y 是

$$2y+1 = (z+1)/2^x$$

的唯一解. 显然，$x\leqslant z, y\leqslant z$.

记

$$l(z) = x, \qquad r(z) = y,$$

则有

$$l(z) = \min_{t\leqslant z}\{\neg(2^{t+1}\mid (z+1))\},$$

$$r(z) = \lfloor([\lfloor (z+1)/2^{l(z)}\rfloor \dot{-} 1)/2\rfloor.$$

这表明，$l(z)$ 和 $r(z)$ 是原始递归函数. 实际上，在 $r(z)$ 的表达式中取整号是不必要的.

例如，$27+1=28=2^2\times 7$，得 $l(27)=2$. 又 $(27+1)/2^2=7=2\times 3+1$，得 $r(27)=3$.

综上所述，我们有：

定理 2.17 函数 $\langle x,y\rangle$，$l(z)$ 和 $r(z)$ 有下述性质：

(1) $\langle x,y\rangle$，$l(z)$ 和 $r(z)$ 都是原始递归函数；

(2) $l(\langle x,y\rangle)=x$，$r(\langle x,y\rangle)=y$；

(3) $\langle l(z),r(z)\rangle = z$；

(4) $l(z)\leqslant z$，$r(z)\leqslant z$.

用 $z=\langle x,y\rangle$ 作为一对数偶 x 和 y 的编码. 定理 2.17(2)和(3)表明，编码和它表示的一对数偶是一一对应的.

2.4.2 Gödel 数

记

$$[a_1,a_2,\cdots,a_n] = \prod_{i=1}^{n} p_i^{a_i},$$

$[a_1,a_2,\cdots,a_n]$称作有穷数列$(a_1,a_2,\cdots,a_n)$的**Gödel 数**. 例如：

$$[2,0,1,3]=2^2\cdot 3^0\cdot 5^1\cdot 7^3=6860.$$

根据定义，对于每一个固定的 n，$[a_1,a_2,\cdots,a_n]$是原始递归函数. 用 Gödel 数作为有穷数列的编码，根据整数素因子分解的唯一性，这种编码具有下述唯一性.

定理 2.18 如果$[a_1,a_2,\cdots,a_n]=[b_1,b_2,\cdots,b_n]$，则

$$a_i=b_i,\quad i=1,2,\cdots,n.$$

应该注意到

$$[a_1,\cdots,a_n]=[a_1,\cdots,a_n,0]=[a_1,\cdots,a_n,0,0]$$
$$=\cdots=[a_1,\cdots,a_n,0,\cdots,0].$$

即，在有穷数列的右端添加任意有限个 0，其 Gödel 数不变. 这是 Gödel 数的一个“缺陷”，但它不会妨碍 Gödel 数的使用.

由于 $1=2^0=2^0\cdot 3^0=2^0\cdot 3^0\cdot 5^0=\cdots$，故 1 是数列(0)，(0，0)，(0，0，0)，…的 Gödel 数. 而这些数列都是在空数列(长度为 0 的数列，即不含任何数的数列)的右端添加若干个 0，所以我们约定空数列的 Gödel 数等于 1.

设 $x=[a_1,\cdots,a_n]$，记

$$(x)_i=\begin{cases}a_i & \text{若 } i=1,2,\cdots,n\\ 0 & \text{否则，}\end{cases}$$

并且规定对于每一个 i，$(0)_i=0$. 于是，我们定义了一个 2 元全函数$(x)_i$，以 x 和 i 为自变量.

不难看到

$$(x)_i=\min_{i\leqslant x}\{\neg\,(p_i^{i+1}\mid x)\}.$$

因此，$(x)_i$ 是原始递归函数. 根据有界极小化的定义，上式对于 $(0)_i$ 和 $(x)_0$ 这些特殊情况也都成立.

把以 x 为 Gödel 数的最短数列(最后一个数不为 0 的数列)的长度记作 $\mathrm{Lt}(x)$，即当 $x=0$ 或 $x=1$ 时，$\mathrm{Lt}(x)=0$；当 $x>1$ 时，$\mathrm{Lt}(x)=n$，其中$(x)_n>0$ 并且对所有的 $i>n$ 有$(x)_i=0$，亦即 p_n 整除 x 并且所有的 $p_i(i>n)$不整除 x.

根据定义，有下述等式

$$\mathrm{Lt}(x) = \min_{i\leqslant x}\{(\forall\ j)_{\leqslant x}(j \leqslant i \vee (x)_j = 0)\},$$

故 $\mathrm{Lt}(x)$是原始递归的.

现将关于 Gödel 数的主要性质综合如下：

定理 2.19

(1) 下述函数都是原始递归的：

a. 对每一个固定的 n，$[a_1,\cdots,a_n]$.

b. $(x)_i$，这里把它看作 x 和 i 的 2 元函数.

c. $\mathrm{Lt}(x)$.

(2)

$$([a_1,\cdots,a_n])_i = \begin{cases} a_i & \text{若 } 1\leqslant i\leqslant n \\ 0 & \text{否则.} \end{cases}$$

(3) 如果 $n\geqslant \mathrm{Lt}(x)$且 $x\neq 0$，则$[(x)_1,\cdots,(x)_n]=x$.

2.5 原始递归运算

在 2.1 节给出最基本的原始递归运算形式(2.1)和(2.2). 本节介绍另外三种较为复杂的原始递归形式，它们是多变量递归、多步递归和联立递归. 证明原始递归函数和可计算函数在这些递归运算下是封闭的. 证明使用的主要工具是配对函数和 Gödel 数. 因此，本节既是对 2.1 节的补充、又是配对函数和 Gödel 数的应用.

2.5.1 联立递归

设 f_1 和 f_2 是两个 n 元全函数(当 $n=0$ 时是两个常数)，g_1 和 g_2 是两个 $n+3$ 元全函数. 为方便起见，记 $\boldsymbol{x}=(x_1,\cdots,x_n)$. 例如 $f_1(\boldsymbol{x})=f_1(x_1,x_2,\cdots,x_n)$，$g_1(t,z_1,z_2,\boldsymbol{x})=g_2(t,z_1,z_2,x_1,\cdots,x_n)$ 等. 下述递归等式可以定义两个 $n+1$ 元全函数 h_1 和 h_2：

$$
\begin{aligned}
&h_1(x,0) = f_1(x)\\
&h_2(x,0) = f_2(x)\\
&h_1(x,t+1) = g_1(t,h_1(x,t),h_2(x,t),x)\\
&h_2(x,t+1) = g_2(t,h_1(x,t),h_2(x,t),x)
\end{aligned}
\tag{2.3}
$$

这是由函数的有序对(f_1,f_2)和(g_1,g_2)递归得到有序对(h_1,h_2). 为了将(2.3)化成(2.2)的形式,令

$$H(x,y) = \langle h_1(x,y),h_2(x,y)\rangle,$$

$$F(x) = \langle f_1(x),f_2(x)\rangle,$$

$$G(y,z,x) = \langle g_1(y,l(z),r(z),x),g_2(y,l(z),r(z),x)\rangle.$$

有

$$
\begin{aligned}
&H(x,0) = F(x),\\
&H(x,t+1) = G(t,H(x,t),x).
\end{aligned}
\tag{2.4}
$$

定理 2.20 设f_1,f_2和g_1,g_2都是原始递归(可计算)函数,则由(2.3)式定义的h_1和h_2也是原始递归(可计算)函数.

证:由已知条件,F和G是(原始)递归的. 根据(2.4)式和定理 2.3、2.4,H是(原始)递归的,从而$h_1=l(H),h_2=r(H)$也是(原始)递归的. □

(2.3)式很容易推广到$m\geqslant 3$个函数的情况,由m个n元全函数$f_1,\cdots,f_m$和m个$n+m+1$元全函数$g_1,\cdots,g_m$联立递归得到m个$n+1$元全函数$h_1,\cdots,h_m$. 也有与上述定理相同的结论.

2.5.2 多步递归

多步递归又叫**串值递归**. 在(2.1)和(2.2)式中,在递归计算的过程中每一步只用到上一步的函数值. 现在把它推广到使用前面已经算得的若干个函数值,甚至全部函数值. Fibonacci 数列就是一个典型的例子,它由下述递归等式给出:

$$
\begin{aligned}
&f_0 = 1,\\
&f_1 = 1,\\
&f_{t+1} = f_t + f_{t-1},\quad t\geqslant 1.
\end{aligned}
$$

从计算 f_2 开始，每一步用到前面的 2 个函数值. 为了说明原始递归函数和可计算函数的递归运算是封闭的，我们先看一个更一般性的例子.

设

$$\begin{aligned}&\phi(0)=1,\\&\phi(t+1)=\phi(t)+2\phi(t-1)+\cdots+(t+1)\phi(0).\end{aligned}\tag{2.5}$$

它的函数值依次是 1,1,3,8,21,…. 为了说明这个函数是原始递归的，我们要设法把(2.5)式化成(2.1)的形式. 这里是由数列$(\phi(0),\phi(1),\cdots,\phi(t))$计算出 $\phi(t+1)$，因而需要把这个数列看成一个数. 可以利用 Gödel 数进行编码. 令

$$\Phi(t)=[\phi(0),\phi(1),\cdots,\phi(t)]=\prod_{i=0}^{t}p_{i+1}^{\phi(i)},$$

有

$$\phi(i)=(\Phi(t))_{i+1},\quad 0\leqslant i\leqslant t.$$

于是

$$\begin{aligned}\phi(t+1)&=\sum_{i=0}^{t}(i+1)\phi(t-i)\\&=\sum_{i=0}^{t}(i+1)(\Phi(t))_{t-i+1}\end{aligned}$$

令 $g(t,y)=\sum_{i=0}^{t}(i+1)(y)_{t-i+1}$，上式可写成

$$\phi(t+1)=g(t,\Phi(t)).$$

从而得到下述关于 $\Phi(t)$的递归等式：

$$\begin{aligned}\Phi(0)&=[\phi(0)]=2,\\\Phi(t+1)&=[\phi(0),\phi(1),\cdots,\phi(t+1)]\\&=[\phi(0),\phi(1),\cdots,\phi(t)]\cdot p_{t+2}^{\phi(t+1)}\\&=\Phi(t)\cdot p_{t+2}^{g(t,\Phi(t))}.\end{aligned}$$

令 $G(t,y)=y\cdot p_{t+2}^{g(t,y)}$，上述递归等式可表示成

$$\begin{aligned}&\Phi(0)=2,\\&\Phi(t+1)=G(t,\Phi(t)).\end{aligned}$$

因为 $g(t,y)$是原始递归函数，进而 $G(t,y)$也是原始递归函数，所以 $\Phi(t)$是原始递归函数. 从而，得证 $\phi(t)=(\Phi(t))_{t+1}$是原始递归函数.

一般地，设 $f(x)$是 n 元全函数，$g(t,y,x)$是 $n+2$ 元全函数，这里 $x=(x_1,\cdots,x_n)$. 考虑由下述递归等式定义的 $n+1$ 元全函数 $h(x,t)$：

$$\begin{aligned} &h(x,0)=f(x),\\ &h(x,t+1)=g(t,[h(x,0),\cdots,h(x,t)],x). \end{aligned} \tag{2.6}$$

令

$$\begin{aligned} H(x,t)&=[h(x,0),\cdots,h(x,t)],\\ F(x)&=2^{f(x)},\\ G(t,y,x)&=y\cdot p_{t+2}^{g(t,y,x)}, \end{aligned}$$

则

$$\begin{aligned} &H(x,0)=F(x),\\ &H(x,t+1)=G(t,H(x,t),x). \end{aligned} \tag{2.7}$$

如果 f 和 g 是原始递归(可计算)函数，则 F 和 G 也是原始递归(可计算)函数. 由(2.7)式，H 是原始递归(可计算)函数，从而 $h(x,t)=(H(x,t))_{t+1}$也是原始递归(可计算)函数. 于是，得到下述定理：

定理 2.21 设 f 和 g 是原始递归(可计算)函数，则由(2.6)式给出的 h 也是原始递归(可计算)函数.

经常使用的多步递归是在每一步用到前面的 k 个函数值，这里 $k\geqslant 2$ 是一个固定常数. 如 Fibonacci 数列就是这样.

设 $f_0,f_1,\cdots,f_{k-1}$是 k 个 n 元全函数，g 是 $n+k+1$ 元全函数，$\omega_1,\omega_2,\cdots,\omega_k$ 是 k 个 $n+1$ 元全函数，并且对任意的 x 和 t，$\omega_i(x,t)\leqslant t, i=1,2,\cdots,k$. $n+1$ 元函数 h 由下述等式给出：

$$\begin{aligned} &h(x,0)=f_0(x),\\ &\vdots\\ &h(x,k-1)=f_{k-1}(x), \end{aligned}$$

$$h(x,t+1)=g(t,h(x,\omega_1(x,t)),$$
$$\cdots,h(x,\omega_k(x,t)),x),t\geqslant k-1. \tag{2.8}$$

令

$$g'(t,y,x)=\begin{cases}f_{t+1}(x) & \text{若 } t<k-1\\ g(t,(y)_{\omega_1(x,t)+1},\cdots,(y)_{\omega_k(x,t)+1},x), & \text{若 } t\geqslant k-1\end{cases}$$

则(2.8)可表成(2.6)的形式：

$$h(x,0)=f_0(x),$$
$$h(x,t+1)=g'(t,[h(x,0),\cdots,h(x,t)],x).$$

于是，我们有：

推论 2.22 设 $f_0,\cdots,f_{k-1},\omega_1,\cdots,\omega_k$ 和 g 都是原始递归(可计算)函数，且对任意的 $x_1,\cdots,x_n,t$ 有 $\omega_i(x_1,\cdots,x_n,t)\leqslant t,1\leqslant i\leqslant k$，则由(2.8)式定义的 h 也是原始递归(可计算)函数.

2.5.3 多变量递归

前面讲的递归都是单变量递归.在递归计算过程中只有一个变量 t 在变动，而其余的变量保持不变，实际上都是参数.

现在考虑双变量递归.设 f_1 和 f_2 是$n+1$元全函数，g 是 $n+4$ 元全函数，$n+2$ 元全函数 h 由下述等式给出

$$h(x,0,t_2)=f_1(x,t_2),$$
$$h(x,t_1+1,0)=f_2(x,t_1), \tag{2.9}$$
$$h(x,t_1+1,t_2+1)=g(t_1,t_2,h(x,t_1+1,t_2),h(x,t_1,t_2+1),x).$$

在第 2 式等号的左端用 $h(x,t_1+1,0)$面不是用 $h(x,t_1,0)$，是为了避免第1、2 两个式子同时给出 $h(x,0,0)$的值而可能产生冲突.如果在第 2 式用 $h(x,t_1,0)$，则必须对 f_1 和 f_2 加以限制：$f_1(x,0)=f_2(x,0)$.

根据(2.9)式，h 的值可像图 2.1 那样逐个计算.在两条坐标轴上 h 的值已知，逐条计算对角线上 h 的值.(t_1,t_2)点在第 t_1+t_2 条对角线上.第 $k+1$ 条对角线上的点的 h 值由第 k 条对角线上相邻两点的 h 值算得.

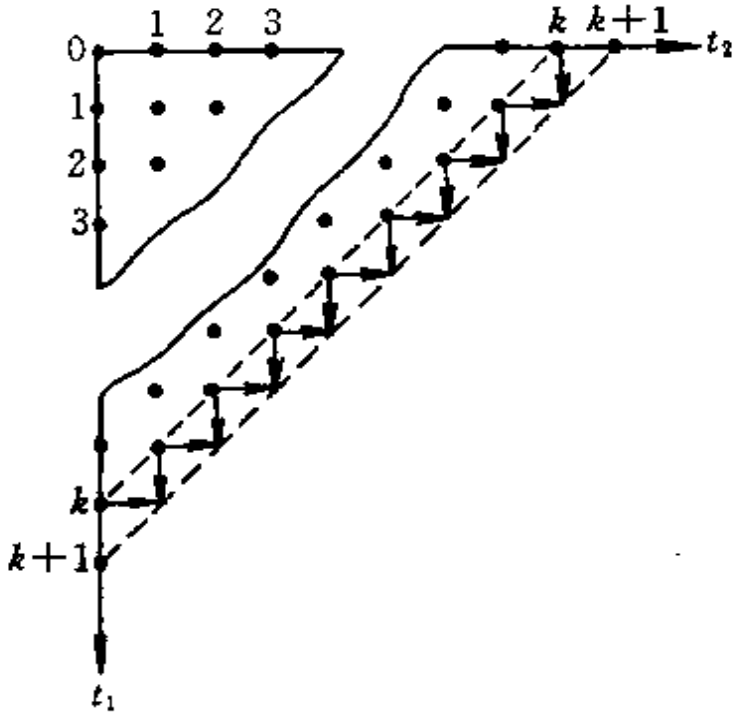


图 2.1

把 h 在第 k 条对角线上的函数值数列的 Gödel 数记作 $H(x,k)$，即

$$H(x,k) = [h(x,0,k),h(x,1,k-1),\cdots,h(x,k,0)]$$
$$= \prod_{i=0}^{k} p_{i+1}^{h(x,i,k-i)}.$$

由(2.9)式得，

$$H(x,0) = 2^{f_1(x,0)},$$

$$H(x,k+1) = p_1^{h(x,0,k+1)} \cdot \{\prod_{i=1}^{k} p_{i+1}^{h(x,i,k+1-i)}\} \cdot p_{k+2}^{h(x,k+1,0)}$$

$$= p_1^{f_1(x,k+1)} \cdot \{\prod_{i=1}^{k} p_{i+1}^{g(i-1,k-i,h(x,i,k-i),h(x,i-1,k+1-i),x)}\}$$

$$\cdot p_{k+2}^{f_2(x,k)}.$$

令

$$F(x) = 2^{f_1(x,0)},$$

$$G(k,y,x) = p_1^{f_1(x,k+1)} \cdot \{\prod_{i=1}^{k} p_{i+1}^{g(i-1,k-i,(y)_{i+1},(y)_i,x)}\} \cdot p_{k+2}^{f_2(x,k)},$$

则上述等式可写成

$$H(x,0) = F(x),$$
$$H(x,k+1) = G(k,H(x,k),x).$$

注意到 $h(x,t_1,t_2)=(H(x,t_1+t_2))_{t_1+1}$，从而有：

定理 2.23 设 f_1,f_2 和 g 都是原始递归(可计算)函数，则由(2.9)式定义的函数 h 也是原始递归(可计算)函数.

2.6 Ackermann 函数

Ackermann 函数是可计算的、但不是原始递归的，从而证明原始递归函数类是可计算函数类的真子集.

Ackermann 函数的定义如下：

$$A(0,x) = x+1,$$
$$A(k+1,0) = A(k,1),$$
$$A(k+1,x+1) = A(k,A(k+1,x)).$$

它最初的一些值如表 2-1 所示. 函数 $A(k,x)$ 的值随着自变量，特别是随着 k 的增加而增加的速度非常快. 为了在直觉上对此有更深刻的印象，让我们来看它的一个变种 $B(k,x)$，它们仅当 $x=0$ 时不同.

表 2-1 Ackermann 函数

k \ x	0	1	2	3	4	5
0	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	5	7	9	11	13
3	5	13	29	61	125	253
4	13	32765	…			
5	32765	…				

$B(k,x)$的定义如下：

$$B(0,x)=x+1,$$

$$B(k,0)=\begin{cases}2 & \text{若 } k=1\\ 0 & \text{若 } k=2\\ 1 & \text{若 } k\geqslant 3,\end{cases}$$

$$B(k+1,x+1)=B(k,B(k+1,x)).$$

记 $f_k(x)=B(k,x)$. 于是，

$$f_0(x)=x+1$$

$$f_k(0)=\begin{cases}2 & \text{若 } k=1\\ 0 & \text{若 } k=2\\ 1 & \text{若 } k\geqslant 3,\end{cases}$$

$$f_{k+1}(x+1)=f_k(f_{k+1}(x)).$$

最初几个 $f_k(x)$如下：

$$f_0(x)=x+1,$$

$$f_1(x)=x+2,$$

$$f_2(x)=2x,$$

$$f_3(x)=2^x,$$

$$f_4(x)=\underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{x\text{个}},$$

$$f_5(x)=\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}\ \text{个}\ 2^{2^{\cdot^{\cdot^{\cdot^2}}}}\ \text{个}\ \cdots\ 2^2\ \text{个}\ 2\ \text{个}}_{x\text{个}},$$

例如，$f_5(0)=1$，$f_5(1)=2$，$f_5(2)=2^2=4$，$f_5(3)=2^{2^2}=65536$，

$f_5(4)=2^{2^{\cdot^{\cdot^{\cdot^2}}}}$ 65536 个. 函数值的增长速度令人吃惊！这也正是Ackermann函数不是原始递归函数的原因所在.

首先给出 Ackermann 函数的一些性质.

引理 2.24 Ackermann 函数具有下述性质：

(1) $A(k,x)>x$；

(2) $A(k,x+1)>A(k,x)$；

(3) $A(k,x)>k$；

(4) $A(k+1,x)>A(k,x)$;

(5) $A(k+1,x)\geqslant A(k,x+1)$;

(6) $A(k+2,x)>A(k,2x)$.

证:(1) 对 k 作归纳证明:当 $k=0$ 时,$A(0,x)=x+1>x$,结论成立. 假设对 k 结论成立,即对任意的 x,$A(k,x)>x$. 要证对 $k+1$结论也成立,即对任意的 x,$A(k+1,x)>x$.

为此,再对 x 作归纳证明. 当 $x=0$ 时,$A(k+1,0)=A(k,1)$. 由对 k 的归纳假设,$A(k,1)>1$. 得证 $A(k+1,0)>0$,结论成立. 假设对 x 结论成立,即 $A(k+1,x)>x$. 要证对 $x+1$ 结论也成立,即 $A(k+1,x+1)>x+1$.

由对 k 的归纳假设,$A(k+1,x+1)=A(k,A(k+1,x))>A(k+1,x)$. 从而,$A(k+1,x+1)\geqslant A(k+1,x)+1$. 又由对 x 的归纳假设,得证 $A(k+1,x+1)>x+1$.

(2) 分两种情况讨论:

当 $k=0$ 时,$A(0,x)=x+1$,结论显然成立;

当 $k\geqslant 1$ 时,由定义和性质(1),有

$$A(k,x+1)=A(k-1,A(k,x))>A(k,x).$$

下面先证性质(5):

(5)对 x 作归纳证明:当 $x=0$ 时,$A(k+1,0)=A(k,1)$,结论成立. 假设对 x 结论成立,即 $A(k+1,x)\geqslant A(k,x+1)$. 要证对 $x+1$结论也成立,即 $A(k+1,x+1)\geqslant A(k,x+2)$.

由归纳假设和性质(1),$A(k+1,x)\geqslant A(k,x+1)\geqslant x+2$. 再由定义和性质(2),得 $A(k+1,x+1)=A(k,A(k+1,x))\geqslant A(k,x+2)$.

(3) 重复使用性质(5)可得 $A(k,x)\geqslant A(0,x+k)$. 而 $A(0,x+k)=x+k+1$,得证 $A(k,x)>k$.

(4) 由性质(5)和(2),$A(k+1,x)\geqslant A(k,x+1)>A(k,x)$.

(6)对 x 作归纳证明:当 $x=0$ 时,由性质(4),$A(k+2,0)>A(k,0)$,结论成立. 假设对 x 结论成立,即 $A(k+2,x)>A(k,$

$2x)$. 要证对 $x+1$ 结论也成立，即 $A(k+2,x+1)>A(k,2x+2)$.

由定义及性质(4)，$A(k+2,x+1)=A(k+1,A(k+2,x))>A(k,A(k+2,x))$. 由归纳假设，$A(k+2,x)>A(k,2x)$. 而 $A(k,2x)>2x$，得 $A(k+2,x)\geqslant A(k,2x)+1\geqslant(2x+1)+1$. 于是，得证 $A(k+2,x+1)>A(k,2(x+1))$. □

引理 2.25 设 n 元全函数 f 由 n 元全函数 $g_1,\cdots,g_m$ 和 m 元全函数 h 合成得到的，即

$$f(x_1,\cdots,x_n)=h(g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n)).$$

如果存在自然数 $k_1,\cdots,k_m$ 和 k_0 使得对所有的 $x_1,\cdots,x_n$ 和 $y_1,\cdots,y_m$ 有

$$g_i(x_1,\cdots,x_n)<A(k_i,\max\{x_1,\cdots,x_n\}),\quad 1\leqslant i\leqslant m,$$

和

$$h(y_1,\cdots,y_m)<A(k_0,\max\{y_1,\cdots,y_m\}),$$

则对所有的 $x_1,\cdots,x_n$，有

$$f(x_1,\cdots,x_n)<A(k,\max\{x_1,\cdots,x_n\}),$$

其中 $k=\max\{k_0,k_1,\cdots,k_m\}+2$.

证：记 $x^*=\max\{x_1,\cdots,x_n\}$. 注意到 $k\geqslant 2$，由引理 2.24(5)和 Ackermann 函数的定义，有

$$A(k,x^*)\geqslant A(k-1,x^*+1)=A(k-2,A(k-1,x^*)).$$

由于 $k-1>k_i(1\leqslant i\leqslant m)$，根据引理 2.24(4)和假设的条件，有

$$A(k-1,x^*)>A(k_i,x^*)>g_i(x_1,\cdots,x_n),\quad 1\leqslant i\leqslant m.$$

于是，

$$A(k-1,x^*)>\max\{g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n)\}.$$

从而，

$$A(k,x^*)>A(k-2,\max\{g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n)\}).$$

而 $k-2\geqslant k_0$，所以

$$\begin{aligned}A(k,x^*)&>A(k_0,\max\{g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n)\})\\&>h(g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n))\end{aligned}$$

$$=f(x_1,\cdots,x_n).$$ □

引理 2.26 设 $n+1$ 元全函数 f 由 n 元全函数 g 和 $n+2$ 元全函数 h 递归得到，即

$$f(x_1,\cdots,x_n,0)=g(x_1,\cdots,x_n),$$

$$f(x_1,\cdots,x_n,y+1)=h(y,f(x_1,\cdots,x_n,y),x_1,\cdots,x_n).$$

如果存在自然数 k_g 和 k_h 使得对所有的 $x_1,\cdots,x_n,y$ 和 z，有

$$g(x_1,\cdots,x_n)<A(k_g,\max\{x_1,\cdots,x_n\})$$

和

$$h(y,z,x_1,\cdots,x_n)<A(k_h,\max(x_1,\cdots,x_n,y,z\}),$$

则对所有的 $x_1,\cdots,x_n$ 和 y，有

$$f(x_1,\cdots,x_n,y)<A(k,\max\{x_1,\cdots,x_n,y\}),$$

其中 $k=\max\{k_g,k_h\}+3$，当 $n=0$ 时 g 为一固定常数.

证：记 $x^*=\max\{x_1,\cdots,x_n\}$. 由引理 2.24(6)，

$$\begin{aligned}A(k,\max\{x_1,\cdots,x_n,y\})&=A(k,\max\{x^*,y\})\\&>A(k-2,2\max\{x^*,y\})\\&\geqslant A(k-2,x^*+y).\end{aligned}$$

因此，只要证 $A(k-2,x^*+y)>f(x_1,\cdots,x_n,y)$.

对 y 进行归纳证明. 当 $y=0$ 时，由于 $k-2>k_g$，有

$$\begin{aligned}A(k-2,x^*+0)&>A(k_g,x^*)>g(x_1,\cdots,x_n)\\&=f(x_1,\cdots,x_n,0),\end{aligned}$$

结论成立. 假设对 y 结论成立，下面考虑 $y+1$ 的情况.

由引理 2.24(2)，得到

$$A(k-2,x^*+y)>x^*+y\geqslant\max\{x_1,\cdots,x_n,y\}.$$

再由归纳假设 $A(k-2,x^*+y)>f(x_1,\cdots,x_n,y)$，得到

$$A(k-2,x^*+y)>\max\{x_1,\cdots,x_n,y,f(x_1,\cdots,x_n,y)\}.$$

注意到 $k-3\geqslant k_h$，由 Ackermann 函数的定义、引理 2.24(2)、(4)以及假设的条件，得到

$$A(k-2,x^*+y+1)=A(k-3,A(k-2,x^*+y))$$

$$>A(k_h,\max\{x_1,\cdots,x_n,y,f(x_1,\cdots,x_n,y)\})$$
$$>h(y,f(x_1,\cdots,x_n,y),x_1,\cdots,x_n)$$
$$=f(x_1,\cdots,x_n,y+1).$$ □

定理 2.27 对于任意的 n 元原始递归函数 f，存在自然数 k 使得对所有的 $x_1,\cdots,x_n$，有

$$f(x_1,\cdots,x_n)<A(k,\max\{x_1,\cdots,x_n\}).$$

证：根据引理 2.22 和 2.23，只要证明结论对初始函数成立. 实际上，对于 $s(x)$，$n(x)$ 和 $u_i^n(x)$，可以分别取 $k=1,0,0$. □

引理 2.28 $A(k,x)$ 和 $A(x,x)$ 都不是原始递归函数.

证：只需证 $A(x,x)$ 不是原始递归的. 用反证法. 记 $f(x)=A(x,x)$，假设它是原始递归的. 由定理 2.27，存在常数 k 使得对所有的 x，有

$$f(x)<A(k,x).$$

取 $x=k$，得

$$f(k)<A(k,k)=f(k),$$

矛盾. □

最后，证明 Ackermann 函数是可计算的. 根据定义，函数值的计算可用递归过程实现. 设当前要计算 $A(k,x)$ 的值. 当 $k\neq 0$，$x\neq 0$ 时，由公式 $A(k,x)=A(k-1,A(k,x-1))$ 将计算 $A(k,x)$ 转化为计算 $x'=A(k,x-1)$ 和 $A(k-1,x')$. 于是，当前要计算的变成 $A(k,x-1)$，同时将 $k-1$ 存入栈. 等到 x' 的值计算出来后（很可能是在若干步之后），再将这个 $k-1$ 从栈中取出，计算 $A(k-1,x')$. 在下面的程序中，继续沿用前面的符号，用 K 和 X 作为输入变量，输出变量还是 Y，S 表示栈，L 表示栈的长度. $S\leftarrow(K,S)$ 是把 K 的值推入栈 S，而

$$K\leftarrow l(S)$$
$$S\leftarrow r(S)$$

是把 K 从堆顶托出.

计算 $A(k,x)$ 的程序如下：

[A] IF $K\neq 0$ GOTO B

$X\leftarrow X+1$ /注：$A(0,x)=x+1$/

IF $L=0$ GOTO E

$L\leftarrow L-1$

$K\leftarrow l(S)$

$S\leftarrow r(S)$

GOTO A

[B] IF $X\neq 0$ GOTO C

$K\leftarrow K-1$ /注：$A(k,0)=A(k-1,1)$/

$X\leftarrow 1$

GOTO A

[C] $X\leftarrow X-1$ /注：$A(k,x)=A(k-1,A(k,x-1))$/

$L\leftarrow L+1$

$S\leftarrow (K-1,S)$

GOTO A

[E] $Y\leftarrow X$

于是，我们证明了下述定理.

定理 2.29 Ackermann 函数 $A(k,x)$是可计算的，但不是原始递归的.

$A(x,x)$也是非原始递归的可计算函数.

习　题

1. 试证明下述函数是原始递归的：

(1)

$$E(x)=\begin{cases}1 & \text{若 } x \text{ 为偶数}\\ 0 & \text{若 } x \text{ 为奇数.}\end{cases}$$

(2)

$$H(x)=\begin{cases}x/2 & \text{若 } x \text{ 为偶数}\\ (x-1)/2 & \text{若 } x \text{ 为奇数.}\end{cases}$$

2. 试证明：仅在有穷个点取非零值，而在其余的点的值均为零的函数必为原始递归函数.

3. 设 $g(x)$是一个原始递归函数，又设 $f(0,x)=g(x)$，$f(n+1,x)=f(n,f(n,x))$. 证明 $f(n,x)$是原始递归的.

4. 设 $f(0)=0$，$f(1)=1$，$f(2)=2^2$，$f(3)=3^{3^3}=3^{27}$等. 一般地，$f(n)$等于高度为 n 的一叠 n，这些 n 都作为指数. 试证明 f 是原始递归的.

5. 设 $g(x)$是一个原始递归函数，试证明下述 $f(x,y)$是原始递归函数：

(1) $f(x,0)=g(x)$

$f(x,y+1)=f(f(\cdots f(f(x,y),y-1),\cdots),0)$.

(2) $f(x,0)=g(x)$

$f(x,y+1)=f(f(\cdots f(f(x,0),1),\cdots),y)$.

(3) $f(0)=g(0)+1$

$f(1)=g(g(0)+1)+1$

$\cdots$

$f(x)=\underbrace{g(g(\cdots(g}_{x+1\text{个 }g}(0)+1)\cdots)+1)+1$

6. 设 $\sigma(0)=0$；当 $x\neq 0$ 时 $\sigma(x)$是 x 的所有因子之和[例如，$\sigma(6)=1+2+3+6=12$]. 试证明 $\sigma(x)$是原始递归函数.

7. 设 $\pi(x)$是小于等于 x 的素数的个数. 试证明 $\pi(x)$是原始递归函数.

8. 设 $\phi(x)$是小于 x 且与 x 互素的正整数的个数. $\phi(x)$称作 Euler 函数. 试证明 $\phi(x)$是原始递归函数.

9. 设 $h(x)$是使得 $n\leqslant\sqrt{2}\,x<n+1$ 的整数 n. 试证明 $h(x)$是原始递归函数.

10. 设 $h(x)$是使得 $n\leqslant(1+\sqrt{2})x<n+1$ 的整数 n. 试证明 $h(x)$是原始递归函数.

11. 设按照从小到大排列，$u(n)$是第 n 个两平方数之和. 试证明 $u(n)$是原始递归函数.

12. 设 $R(x,t)$是原始递归谓词，定义

$$g(x,y)=\max_{t\leqslant y}R(x,t)$$

如下：当存在 $t\leqslant y$ 使 $R(x,t)$为真时，$g(x,y)$等于这样的 t 的最大值；当不存在这样的 t 时，$g(x,y)=0$. 试证明 $g(x,y)$是原始递归函数.

13. (1) Cantor 编码 $\pi(x,y)$的定义如表 2-2 所示.

(2) 若 $\pi(x,y)=z$，则 $\sigma_1(z)=x,\sigma_2(z)=y$.

(3) $\sigma(z)=\sigma_1(z)+\sigma_2(z)$.

试证明：$\pi(x,y),\sigma_1(z),\sigma_2(z),\sigma(z)$都是原始递归的.

表 2-2　Cantor 编码 $\pi(x,y)$

x \ y	0	1	2	3	…
0	0	1	3	6	…
1	2	4	7		
2	5	8			
3	9				
⋮	⋮				

14. 设 $g(x)$和 $h(x)$是原始递归函数，$f(x,y)$由下式给出

$$f(x,0)=g(x)$$

$$f(x,y+1)=h(\sum_{j=0}^{x}f(j,y)).$$

试证明：$f(x,y)$是原始递归的.

15. 令 $g_i(x)=A(i,x),h_i(x)=A(x,i)$，这里 $A(x,y)$是 Ackermann 函数. 试证明：对每一个 i，$g_i(x)$是原始递归函数，$h_i(x)$不是原始递归函数.

16. 利用 Cantor 编码(第 13 题)把双变量递归(2.9)化成多步递归(2.8)的形式.

第三章　通 用 程 序

3.1　程序的代码

本节叙述程序的一种编码方法，每一个 $\mathscr{S}$ 程序 $\mathscr{P}$ 有一个数与之对应，称作 $\mathscr{P}$ 的**代码**，记作 $\#(\mathscr{P})$. 从 $\mathscr{P}$ 可以得到它的代码 $\#(\mathscr{P})$；反之，任给一个数 p，也可以译码得到它所表示的程序 $\mathscr{P}$，即使得 $\#(\mathscr{P})=p$.

首先要排定所有变量和标号的顺序. 变量的排列顺序为：

$$Y X_1 Z_1 X_2 Z_2 X_3 Z_3 \cdots,$$

标号的排列顺序为：

$$A_1 A_2 A_3 \cdots.$$

变量 V 和标号 L 在上述排列中的序号称作它们的编号，分别记作 $\#(V)$ 和 $\#(L)$. 例如，$\#(Y)=1$，$\#(X)=2$，$\#(Z)=3$，$\#(A_3)=3$. 输入变量的编号为偶数，编号为 1 的变量是 Y，其余奇数编号的变量都是中间变量. 这里所说的变量和标号严格限定在 $\mathscr{S}$ 语言规定的范围内.

指令 I 的代码 $\#(I)$ 规定如下：

$$\#(I) = \langle a, \langle b, c \rangle \rangle,$$

其中

(1) 若 I 不带标号，则 $a=0$；若 I 带标号 L，则 $a=\#(L)$.

(2) 若 I 中的变量是 V，则 $C=\#(V)-1$.

(3) 若 I 中的语句是 $V \leftarrow V$，$V \leftarrow V+1$，$V \leftarrow V-1$，则 b 分别等于 0，1，2.

(4) 若 I 中的语句是 IF $V \neq 0$ GOTO L，则 $b=\#(L)+2$.

例如，指令 $X \leftarrow X+1$ 的代码等于

$$\langle 0,\langle 1,1\rangle\rangle = \langle 0,5\rangle = 10.$$

指令 $[A]$ $X\leftarrow X+1$ 的代码等于

$$\langle 1,\langle 1,1\rangle\rangle = \langle 1,5\rangle = 21.$$

指令 IF $Z\neq 0$ GOTO A 的代码等于

$$\langle 0,\langle 3,2\rangle\rangle = \langle 0,39\rangle = 78.$$

反过来，给定一个数 q，不难得到唯一的指令 I 使得 $\#(I)=q$. 由 $l(q)$ 确定 I 是否带标号和带什么标号. 由 $l(r(q))$ 和 $r(r(q))$ 分别确定 I 中语句的类型和变量. 如果是条件转移语句，$l(r(q))$ 同时给出了转移语句中的标号.

例如，设 $\#(I)=95$. 由 $95+1=2^5\times 3$，得 $l(95)=5$，I 带标号 A_5. 由 $r(95)=1$，$1+1=2^1\times 1$，$l(r(95))=1$，$r(r(95))=0$，知 I 中的语句是 $Y\leftarrow Y+1$. 所以，I 为：

$$[A_5]\quad Y\leftarrow Y+1.$$

现在来规定程序的代码. 设程序 $\mathscr{P}$ 由指令 $I_1, I_2, \cdots, I_k$ 组成，则 $\mathscr{P}$ 的代码为：

$$\#(\mathscr{P}) = [\#(I_1), \#(I_2), \cdots, \#(I_k)] - 1.$$

例如，设 $\mathscr{P}$ 是计算 $\alpha(x)$ 的程序

IF $X\neq 0$ GOTO A

$Y\leftarrow Y+1$

那么，

$$\#(I_1)=\langle 0,\langle 3,1\rangle\rangle=46,$$

$$\#(I_2)=\langle 0,\langle 1,0\rangle\rangle=2,$$

$$\#(\mathscr{P})=[46,2]-1=2^{46}\times 3^2-1.$$

根据约定，空数列的 Gödel 数为 1，故空程序的代码为 0.

注意到不带标号的指令 $Y\leftarrow Y$ 的代码是 $\langle 0,\langle 0,0\rangle\rangle=0$. 由于在数列的尾部添加若干 0 不改变其 Gödel 数，所以在程序的末尾添加若干条指令 $Y\leftarrow Y$ 不改变程序的代码. 从而这些不同的程序具有相同的代码. 不过，这种多义性一般说来是无害的. 在程序的末尾添加若干条 $Y\leftarrow Y$ 不会改变程序的功能. 但是，如果能避免这种多义性总是无害的，并且能减少不必要的麻烦. 为此，我们规定：

不允许程序的最后一条指令是不带标号的 $Y\leftarrow Y$. 这样一来，任给一个数 p，存在唯一的程序 $\mathscr{P}$ 使得 $\#(\mathscr{P})=p$. 从而，用这样的编码方式给出程序和数(程序的代码)之间的一一对应关系.

例如，设 $\#(\mathscr{P})=125$，求程序 $\mathscr{P}$.

由于 $125+1=2\times 3^2\times 7=[1,2,0,1]$，$\mathscr{P}$ 由 4 条指令组成，其编码分别为 1,2,0,1. 而

$$1=\langle 1,0\rangle=\langle 1,\langle 0,0\rangle\rangle,$$
$$2=\langle 0,1\rangle=\langle 0,\langle 1,0\rangle\rangle,$$
$$0=\langle 0,0\rangle=\langle 0,\langle 0,0\rangle\rangle,$$

故 $\mathscr{P}$ 为

$$\begin{array}{ll}[A] & Y\leftarrow Y\\ & Y\leftarrow Y+1\\ & Y\leftarrow Y\\ [A] & Y\leftarrow Y\end{array}$$

它计算函数 $f(x)=1$. 尽管谁也不会编写出这样的程序，但它确实是一个合法的程序.

3.2 停 机 问 题

上一节给出程序与自然数之间的一一对应，共有可数个程序. 每一个(部分)可计算函数都有计算它的程序(可能不只一个)，故至多有可数个(部分)可计算函数. 而自然数集 N 上的函数全体的势是 $\aleph>\aleph_0$，这样就得到下述结论：一定存在不是可计算的全函数和不是部分可计算的部分函数.

下面给出这样的例子：任给一个程序 $\mathscr{P}$ 和一个数 x，问程序 $\mathscr{P}$ 对输入 x 的计算最终是否停止？这个问题称作**停机问题**. 用谓词 $\mathrm{HALT}(x,y)$ 描述这个问题. 谓词 $\mathrm{HALT}(x,y)$ 定义如下：

$\mathrm{HALT}(x,y)\Leftrightarrow$ 以 y 为代码的程序对输入 x 的计算最终停止，

即

$$\mathrm{HALT}(x,y) \Leftrightarrow \psi_{\mathscr{P}}^{(1)}(x)\downarrow, \text{其中 } \#(\mathscr{P}) = y.$$

定理 3.1 HALT(x,y)和 HALT(x,x)不是可计算的.

证:只需证 HALT(x,x)不是可计算的. 假设不然,则可构造程序 $\mathscr{D}$ 如下:

$$[A] \quad \text{IF HALT}(X,X) \text{ GOTO } A$$

显然,

$$\psi_{\mathscr{D}}(x) = \begin{cases} 0 & \text{若} \neg \mathrm{HALT}(x,x) \\ \uparrow & \text{若 HALT}(x,x). \end{cases}$$

记 $\#(\mathscr{D}) = y_0$,则对任意的 x,有

$$\mathrm{HALT}(x,y_0) \Leftrightarrow \psi_{\mathscr{D}}(x) = 0 \Leftrightarrow \neg\, \mathrm{HALT}(x,x).$$

令 $x = y_0$,得

$$\mathrm{HALT}(y_0,y_0) \Leftrightarrow \neg\, \mathrm{HALT}(y_0,y_0),$$

矛盾. □

3.3 通 用 程 序

对于每一个 $n>0$,定义:

$$\Phi^{(n)}(x_1,\cdots,x_n,y) = \psi_{\mathscr{P}}^{(n)}(x_1,\cdots,x_n),$$

这里 $\#(\mathscr{P}) = y$.

令 $y=0,1,2,\cdots$ 可以枚举出所有的 $\mathscr{S}$ 程序. 因此,对于每一个 $n>0$,

$$\Phi^{(n)}(x_1,\cdots,x_n,0), \Phi^{(n)}(x_1,\cdots,x_n,1),\cdots$$

枚举出所有的 n 元部分可计算函数. 当 y 是一个固定的常数、而不作为自变量时,记

$$\Phi_y^{(n)}(x_1,\cdots,x_n) = \Phi^{(n)}(x_1,\cdots,x_n,y),$$

即

$$\Phi_y^{(n)}(x_1,\cdots,x_n) = \psi_{\mathscr{P}}^{(n)}(x_1,\cdots,x_n),$$

这里 $\#(\mathscr{P}) = y$. 当 $n=1$ 时,常略去上标,写成 $\Phi(x,y)$ 和 $\Phi_y(x)$.

定理 3.2(通用性定理)　对于每一个 $n>0$，函数 $\Phi^{(n)}(x_1,\cdots,x_n,y)$是部分可计算的.

在证明这个定理之前，先解释一下它的含义. 由于 $\Phi^{(n)}(x_1,\cdots,x_n,y)$是部分可计算的，故存在计算这个函数的程序 $\mathscr{U}_n$. 根据函数 $\Phi^{(n)}$的定义，$\mathscr{U}_n$ 有这样的能力：任给一个程序 $\mathscr{P}$(以它的编码 $\#(\mathscr{P})=y$ 的形式)和对 $\mathscr{P}$ 的输入 $x_1,\cdots,x_n$，以 $x_1,\cdots,x_n,y$ 作为 $\mathscr{U}_n$ 的输入，其计算结果恰好等于程序 $\mathscr{P}$ 以 $x_1,\cdots,x_n$ 为输入的计算结果. 换句话说，$\mathscr{U}_n$ 可以完成任何程序的计算. 因此，它是一个通用程序. 通用程序是现代通用电子计算机的数学模型，早在 1936 年 A. Turing 已经证明了它的存在性(以通用 Turing 机的形式). 从历史上，这个定理预示了现代通用电子计算机的可能性.

证：$\mathscr{U}_n$ 的工作方式类似一个解释程序. 设 $\#(\mathscr{P})=y$，$\mathscr{U}_n$ 取出 $\mathscr{P}$ 的一条指令，译出这条指令并完成其功能，然后执行下一条指令，直到计算结束.

$\mathscr{U}_n$ 用输入变量 $X_1,\cdots,X_n$ 分别表示 $\Phi^{(n)}(x_1,\cdots,x_n,y)$的自变量 $x_1,\cdots,x_n$，用 X_{n+1}表示 y，函数值由输出变量 Y 给出.

为了描述程序 $\mathscr{P}$ 在计算过程中的当前情况，只需指明 $\mathscr{P}$ 中所有变量的当前值和即将执行的指令. 用 K 表示 $\mathscr{P}$ 即将执行第 K 条指令. 用 S 以 Gödel 数的形式存储 $\mathscr{P}$ 的所有变量的当前值，具体地说

$$S=[Y,X_1,Z_1,X_2,Z_2,\cdots,X_n,Z_n,\cdots,X_d,Z_d],$$

其余变量 $X_i,Z_i,(i>d)$的值为 0. 下面按这个顺序来称呼变量，如第 1 个变量是 Y，第 2 个变量是 X_1，第 3 个变量是 Z_1，….

程序 $\mathscr{U}_n$ 如下：

$W\leftarrow X_{n+1}+1$　　/$W\leftarrow y+1$/

$S\leftarrow\prod_{i=1}^{n}p_{2i}^{X_i}$　　/将初始状态赋给 S/

$K\leftarrow 1$　　/从第 1 条指令开始执行/

[C]　IF $K=\mathrm{Lt}(W)+1\vee K=0$ GOTO F

$U\leftarrow r((W)_K)$　　/设$\#(I_K)=\langle a,\langle b,c\rangle\rangle$,$U\leftarrow\langle b,c\rangle$/

$P\leftarrow p_{r(U)+1}$　　/当前要用第$c+1$个变量V/

IF $l(U)=0$ GOTO N　　/I_K的语句为$V\leftarrow V$/

IF $l(U)=1$ GOTO A　　/I_K的语句为$V\leftarrow V+1$/

IF $\neg(P|S)$ GOTO N　　/V的值为0/

IF $l(U)=2$ GOTO M　　/I_K的语句为$V\leftarrow V-1$/

$K\leftarrow\min_{i\leqslant Lt(W)}\{l((W)_i)+2=l(U)\}$

GOTO C

[M] $S\leftarrow\llcorner S/P\lrcorner$　　/执行$V\leftarrow V-1$/

GOTO N

[A] $S\leftarrow S\cdot P$　　/执行$V\leftarrow V+1$/

[N] $K\leftarrow K+1$

GOTO C

[F] $Y\leftarrow(S)_1$　　/计算结束/

说明：设$\mathscr{P}$有m条指令$I_1,I_2,\cdots,I_m$，则$W=[\#(I_1),\cdots,\#(I_m)]$，$m=\mathrm{Lt}(W)$. S的初值为$\mathscr{P}$的初始状态，$S=[0,x_1,0,\cdots,x_n,0,\cdots,0]$.

即将执行第K条指令I_K，设$\#(I_K)=(W)_K=\langle a,\langle b,c\rangle\rangle$，则$U=r((W)_K)=\langle b,c\rangle$，$l(U)=b$，$r(U)=c$. I_K中使用第$c+1$个变量，设为V. I_K的语句类型由b确定. 若$b=0$，则I_K的语句是$V\leftarrow V$，不做任何运算，下一步执行I_{K+1}；若$b=1$，则I_K的语句是$V\leftarrow V+1$，要在S的p_{c+1}的指数上加1，即S乘以p_{c+1}. 以A为标号的宏指令完成这个运算；若$b\geqslant 2$，则I_K的语句是$V\leftarrow V-1$或IF $V\neq 0$ GOTO A_j. 当$V=0$，即S不被p_{c+1}整除时，这两个语句都不做任何运算；当$b=2$且$V>0$时执行$V\leftarrow V-1$，要将S的p_{c+1}的指数减1，即S除以p_{c+1}. 以M为标号的宏指令完成这个运算；当$b\geqslant 3$且$V>0$时执行语句IF $V\neq 0$ GOTO A_j，程序转去执行$\mathscr{P}$中以A_j为标号的第一条指令，其中$j=b-2$. 令

$$t=\min_{i\leqslant Lt(W)}\{l((W)_i)+2=b\},$$

I_t 是以 A_t 为标号的第一条指令，将 t 赋给 K. 注意到，当 $\mathscr{P}$ 没有以 A_t 为标号的指令时 $t=0$，故当 $K=m+1$ 或 $K=0$ 时计算结束.

□

对每一个 $n>0$，定义谓词

$\mathrm{STP}^{(n)}(x_1,\cdots,x_n,y,t)$⇔代码为 y 的程序对输入 $x_1,\cdots,x_n$ 至多在 t 步之后计算结束

⇔代码为 y 的程序关于输入 $x_1,\cdots,x_n$ 的计算的长度小于等于 $t+1$.

下述定理表明，任给一个程序 $\mathscr{P}$ 和输入 $x_1,\cdots,x_n$ 以及 t，能够判断 $\mathscr{P}$ 对输入 $x_1,\cdots,x_n$ 的计算在 t 步之内是否结束.

定理 3.3（计步定理） 对每一个 $n>0$，谓词 $\mathrm{STP}^{(n)}(x_1,\cdots,x_n,y,t)$ 是可计算的.

证：只需对 $\mathscr{U}_n$ 做稍许修改就可得到计算这个谓词的程序. 添加一个输入变量 X_{n+2} 存放 t 值，添加变量 Q 记录程序 $\mathscr{P}$ 执行的步数，这里 $\#(\mathscr{P})=y$. $\mathscr{U}_n$ 大循环一次，即 $\mathscr{P}$ 执行一步，Q 的值加 1. 若 $\mathscr{P}$ 在 t 步之内（即 $Q\leqslant t$）计算结束，则 $Y=1$. 若 $\mathscr{P}$ 执行 t 步仍未结束，则输出 $Y=0$ 并停止计算.

程序清单如下，其中添加或修改的部分标有记号（*）.

$W\leftarrow X_{n+1}+1$

$S\leftarrow\prod_{i=1}^{n}p_{2i}^{X_i}$

$K\leftarrow 1$

[C] $Q\leftarrow Q+1$ （*）

IF $K=\mathrm{Lt}(W)+1\vee K=0$ GOTO F

IF $Q>X_{n+2}$ GOTO E （*）

$U\leftarrow r((W)_K)$

$P\leftarrow p_{r(U)+1}$

IF $l(U)=0$ GOTO N

IF $l(U)=1$ GOTO A

IF $\neg(P|S)$ GOTO N

IF $l(U)=2$ GOTO M

$K\leftarrow\min_{i\leqslant \mathrm{Lt}(W)}\{l((W)_i)+2=l(U)\}$

GOTO C

$[M]$ $S\leftarrow \lfloor S/P \rfloor$

GOTO N

$[A]$ $S\leftarrow S\cdot P$

$[N]$ $K\leftarrow K+1$

GOTO C

$[F]$ $Y\leftarrow 1$ (*) □

3.4 参 数 定 理

定理 3.4(参数定理) 对于每一个 $n,m>0$,存在原始递归函数 $S_m^n(u_1,\cdots,u_n,y)$使得

$$\Phi^{(m+n)}(x_1,\cdots,x_m,u_1,\cdots,u_n,y)$$
$$=\Phi^{(m)}(x_1,\cdots,x_m,S_m^n(u_1,\cdots,u_n,y)).$$

参数定理又称作 s-m-n 定理. 对于固定的 $u_1,\cdots,u_n$ 和 y,等式左边是一个 m 元部分可计算函数,存在 q 使得

$$\Phi^{(m+n)}(x_1,\cdots,x_m,u_1,\cdots,u_n,y)=\Phi^{(m)}(x_1,\cdots,x_m,q).$$

显然,q 的值依赖于 $u_1,\cdots,u_n$ 和 y. 定理告诉我们,q 是 $u_1,\cdots,u_n,y$ 的原始递归函数.

证:对 n 作归纳证明.

当 $n=1$ 时,要证存在原始递归函数 $S_m^1(x,y)$使得

$$\Phi^{(m+1)}(x_1,\cdots,x_m,u,y)=\Phi^{(m)}(x_1,\cdots,x_m,S_m^1(u,y)).$$

对于任意给定的 u 和 y,设 $y=\#(\mathscr{P})$,$S_m^1(u,y)=\#(\mathscr{Q})$. 程序 $\mathscr{Q}$ 对输入 $x_1,\cdots,x_m$ 的计算结果应该和程序 $\mathscr{P}$ 对输入 $x_1,\cdots,x_m,u$

的计算结果相同. 在程序 $\mathscr{P}$ 的前面添加 u 条 $X_{m+1}\leftarrow X_{m+1}+1$ 即可得到这样的程序 $\mathscr{Q}$:

$$\left.\begin{array}{l} X_{m+1}\leftarrow X_{m+1}+1 \\ \vdots \\ X_{m+1}\leftarrow X_{m+1}+1 \end{array}\right\} u\text{ 条}$$
$$\mathscr{P}$$

$\mathscr{P}$ 有 $\mathrm{Lt}(y+1)$条指令,$\mathscr{Q}$ 有 $u+\mathrm{Lt}(y+1)$条指令,其代码为

$$\#(\mathscr{Q}) = [\#(I_1),\cdots,\#(I_u),(y+1)_1,\cdots,(y+1)_{\mathrm{Lt}(y+1)}] \dot{-} 1,$$

其中 $I_1,\cdots,I_u$ 均为 $X_{m+1}\leftarrow X_{m+1}+1$. 注意到 $X_{m+1}\leftarrow X_{m+1}+1$ 的代码为$\langle 0,\langle 1,2m+1\rangle\rangle = 16m+10$,故

$$S_m^1(u,y) = (\prod_{i=1}^{u} p_i)^{16m+1}(\prod_{j=1}^{\mathrm{Lt}(y+1)} p_{x+j}^{(y+1)_j}) \dot{-} 1.$$

这是一个原始递归函数,得证当 $n=1$ 时结论成立.

设当 $n=k$ 时结论成立. 由 $n=1$ 及归纳假设,有

$$\begin{aligned}&\Phi^{(m+k+1)}(x_1,\cdots,x_m,u_1,\cdots,u_{k+1},y)\\ &\quad = \Phi^{(m+k)}(x_1,\cdots,x_m,u_1,\cdots,u_k,S_{m+k}^1(u_{k+1},y))\\ &\quad = \Phi^{(m)}(x_1,\cdots,x_m,S_m^k(u_1,\cdots,u_k,S_{m+k}^1(u_{k+1},y))),\end{aligned}$$

其中 $S_{m+k}^1(u_{k+1},y)$和 $S_m^k(u_1,\cdots,u_k,z)$都是原始递归函数. 令

$$S_m^{k+1}(u_1,\cdots,u_{k+1},y) = S_m^k(u_1,\cdots,u_k,S_{m+k}^1(u_{k+1},y)),$$

这是一个原始递归函数,且

$$\begin{aligned}&\Phi^{(m+k+1)}(x_1,\cdots,x_m,u_1,\cdots,u_{k+1},y)\\ &\quad = \Phi^{(m)}(x_1,\cdots,x_m,S_m^{k+1}(u_1,\cdots,u_{k+1},y)),\end{aligned}$$

得证当 $n=k+1$ 时结论也成立. □

推论 3.5 设 $f(x,t)$是部分可计算函数,记 $f_t(x)=f(x,t)$,则存在原始递归函数 $q(t)$使得

$$\Phi_{q(t)}(x) = f_t(x).$$

证:$f(x,t)$是部分可计算的,存在 y_0 使得 $\Phi^{(2)}(x,t,y_0)=f(x,t)$. 由参数定理,存在原始递归函数 $S_1^1(t,y)$使得

$$\Phi^{(2)}(x,t,y) = \Phi(x,S_1^1(t,y)).$$

令 $q(t)=S_1^1(t,y_0)$，$q(t)$是原始递归的，且

$$\Phi(x,q(t)) = \Phi^{(2)}(x,t,y_0) = f(x,t),$$

即

$$\Phi_{q(t)}(x) = f_t(x). \qquad \square$$

参数定理(包括推论 3.5 在内)是一个非常有用的工具，下面举例说明它的应用.

[例 3.1] 证明：存在原始递归函数 $g(u,v)$使得

$$\Phi_u(\Phi_v(x)) = \Phi_{g(u,v)}(x).$$

证：$\Phi_u(\Phi_v(x))=\Phi(\Phi(x,v),u)$是 x,u,v 的部分可计算函数，存在 y_0 使得

$$\Phi^{(3)}(x,u,v,y_0) = \Phi_u(\Phi_v(x)).$$

由参数定理，

$$\Phi^{(3)}(x,u,v,y) = \Phi(x,S_1^2(u,v,y)).$$

取 $g(u,v)=S_1^2(u,v,y_0)$，$g(u,v)$是原始递归函数，且

$$\Phi^{(3)}(x,u,v,y_0) = \Phi(x,g(u,v)),$$

即

$$\Phi_u(\Phi_v(x)) = \Phi_{g(u,v)}(x). \qquad \square$$

[例 3.2] 谓词 $T(x)$不是可计算的，其中

$$T(x) \Leftrightarrow \Phi_x(t) \text{ 是一个全函数}$$
$$\Leftrightarrow \text{对所有的 } t, \Phi_x(t)\downarrow.$$

证：令

$$f(t,x) = \begin{cases} 0 & \text{若 HALT}(x,x) \\ \uparrow & \text{否则,} \end{cases}$$

记 $f_x(t)=f(t,x)$，则当 HALT(x,x)时，$f_x(t)=n(t)$是一个全函数；当$\neg$ HALT(x,x)时，$f_x(t)=\varnothing$处处无定义. 所以

(1) $f_x(t)$是全函数$\Leftrightarrow$HALT(x,x)；

(2) 注意到 HALT$(x,x)\Leftrightarrow\Phi(x,x)\downarrow$，故 $f(t,x)=n(\Phi(x,x))$是部分可计算的. 由推论 3.5，存在原始递归函数 $q(x)$使得

$f_x(t) \simeq \Phi_{q(x)}(t)$.

于是，

$$\begin{aligned}\mathrm{HALT}(x,x) &\Leftrightarrow f_x(t) \text{ 是全函数}，\\ &\Leftrightarrow \Phi_{q(x)}(t) \text{ 是全函数}，\\ &\Leftrightarrow T(q(x)).\end{aligned}$$

假如 $T(x)$是可计算的，则 $T(q(x))$是可计算的，推出 $\mathrm{HALT}(x,x)$是可计算的，矛盾. 得证 $T(x)$不是可计算的. □

例 3.2 采用的证明方法是具有普遍性的. 它通过可计算函数 $q(x)$，把谓词 $\mathrm{HALT}(x,x)$转化成 $T(q(x))$. 由于 $\mathrm{HALT}(x,x)$不是可计算的，推出 $T(x)$也不是可计算的. 这种利用已知不可计算的谓词来证明另一谓词是不可计算的方法称作归约法. 现将归约的定义及其性质叙述如下.

定义 3.1 设 $P(x)$，$Q(x)$是两个谓词，如果函数 $q(x)$满足下述条件：

(1) $q(x)$是可计算的，

(2) $P(x) \Leftrightarrow Q(q(x))$，

则称 $q(x)$是 $P(x)$到 $Q(x)$的**归约**.

如果存在 $P(x)$到 $Q(x)$的归约，则称 $P(x)$**可归约到** $Q(x)$.

归约具有下述重要性质：

定理 3.6 如果谓词 $P(x)$可归约到 $Q(x)$，那么

(1) 若 $Q(x)$是可计算的，则 $P(x)$也是可计算的；

(2) 若 $P(x)$不是可计算的，则 $Q(x)$也不是可计算的.

例如，例 3.2 中的 $q(x)$是 $\mathrm{HALT}(x,x)$到 $T(x)$的归约. 归约的上述性质提供了证明谓词不可计算性的一条途径.

[**例 3.3**] 谓词 $E(x,y)$不是可计算的，其中

$$E(x,y) \Leftrightarrow \text{对所有的 } t, \Phi_x(t) = \Phi_y(t).$$

证：只需证存在 y_0 使得 $E(x,y_0)$不是可计算的. 取 y_0 使得 $\Phi_{y_0}(t) \equiv u_1^1(t)$，记 $G(x) = E(x,y_0)$，则

$$G(x) \Leftrightarrow \Phi_x(t) \equiv u_1^1(t).$$

设辅助函数

$$f(t,x)=\begin{cases}t & \text{若 } \Phi(t,x)\downarrow\\ \uparrow & \text{否则}\end{cases}$$

并且记 $f_x(t)=f(t,x)$，则

(1) $f_x(t)\equiv u_1^1(t)\Leftrightarrow\Phi_x(t)$是全函数$\Leftrightarrow T(x)$；

(2) $f(t,x)=u_2^{(2)}(\Phi(t,x),t)$是部分可计算的.

由推论 3.5，存在原始递归函数 $q(x)$使 $f_x(t)=\Phi_{q(x)}(t)$. 于是，

$$T(x)\Leftrightarrow\Phi_{q(x)}(t)\equiv u_1^1(t)\Leftrightarrow G(q(x)).$$

因此，$q(x)$是 $T(x)$到 $G(x)$的归约. 例 3.2 已经证明 $T(x)$不是可计算的，故 $G(x)$也不是可计算的. □

3.5 递 归 定 理

定理 3.7(递归定理) 设 $g(x,x_1,\cdots,x_m)$是 $m+1$ 元部分可计算函数，则存在数 e 使得

$$\Phi_e^{(m)}(x_1,\cdots,x_m)=g(e,x_1,\cdots,x_m).$$

证：考虑部分可计算函数 $g(S_m^1(t,t),x_1,\cdots,x_m)$，其中 $S_m^1(x,y)$是参数定理中的原始递归函数. 于是，存在 z_0 使

$$\begin{aligned}g(S_m^1(t,t),x_1,\cdots,x_m)&=\Phi^{(m+1)}(x_1,\cdots,x_m,t,z_0)\\&=\Phi^{(m)}(x_1,\cdots,x_m,S_m^1(t,z_0)).\end{aligned}$$

令 $t=z_0$，$e=S_m^1(z_0,z_0)$，得

$$g(e,x_1,\cdots,x_m)=\Phi_e^{(m)}(x_1,\cdots,x_m).$$ □

定理 3.8(不动点定理) 设 $f(t)$是一个可计算函数，则存在数 e 使得对所有的 x，

$$\Phi_{f(e)}(x)=\Phi_e(x).$$

证：考虑 $g(t,x)=\Phi(f(t),x)$，这是一个部分可计算函数. 由递归定理，存在数 e 使得

$$\Phi_e(x)=g(e,x)=\Phi_{f(e)}(x).$$ □

[**例 3.4**] 存在数 e 使得对所有的 x，

$$\Phi_e(x) = e.$$

证：取 $g(t,x)=t$. 由递归定理，存在数 e 使得对所有的 x，有

$$\Phi_e(x) = g(e,x) = e. \qquad \square$$

这个 e 很有点意思：设 $e=\#(\mathscr{P})$，不管输入什么，$\mathscr{P}$ 总是输出自己的代码. 换句话说，$\mathscr{P}$ 在不断地复制出自身.

习　题

1. 设 $\mathscr{P}$ 为例 1.1 中的程序，求 $\#(\mathscr{P})$.

2. 设 $\#(\mathscr{P})=575$，试写出程序 $\mathscr{P}$.

3. 证明 HALT$(0,x)$不是可计算的.

4. 试证明对于每一个 $n>0$，STP$^{(n)}$是原始递归的.

5. 证明不存在可计算函数 $f(x)$使得当 $\Phi(x,x)\downarrow$ 时 $f(x)=\Phi(x,x)+1$.

6. (a) 设 $g(x)$，$h(x)$是部分可计算函数. 试证明存在部分可计算函数 $f(x)$使得 $f(x)\downarrow$ 当且仅当 $g(x)\downarrow$ 或 $h(x)\downarrow$，并且当 $f(x)\downarrow$ 时 $f(x)=g(x)$ 或 $f(x)=h(x)$.

(b) 对于任何部分可计算函数 $g(x)$，$h(x)$，是否都存在 f 满足(a)的所有要求并且当 $g(x)\downarrow$ 时 $f(x)=g(x)$？试证明之.

7. 设 $f(x,y)$是部分可计算函数，试证明存在原始递归函数 $g(u,v)$使得

$$\Phi_{g(u,v)}(x) = f(\Phi_u(x),\Phi_v(x)).$$

8. 证明存在原始递归函数 $g(u,v,w)$使得

$$\Phi^{(3)}(u,v,w,x) = \Phi_{g(u,v,w)}(x).$$

9. 设 $g(x)$是部分可计算函数，如果存在可计算函数 $f(x)$使得当 $g(x)\downarrow$ 时 $f(x)=g(x)$，则称 $g(x)$是可扩张的. 定义谓词 EXT 如下：

$$\mathrm{EXT}(y)\Leftrightarrow\Phi_y(x)\text{ 是可扩张的.}$$

试证明：EXT 不是可计算的.

10. 证明存在原始递归函数 $h(u)$使得

$$\Phi(x,h(u)) = \Phi(x,\Phi(h(u),u)).$$

第四章　字符串计算

4.1　字符串的数字表示

设字母表 $A=\{s_1,s_2,\cdots,s_n\}$，对 A 中的元素规定一个顺序，如所列的那样. 对每一个 $u\in A^*$，定义一个数 x 与它对应. 当 $u=\varepsilon$ 时，$x=0$；当 $u=s_{i_k}s_{i_{k-1}}\cdots s_{i_0}$ 时，

$$x=i_k\cdot n^k+i_{k-1}\cdot n^{k-1}+\cdots+i_1\cdot n+i_0, \qquad (4.1)$$

其中 $1\leqslant i_j\leqslant n, j=0,1,\cdots,k$. u 称作 x 的**以 n 为底的表示**或 x 的 **n 进制表示**.

例如，$A=\{a,b,c\}$，元素的顺序如所示的那样. $u=bcab$ 是

$$x=2\times 3^3+3\times 3^2+1\times 3+2=86$$

的 3 进制表示.

任给一个数 x，可以求出它的 n 进制表示. 当 $x=0$ 时，它的 n 进制表示是空串 ε. 当 $x>0$ 时，可以求出 $i_0,i_1,\cdots,i_k$ 使得(4.1)式成立. 为此，先定义两个原始递归函数.

$$R^+(x,y)=\begin{cases}R(x,y) & \text{若 } \neg(y|x)\\ y & \text{否则,}\end{cases}$$

$$Q^+(x,y)=\begin{cases}\llcorner x/y\lrcorner & \text{若 } \neg(y|x)\\ \llcorner x/y\lrcorner \dot{-} 1 & \text{否则,}\end{cases}$$

其中 $\llcorner x/y\lrcorner$ 和 $R(x,y)$ 已在 2.3 节给出，都是原始递归的. 设 $y>0$，当 y 不整除 x 时，$Q^+(x,y)$ 和 $R^+(x,y)$ 就是 x 除以 y 的整数部分和余数；当 y 整除 x 时，“余数”$R^+(x,y)$ 不是 0，而是 y，相应地，“商”$Q^+(x,y)$ 也不是 x/y，而是 $x/y\dot{-}1$. 但是，除法中被除数、除数、商和余数的关系在这里仍成立：

$$x = y \cdot Q^+(x,y) + R^+(x,y),$$

只是这里 $R^+(x,y)$ 可能取值 $1,2,\cdots,y$，而不是 $0,1,\cdots,y-1$.

令

$$u_0 = x, u_{m+1} = Q^+(u_m,n), \tag{4.2}$$

由(4.1)，有

$$\begin{aligned} u_0 &= i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \cdots + i_1 \cdot n + i_0, \\ u_1 &= i_k \cdot n^{k-1} + i_{k-1} \cdot n^{k-2} + \cdots + i_1, \\ &\vdots \\ u_k &= i_k. \end{aligned} \tag{4.3}$$

于是，

$$i_m = R^+(u_m,n), \quad m = 0,1,\cdots,k. \tag{4.4}$$

令

$$g(m,n,x) = u_m, \tag{4.5}$$

$$h(m,n,x) = R^+(g(m,n,x),n). \tag{4.6}$$

由(4.2)，g 满足下述递归等式

$$g(0,n,x) = x,$$

$$g(m+1,n,x) = Q^+(g(m,n,x),n)$$

故 $g(m,n,x)$ 和 $h(m,n,x)$ 都是原始递归函数.

由(4.4)，有

$$i_m = h(m,n,x), \quad m = 0,1,\cdots,k.$$

这表明通过原始递归运算可以得到 x 的 n 进制表示.

任给 $x \in N$，令 $\mathrm{LENTH}_n(x)$ 等于 x 的 n 进制表示的长度. 注意到长度为 $k+1$ 的最小的 n 进制数是 $\sum_{i=0}^{k} n^i$，故

$$\mathrm{LENTH}_n(x) = \min_{k \leqslant x}\left\{ \sum_{i=0}^{k} n^i > x \right\}.$$

函数 LENTH_n 是原始递归的.

数的这种表示方法和通常的记数法的原理是一样的，区别在于当以 n 为底时，我们使用数字 $1,2,\cdots,n$(分别用 n 个符号 $s_1,s_2,$

…,s_n 表示)，而不是 0,1,…,$n-1$. 例如，在通常用的二进制数中有 2 个数字 0 和 1，4 和 5 分别表示为 100 和 101. 而在我们的二进制数中使用数字 1 和 2，4 和 5 分别表示为 12 和 21. 这种表示方法有两点好处：

(1) 避免了不唯一性. 在通常用的记数法中，11，011，0011，00011，…是一个数. 现在避免了这种不唯一性，数和它的 n 进制表示是一一对应的.

(2) 可以以 1 为底. 在通常使用的记数法中，不能用 1 做底，没有一进制数. 现在可以用 1 做底. 设 $A=\{s_1\}$，数 x 的一进制表示是 s_1^x.

设字母表 $A=\{s_1,s_2,\cdots,s_n\}$，f 是 A 上的 m 元字函数. 把 A 上的字符串看作数的 n 进制表示，于是 f 自动地成为一个 m 元数论函数. 把 f 称作这个数论函数的 n 进制表示或以 n 为底的表示. 通常不必严格地区分它们，使用同一个函数符号表示，并根据上下文把它看作数论函数或 A 上的字函数. 这样一来，部分可计算、可计算、原始递归的概念都可以自然地移植到 A 上的字函数.

定义 4.1 设 A 是包含 n 个符号的字母表，f 是 A 上的字函数. 如果 f 以 n 为底所表示的数论函数是部分可计算的(可计算的、原始递归的)，则称 f 是**部分可计算的**(**可计算的，原始递归的**).

例如，设 $A=\{a,b\}$，$f(w)=wa$，$\forall\ w\in A^*$. 规定 A 中符号的顺序是 a,b，函数 f 以 2 为底表示函数 $2x+1$. 因为 $2x+1$ 是原始递归的，所以 $f(w)=wa$ 是原始递归的.

A^* 上的谓词可以看作取值 s_1(真值)或 ε(假值)的全函数，所以 A^* 上的可计算谓词和原始递归谓词都有明确的定义. 定义 4.1 可以推广到在 N 上取值的 A^* 上的函数.

但是，在定义 4.1 正式生效之前，我们必须证明这个定义与指定的 A 中符号的顺序无关. 当指定的顺序不同时，同一个字符串表示的数一般是不同的，同一个字函数表示的数论函数一般也是

不同的.例如,对上面的例子,若指定顺序 b,a,则 $f(w)=wa$ 表示的数论函数是 $2x+2$. 而 A 中符号顺序完全是人为指定的.幸运的是有下述引理,因此定义 4.1 是有效的.

引理 4.1 A 上的字函数表示的数论函数是否是部分可计算的(可计算的、原始递归的),与 A 中符号的指定顺序无关.

证:设字母表 $A=\{s_1,s_2,\cdots,s_n\}$,π 是 $\{1,2,\cdots,n\}$ 到自身的双射.取顺序 Ⅰ :$s_1,s_2,\cdots,s_n$;顺序 Ⅱ :$s_{\pi(1)},s_{\pi(2)},\cdots,s_{\pi(n)}$. 任给 $w\in A^*$,在顺序 Ⅰ 和顺序 Ⅱ 下 w 以 n 为底分别表示数 x 和 x',令 $\eta(x)=x'$.

$$\eta(x)=\sum_{i=0}^{k}\pi(h(i,n,x))\cdot n^i,$$

$$\eta^{-1}(x')=\sum_{i=0}^{k}\pi^{-1}(h(i,n,x'))\cdot n^i,$$

这里 $k=\mathrm{LENTH}_n(x)-1=\mathrm{LENTH}_n(x')-1$,与顺序无关.$\eta$ 是 N 到自身的双射函数,η^{-1}是它的逆.根据上式,η 和 η^{-1}都是原始递归的.

设 f 是 A 上的 m 元字函数,在顺序 Ⅰ 和顺序 Ⅱ 下以 n 为底分别表示数论函数 f_1 和 f_2,则

$$f_2(x_1,\cdots,x_m)=\eta(f_1(\eta^{-1}(x_1),\cdots,\eta^{-1}(x_m))),$$

$$f_1(x_1,\cdots,x_m)=\eta^{-1}(f_2(\eta(x_1),\cdots,\eta(x_m))).$$

因此,f_1 是部分可计算的(可计算的、原始递归的)当且仅当 f_2 是部分可计算的(可计算的、原始递归的). □

下面给出若干原始递归函数,后面要用到它们.

1. *长度函数* $|w|$

作为 A 上的字函数,$|w|$ 等于 w 的长度.作为数论函数,$|x|=\mathrm{LENTH}_n(x)$. 它是原始递归的.

2. *连接函数* $\mathrm{CONCAT}_n^{(m)}(u_1,\cdots,u_m)$

$\mathrm{CONCAT}_n^{(m)}(u_1,u_2,\cdots,u_m)=u_1u_2\cdots u_m$,即把 $u_1,u_2,\cdots,u_m$ 连接起来.m 是自变量的个数,可以省略.n 是字母表中符号的个数,

即以 n 为底表示数.

当用字符串形式表示时,函数值与 n 无关. 例如,

$$\mathrm{CONCAT}_2(s_2s_1, s_1s_2) = s_2s_1s_1s_2.$$

把底换成 5,同样有

$$\mathrm{CONCAT}_5(s_2s_1, s_1s_2) = s_2s_1s_1s_2.$$

如果把 CONCAT_n 看成 N 上的函数,情况就不同了. 由于 s_2s_1 以 2 和 5 为底表示的数分别是 5 和 11,s_1s_2 以 2 和 5 为底分别表示 4 和 7,$s_2s_1s_1s_2$ 以 2 和 5 为底分别表示 24 和 282,所以刚才的两个式子现在变成

$$\mathrm{CONCAT}_2(5,4) = 24$$

和

$$\mathrm{CONCAT}_5(11,7) = 282.$$

又如,

$$\mathrm{CONCAT}_2(2,3) = 11,$$

而

$$\mathrm{CONCAT}_5(2,3) = 13.$$

下面证明 $\mathrm{CONCAT}_n^{(m)}(u_1, u_2, \cdots, u_m)$是原始递归的.

当 $m=2$ 时,

$$\mathrm{CONCAT}_n(u,v) = u \cdot n^{|v|} + v$$

是原始递归的.

根据定义,有下述等式

$$\mathrm{CONCAT}_n^{(1)}(u) = u,$$

$$\mathrm{CONCAT}_n^{(m+1)}(u_1, \cdots, u_{m-1}) = \mathrm{CONCAT}_n^{(2)}(\mathrm{CONCAT}_n^{(m)}(u_1, \cdots, u_m), u_{m+1}).$$

用对 m 的归纳证明,容易证明对任意的 $m, n \geqslant 1$,$\mathrm{CONCAT}_n^{(m)}(u_1, \cdots, u_m)$是原始递归的.

3. 子串函数

(1) $\mathrm{RTEND}_n(w) = h(0, n, w)$,其中 h 由(4.6)式给出.

作为字函数，$\mathrm{RTEND}_n(\varepsilon)=\varepsilon$；当 $w\neq\varepsilon$ 时，$\mathrm{RTEND}_n(w)$等于 w 右端的第一个符号.

(2) $\mathrm{LTEND}_n(w)=h(|w|\dot{-}1,n,w)$.

$\mathrm{LTEND}_n(\varepsilon)=\varepsilon$；当 $w\neq\varepsilon$ 时，$\mathrm{LTEND}_n(w)$等于 w 左端的第一个符号.

(3) $\mathrm{RTRUNC}_n(w)=g(1,n,w)$，其中 g 由(4.5)式给出.

$\mathrm{RTRUNC}_n(\varepsilon)=\varepsilon$；当 $w\neq\varepsilon$ 时，$\mathrm{RTRUNC}_n(w)$等于 w 删去右端第一个符号后得到的子串. 常把 $\mathrm{RTRUNC}_n(w)$记作 w^-.

(4) $\mathrm{LTRUNC}_n(w)=w\dot{-}\mathrm{LTEND}(w)\cdot n^{|w|\dot{-}1}$.

$\mathrm{LTRUNC}_n(\varepsilon)=\varepsilon$；当 $w\neq\varepsilon$ 时，$\mathrm{LTRUNC}_n(w)$等于 w 删去左端第一个符号后得到的子串.

这 4 个于字符串函数都是原始递归的.

4. 换底函数

设字母表 $A=\{s_1,s_2,\cdots,s_n\}$和 $\tilde{A}=\{s_1,s_2,\cdots,s_l\}$，这里 $n<l$. 任给数 x，设 $w\in A^*$ 是 x 的 n 进制表示，w 也是 $\tilde{A}$ 上的字符串，把以 l 为底 w 所表示的数记作 $\mathrm{UPCHANGE}_{n,l}(x)$.

例如，$s_1s_2s_1$ 以 2 为底表示 9，以 5 为底表示 36，故 $\mathrm{UPCHANGE}_{2,5}(9)=36$.

反之，任给数 x，设 $w\in\tilde{A}^*$ 是 x 的 l 进制表示，删去 w 中不属于 A 的符号后得到 $w'\in A^*$，把以 n 为底 w' 所表示的数记作 $\mathrm{DOWNCHANGE}_{n,l}(x)$.

例如，由上例，$\mathrm{DOWNCHANGE}_{2,5}(36)=9$. 又如，$s_1s_4s_2s_3$ 以 5 为底表示 238，删去 s_3 和 s_4 得到 s_1s_2，s_1s_2 以 2 为底表示 4，故 $\mathrm{DOWNCHANGE}_{2,5}(238)=4$.

下述表达式表明这两个函数是原始递归的.

$$\mathrm{UPCHANGE}_{n,l}(x)=\sum_{i=0}^{k}h(i,n,x)\cdot l^i,$$

其中 $k=\mathrm{LENTH}_n(x)\dot{-}1$.

$$\mathrm{DOWNCHANGE}_{n,l}(x)=\sum_{i=0}^{m}h(i,l,x)\cdot\beta(i,x)\cdot l^{\tau(i,x)}$$

其中 $m=\mathrm{LENTH}_l(x)\dot{-}1$；当 $h(i,l,x)\leqslant n$ 时 $\beta(i,x)=1$，否则 $\beta(i,x)=0$；$\tau(i,x)=\sum_{j=0}^{i}\beta(j,x)\dot{-}1$. h 由(4.6)式给出.

4.2 程序设计语言 $\mathscr{S}_n$

语言 $\mathscr{S}_n$ 是为字符串运算设计的. 设字母表 A，A 中有 n 个符号. 语言 $\mathscr{S}_n$ 有 3 种类型的语句：

(1) 对每一个 $a\in A$，$V\leftarrow aV$. 在变量 V 当前值(是一个字符串)的左端添加 a 后重新赋给 V.

(2) $V\leftarrow V^-$. 当 $V\neq\varepsilon$ 时，删去 V 当前值右端的第一个符号后重新赋给 V；当 $V=\varepsilon$ 时，V 的值保持不变.

(3) 对每一个 $a\in A$ 和标号 L，

IF V ENDS a GOTO L.

如果 V 右端第一个符号是 a，则下一步执行标号为 L 的第一条指令；否则执行下一条指令 .

有关语言的其他规定，如变量、标号、程序的执行和计算的函数等等都和语言 $\mathscr{S}$ 相同. 这里不再引入空语句，因为是否有空语句不影响语言的计算能力.

定义 4.2 设字母表 A 中有 n 个符号，如果 A^* 上的 m 元部分函数能用 $\mathscr{S}_n$ 程序计算，则称这个函数是在 $\mathscr{S}_n$ 中部分可计算的，简称 $\mathscr{S}_n$ **部分可计算的**. 在 $\mathscr{S}_n$ 中部分可计算的全函数称作在 $\mathscr{S}_n$ 中可计算的，简称 $\mathscr{S}_n$ **可计算的**.

实际上，一个函数是 $\mathscr{S}_n$ 部分可计算的当且仅当它是部分可计算的. 我们将在本章完成这个定理的证明.

设 $A=\{s_1,s_2,\cdots,s_n\}$，下面给出 $\mathscr{S}_n$ 中的几条宏指令及其展开.

(1) IF $V \neq \varepsilon$ GOTO L

宏展开：

IF V ENDS s_1 GOTO L
IF V ENDS s_2 GOTO L
⋮
IF V ENDS s_n GOTO L

这 n 条指令可缩写成

IF V ENDS s_i GOTO L $(1 \leqslant i \leqslant n)$,

后面常用类似的缩写形式.

(2) $V \leftarrow \varepsilon$

宏展开：

$[A]$ $V \leftarrow V^-$
IF $V \neq \varepsilon$ GOTO A

(3) GOTO L

宏展开：

$Z \leftarrow s_1 Z$
IF $Z \neq \varepsilon$ GOTO L

(4) $V' \leftarrow V$

宏展开：

$Z \leftarrow \varepsilon$
$V' \leftarrow \varepsilon$
$[A]$ IF V ENDS s_1 GOTO B_1
IF V ENDS s_i GOTO B_i $(2 \leqslant i \leqslant n)$
GOTO C
$[B_i]$ $V \leftarrow V^-$
$V' \leftarrow s_i V'$
$Z \leftarrow s_i Z$
GOTO A $\}$ $(1 \leqslant i \leqslant n)$
$[C]$ IF Z ENDS s_1 GOTO D_1

IF Z ENDS s_i GOTO D_i, $(2\leqslant i\leqslant n)$

GOTO E

$[D_i]$ $Z\leftarrow Z'$

$V\leftarrow s_iV$ $\}$ $(1\leqslant i\leqslant n)$

GOTO C

这里有 n 个分别以标号 B_i 开始的 4 条指令组成的程序段，i 的值取 $1,2,\cdots,n$. 最后的 3 条指令也与此类似. 这个宏展开的思想和例 1.4 在 $\mathscr{S}'$ 中 $V'\leftarrow V$ 的宏展开的思想一样. 在这里要从右到左逐个检查 V 的符号，若是 s_i，则转到以标号 B_i 为首的程序段去处理.

(5) $V\leftarrow V+1$

先看几个例子. 取 $n=10$，注意这里使用的数字是 $1,2,\cdots,9,X$，其中 X 表示 10.

245	249	24X	2XX	XXX
$+\ 1$	$+\ 1$	$+\ 1$	$+\ 1$	$+\ 1$
246	24X	251	311	1111

我们的加法运算和普通的加法运算基本一样，但在进位时有点不同. 逢 10 得到 X，不进位. X 加 1 得到 11 时才进位，并留下 1. 图 4.1 给出计算的框图.

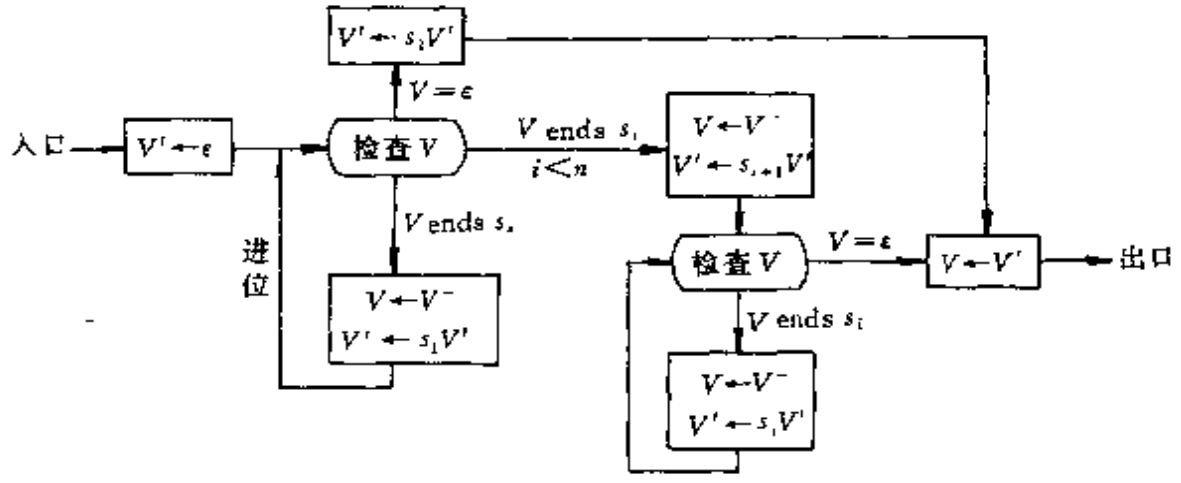


图 4.1 $V\leftarrow V+1$ 在 $\mathscr{S}_n$ 中的计算框图

宏指令 $V\leftarrow V+1$ 的展开清单：

$V'\leftarrow\epsilon$

$[B]$ IF V ENDS s_1 GOTO A_1

IF V ENDS s_i GOTO A_i $(2\leqslant i\leqslant n)$

GOTO F

$[A_i]$ $V\leftarrow V^-$

$V'\leftarrow s_{i+1}V'$ $(1\leqslant i\leqslant n-1)$

GOTO C

$[A_n]$ $V\leftarrow V^-$

$V'\leftarrow s_1V'$

GOTO B

$[C]$ IF V ENDS s_1 GOTO D_1

IF V ENDS s_i GOTO D_i $(2\leqslant i\leqslant n)$

GOTO F

$[D_i]$ $V\leftarrow V^-$

$V'\leftarrow s_iV'$ $(1\leqslant i\leqslant n)$

GOTO C

$[F]$ $V\leftarrow V'$

(6) $V\leftarrow V-1$

这是 $\mathscr{S}$ 中的减量语句，它的功能是计算 $V\dot{-}1$ 并且把结果重新赋给 V. 我们先做 n 个减法，仍取 $n=10$.

$$\begin{array}{cccc} 245 & 241 & 211 & 111 \\ -\ 1 & -\ 1 & -\ 1 & -\ 1 \\ \hline 244 & 23\overline{X} & 1\overline{XX} & \overline{XX} \end{array}$$

若最低位是 $s_i(2\leqslant i\leqslant n)$，则把这个 s_i 改成 s_{i-1}. 若最低位是 s_1，则要向左借位. 计算框图如图 4.2 所示.

宏指令 $V\leftarrow V-1$ 的展开清单：

$V'\leftarrow\epsilon$

$[B]$ IF V ENDS s_1 GOTO A_1

IF V ENDS s_i GOTO A_i $(2\leqslant i\leqslant n)$

GOTO E

$[A_i]$ $V\leftarrow V^-$
$V'\leftarrow s_{i-1}V'$
GOTO C
$(2\leqslant i\leqslant n)$

$[A_1]$ $V\leftarrow V^-$
IF $V\neq\varepsilon$ GOTO C_2
GOTO F

$[C_2]$ $V'\leftarrow s_nV'$
GOTO B

$[C]$ IF V ENDS s_1 GOTO D_1
IF V ENDS s_i GOTO D_i $(2\leqslant i\leqslant n)$
GOTO F

$[D_i]$ $V\leftarrow V^-$
$V'\leftarrow s_iV'$
GOTO C
$(1\leqslant i\leqslant n)$

$[F]$ $V\leftarrow V'$

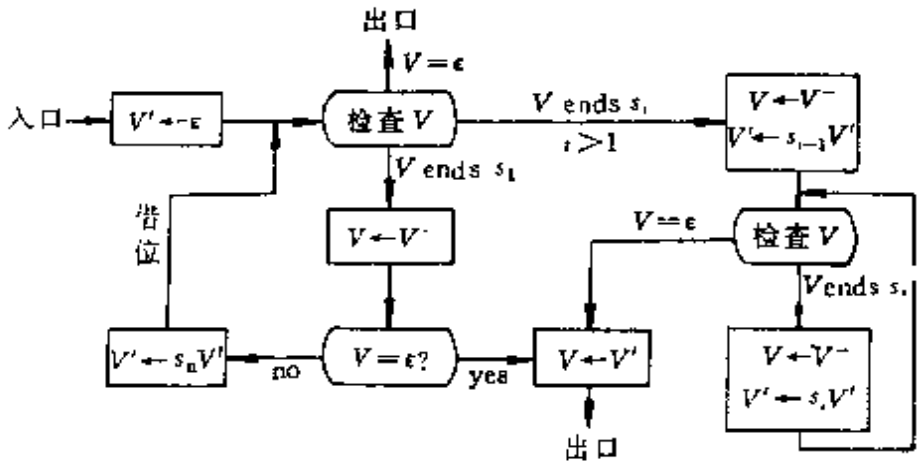


图 4.2 $V\leftarrow V-1$ 在 $\mathscr{S}_n$ 中的计算框图

现在讨论语言 $\mathscr{S}$ 和语言 $\mathscr{S}_n$ 的关系，先看 $\mathscr{S}_1$. 实际上，$\mathscr{S}_1$ 和 $\mathscr{S}$ 是相同的. 设字母表 $A=\{s_1\}$，数 x 的一进制表示是 s_1^x. 不难看到，$\mathscr{S}_1$ 中的语句

$$V\leftarrow s_1V \quad 和 \quad V\leftarrow V^-$$

的数值效果分别与 $\mathscr{S}$ 中的语句

$$V \leftarrow V+1 \quad 和 \quad V \leftarrow V-1$$

相同. 在 $\mathscr{S}_1$ 中，条件“V ENDS s_1”等价于条件“$V \neq \varepsilon$”，从而也等价于 $\mathscr{S}$ 中的条件“$V \neq 0$”. 于是，$\mathscr{S}_1$ 中的条件转移语句

$$\text{IF } V \text{ ENDS } s_1 \text{ GOTO } L$$

等价于 $\mathscr{S}$ 中的条件转移语句

$$\text{IF } V \neq 0 \text{ GOTO } L$$

因此，对于每一个 $\mathscr{S}_1$ 程序，把程序的每一个语句都改写成相对应的 $\mathscr{S}$ 语句，就可得到计算效果相同的 $\mathscr{S}$ 程序. 反之，也是一样. 于是，得到下述结论：一个函数是部分可计算的当且仅当它是 $\mathscr{S}_1$ 部分可计算的.

当 n 为任意固定的正整数时，上述结论仍然成立，但情况要复杂一些. 我们先证明这个结论的一半.

定理 4.2 对于每一个 $n>0$，部分可计算函数都是 $\mathscr{S}_n$ 部分可计算的.

证：设部分可计算函数 f 由 $\mathscr{S}$ 程序 $\mathscr{P}$ 计算，不妨设 $\mathscr{P}$ 中不含空语句. 把 $\mathscr{P}$ 中的每一条语句 $V \leftarrow V+1$，$V \leftarrow V-1$ 和 IF $V \neq 0$ GOTO L 分别替换成 $\mathscr{S}_n$ 中的宏指令 $V \leftarrow V+1$，$V \leftarrow V-1$ 和 IF $V \neq \varepsilon$ GOTO L，得到一个 $\mathscr{S}_n$ 程序. 这个程序和 $\mathscr{P}$ 计算同一个函数 f，因此 f 是 $\mathscr{S}_n$ 部分可计算的. □

上述证明采用了模拟方法. 用语言 $\mathscr{A}$ 模拟语言 $\mathscr{B}$，是用语言 $\mathscr{A}$ 的程序段来实现语言 $\mathscr{B}$ 的程序中的每一条指令，得到一个功能完全相同的语言 $\mathscr{A}$ 的程序. 从而证明语言 $\mathscr{B}$ 能计算的东西都能用语言 $\mathscr{A}$ 计算. 模拟是比较两个计算模型的计算能力的主要手段，在后面将多次用到.

实际上，定理 4.2 的逆也成立，并且同样可以用模拟方法证明. 我们把这个证明留给读者. 在本章，定理 4.2 的逆是后面几个定理的直接结果.

4.3 Post-Turing 语言 $\mathscr{T}$

语言 $\mathscr{T}$ 也是一种面向字符串运算的程序设计语言. 但是，与 $\mathscr{S}$，$\mathscr{S}_n$ 不同，它不使用变量，而是把要处理的字符串存放在一条带上.

如图 4.3 所示，带被分成无数个小方格，两头可以无限制地延伸，每一个小方格内可以存放一个符号. 有一个读写头，读写头总在扫描一个方格. 它能知道这个方格内存放着什么符号且能按照程序的命令做下述动作：改写被扫描方格内的符号(打印一个新符号)，向左移动一格，或者向右移动一格.

图 4.3 Post-Turing 语言的存储装置

Post-Turing 语言使用下述语句：

(1) PRINT a

打印符号 a，即用 a 代替正在被扫描的方格内的符号.

(2) IF a GOTO L

如果正在被扫描的方格内的符号是 a，则下一步执行标号为 L 的第一条指令；否则顺序执行下一条指令.

(3) RIGHT

读写头右移一格.

(4) LEFT

读写头左移一格.

设字母表 $A=\{s_1, s_2, \cdots, s_n\}$. 除 A 中的符号外，Post-Turing 程序要使用一个空白符 $s_0 \notin A$，常把 s_0 记作 B. 一个方格内存放着

s_0 表示这个方格是“空的”. 还可以把 s_0 当作标点符号使用，用来分隔字符串. 此外，程序还可以使用一些工作符号. 程序使用的所有符号(包括 A 中的符号、工作符号和空白符 B)称作**带字母表**. 在计算的每一步，带上只有有穷个非空白符，其余方格内都存放着 B. 带的内容和读写头的位置合称为**带格局**. 带格局可以表示成

$$\begin{array}{c} a_1a_2\cdots a_j\cdots a_k \\ \uparrow \end{array}$$

每一个 a_i 是带字母表中的一个符号，带的其余部分(a_1 的左边和 a_k 的右边)全是 B，箭头指明了读写头的位置.

Post-Turing 程序 $\mathscr{P}$ 计算的 A^* 上的 m 元部分函数 $\psi_{\mathscr{P}}^{(m)}$ 定义如下：对任意给定的输入 $x_1, x_2, \cdots, x_m \in A^*$，从初始带格局

$$\begin{array}{c} Bx_1Bx_2B\cdots Bx_m \\ \uparrow \end{array}$$

开始，如果计算最终停止，则 $\psi_{\mathscr{P}}^{(m)}(x_1, x_2, \cdots, x_m)$ 等于计算停止时从带的内容中删去不属于 A 的符号后得到的字符串；如果计算永不停止，则 $\psi_{\mathscr{P}}^{(m)}(x_1, x_2, \cdots, x_m)$ 无定义.

定义 4.3 设 f 是 A^* 上的 m 元部分函数，$\mathscr{P}$ 是一个 Post-Turing程序. 如果对任意的 $x_1, \cdots, x_m \in A^*$，有

$$f(x_1, \cdots, x_m) = \psi_{\mathscr{P}}^{(m)}(x_1, \cdots, x_m),$$

则称程序 $\mathscr{P}$ **计算**函数 f.

如果再附加两个条件：

(1) $\mathscr{P}$ 只使用 A 中的符号和 s_0，

(2) 当 $\mathscr{P}$ 停止计算时，带格局为

$$\begin{array}{c} Bf(x_1, \cdots, x_m) \\ \uparrow \end{array}$$

则称程序 $\mathscr{P}$ **严格地计算** f.

后面将会证明，如果存在 Post-Turing 程序计算 f，则存在 Post-Turing 程序严格地计算 f.

定义 4.4 设 f 是 A^* 上的部分函数，如果存在 Post-Turing 程序计算 f，则称 f 是在 $\mathscr{T}$ 中部分可计算的，简称 $\mathscr{T}$ **部分可计**

算的.

在 $\mathscr{T}$ 中部分可计算的全函数称作在 $\mathscr{T}$ 中可计算函数，简称 $\mathscr{T}$ **可计算函数**.

[**例 4.1**] 设字母表 $A=\{s_1,s_2,\cdots,s_n\}$，Post-Turing 程序 $\mathscr{P}$ 如下：

$[A]$ RIGHT
IF s_i GOTO A $(1\leqslant i\leqslant n)$
PRINT s_1
$[B]$ LEFT
IF s_i GOTO B $(1\leqslant i\leqslant n)$

程序 $\mathscr{P}$ 严格地计算函数 $f(x)=xs_1$. 对于输入 $x\in A^*$，计算停止在带格局

$$\overset{\uparrow}{B}xs_1.$$

[**例 4.2**] 给出计算函数 $f(s_1^k)=s_1^{2k}$ 的 Post-Turing 程序.

对输入 $w=s_1^k$，只需再复制一个 w. 把左端第一个 s_1 改写成 M，并且在 w 的右边复制一个 s_1，然后返回到左端把 M 改写成 s_1. 重复这个过程，直到把 w 的每一个 s_1 都改写成 M、又改写回 s_1 为止. 程序使用 3 个符号 s_1，B 和 M. M 是一个工作符号，用来标示哪些 s_1 已被复制. 程序清单如下：

$[A]$ RIGHT
IF B GOTO E
PRINT M
$[B]$ RIGHT
IF s_1 GOTO B
$[C]$ RIGHT
IF s_1 GOTO C
PRINT s_1
$[D]$ LEFT

IF s_1 GOTO D

IF B GOTO D

PRINT s_1

IF s_1 GOTO A

程序停止计算时的带格局为

$$Bs_1\cdots s_1Bs_1\cdots s_1$$
$$\uparrow$$

程序以一进制的方式计算 $2x$，但它不是严格地计算.

[**例 4.3**] 给出一个 Post-Turing 程序以一进制方式严格地计算函数

$$f(x)=\begin{cases}x/2 & \text{若 } x \text{ 是偶数}\\ \uparrow & \text{若 } x \text{ 是奇数}\end{cases}$$

程序只能使用 2 个符号 s_1 和 B. 输入 $w=s_1^i$，把左端的 2 个 s_1 改写成 BB，在 w 的右端写一个 s_1，然后回到左端. 重复这个过程. 如果能恰好把 w 的 s_1 都改写成 B，则计算结束；否则让程序进入死循环. 当然，还应考虑到计算结束时读写头的位置. 程序清单如下：

```
[A]  RIGHT
     IF B GOTO E
     PRINT B              / 删去 1 个 s1/
     RIGHT
[C]  IF B GOTO C          / 进入死循环 /
     PRINT B              / 再删去 1 个 s1/
[B]  RIGHT                / 向右查寻 /
     IF s1 GOTO B
[D]  RIGHT
     IF s1 GOTO D
     PRINT s1             /2 个 s1 换 1 个 s1/
[G]  LEFT                 / 返回左端 /
```

IF s_1 GOTO G

$[H]$ LEFT

IF s_1 GOTO H

IF B GOTO A

4.4 用 $\mathscr{T}$ 模拟 $\mathscr{S}_n$

本节给出用 $\mathscr{T}$ 模拟 $\mathscr{S}_n$ 的方法，并且模拟时除 s_0 之外不添加其他的符号. 从而证明任何在 $\mathscr{S}_n$ 中部分可计算的函数都可以用 $\mathscr{T}$ 程序严格地计算.

设字母表 $A=\{s_1,\cdots,s_n\}$. 先给出几条宏指令.

(1) GOTO L

宏展开

IF s_i GOTO L $\quad(0\leqslant i\leqslant n)$

注意我们只使用符号 $s_0,s_1,\cdots,s_n$.

(2) RIGHT TO NEXT BLANK

功能：将读写头移到右边下一个 B.

宏展开

$[A]$ RIGHT

IF B GOTO E

GOTO A

(3) LEFT TO NEXT BLANK

功能：将读写头移到左边下一个 B.

宏展开

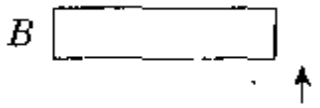
$[A]$ LEFT

IF B GOTO E

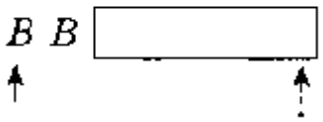
GOTO A

(4) MOVE BLOCK RIGHT

功能：把带格局



改变成



其中方框表示一个不含 B 的字符串. 即，把读写头当前位置和左边第一个 B 之间的字符串右移一格，在其左端空出一个“空白”B.

宏展开如下：

$[C]$ LEFT

IF s_i GOTO A_i $(0\leqslant i\leqslant n)$

$[A_i]$ RIGHT

PRINT s_i

LEFT

GOTO C

$(1\leqslant i\leqslant n)$

$[A_0]$ RIGHT

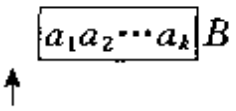
PRINT s_0

LEFT

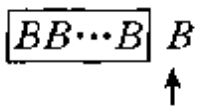
(5) ERASE A BLOCK

功能：把读写头当前位置与右边第一个空白符 B 之间的一段内容都清成空白，并且将读写头移至原右边第一个空白符 B 处.

即把带格局



改变成



其中 $a_1, a_2, \cdots, a_k$ 都不是 B.

宏展开

$[A]$ RIGHT

IF B GOTO E

PRINT B

GOTO A

下面还采用这样的缩写，在一条宏指令后面放上[k]，表示该宏指令重复 k 次. 如

RIGHT TO NEXT BLANK [3]

是

RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK

的缩写.

下面说明如何用 Post-Turing 程序模拟 $\mathscr{S}_n$ 程序. 设 $\mathscr{P}$ 是一个 $\mathscr{S}_n$ 程序，它使用输入变量 $X_1,\cdots,X_m$，输出变量 Y 和中间变量 $Z_1,\cdots,Z_k$. 将它们按下述顺序排列：

$$X_1,\cdots,X_m,Z_1,\cdots,Z_k,Y$$

分别记作 $V_1,V_2,\cdots,V_l$，这里 $l=m+k+1$. 在带上按这个顺序存放它们，每两个变量之间用一个 B 分隔开. 一步一步地模拟程序 $\mathscr{P}$ 的计算. 在模拟每一步的开始时，带格局形如

$$Bx_1Bx_2\cdots Bx_mBz_1\cdots Bz_kBy$$
$$\uparrow$$

其中 $x_1,x_2,\cdots,x_m,z_1,\cdots,z_k,y$ 依次是 $X_1,X_2,\cdots,X_m,Z_1,\cdots,Z_k,Y$ 的当前值. 并在每一步模拟完成时，把带格局恢复到上述形状，为下一步模拟做好准备.

(1) 模拟 $V_j\leftarrow s_iV_j$

把读写头移到 V_l 右边的 B 处，依次将 $V_l,\cdots,V_j$ 右移一格，在 V_j 的左边空出一个空格，插入 s_i，最后将读写头移回到原处. 模拟程序的清单如下：

RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [$l-j+1$]
RIGHT
PRINT s_i
LEFT TO NEXT BLANK [j]

(2) 模拟 $V_j \leftarrow V_j$

要注意 V_j 的值为 ϵ 时的特殊情况. 将读写头右移到 V_j 右边的 B 处,左移一格. 若是 B,则说明 V_j 的值为 ϵ. 把读写头恢复到原来的位置即可;若不是 B,则依次把 $V_j, V_{j-1}, \cdots, V_1$ 右移一格(这样做的结果是删去了 V_j 最右端的符号)并把读写头放到正确位置上. 模拟程序的清单如下:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF B GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j - 1]
```

(3) IF V_j ENDS s_i GOTO L

将读写头右移到 V_j 右边的 B 处,再左移一格检查 V_j 最右端的符号. 若这个符号是 s_i,则转向 L;否则不转向 L. 当然,不论是转还是不转,都要把读写头恢复到原来的位置. 模拟程序的清单如下:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF s_i GOTO C
        GOTO D
[C]     LEFT TO NEXT BLANK [j]
        GOTO L
[D]     RIGHT
        LEFT TO NEXT BLANK [j]
```

倒数第 2 行的 RIGHT 是考虑到 V_j 的值为 ϵ 的情况. 若 $V_j \neq \epsilon$,此时读写头扫描 V_j 最右端的符号,V_1 左端的 B(应把读写头恢复到这个位置)是读写头左边的第 j 个 B;若 $V_j = \epsilon$,此时读写头扫描

V_{j-1}右边的 B，V_1 左端的 B 是读写头左边第 $j-1$ 个 B. 为了把这两种情况统一起来，读写头右移一格. 当 $V_j=\varepsilon$ 时，读写头左边多了一个 B，当 $V_j\neq\varepsilon$ 时，读写头左边 B 的个数不变. 这样一来，到 V_1 左端的 B 为止，读写头左边都恰好有 j 个 B.

按照上述说明，用 Post-Turing 程序模拟程序 $\mathscr{P}$ 的每一条指令，得到一个 Post-Turing 程序. 对于输入 $x_1,\cdots,x_m$，这个程序最终停止计算当且仅当 $\mathscr{P}$ 最终停止计算，并且当计算停止时，带格局为

$$\overset{\uparrow}{B}x_1Bx_2\cdots Bx_mBz_1\cdots Bz_kBy$$

其中 $x_1,\cdots,x_m,z_1,\cdots,z_k,y$ 分别为 $\mathscr{P}$ 停止计算时变量 $X_1,\cdots,X_m,Z_1,\cdots,Z_k,Y$ 的值. 为了使这个 Post-Turing 程序恰好输出 y，还需要删去除 y 之外的所有字符串. 为此在程序的末尾添加

ERASE A BLOCK $[l-1]$

如果程序 $\mathscr{P}$ 中存在标号 E，有语句 IF V_j ENDS s_i GOTO E，但没有以 E 为标号的指令(不妨假设只有一个这样的标号)，则在这段程序的开头加标号$[E]$. 这样就完成了对程序 $\mathscr{P}$ 的模拟，于是得到下述定理.

定理 4.3 如果函数 f 在 $\mathscr{S}_n$ 中是部分可计算的，则存在 Post-Turing 程序严格地计算 f.

4.5 用 $\mathscr{S}$ 模拟 $\mathscr{T}$

本节将要证明下述定理：

定理 4.4 如果部分函数 f 可以用 Post-Turing 程序计算，则 f 是部分可计算的.

于是语言 $\mathscr{S}$、$\mathscr{S}_n$ 和 $\mathscr{T}$ 之间的关系如图 4.4 所示. 图中的蕴涵关系形成一个封闭的圈，所以圈上的 4 个概念是相互等价的. 将它总结成下述定理.

定理 4.5 设字母表 A 有 n 个符号，f 是 A^* 上的 m 元部分函数，则下述命题是等价的：

(1) f 是部分可计算的.

(2) f 在 $\mathscr{S}_n$ 中是部分可计算的.

(3) f 可以用 Post-Turing 程序严格地计算.

(4) f 可以用 Post-Turing 程序计算.

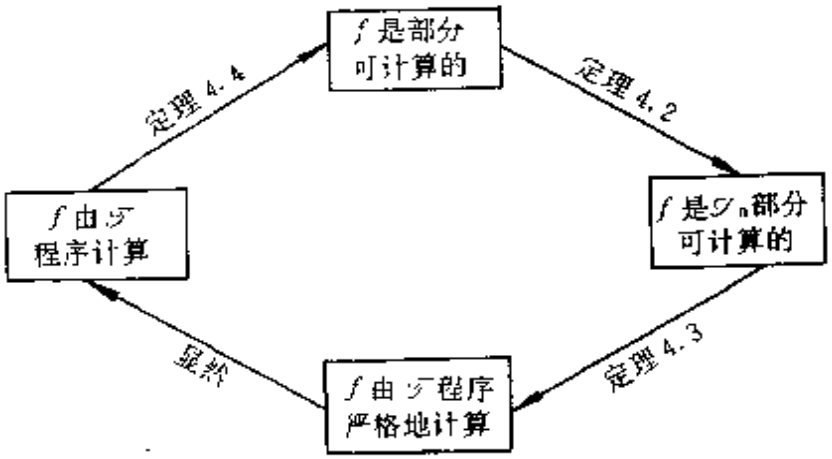


图 4.4

可计算性的这些不同的概念都是等价的，从而表明把它们作为人们直观的可计算性的严格地形式描述是正确的. 后面我们还能继续看到这样的证据.

推论 4.6 对于任意的 $n, l \geqslant 1$，N 上的 m 元部分函数在 $\mathscr{S}_n$ 中是部分可计算的当且仅当它在 $\mathscr{S}_l$ 中是部分可计算的.

证：这两个条件都等价于这个函数是部分可计算的. □

推论 4.7 每一个 N 上的部分可计算函数都可以用仅使用符号 s_0, s_1 的 Post-Turing 程序严格地计算.

证：每一个部分可计算函数在 $\mathscr{S}_1$ 中是部分可计算的，从而可以用仅使用符号 s_0, s_1 的 Post-Turing 程序严格地计算. □

现在我们回过头来证明定理 4.4.

设 f 是 A^* 上的 m 元部分函数，这里 $A=\{s_1, \cdots, s_n\}$. $\mathscr{P}$ 是计算 f 的 Post-Turing 程序，这里不假设 $\mathscr{P}$ 严格地计算 f. 除 $s_1, \cdots,$

s_n 和 B 外，$\mathscr{P}$ 还要使用符号 $s_{n+1},\cdots,s_r(r\geqslant n)$. 将 $\mathscr{P}$ 使用的符号排列如下：

$$s_1,\cdots,s_n,s_{n+1},\cdots,s_r,B$$

这次把 B 记作 s_{r+1}. 要构造一个 $\mathscr{S}$ 程序 $\mathscr{Q}$ 以 $r+1$ 为底计算 f. $\mathscr{Q}$ 由 3 段组成

BEGINNING

MIDDLE

END

核心部分是 MIDDLE，它逐条地模拟 $\mathscr{P}$. BEGINNING 的任务是按照 MIDDLE 的要求把输入转化成需要的数据. END 的任务是在模拟 $\mathscr{P}$ 的计算结束后，从结果中抽取出输出.

$\mathscr{P}$ 的带格局可以表示成

$$\cdots BB\alpha s_i\beta BB\cdots$$
$$\uparrow$$

读写头扫描符号 s_i. α 是一个非空字符串，它的左边全是空白符 B. β 也是一个非空字符串，它的右边全是空白符 B. 在这种表示中，α 不是唯一的，在 α 的左端多加几个 B 或少几个 B 都是一样. 同样地，在 β 的右端也可以多加几个 B 或少几个 B. 如带格局

$$\cdots Bs_1s_2s_1Bs_2s_2s_1B\cdots$$
$$\uparrow$$

α 可以为 s_1s_2，也可以为 Bs_1s_2 或 BBs_1s_2 等. β 可以为 $Bs_2s_2s_1$，也可以为 $Bs_2s_2s_1B$ 或 $Bs_2s_2s_1BB$ 等. $\mathscr{Q}$ 用 3 个变量 L、H 和 R 记录 $\mathscr{P}$ 的带格局. H 存放读写头正在扫描的符号 s_i 的值 i. L 存放 α 以 $r+1$ 为底表示的数，R 存放 β 以 $r+1$ 为底表示的数. 对于刚才的例子，设 $r=2$，取 $\alpha=s_1s_2$，$\beta=Bs_2s_2s_1$，则

$$H=1,$$
$$L=1\times 3+2=5,$$
$$R=3\times 3^3+2\times 3^2+2\times 3+1=106.$$

下面说明如何用 $\mathscr{S}$ 模拟 $\mathscr{T}$ 的语句.

(1) PRINT s_i

$$H \leftarrow i$$

(2) IF s_i GOTO L

$$\text{IF } H = i \text{ GOTO } L$$

(3) RIGHT

$$L \leftarrow \text{CONCAT}_{r+1}(L, H)$$

$$H \leftarrow \text{LTEND}_{r+1}(R)$$

$$R \leftarrow \text{LTRUNC}_{r+1}(R)$$

$$\text{IF } R \neq 0 \text{ GOTO } E$$

$$R \leftarrow r+1$$

最后两句是考虑到下述情况：设带格局为

$$\cdots B\alpha s_i s_j B \cdots$$
$$\uparrow$$

并且 $\beta = s_j$，读写头右移一格后，带格局为

$$\cdots B\alpha s_i s_j B \cdots$$
$$\uparrow$$

由于要求 $\beta \neq \varepsilon$，故应取 $\beta = B$. 相应地，R 的值应为 $r+1$.

(4) LEFT

$$R \leftarrow \text{CONCAT}_{r+1}(H, R)$$

$$H \leftarrow \text{RTEND}_{r+1}(L)$$

$$L \leftarrow \text{RTRUNC}_{r+1}(L)$$

$$\text{IF } L \neq 0 \text{ GOTO } E$$

$$L \leftarrow r+1$$

把 $\mathscr{P}$ 的每一条语句都替换成模拟它的程序就得到 MIDDLE.

由于 f 是 A^* 上的 m 元函数，而 A 包含 n 个符号，故 $\mathscr{Q}$ 的输入变量 $X_1, X_2, \cdots, X_m$ 的初值应是输入字符串 $x_1, x_2, \cdots, x_m$ 以 n 为底表示的数. BEGINNING 把它们转换成以 $r+1$ 为底的数，并给出 L, H, R 的初值. $\mathscr{P}$ 的初始带格局为

$$B\underset{\uparrow}{x_1}Bx_2B\cdots x_mB\cdots$$

于是，BEGINNING 为

$$L\leftarrow r+1$$

$$H\leftarrow r+1$$

$$Z_i\leftarrow \mathrm{UPCHANGE}_{n,r+1}(X_i)\quad (1\leqslant i\leqslant m)$$

$$R\leftarrow \mathrm{CONCAT}_{r+1}(Z_1,r+1,\cdots,Z_m,r+1)$$

最后，END 把 L,H,R 合并存入一个变量，得到 $\mathscr{P}$ 停止计算时带上的内容(如果计算最终停止的话)，删去不属于 A 的符号后转换成以 n 为底的数作为 $\mathscr{Q}$ 的输出. END 为

$$Z\leftarrow \mathrm{CONCAT}_{r+1}(L,H,R)$$

$$Y\leftarrow \mathrm{DOWNCHANGE}_{n,r+1}(Z)$$

至此，完成了用 $\mathscr{S}$ 模拟 $\mathscr{T}$ 的描述. □

习　题

1. 计算：

(1) $\mathrm{CONCAT}_3(12,15)$，$\mathrm{CONCAT}_5(12,15)$.

(2) $\mathrm{UPCHANGE}_{3,7}(15)$，$\mathrm{UPCHANGE}_{2,7}(15)$，$\mathrm{UPCHANGE}_{2,10}(15)$，$\mathrm{DOWNCHANGE}_{3,7}(15)$，$\mathrm{DOWNCHANGE}_{2,7}(15)$，$\mathrm{DOWNCHANGE}_{2,10}(20)$.

2. 写出计算 $\mathrm{UPCHANGE}_{n,l}(x)$和 $\mathrm{DOWNCHANGE}_{n,l}(x)$的 $\mathscr{S}$ 程序.

3. 设字母表 A. $\forall x\in A^*$，$f(x)=x^R$，这里 x^R 是 x 的反转，即将 x 的字符顺序颠倒过来. 如，$(abb)^R=bba$. 试证明：$f(x)$是原始递归的.

4. 证明 $f(x)=x^R$ 是 $\mathscr{S}_n$ 可计算的. (x^R 的定义见上题.)

5. 用 $\mathscr{S}$ 模拟 $\mathscr{S}_n$.

6. 用 Post-Turing 程序严格地计算 $f(x)=x^R$(x^R 的定义见第 3 题).

7. 用 Post-Turing 程序以一进制方式严格地计算下述函数：

(1) $2x$.　　(2) $x-y$.

第五章　递归可枚举集

5.1　递归集和递归可枚举集

本章讨论集合识别的可计算性. 所谓**集合识别问题**，是指对于给定的集合，任给一个元素，问这个元素是否属于该集合？集合识别问题又称作**集合的成员资格问题**. 这一节限制在自然数集 N 上.

设 $B\subseteq N$，B 的**特征函数** χ_B 是一个谓词，

$$\chi_B(x)\Leftrightarrow x\in B,\quad \forall x\in N,$$

或

$$\chi_B(x)=\begin{cases}1 & \text{若 } x\in B\\ 0 & \text{否则}\end{cases}$$

B 可用它的特征函数表示成

$$B=\{x\in N\mid \chi_B(x)\}.$$

定义 5.1　设 $B\subseteq N$，如果 B 的特征函数 χ_B 是原始递归(可计算)的，则称集合 B 是**原始递归(递归)的**.

如果存在部分可计算函数 g 使得

$$B=\{x\in N\mid g(x)\downarrow\},$$

则称集合 B 是**递归可枚举的**(缩写为 r.e.).

根据上述定义，如果 B 是递归集，则存在 $\mathscr{S}$ 程序 $\mathscr{P}$，它对所有的输入都停机. 当输入 $x\in B$ 时，$\mathscr{P}$ 输出 1；当 $x\notin B$ 时，$\mathscr{P}$ 输出 0. 如果 B 是 r.e. 集，则存在 $\mathscr{S}$ 程序 $\mathscr{Q}$，当输入 $x\in B$ 时，$\mathscr{Q}$ 停机；当 $x\notin B$ 时，$\mathscr{Q}$ 永不停机. 非形式地说，若 B 是递归集，则存在一个算法判断任给的数 x 是否属于 B. 若 B 是 r.e. 集，则只能有这样的算法，当 $x\in B$ 时算法给出肯定的回答，而当 $x\notin B$ 时算法不能

给出回答. 因为在计算停止之前，无法判断是最终会停机而尚未停机、还是永不停机. 也就是说，这个算法只能肯定 $x\in B$，而不能肯定 $x\notin B$. 这是递归可枚举集与递归集的本质区别.

定理 5.1 如果集合 B 和 C 是(原始)递归的，则集合 $B\cup C$，$B\cap C$ 和 $\overline{B}$ 都是(原始)递归的.

证：设 B 和 C 的特征函数分别是 χ_B 和 χ_C，则

$$B\cup C=\{x\in N\mid \chi_B(x)\vee\chi_C(x)\},$$

$$B\cap C=\{x\in N\mid \chi_B(x)\wedge\chi_C(x)\},$$

$$\overline{B}=\{x\in N\mid \neg\,\chi_B(x)\}.$$

根据定义 5.1 和定理 2.6 立即得到所需结论. □

定理 5.2 递归集必是递归可枚举集.

证：设 B 是一个递归集，它的特征函数 χ_B 是递归的. 考虑程序 $\mathscr{P}$：

$$[A]\quad \text{IF}\ \neg\,\chi_B(X)\ \text{GOTO}\ A$$

显然，对所有的 $x\in N$，

$$\chi_B(x)\Leftrightarrow \text{对输入}\ x,\mathscr{P}\ \text{最终停机},$$

故

$$B=\{x\in N\mid \psi_{\mathscr{P}}(x)\downarrow\},$$

得证 B 是 r.e.. □

定理 5.3 集合 B 是递归的当且仅当 B 和 $\overline{B}$ 是递归可枚举的.

证：设 B 是递归的. 由定理 5.1，$\overline{B}$ 也是递归的. 再由定理5.2，B 和 $\overline{B}$ 是 r.e..

反之，设 B 和 $\overline{B}$ 是 r.e.，则存在部分可计算函数 $g(x)$和 $h(x)$ 使得

$$B=\{x\in N\mid g(x)\downarrow\},$$

$$\overline{B}=\{x\in N\mid h(x)\downarrow\}.$$

设 $g(x)$和 $h(x)$分别由程序 $\mathscr{P}$ 和 $\mathscr{Q}$ 计算，要利用 $\mathscr{P}$ 和 $\mathscr{Q}$ 构造计

算谓词$\chi_B(x)$的程序$\mathscr{R}$.$\mathscr{R}$执行$\mathscr{P}$一步、执行$\mathscr{Q}$一步.如果$\mathscr{P}$,$\mathscr{Q}$都不停机,则回到初始状态,重新执行$\mathscr{P}$二步、执行$\mathscr{Q}$二步,….由于$x\in B$和$x\in\overline{B}$必有且只有一个为真,$g(x)$和$h(x)$必有且只有一个有定义,因此$\mathscr{P}$和$\mathscr{Q}$必有且只有一个在某一步停机.当$\mathscr{P}$停机时,$\mathscr{R}$令$Y=1$并停机;当$\mathscr{Q}$停机时,$\mathscr{R}$令$Y=0$并停机.设$p=\#(\mathscr{P})$,$q=\#(\mathscr{Q})$,程序$\mathscr{R}$如下:

$[A]$ IF $\mathrm{STP}^{(1)}(X,p,T)$ GOTO C
IF $\mathrm{STP}^{(1)}(X,q,T)$ GOTO E
$T\leftarrow T+1$
GOTO A
$[C]$ $Y\leftarrow Y+1$ □

定理证明中利用计步函数联合程序$\mathscr{P}$和$\mathscr{Q}$构造$\mathscr{R}$的方法是很有用的,下一个定理的证明将再次用到这个方法.

定理 5.4 如果集合B和C是递归可枚举的,则$B\cup C$和$B\cap C$也是递归可枚举的.

证:设部分可计算函数$g(x)$和$h(x)$使得

$$B=\{x\in N\mid g(x)\downarrow\},$$
$$C=\{x\in N\mid h(x)\downarrow\},$$

则

$$B\cap C=\{x\in N\mid g(x)\downarrow\ \wedge\ h(x)\downarrow\}.$$

考虑程序

$Z\leftarrow X$
$Y\leftarrow g(X)$
$Y\leftarrow h(Z)$

设它计算的函数为$f(x)$,则有

$$f(x)\downarrow\Leftrightarrow g(x)\downarrow\ \wedge\ h(x).$$

所以

$$B\cap C=\{x\in N\mid f(x)\downarrow\},$$

得证$B\cap C$是r.e..

而

$$B \cup C = \{x \in N \mid g(x)\downarrow \ \vee \ h(x)\downarrow\}.$$

设 $g(x)$和 $h(x)$分别由代码为 p 和 q 的程序计算. 考虑程序

$[A]$ IF $\mathrm{STP}^{(1)}(X,p,T)$ GOTO E
IF $\mathrm{STP}^{(1)}(X,q,T)$ GOTO E
$T \leftarrow T+1$
GOTO A

设它计算的函数为 $r(x)$,则

$$r(x)\downarrow \Leftrightarrow g(x)\downarrow \ \vee \ h(x)\downarrow,$$

从而

$$B \cup C = \{x \in N \mid r(x)\downarrow\},$$

得证 $B \cup C$ 是 r. e.. □

对每一个 $n \in N$,记

$$\mathrm{W}_n = \{x \in N \mid \Phi(x,n)\downarrow\}.$$

当 n 遍取全体自然数时,$\Phi(x,n)$给出所有的部分可计算函数,故有:

定理 5.5(枚举定理) 集合 B 是递归可枚举的当且仅当存在 $n \in N$ 使得 $B=\mathrm{W}_n$.

由这个定理,序列 $\mathrm{W}_0,\mathrm{W}_1,\mathrm{W}_2,\cdots$枚举出全部 r. e. 集. 这也是定理名字的来源.

5.2 递归语言和递归可枚举语言

设字母表 $A=\{s_1,s_2,\cdots,s_n\}$,A^* 的任何子集称作 A 上的语言,简称**语言**. 语言 L 的特征函数

$$\chi_L(w) \Leftrightarrow w \in L, \forall\ w \in A^*,$$

或

$$\chi_L(w)=\begin{cases}1 & \text{若 } w \in L\\ 0 & \text{若 } w \in A^*-L.\end{cases}$$

类似定义 5.1,关于语言有下述定义.

定义 5.2 设字母表 A，$L\subseteq A^*$. 如果 L 的特征函数 χ_L 是原始递归(可计算)的，则称语言 L 是**原始递归(递归)的**.

如果存在 A^* 上的部分可计算函数 g 使得

$$L=\{w\in A^*\mid g(w)\downarrow\},$$

则称语言 L 是**递归可枚举的**(缩写成 r. e.).

可以像上一章那样，指定 A 中符号的顺序，A 上的字符串成为数的 n 进制表示. 于是，A 上的语言 L 成为 N 的子集，并且 L 是递归可枚举的(递归的、原始递归的)当且仅当 L 作为 N 的子集是递归可枚举的(递归的、原始递归的). 因此，上一节的定理 5.1—5.4 对于 A 上的语言同样成立. 将它们叙述如下：

定理 5.1′ 如果字母表 A 上的语言 L_1 和 L_2 是(原始)递归的，则 $L_1\cup L_2$，$L_1\cap L_2$ 和 $\overline{L}_1=A^*-L_1$ 都是(原始)递归的.

定理 5.2′ A 上的递归语言必是递归可枚举的.

定理 5.3′ A 上的语言 L 是递归的当且仅当 L 和 $\overline{L}=A^*-L$ 都是递归可枚举的.

定理 5.4′ 如果 A 上的语言 L_1 和 L_2 是递归可枚举的，则 $L_1\cup L_2$ 和 $L_1\cap L_2$ 是递归可枚举的.

由定理 4.5 可以得到下述两个定理.

定理 5.6 设字母表 A，A 有 n 个符号，$L\subseteq A^*$，则下述命题是等价的：

(1) L 是递归可枚举的.

(2) 存在 $\mathscr{S}_n$ 部分可计算函数 g 使得

$$L=\{w\in A^*\mid g(w)\downarrow\}.$$

(3) 存在可以用 Post-Turing 程序严格地计算的函数 g 使得

$$L=\{w\in A^*\mid g(w)\downarrow\}.$$

(4) 存在可以用 Post-Turing 程序计算的函数 g 使得

$$L=\{w\in A^*\mid g(w)\downarrow\}.$$

定理 5.7 设字母表 A，A 有 n 个符号，$L\subseteq A^*$，则下述命题是等价的：

(1) L 是递归的.

(2) χ_L 在 $\mathscr{S}_n$ 中是可计算的.

(3) χ_L 可以用 Post-Turing 程序严格地计算.

(4) χ_L 可以用 Post-Turing 程序计算.

设 $A\subset\tilde{A}, L\subseteq A^*$. 于是,$L$ 既是 A 上的语言(以 A 为字母表)、又是 $\tilde{A}$ 上的语言(以 $\tilde{A}$ 为字母表). 人们自然要问:L 是否是 r. e.(递归的)和把它看作 A 上的语言,还是 $\tilde{A}$ 上的语言有关吗? 我们当然希望是无关的. 答案也确实如此.

定理 5.8 设 $A\subset\tilde{A}, L\subseteq A^*$,那么 L 是字母表 A 上的递归可枚举语言当且仅当 L 是字母表 $\tilde{A}$ 上的递归可枚举语言.

证:设 $A=\{s_1,s_2,\cdots,s_n\}$,$\tilde{A}=\{s_1,s_2,\cdots,s_l\}$,这里 $n<l$.

设 L 是 A 上的 r. e. 语言,则存在 A^* 上的部分函数 g,g 由 Post-Turing 程序 $\mathscr{P}$ 计算,并且

$$L=\{w\in A^*\mid g(w)\downarrow\}.$$

定义 $\tilde{A}^*$ 上的部分函数

$$\tilde{g}(w)=\begin{cases} g(w) & \text{若 } w\in A^* \\ \uparrow & \text{若 } w\in \tilde{A}^*-A^*. \end{cases}$$

显然,

$$L=\{w\in \tilde{A}^*\mid \tilde{g}(w)\downarrow\}.$$

要证 $\tilde{g}$ 是部分可计算的,为此构造计算 $\tilde{g}$ 的 Post-Turing 程序 $\mathscr{Q}$ 如下:首先检查输入 $w\in\tilde{A}^*$ 的每一个符号. 如果 w 含有 $\tilde{A}-A$ 中的符号,则进入永不停机的死循环;否则 $w\in A^*$,恢复到初始带格局后执行程序 $\mathscr{P}$,从而得证 L 是 $\tilde{A}$ 上的 r. e. 语言.

程序 $\mathscr{Q}$ 如下:

$[A]$ RIGHT

IF s_i GOTO A $\quad(1\leqslant i\leqslant n)$

$[B]$ IF s_i GOTO B $\quad(n+1\leqslant i\leqslant l)$

LEFT TO NEXT BLANK

$\mathscr{P}$

反之，设 L 是 $\tilde{A}$ 上的 r.e. 语言，那么存在 $\tilde{A}^*$ 上的部分函数 f，f 由 Post-Turing 程序 $\mathscr{P}$ 计算，并且

$$L = \{w \in \tilde{A}^* \mid f(w)\downarrow\}.$$

由于 $L \subseteq A^*$，对任意的 $w \in \tilde{A}^* - A^*$，有 $f(w)\uparrow$. 如下构造 A^* 上的部分函数 g，对任意的 $w \in A^*$，若 $f(w)\downarrow$，则 $g(w)$ 等于 $f(w)$ 删去 $\tilde{A}-A$ 中的符号后得到的字符串；若 $f(w)\uparrow$，则 $g(w)\uparrow$. 显然

$$L = \{w \in A^* \mid g(w)\downarrow\}.$$

再考虑程序 $\mathscr{P}$，取 A 作字母表，把 $\tilde{A}-A$ 中的符号全都作为工作符号，则它计算 g. 从而得证 L 也是 A 上的 r.e. 语言. □

定理 5.9 设 $A \subset \tilde{A}$，$L \subseteq A^*$，则 L 是 A 上的递归语言当且仅当 L 是 $\tilde{A}$ 上的递归语言.

证：设 L 是 A 上的递归语言，则 L 和 $A^* - L$ 是 A 上的 r.e. 语言. 由定理 5.10，L 和 $A^* - L$ 也都是 $\tilde{A}$ 上的 r.e. 语言. 而

$$\tilde{A}^* - L = (\tilde{A}^* - A^*) \cup (A^* - L).$$

不难证明 $\tilde{A}^* - A^*$ 是 $\tilde{A}$ 上的 r.e. 语言，从而 $\tilde{A}^* - L$ 也是 $\tilde{A}$ 上的 r.e. 语言. 由 L 和 $\tilde{A}^* - L$ 都是 $\tilde{A}$ 上的 r.e. 语言，得证 L 是 $\tilde{A}$ 上的 r.e. 语言.

反之，设 L 是 $\tilde{A}$ 上的递归语言，则 L 和 $\tilde{A}^* - L$ 都是 $\tilde{A}$ 上的 r.e. 语言. 由于 $L \subseteq A^*$，故 L 也是 A 上的 r.e. 语言. 而

$$A^* - L = (\tilde{A}^* - L) \cap A^*.$$

A^* 显然是 A 上的 r.e. 语言，也是 $\tilde{A}$ 上的 r.e. 语言，因此 $A^* - L$ 是 $\tilde{A}$ 上的 r.e. 语言. 注意到 $A^* - L \subseteq A^*$，故 $A^* - L$ 也是 A 上的 r.e. 语言. 得证 L 是 A 上的递归语言. □

5.3 非递归集和非递归可枚举集

一方面，自然数集合的所有子集是不可数的. 另一方面，只有可数个可计算谓词和可数个部分可计算函数，从而只有可数个递

归集和可数个 r.e. 集. 因此，一定存在非递归集和非 r.e. 集. 本节将给出这样的例子.

令

$$\mathrm{K} = \{n \in N \mid n \in \mathrm{W}_n\}.$$

定理 5.10 K 是递归可枚举的、但不是递归的.

证：由定义，

$$\mathrm{K} = \{n \in N \mid \Phi(n,n) \downarrow\}.$$

$\Phi(n,n)$是部分可计算的，故 K 是 r.e..

由定义，K 的特征函数是 HALT(n,n)，即

$$\mathrm{K} = \{n \in N \mid \mathrm{HALT}(n,n)\}.$$

已知 HALT(n,n)不是可计算谓词(定理 3.1)，故 K 不是递归集.

□

推论 5.11 $\overline{\mathrm{K}}$ 不是递归可枚举的.

证：由定理 5.10 和定理 5.3 推出. □

设 $\mathscr{F}$ 表示一元部分可计算函数的全体，$\Gamma \subseteq \mathscr{F}$. 记

$$R_\Gamma = \{t \in N \mid \Phi_t \in \Gamma\}$$

R_Γ 称作 Γ 的**下标集**.

定理 5.12(Rice 定理) 设 $\Gamma \subseteq \mathscr{F}$. 如果 $\Gamma \neq \mathscr{F}$ 且 $\Gamma \neq \varnothing$，则 R_Γ 不是递归的.

证：因为 $\Gamma \neq \mathscr{F}$ 且 $\Gamma \neq \varnothing$，必存在部分可计算函数 $f(x)$和 $g(x)$，使得 $f(x) \in \Gamma$，$g(x) \notin \Gamma$. 假设 R_Γ 是递归的. 令

$$h(t,x) = \begin{cases} g(x) & \text{若 } t \in R_\Gamma \\ f(x) & \text{否则} \end{cases}$$

$h(t,x)$是部分可计算的. 由递归定理，存在数 e 使得

$$\Phi_e(x) = h(e,x) = \begin{cases} g(x) & \text{若 } e \in R_\Gamma \\ f(x) & \text{否则} \end{cases}$$

于是，

$$e \in R_\Gamma \Rightarrow \Phi_e(x) = g(x) \notin \Gamma \Rightarrow e \notin R_\Gamma,$$

$$e \notin R_\Gamma \Rightarrow \Phi_e(x) = f(x) \in \Gamma \Rightarrow e \in R_\Gamma,$$

即

$$e \in R_r \quad \Leftrightarrow \quad e \notin R_r,$$

矛盾. 所以, R_r 不是递归的. □

Rice 定理表明下述集合都是非递归的:

$$A=\{t\in N \mid \forall x \ \ \Phi_t(x)=\Phi_a(x)\},$$

$$B=\{t\in N \mid \Phi_t(a)\downarrow\},$$

$$C=\{t\in N \mid \exists x \ \ \Phi_t(x)=a\},$$

$$D=\{t\in N \mid \text{除有限个 } x \text{ 值外}, \Phi_t(x)\downarrow\},$$

$$E=\{t\in N \mid \Phi_t(x) \text{是原始递归函数}\},$$

$$T=\{t\in N \mid \Phi_t(x) \text{是全函数}\},$$

其中 a 是任意给定的常数.

T 是所有可计算函数组成的集合的下标集. 在 7.4 节中将证明它不仅是非递归的, 也是非 r.e.. 下面先证明 $\overline{T}$ 是非 r.e., 从而得到这样一个集合, 它和它的补都是非 r.e..

定理 5.13 $\overline{T}$ 不是递归可枚举的.

证: 假设 $\overline{T}$ 是 r.e. 集, 则存在部分可计算函数 $f(t)$ 使得

$$\overline{T} = \{t \in N \mid f(t)\downarrow\}.$$

作部分可计算函数

$$g(t,x) = u_2^2(f(t),x) = \begin{cases} x & \text{若 } f(t)\downarrow \\ \uparrow & \text{否则} \end{cases}$$

由递归定理, 存在 $e\in N$ 使得

$$\Phi_e(x) = g(e,x) = \begin{cases} x & \text{若 } f(e)\downarrow \\ \uparrow & \text{否则} \end{cases}$$

于是,

$$e\in\overline{T} \Rightarrow f(e)\downarrow \quad \Rightarrow \quad \forall x\, \Phi_e(x)=x \quad \Rightarrow \quad e\in T,$$

$$e\in T \Rightarrow f(e)\uparrow \quad \Rightarrow \quad \forall x\, \Phi_e(x)\uparrow \quad \Rightarrow \quad e\in\overline{T}.$$

从而得到 $e\in\overline{T} \quad \Leftrightarrow \quad e\in T$, 矛盾. 所以, $\overline{T}$ 不是 r.e.. □

习　题

1. 设 f 是一个全函数，记 $B=\{f(n)\mid n\in N\}$. 试证明：

(1) 若 f 是可计算的，则 B 是 r. e..

(2) 若 f 是可计算的和严格增加的($\forall n\ f(n)<f(n+1)$)，则 B 是递归的.

2. 设 A,B 是 N 的非空子集，定义

$$A\odot B=\{2x\mid x\in A\}\cup\{2x+1\mid x\in B\},$$

$$A\otimes B=\{\langle x,y\rangle\mid x\in A,y\in B\}.$$

试证明：

(1) $A\odot B$ 是递归的当且仅当 A 和 B 是递归的.

(2) $A\otimes B$ 是递归的当且仅当 A 和 B 是递归的.

3. 证明下述集合是 r. e.，其中 a 是一个常数：

$$B_1=\{x\in N\mid a\in \mathrm{dom}\Phi_x\},$$

$$B_2=\{x\in N\mid a\in \mathrm{ran}\Phi_x\},$$

$$B_3=\{x\in N\mid \mathrm{dom}\Phi_x\neq\varnothing\}.$$

4. 设 A 是 r. e. 集，试证明 $\bigcup\limits_{n\in A}W_n$ 是 r. e. 集.

5. 用对角化方法证明 K 不是递归集.

6. 证明第3题中的集合的补集都不是 r. e..

第六章　Turing 机

6.1　Turing 机的基本模型

A. Turing 于 1936 年提出一种计算模型，现在称之为 Turing 机，是最重要的计算模型之一. Turing 机有很多种形式，本节介绍一种基本的形式，叫做基本 Turing 机.

它有一条类似 Post-Turing 语言的带作为存储装置和一个控制器，控制器带一个读写头(又叫做带头)，如图 6.1 所示. 带的两端是无穷的，被划分成无穷多个小方格. 每个小方格内可以存放一个符号. 控制器有有穷个状态. 在计算的每一步，控制器总处于某个状态，读写头扫描一个方格. 根据控制器当前所处的状态和读写头扫描的方格内的符号(以后简称读写头扫描的符号)，机器完成下述三个动作中的一个：

(1) 改写被扫描方格的内容，控制器转换到一个新状态；

(2) 读写头向左移动一格，控制器转换到一个新状态；

(3) 读写头向右移动一格，控制器转换到一个新状态.

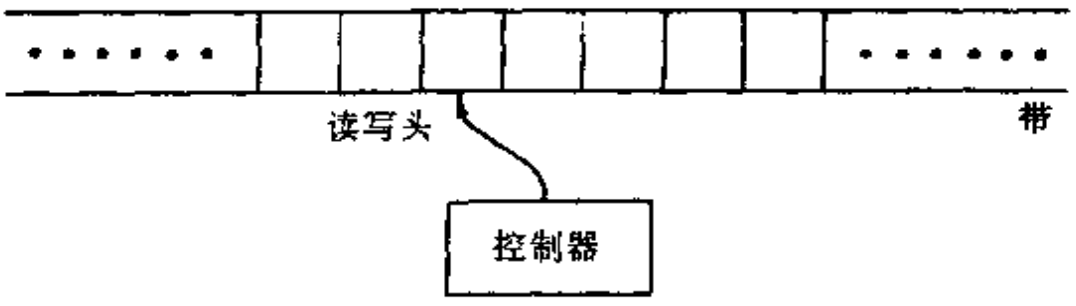


图 6.1　TM 示意图

定义 6.1　一台基本 Turing 机 $\mathscr{M}$ 由下述 6 部分组成：

(1) 状态集 Q，Q 是一个非空有穷集合；

(2) 带字母表 C，C 是一个非空有穷集合；

(3) 动作函数 δ，δ 是 $Q\times C$ 到 $(C\cup\{L, R\})\times Q$ 的部分函数；

(4) 输入字母表 A，$A\subseteq C-\{B\}$；

(5) 空白符 B，$B\in C$；

(6) 初始状态 q_1，$q_1\in Q$.

记作 $\mathscr{M}=(Q,A,C,\delta,B,q_1)$. 今后如无特别说明，Turing 机均指基本 Turing 机，缩写为 TM.

Turing 机的每一步计算由动作函数 δ 确定. 设当前的状态 q，被扫描的符号 s，

(1) 若 $\delta(q,s)=(s',q')$，则把被扫描的方格的内容改写成 s' 并且转换到状态 q'，读写头的位置保持不变；

(2) 若 $\delta(q,s)=(L,q')$，则读写头左移一格并且转换到状态 q'，带的内容保持不变；

(3) 若 $\delta(q,s)=(R,q')$，则读写头右移一格并且转换到状态 q'，带的内容保持不变；

(4) 若 $\delta(q,s)$无定义，则停止计算.

Turing 机的**格局**包括带的内容、读写头的位置和控制器的状态，可表示成

$$\begin{array}{c} a_1a_2\cdots a_j\cdots a_{k-1}a_k \\ \uparrow \\ q \end{array}$$

它表示带的内容是 $a_1a_2\cdots a_k$，两端其余部分全是 B，读写头正在扫描 a_j，控制器处于状态 q. 如果 $\delta(q,a_j)$ ↑，则称这个格局是**停机格局**.

设 σ 和 τ 是两个格局. 如果 σ 经过一步计算变成 τ，则记作

$$\sigma \underset{\mathscr{M}}{\vdash} \tau.$$

如果存在 $\sigma_1,\sigma_2,\cdots,\sigma_k$ 使得

$$\sigma=\sigma_1 \underset{\mathscr{M}}{\vdash} \sigma_2 \underset{\mathscr{M}}{\vdash} \cdots \underset{\mathscr{M}}{\vdash} \sigma_k=\tau,$$

则记作

$$\sigma \overset{+}{\underset{\mathscr{M}}{\vdash}} \tau.$$

当不需要指明 Turing 机时，$\sigma \underset{\mathscr{M}}{\vdash} \tau$ 和 $\sigma \overset{+}{\underset{\mathscr{M}}{\vdash}} \tau$ 分别简记作 $\sigma \vdash \tau$ 和 $\sigma \overset{+}{\vdash} \tau$.

如果格局的序列 $\sigma_1, \sigma_2, \cdots$（有穷的或无穷的）使得 $\sigma_1 \vdash \sigma_2 \vdash \cdots$，并且当序列有穷时最后一个格局 σ_k 是停机格局，则称这个序列是 Turing 机的一个**计算**.

任意给定输入 $x_1, x_2, \cdots, x_m \in A^*$，Turing 机总是从初始状态 q_1 开始计算. 格局 σ_1

$$\begin{array}{l} Bx_1Bx_2\cdots Bx_m \\ \uparrow \\ q_1 \end{array}$$

称作**初始格局**. 从初始格局 σ_1 开始计算有两种可能：

（1）计算是一个有穷序列 $\sigma_1, \sigma_2, \cdots, \sigma_k$，其中 σ_k 是停机格局，此时称 Turing 机停机在格局 σ_k.

（2）计算是一个无穷序列 $\sigma_1, \sigma_2, \cdots$，此时称 Turing 机永不停机.

定义6.2 设 Turing 机 $\mathscr{M}=(Q, A, C, \delta, B, q_1)$，对每一个正整数 m，$\mathscr{M}$ 计算 A^* 上的 m 元部分函数 $\psi_{\mathscr{M}}^{(m)}$ 定义如下：对于所有的 $x_1, x_2, \cdots, x_m \in A^*$，以 $x_1, x_2, \cdots, x_m$ 作为输入，从初始格局 σ_1 开始计算，如果 $\mathscr{M}$ 最终停机在格局 σ_k，删去 σ_k 的带内容中所有不属于 A 的符号，得到字符串 $y \in A^*$，则 $\psi_{\mathscr{M}}^{(m)}(x_1, x_2, \cdots, x_m)=y$；如果 $\mathscr{M}$ 永不停机，则 $\psi_{\mathscr{M}}^{(m)}(x_1, x_2, \cdots, x_m)$ 无定义.

定义6.3 设 Turing 机 $\mathscr{M}=(Q, A, C, \delta, B, q_1)$，$f$ 是 A^* 上的 m 元部分函数. 如果对于所有的 $x_1, x_2, \cdots, x_m \in A^*$，有

$$\psi_{\mathscr{M}}^{(m)}(x_1, x_2, \cdots, x_m) = f(x_1, x_2, \cdots, x_m),$$

则称 $\mathscr{M}$ **计算** f.

如果 $\mathscr{M}$ 计算 f，并且

（1）$C=A \cup \{B\}$，即除 A 中的符号和空白符 B 外，$\mathscr{M}$ 不使用

其他的工作符号；

(2) 当 $\mathscr{M}$ 停机时，停机格局为

$$\begin{array}{l} Bf(x_1,x_2,\cdots,x_m) \\ \uparrow \\ q \end{array}$$

则称 $\mathscr{M}$ **严格地计算** f.

和 Post-Turing 程序一样，后面将证明如果存在 Turing 机计算 f，则也存在 Turing 机严格地计算 f.

[**例6.1**] 设 Turing 机 $\mathscr{M}=(Q,A,C,\delta,B,q_1)$，其中 $Q=\{q_i|i=1,2,3,4,5\}$，$A=\{0,1\}$，$C=\{0,1,B\}$，δ 由表6-1给出. 表中空白的地方表示 δ 无定义.

表6-1 一台 TM

δ	0	1	B
q_1			(R,q_2)
q_2	(R,q_2)	(R,q_2)	(L,q_3)
q_3	(B,q_5)	(L,q_4)	
q_4	(L,q_4)	(L,q_4)	(B,q_1)
q_5			

当输入1010时，计算如下：

$$\begin{array}{l} B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,0\,B \\ \uparrow \quad\quad\quad\quad \uparrow \quad\quad\quad\quad \uparrow \quad\quad\quad\quad \uparrow \\ q_1 \quad\quad\quad\quad q_2 \quad\quad\quad\quad q_3 \quad\quad\quad\quad q_2 \\ \vdash B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,0\,B \vdash B\,1\,0\,1\,B\,B \\ \quad\quad\quad\quad \uparrow \quad\quad\quad\quad \uparrow \quad\quad\quad\quad \uparrow \quad\quad\quad\quad \uparrow \\ \quad\quad\quad\quad q_2 \quad\quad\quad\quad q_2 \quad\quad\quad\quad q_3 \quad\quad\quad\quad q_5 \end{array}$$

最后一个格局是停机格局，计算结束.

当输入1001时，计算如下：

$$
\begin{array}{l}
B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \\
\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \\
q_1 \qquad\qquad q_2 \qquad\qquad q_2 \qquad\qquad q_2 \\
\vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \\
\quad\ \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \\
\quad\ q_2 \qquad\qquad q_2 \qquad\qquad q_3 \qquad\qquad q_4 \\
\vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \vdash B\,1\,0\,0\,1\,B \\
\quad\ \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow \\
\quad\ q_4 \qquad\qquad q_4 \qquad\qquad q_4 \qquad\qquad q_1 \\
\vdash \cdots
\end{array}
$$

经过11步计算又回到初始格局，如此重复下去，永不停机.

一般地，设输入 x，读写头右移一格并且转移到状态 q_2. 若 $x=\varepsilon$，此时扫描 B，读写头左移一格并且转移到 q_3，停机. 若 $x\neq\varepsilon$，在状态 q_2 下继续右移，直到读完 x，扫描 x 右边的第一个 B. 读写头左移一格扫描 x 右端的第一个符号. 若这个符号是0，则把这个0删去(即改写成 B)，然后停机；若这个符号是1，则读写头再左移一格并且转移到 q_4. 然后在状态 q_4 下继续左移，直到扫描 x 左边的第一个 B. 把状态转移到 q_1，恢复到初始格局. 如此重复上去，永不停机.

对所有的 $x\in\{0,1\}^*$，

$$\varphi_{\mathscr{M}}^{(1)}(x)=\begin{cases}x^- & \text{若 } x \text{ 以 } 0 \text{ 结束或 } x=\varepsilon\\ \uparrow & \text{否则}\end{cases}$$

$\mathscr{M}$ 以二进制数的形式计算 N 上的部分函数

$$f(x)=\begin{cases}x/2 & \text{若 } x \text{ 是偶数}\\ \uparrow & \text{若 } x \text{ 是奇数.}\end{cases}$$

把 $Q\times C$ 到 $(C\cup\{L,R\})\times Q$ 的二元关系的元素 $((q,s),(s',q'))$，$((q,s),(L,q'))$，$((q,s),(R,q'))$ 分别记作 $qss'q'$，$qsLq'$，$qsRq'$，并把它们称作4元组. 于是，δ 也可以表示成上述3种类型的4元组的有穷集合，在这个集合中没有两个4元组以相同的 qs 开始. 例如，例6.1中的状态转移函数

$$\delta=\{q_1BRq_2,q_20Rq_2,q_21Rq_2,q_2BLq_3,$$
$$q_30Bq_5,q_31Lq_4,q_10Lq_4,q_41Lq_4,q_4BBq_1\}.$$

δ 还可以用有向图描述，这个有向图称作**状态转移图**. 例6.1中的 Turing 机的状态转移图如图6.2所示.

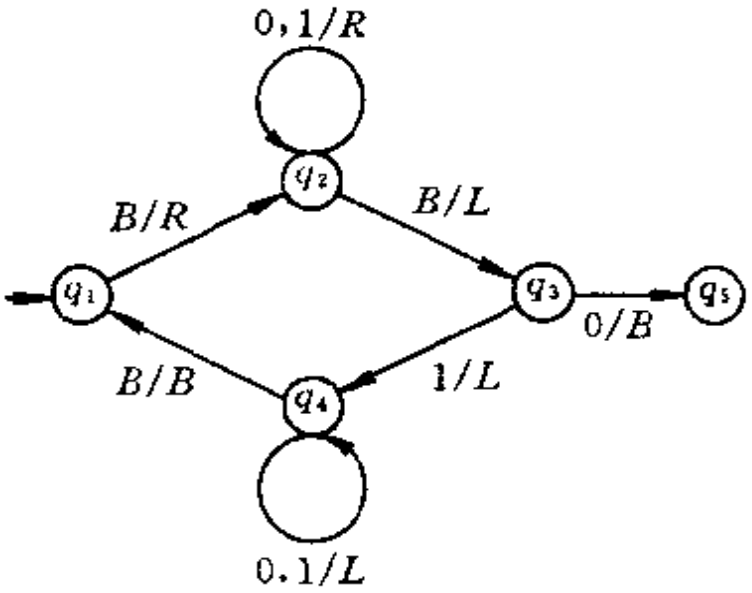


图 6.2

用节点表示状态，弧(q_1,q_2)旁的 B/R 表示 $\delta(q,B)=(R,q_2)$，弧(q_3,q_5)旁的$0/B$ 表示 $\delta(q_3,0)=(B,q_5)$. 顶点 q_2的环旁的$0,1/R$是$0/R$ 与$1/R$ 的缩写. q_1旁的小箭头→表示它是初始状态.

[**例6.2**] 设计一台 Turing 机计算函数

$$f(x)=xx,\forall\ x\in\{0,1\}^*.$$

解：这是一台复制机，在 x 的右边复制出一个 x. 为了区分原来的 x 和复制出的 x，用 B 把它们分隔开来. 在停机格局中，带的内容为 xBx. Turing 机做如下工作：使用两个工作符号 a 和 b，分别表示已被复制或即将复制的0和1，待复制完成后再将它们分别恢复成0或1. 若左端第一个未被复制的符号是0，则把它改写成 a，然后向右查寻，越过一个 B 后找到右边的第一个 B，把这个 B 改写成0. 类似地，若左端第一个未被复制的符号是1，则把它改写成

b，然后向右查寻，越过一个 B 后找到右边的第一个 B，把这个 B 改写成1. 不论是哪一种情况，在复写一个符号后向左查寻，直到遇到 a 或 b 为止. 右移一格，若扫描到的是0或1则重复上述过程. 若扫描到的是 B，则表明复制工作已经完成. 读写头向左移动，把 a 和 b 分别恢复成0和1，直到完成这项工作为止. Turing 机的状态转移图如图6.3所示.

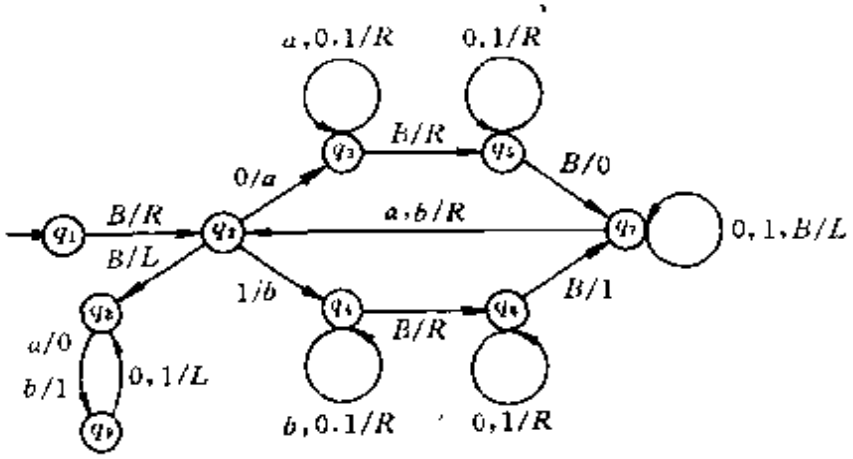


图6.3 复制机

[**例6.3**] 设计一台 Turing 机以1为底计算 $x \cdot y$，即任给 $x, y \in N$，输入 1^x 和 1^y，输出 1^{xy}.

解：记 $u=1^x, v=1^y$. 基本思想是：对 u 中的每一个1，复制一个 v. Turing 机的状态转移图如图6.4所示. 小循环 $q_4 \to q_5 \to q_6 \to q_7 \to q_4$ 用来复制 v. 复制 v 完成时，原来的 v 被改写成 a^y，a 是工作符号. 用 $q_8 \to q_9 \to q_8$ 把 a^y 恢复成 1^y. 大循环 $q_2 \to q_3 \to \cdots \to q_{10} \to q_2$ 一次把 u 中的一个1改写成 a 并复制出一个 v. 当回到 q_2 发现 u 中的 x 个1已全部被改写成 a 时，整个复制工作已全部完成. 此时，原来的 u 已被改写成 a^x，但 v 是原来的样子 1^y. 为了能得到正确的函数值，只需把 v 再次改写成 a^y. 用 $q_{11} \to q_{12} \to q_{11}$ 完成这项改写工作. 任给 $x, y \in N$，初始格局为

$$B1^xB1^y$$
$$\uparrow$$
$$q_1$$

停机在格局

$$Ba^xBa^yB1^{xy}$$
$$\uparrow$$
$$q_{11}$$

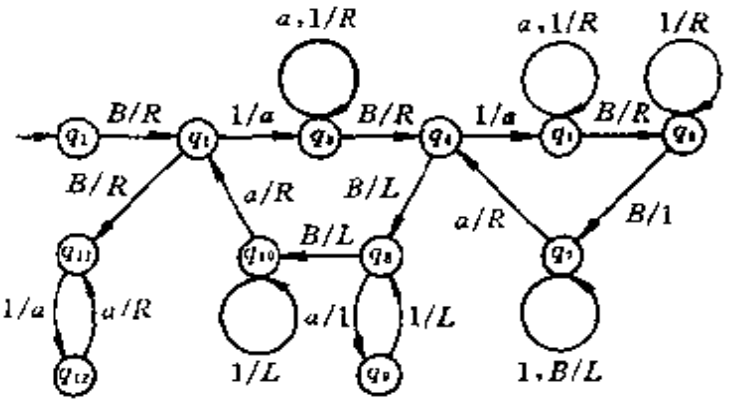


图6.4 乘法机

6.2 Turing 机与可计算性

Turing 机和前面介绍的几种计算模型(语言 $\mathscr{L}$,$\mathscr{L}_n$ 以及 $\mathscr{T}$)是等价的,即一个部分函数可以用 Turing 机计算当且仅当它在这些模型中是部分可计算的. 已经证明这几种模型是等价的,因此只需证明 Turing 机与其中的任何一种是等价的. 我们选择 Post-Turing语言,证明 Turing 机和 Post-Turing 语言等价.

引理6.1 任何 Post-Turing 程序都能用 Turing 机模拟,且使用相同的带字母表.

证:设 Post-Turing 程序 $\mathscr{P}$ 由指令 $I_1,I_2,\cdots,I_t$ 组成,使用符号 $s_0,s_1,\cdots,s_n$,其中 $s_0=B$ 是空白符. 构造模拟 $\mathscr{P}$ 的 Turing 机 $\mathscr{M}$ 如下:

$\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$,这里 $C=\{s_0,s_1,\cdots,s_n\}$,$Q=\{q_1,q_2,\cdots,q_{t+1}\}$. 前 t 个状态分别对应于 t 条指令、$\mathscr{M}$ 处于 q_i 当且仅当 $\mathscr{P}$ 执行指令 I_i,$1\leqslant i\leqslant t$. q_{t+1}是停机状态. 对所有的 $s_i(0\leqslant i\leqslant n)$,$\delta(q_{t+1},s_i)\uparrow$. $\mathscr{M}$ 一进入 q_{t+1},立即停机.

δ 的定义如下：对每一个 $i(1\leqslant i\leqslant t)$，

若 I_i 的语句是 PRINTs_k，则

$$\delta(q_i,s_j)=(s_k,q_{i+1}),\quad j=0,1,\cdots,n.$$

若 I_i 的语句是 RIGHT，则

$$\delta(q_i,s_j)=(R,q_{i+1}),\quad j=0,1,\cdots,n.$$

若 I_i 的语句是 LEFT，则

$$\delta(q_i,s_j)=(L,q_{i+1}),\quad j=0,1,\cdots,n.$$

若 I_i 的语句是 IF s_k GOTO L，则

$$\delta(q_i,s_k)=(s_k,q_m),$$

$$\delta(q_i,s_j)=(s_j,q_{i+1}),\quad j=0,1,\cdots,n \text{ 且 } j\neq k,$$

这里，当 $\mathscr{P}$ 中有以 L 为标号的指令时，m 是这些指令的最小下标；否则 $m=t+1$. □

引理6.2 任何 Turing 机都能用 Post-Turing 程序模拟，且使用相同的带字母表.

证：设 Turing 机 $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，其中 $Q=\{q_1,q_2,\cdots,q_r\}$，$C=\{s_0,s_1,\cdots,s_n\}$，s_0 是空白符. 如下构造模拟 $\mathscr{M}$ 的 Post-Turing程序 $\mathscr{P}$.

程序 $\mathscr{P}$ 被分成 $r(n+2)$段：r 个以标号 A_i 开始的程序段 $\mathscr{A}_i$ 和 $r(n+1)$个以标号 B_{ij}开始的程序段 $\mathscr{B}_{ij}$，$1\leqslant i\leqslant r$，$0\leqslant j\leqslant n$. 当 $\mathscr{M}$ 处于状态 q_i 时，$\mathscr{P}$ 的计算进入 $\mathscr{A}_i$. 程序段 $\mathscr{A}_i$ 为

$$\begin{aligned}[A_i]\quad &\text{IF } s_0 \text{ GOTO } B_{i0}\\ &\text{IF } s_j \text{ GOTO } B_{ij}\quad (1\leqslant j\leqslant n)\end{aligned}$$

这段程序的作用是根据当前扫描的符号 s_j 转到 $\mathscr{B}_{ij}$. 程序段 $\mathscr{B}_{ij}$与 $\delta(q_i,s_j)$有关.

若 $\delta(q_i,s_j)=(s_l,q_k)$，则 $\mathscr{B}_{ij}$为：

$$\begin{aligned}[B_{ij}]\quad &\text{PRINT } s_l\\ &\text{GOTO } A_k\end{aligned}$$

若 $\delta(q_i,s_j)=(R,q_k)$，则 $\mathscr{B}_{ij}$为：

$$[B_{ij}]\quad \text{RIGHT}$$

GOTO A_k

若 $\delta(q_i,s_j)=(L,q_k)$，则 $\mathscr{B}_{ij}$ 为：

$[B_{ij}]$ LEFT

GOTO A_k

若 $\delta(q_i,s_j)\uparrow$，则 $\mathscr{B}_{ij}$ 为：

$[B_{ij}]$ GOTO E

程序 $\mathscr{P}$ 中没有以 E 为标号的指令.

将所有这些程序段拼接在一起就是所需要的程序 $\mathscr{P}$. 拼接时必须把程序段 $\mathscr{A}_1$ 放在程序的开头（$\mathscr{A}_1$ 对应初始状态 q_1），其余各段程序的顺序无关紧要. 例如，

$\mathscr{A}_1$

$\mathscr{A}_2$

⋮

$\mathscr{A}_r$

$\mathscr{B}_{10}$

$\mathscr{B}_{11}$

⋮

$\mathscr{B}_{rn}$ □

由这两个引理和定理4.5，得到：

定理6.3 设字母表 A，A 上的字函数 f 是部分可计算的当且仅当 f 可以用 Turing 机计算.

推论6.4 设 f 是字母表 A 上的 m 元部分可计算的字函数，那么可以用 Turing 机严格地计算 f.

证：由定理4.5，存在 Post-Turing 程序 $\mathscr{P}$ 严格地计算 f. 按照引理6.1的证明构造模拟 $\mathscr{P}$ 的 Turing 机 $\mathscr{M}$. 这个 $\mathscr{M}$ 严格地计算 f. □

由定理6.3和推论6.4可以得到：

推论6.5 如果字函数 f 可以用 Turing 机计算，则 f 可以用 Turing 机严格地计算.

于是，在定理4.5的等价命题中又可添加两个命题：

(5) f 可以用 Turing 机计算.

(6) f 可以用 Turing 机严格地计算.

推论6.6 每一个 N 上的部分可计算函数都可以用 Turing 机以一进制方式(以$\{1,B\}$为带字母表)严格地计算.

证：由推论4.7和引理6.1可得到证明. □

推论6.6表明，对于每一个 N 上的部分可计算函数 $f(x_1,\cdots,x_m)$，存在 Turing 机

$$\mathscr{M}=(Q,\{1\},\{1,B\},\delta,B,q_1),$$

对所有的 $x_1,x_2,\cdots,x_m\in N$，从初始格局

$$\begin{array}{l} B1^{x_1}B1^{x_2}\cdots B1^{x_m} \\ \uparrow \\ q_1 \end{array}$$

开始计算，如果 $f(x_1,x_2,\cdots,x_m)\downarrow$，则 $\mathscr{M}$ 最终停机在格局

$$\begin{array}{l} B1^{f(x_1,x_2,\cdots,x_m)} \\ \uparrow \\ q \end{array}$$

6.3 Turing 机接受的语言

定义6.4 设 Turing 机 $\mathscr{M}=(Q,A,C,\delta,B,q_1)$，$x\in A^*$. 如果从初始格局

$$\begin{array}{l} Bx \\ \uparrow \\ q_1 \end{array}$$

开始计算，$\mathscr{M}$ 最终停机，则称 $\mathscr{M}$ **接受** x. $\mathscr{M}$ 接受的所有字符串组成的集合称作 $\mathscr{M}$ **接受的语言**，或 $\mathscr{M}$ **识别的语言**，记作$L(\mathscr{M})$. 即

$$L(\mathscr{M})=\{x\in A^*\mid \mathscr{M}\text{ 接受 }x\}.$$

例如，例6.1中的 Turing 机 $\mathscr{M}$ 接受的语言是

$$L(\mathscr{M})=\{x\mid x=\varepsilon\text{ 或 }x=u0,u\in\{0,1\}^*\}.$$

[例6.4] 设计一台 Turing 机接受语言

$$L = \{ww^R \mid w \in \{0,1\}^*\},$$

其中 w^R 表示 w 的反转，$(a_1a_2\cdots a_k)^R = a_k\cdots a_2a_1$.

解：读写头从左到右，检查左端第一个符号和右端第一个符号. 若相同，则把它们删去，回到左端，重复上述动作. 读写头如此左右来回运动，如果在删去右端的符号后已把整个输入 x 删去，则表明 $x \in L$，停机. 如果在某一次检查时发现左右两端的符号不相同，或者在删去右端的符号后只剩下一个符号，则表明 $x \notin L$，计算进入死循环. 状态转移图如图6.5所示. 若在状态 q_2 下扫描到 B，则表明经过若干次来回（每次删去左右端符号各一个，也可以是0次，即 $x = \varepsilon$.）已把输入 x 全部删去，停机. 若在状态 q_9 下扫描到1或在状态 q_{10} 下扫描到0，则表明这个符号与左边对称位置上的符号不相同，计算进入停止在这个状态下的死循环. 若在状态 q_5 或 q_6 下扫描到 B，则表明 x 的长度是奇数，计算也进入死循环.

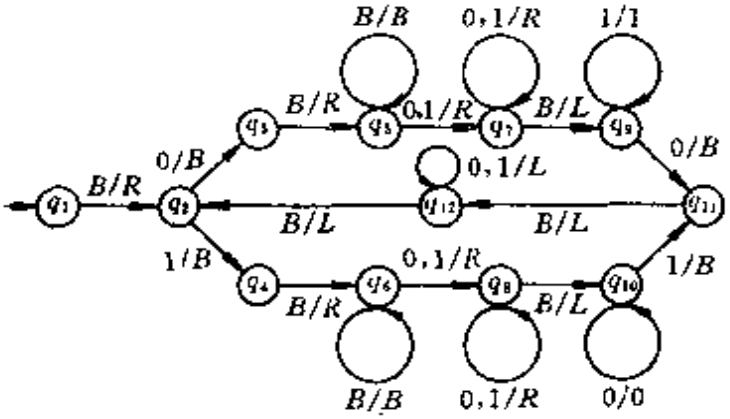


图6.5 接受$\{ww^t \mid w \in \{0,1\}^*\}$的 TM

定理6.7 一个语言能被 Turing 机接受当且仅当它是递归可枚举的.

证：设字母表 A，$L \subseteq A^*$. 如果存在 Turing 机 $\mathscr{M}$ 接受 L，设 $g(x)$ 是 $\mathscr{M}$ 计算的 A^* 上的一元部分函数，则

$$L = \{x \in A^* \mid g(x)\downarrow\}.$$

根据定理6.3，g 是部分可计算的，故 L 是 r.e..

反之，如果 L 是 r.e.，则存在 A^* 上的部分可计算函数 $g(x)$ 使得

$$L = \{x \in A^* \mid g(x)\downarrow\}.$$

根据定理6.3，存在 Turing 机 $\mathscr{M}$ 计算 g. 显然，$\mathscr{M}$ 接受 L. □

推论6.8 设字母表 A，$L \subseteq A^*$. 如果 L 能被 Turing 机接受，则存在使用带字母表 $C=A\cup\{B\}$ 的 Turing 机接受 L.

证：由定理6.7，L 是 r.e.，故存在 A^* 上的部分可计算函数 $g(x)$ 使得

$$L = \{x \in A^* \mid g(x)\downarrow\}.$$

根据推论6.5，存在 Turing 机 $\mathscr{M}$ 严格地计算 g，则 $\mathscr{M}$ 接受 L，且带字母表为 $C=A\cup\{B\}$. □

推论6.9 设 $A \subseteq N$，则 A 是递归可枚举的当且仅当存在以 $\{1,B\}$ 为带字母表的 Turing 机接受语言 $L=\{1^x \mid x\in A\}$.

证：因为 A 是 r.e.，而 $L=\{1^x \mid x\in A\}$ 是 A 的一进制表示，所以 L 也是 r.e.. 由定理6.7和推论6.8，存在所需要的 Turing 机. □

6.4 Turing 机的各种形式

本节介绍几种其他形式的 Turing 机. 尽管在形式上有的受到限制、有的得到加强，但它们的计算能力都与基本 Turing 机相同. 也就是说，它们中的任何一种计算的函数类都是部分可计算函数类，接受的语言类都是 r.e. 语言类.

6.4.1 五元 Turing 机

基本 Turing 机的动作函数可以用4元组的有穷集合来表示，故又称作**四元 Turing 机**. 四元 Turing 机的4元组有3种类型：

(1) $qss'q'$，

(2) $qsLq'$，

(3) $qsRq'$.

四元 Turing 机的一步只能改写一个符号、或者左移一格、或者右移一格. **五元 Turing 机**的一步要做四元 Turing 机两步做的事情，改写一个符号并且左移一格、或者改写一个符号并且右移一格. 五元 Turing 机的动作函数 δ 是从 $Q\times C$ 到 $C\times\{L,R\}\times Q$ 的部分函数，它可以表示成5元组的有穷集合. 5元组有两种类型：

(1) $qss'Lq'$，它表示 $\delta(q,s)=(s',L,q')$，若当前状态是 q，被扫描的符号是 s，则把这个 s 改写成 s'，读写头左移一格，并且转移到状态 q'.

(2) $qss'Rq'$，和上面的类似，不同的是读写头右移一格. 和四元 Turing 机一样，不允许有两个5元组的前两个分量相同.

五元 Turing 机和四元 Turing 机是等价的，它们可以相互模拟.

用五元 Turing 机模拟四元 Turing 机. 设四元 Turing 机 $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，其中 $Q=\{q_1,q_2,\cdots,q_t\}$，$C=\{s_0,s_1,\cdots,s_n\}$. 模拟 $\mathscr{M}$ 的五元 Turing 机 $\mathscr{M}'$ 使用相同的输入字母表 A，带字母表 C，空白符 s_0 以及初始状态 q_1. 状态集 $Q'=\{q_1,\cdots,q_t,q_{t+1},\cdots,q_{2t}\}$，动作函数 δ' 与 δ 的关系列于表6-2，以5元组和4元组的形式给出.

表6-2　用五元 TM 模拟四元 TM

4元组	5元组
(1) $q_is_jLq_l$	$q_is_js_jLq_l$
(2) $q_is_jRq_l$	$q_is_js_jRq_l$
(3) $q_is_js_kq_l$	$q_is_js_kRq_{l+t}$
(4)	$q_{l+t}s_rs_rLq_l$，$1\leqslant l\leqslant t$，$0\leqslant r\leqslant n$

用四元 Turing 机模拟五元 Turing 机. 设五元 Turing 机 $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，其中 $Q=\{q_1,q_2,\cdots,q_t\}$，$C=\{s_0,s_1,\cdots,s_n\}$. 模拟 $\mathscr{M}$ 的四元 Turing 机 $\mathscr{M}'$ 使用相同的输入字母表 A，带字母表 C，空白符 s_0 和初始状态 q_1. 状态集 $Q'=\{q_1,\cdots,q_t,q_{t+1},\cdots,q_{3t}\}$，动作函数 δ' 和 δ 的关系列于表6-3.

表6-3　用四元 TM 模拟五元 TM

5元组	4元组
(1) $q_is_js_kLq_t$	$q_is_js_kq_{t+t}$
(2) $q_is_js_kRq_t$	$q_is_js_kq_{t+2t}$
(3)	$q_{t+t}s_rLq_t$
	$q_{t+2t}s_rRq_t$, $1\leqslant l\leqslant t$, $0\leqslant r\leqslant n$

6.4.2　单向无穷带 Turing 机

基本 Turing 机的带是双向无穷的，两端可以任意地伸长. **单向无穷带 Turing 机**的带仅在一个方向上是无穷的，有一个最左方格，称作带的左端，如图6.6所示. 它有一个特殊的带符号 #，# 被固定放置在带的左端. 它不能被改写，也不能在任何其他方格内打印 #. 对于任何状态 q，$\delta(q,\#)$ 或者等于 (R,q')、或者无定义. 因此，当读写头扫描左端时，或者右移，或者停机，永远不会左移出带外[①].

图6.6　单向无穷带

单向无穷带 Turing 机是基本 Turing 机的特殊类型：有一个特殊的带符号 #，并对 δ 作上述限制. 放置 # 的方格看作带的左端，它左边的方格永远不会被使用.

用单向无穷带 Turing 机模拟基本 Turing 机的关键是要解决如何在带上存放符号. 在双向无穷带上指定一个方格，给它编号1，向右依次编号2，3，…，向左依次编号 -1，-2，…. 沿方格1的左边剪开，将带的左半部折到下面，和右半部并在一起成为一条单向无

① A. Turing 在它的原始模型中使用的是单向无穷带. 在一些书中用这种 Turing 机作为基本 Turing 机.

穷带，如图6.7所示. 这条单向无穷带被分成两道，上道对应原带的右半部，下道对应原带的左半部. 每个方格被分成上下两个小方格，各存放一个符号，例如 a 和 b，把有序对(a,b)看作一个符号.

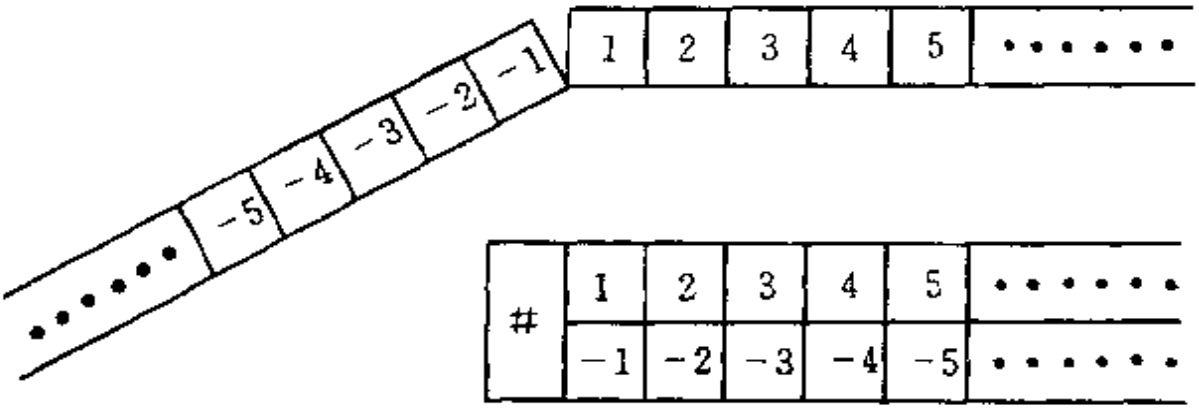


图6.7　用单向无穷带模拟双向无穷带

具体做法如下：设基本 Turing 机 $\mathscr{M}$，它的状态集 $Q=\{q_1,q_2,\cdots,q_t\}$，q_1是初始状态，带字母表 $C=\{s_0,s_1,\cdots,s_n\}$，s_0是空白符. 模拟 $\mathscr{M}$ 的单向无穷带 Turing 机 $\mathscr{M}'$的状态集 $Q'=\{q_i',q_i''\mid 1\leqslant i\leqslant t\}$，$q_i'$ 模拟 q_i 在读写头扫描右半带时的动作，q_i' 只注视上道内存放的符号；类似地，q_i'' 模拟 q_i 在读写头扫描左半带时的动作，q_i'' 只注视下道内存放的符号. $\mathscr{M}'$的带字母表 $C'=\{\#\}\cup\{s_j^r\mid 0\leqslant j,r\leqslant n\}$，$s_j^r$ 表示(s_r,s_j)，即上道为 s_r，下道为 s_j. s_0^0是 $\mathscr{M}'$的空白符. δ'和 δ 的对应关系列于表6-4.

前三组的含义是清楚的. 对于 $\mathscr{M}$ 的每一个4元组，$\mathscr{M}'$有对应的这样一组4元组. 除此之外，$\mathscr{M}'$还应添加第(4)组. 只有在下述情况才会出现在状态 q_i' 下扫描 #：在状态 q_i' 下扫描紧挨 # 的方格. 方格内存放着 s_r^j. 根据 $q_i's_r^jLq_l'$，读写头左移扫描 #，转移到状态 q_l'. 被模拟的 $\mathscr{M}$ 的动作是读写头从方格1左移到方格-1，状态从 q_i 转移到 q_l. 因此，需要添加一个4元组 $q_l'\#Rq_l''$，把读写头移回到原来的位置并将状态转移到注视下道的 q_l''. 类似地，应添加4元组 $q_l''\#Rq_l'$.

表6-4　用单向无穷带 TM 模拟双向无穷带 TM

双向无穷带	单向无穷带
(1) $q_i s_j s_k q_l$	$q'_i s'_r s''_k q'_l$ $q''_i s'_r s'_k q''_l$,　$0 \leqslant r \leqslant n$
(2) $q_i s_j L q_l$	$q'_i s'_r L q'_l$ $q''_i s'_r R q''_l$,　$0 \leqslant r \leqslant n$
(3) $q_i s_j R q_l$	$q'_i s'_r R q'_l$ $q''_i s'_r L q''_l$,　$0 \leqslant r \leqslant n$
(4)	$q'_l \# R q''_l$ $q''_l \# R q'_l$,　$1 \leqslant l \leqslant t$

设 f 是一元部分可计算的字函数，基本 Turing 机 $\mathscr{M}$ 严格地计算 f，$\mathscr{M}'$ 是按上述方式构造的模拟 $\mathscr{M}$ 的单向无穷带 Turing 机. 不难看到，对于每一个 $x=s_{i_1}s_{i_2}\cdots s_{i_k}$，若 $f(x)=s_{j_1}s_{j_2}\cdots s_{j_l}$，则 $\mathscr{M}$ 从初始格局 σ_1

$$\begin{array}{l} s_0 s_{i_1} s_{i_2} \cdots s_{i_k} \\ \uparrow \\ q_1 \end{array}$$

开始，停机在格局

$$\begin{array}{l} s_0 s_{j_1} s_{j_2} \cdots s_{j_l} \\ \uparrow \\ q \end{array}$$

相对应地，$\mathscr{M}'$ 从初始格局 σ'_1

$$\begin{array}{l} \# s_0^0 s_0^1 s_0^2 \cdots s_0^i \# \\ \uparrow \\ q'_1 \end{array}$$

开始，最终停机，并且停机时带的上下道内非 s_0 符号恰好连在一起构成 x. 它可以都在上道从左到右连成 x；也可以都在下道从右到左连成 x；还可以一部分在下道、一部分在上道，在下道从右到左，然后转入上道从左到右连成 x. 给 $\mathscr{M}'$ 添加一些新的4元组(这里不再详细给出这些4元组，但确实是可以做到的)将上面的格局变成下述形状的停机格局

$$\# s_0^0 s_0^1 s_0^2 \cdots s_0^t$$
$$\uparrow$$
$$q'$$

若 $f(x)\uparrow$，从初始格局 σ_1 开始，$\mathscr{M}$ 永不停机. 从初始格局 σ_1' 开始，$\mathscr{M}'$ 也永不停机. 我们将 s_0' 等同于 s_1，则 $\mathscr{M}'$ 计算 f[①].

类似地，任何 $m(m>1)$ 元部分可计算的字函数都可以用单向无穷带 Turing 机计算. 任何递归可枚举语言都可以被单向无穷带 Turing 机接受.

6.4.3 多带 Turing 机

基本 Turing 机只有一条带. **k 带 Turing 机**有 k 条带，这里 $k\geqslant 2$ 是一个固定常数. 每一条带和基本 Turing 机的带一样，被分成无穷多个小方格，每一个方格内可以存放一个符号，带的两边是无穷的(当然也可以是单向无穷的). 有 k 个读写头，每个读写头扫描一条带，可以改写被扫描方格内的符号、左移一格、或者右移一格. 读写头的动作由当前的状态和 k 个读写头从 k 条带上读到的 k 个符号决定，其动作函数 δ 是 $Q\times C^k$ 到 $(C\cup\{L,R\})^k\times Q$ 的部分函数. 图6.8是一台3带 Turing 机的示意图. 计算开始时，输入被存放在第一条带上，其余 $k-1$ 条带全部是空白. 当停止计算时，输出的内容也是存放在第一条带上.

基本 Turing 机是单带 Turing 机. 用 k 带 Turing 机模拟单带 Turing 机是十分简单的. 第一条带的读写头和给定的基本 Turing 机的读写头做一样的动作，其余的保持不动.

用单带 Turing 机模拟 k 带 Turing 机的关键技术仍然是6.4.2小节中采用的多道技术. 设 $\mathscr{M}$ 是一台 k 带 Turing 机，要用

---

① 实际上，可以将 $\{s_0,s_1,\cdots,s_n\}$ 加入 C'，在 $\mathscr{M}'$ 中添加若干4元组使得格局

$$Bs_0s_{i_1}s_{i_2}\cdots s_{i_k} \quad \text{和} \quad \# s_0^0 s_0^1 s_0^2\cdots s_0^t$$
$$\uparrow \qquad\qquad\qquad \uparrow$$
$$q \qquad\qquad\qquad q'$$

可以相互转变.

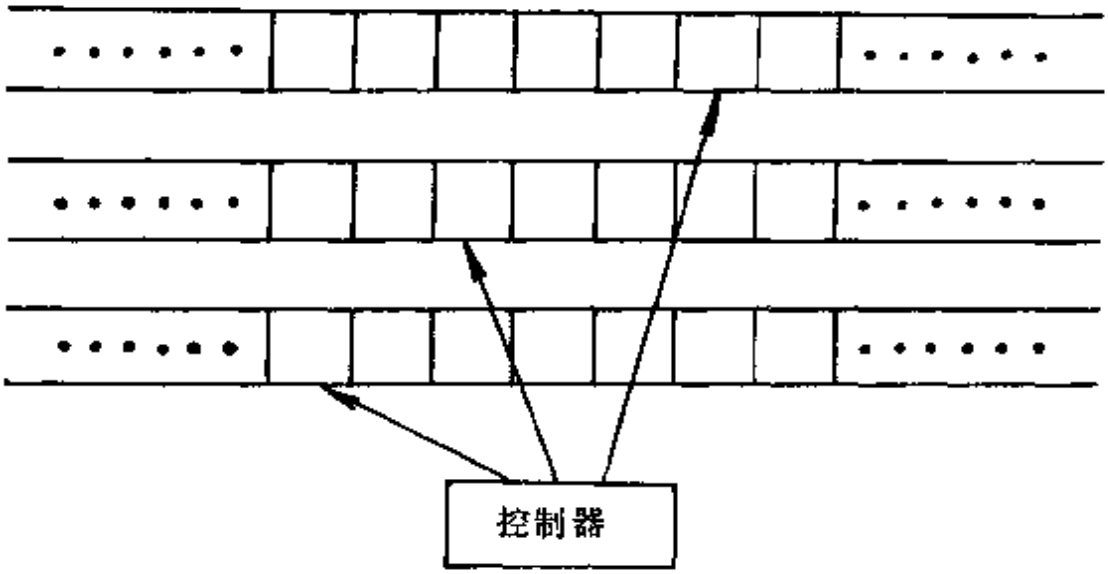


图6.8　多带 TM 示意图

一台基本 Turing 机 $\mathscr{M}'$ 来模拟 $\mathscr{M}$. 把 $\mathscr{M}'$ 的带分成$2k$ 道，用2道模拟 $\mathscr{M}$ 的一条带，其中一道存放着和这条带一样的内容，另一道除一个方格内放置一个特殊的符号 ↑ 外所有方格都是空白. 符号 ↑ 指明了这条带的读写头的位置. 图6.9给出用一条带模拟3条带的示意图. 为了模拟 $\mathscr{M}$ 的一步，$\mathscr{M}'$ 的读写头要从左到右、再从右到左地做一次来回运动. 在模拟的开始，$\mathscr{M}'$ 的读写头位于最左的 ↑ 的左边. 读写头向右移动，读入每一个 ↑ 所指示的符号，直到右边不再有 ↑ 为止. 这时 $\mathscr{M}'$ 已经知道了 $\mathscr{M}$ 的 k 个读写头扫描的符号，从而也知道了 $\mathscr{M}$ 在这一步要做的动作. $\mathscr{M}'$ 的读写头向左往回运动并且模拟这些动作，直到它的左边不再有 ↑ 为止. $\mathscr{M}'$ 要记录读写头右边 ↑ 的个数 t. 读写头向右运动时，每越过一个 ↑，t 减1. 当 $t=0$时，停止向右运动. 读写头向左运动时，每越过一个 ↑，t 加1. 当 $t=k$ 时，停止向左运动. 最后，$\mathscr{M}'$ 还要改变它记录的 $\mathscr{M}$ 的状态. 这就完成了对 $\mathscr{M}$ 的一步计算的模拟. 记录 $\mathscr{M}$ 的状态，读到的符号以及 t 都可以用状态实现. 给出 $\mathscr{M}'$ 详细的形式描述是一件工作量很大的事情，这里只好免去.

此外，还有多维 Turing 机和多头 Turing 机. 二维 Turing 机的存储装置不是一条带，而是一个平面，平面被划分成无数个小方

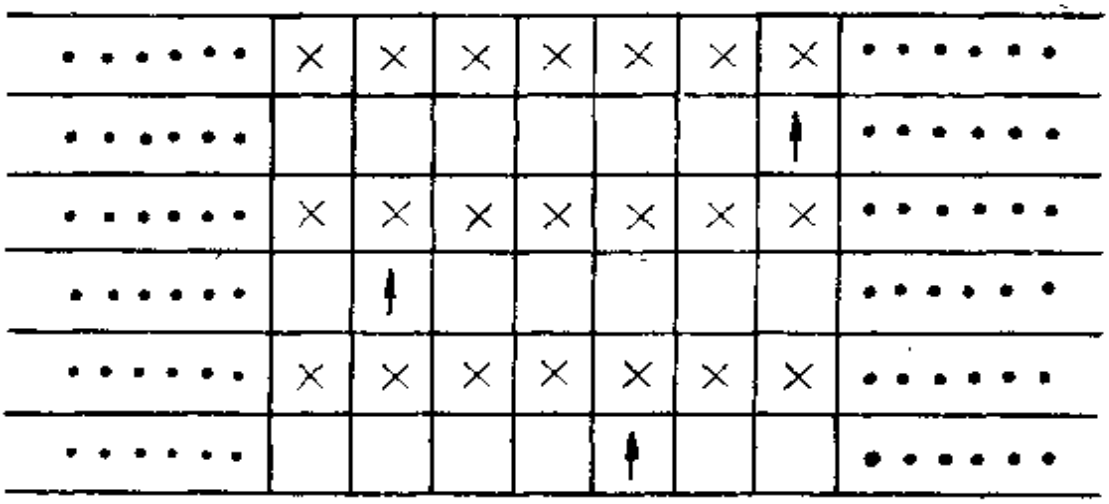


图6.9 用一条带模拟多条带

格，左右上下四个方向上都是无限的. 读写头可以左右上下移动和改写所扫描的方格内的符号. 更一般地，k 维 Turing 机($k \geqslant 2$是一个固定的常数)的存储装置是 k 维空间，每一个存储单元是一个 k 维小立方体. 在$2k$ 个方向上都是无限的，读写头可以在$2k$ 个方向上自由地移动. k 头 Turing 机有一条带和 k 个读写头，每个读写头都可以各自左移、右移、或者改写符号. 它们也和上述几种模型一样，和基本 Turing 机具有相同的计算能力.

6.4.4 离线 Turing 机

离线 Turing 机是具有一条只读输入带的多带 Turing 机，如图6.10所示. 它有一条输入带和 k 条工作带，输入带的带头只读不写，工作带的带头是和普通多带 Turing 机的带头一样的读写头. 给定输入 x，计算开始时把 $\#x\$$ 存放在输入带上，其中 # 和 $ 是两个特殊的带符号，分别表示输入带的左端和右端. 当输入带的带头扫描 # 时只能右移，扫描 $ 时只能左移，使得带头不会移出输入带. 由于输入带的带头只读不写，输入带的内容 $\#x\$$ 在计算中保持不变.

容易用 $k+1$带 Turing 机模拟具有 k 条工作带的离线 Turing 机，只需用一条带作为输入带并且开始计算时首先把 # 和 $ 分别

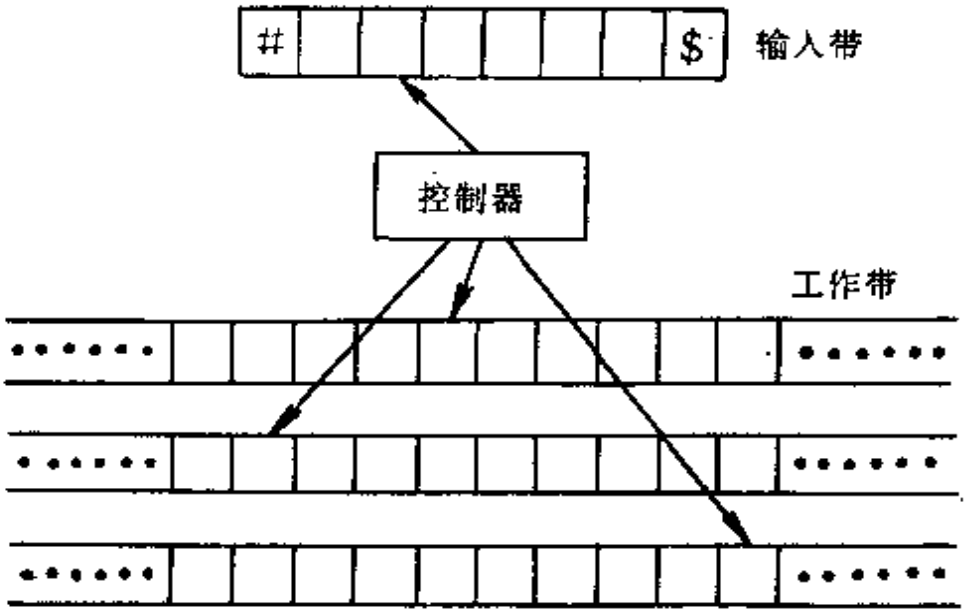


图6.10 离线 Turing 机

放置在输入串的两端. 反之,也容易用具有 k 条工作带的离线 Turing 机模拟 k 带 Turing 机,只要先把输入串写到一条工作带上,然后用 k 条工作带像 k 带 Turing 机一样地进行计算. 因此,离线 Turing 机也和基本 Turing 机具有相同的计算能力.

离线 Turing 机能区分输入数据和中间数据占用的存储单元. 在第十二章考虑计算需要占用的存储单元数时将采用离线 Turing 机作为基本的计算模型. 相应地,前面介绍的各种 Turing 机模型都称作**在线 Turing 机**. 在线 Turing 机计算时不保留输入串,输入串占用的单元也可以用来存放中间数据.

6.5 非确定型 Turing 机

一台**非确定型 Turing 机** $\mathscr{M}$ 也由6部分组成,记作 $\mathscr{M}=(Q, A, C, \delta, B, q_1)$,其中 Q, A, C, B, q_1 与基本 Turing 机的相同,而动作函数 δ 为 $Q\times C$ 到 $(C\cup\{L,R\})\times Q$ 的二元关系. 对于每一对 $q\in Q$ 和 $s\in C$,$\delta(q,s)$ 是 $(C\cup\{L,R\})\times Q$ 的子集,给出若干个可能的动作. 子集的每一个元素给出一个可能的动作. 在当前状态为

q、被扫描的符号为 s 时，若 $\delta(q,s)\neq\varnothing$ 则 $\mathscr{M}$ 执行其中的任意一个动作；若 $\delta(q,s)=\varnothing$ 则停止计算. 和基本 Turing 机一样，动作函数 δ 可以表示成4元组的有穷集合，并且不再需要限制集合内没有多个4元组以相同的 qs 开始.

和基本 Turing 机不同的是，非确定型 Turing 机的动作是不确定的，在每一步有若干个可能的动作. 给定初始格局，非确定型 Turing 机可能有不止一个计算，有的停机早，有的停机晚，甚至还可能永不停机. 它可表示成一棵树，树根是初始格局，每一个节点是一个格局. 从树根开始的每一条路径(可能是无穷的)是一个计算，如图6.11所示. 相对地，基本 Turing 机在每一步的动作是完全确定的. 给定初始格局，它只有唯一的一个计算，因此基本Turing 机是确定型的，称作**确定型 Turing 机**. 用 DTM 作为确定型 Turing机的缩写，NTM 作为非确定型 Turing 机的缩写. DTM 是特殊类型的 NTM. 今后在一般情况下 Turing 机(TM)均指 NTM，它包括 DTM 在内.

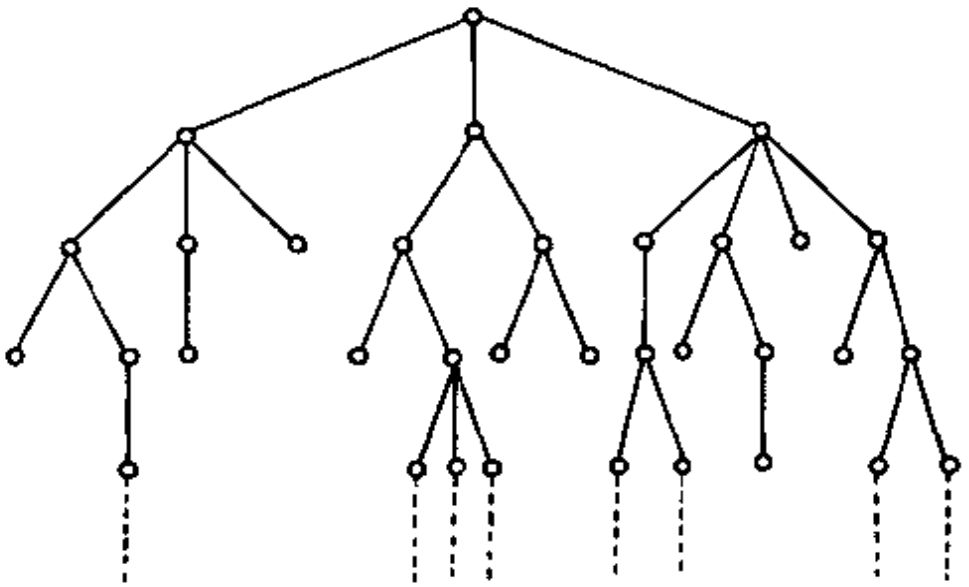


图6.11 NTM 的计算树

通常，非确定型 Turing 机只用做语言接受器，而不再用来计

算函数.

定义6.5 设 NTM $\mathscr{M}=(Q,A,C,\delta,B,q_1)$, $x\in A^*$. 如果存在从初始格局 σ_1

$$\begin{array}{c} Bx \\ \uparrow \\ q_1 \end{array}$$

开始、以某个停机格局结束的计算，则称 $\mathscr{M}$ **接受** x，并把该计算称作接受 x 的计算或关于 x 的**接受计算**. $\mathscr{M}$ 接受的字符串的全体称作 $\mathscr{M}$ **接受的语言**，或 $\mathscr{M}$ **识别的语言**，记作 $L(\mathscr{M})$. 即

$$L(\mathscr{M})=\{x\in A^* \mid \mathscr{M} \text{ 接受 } x\}.$$

根据定义，对于任何 $x\in A^*$，如果 $x\notin L(\mathscr{M})$，则从初始格局 σ_1 开始的任何计算都永不停机；如果 $x\in L(\mathscr{M})$，则一定存在从 σ_1 开始的最终停机的计算. 这种接受 x 的计算可能不止一个，并且不排除存在从 σ_1 开始永不停机的计算.

[例6.5] NTM $\mathscr{M}=(Q,A,C,\delta,B,q_1)$，其中 $Q=\{q_1,q_2\}$, $A=\{0,1\}$, $C=\{0,1,B\}$, $\delta=\{q_10Rq_1, q_11Rq_1, q_1BRq_1, q_10Rq_2, q_200q_2, q_211q_2\}$. 它的状态转移图如图6.12所示.

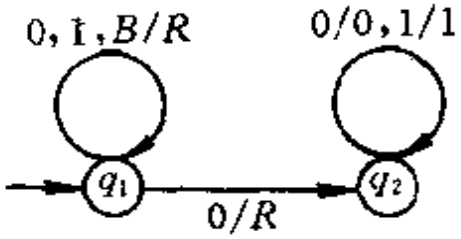


图6.12 NTM $\mathscr{M}$ 的状态转移图

对于输入 $x=100$, $\mathscr{M}$ 可以有好几个计算. 例如，

(1) $\begin{array}{c} B100 \\ \uparrow \\ q_1 \end{array} \vdash \begin{array}{c} B100 \\ \uparrow \\ q_1 \end{array} \vdash \begin{array}{c} B100 \\ \uparrow \\ q_1 \end{array} \vdash \begin{array}{c} B100 \\ \uparrow \\ q_1 \end{array}$

$\vdash \begin{array}{c} B100B \\ \uparrow \\ q_1 \end{array} \vdash \begin{array}{c} B100BB \\ \uparrow \\ q_1 \end{array} \vdash \cdots$

$\mathscr{M}$ 总处于状态 q_1，读写头不断地右移，永不停机.

(2) $B\,1\,0\,0 \vdash B\,1\,0\,0 \vdash B\,1\,0\,0 \vdash B\,1\,0\,0$

$q_1 \qquad q_1 \qquad q_1 \qquad q_2$

$\vdash B\,1\,0\,0 \vdash \cdots$

q_2

$\mathscr{M}$ 在状态 q_2 下，读写头重复在它扫描的那个方格内写0，陷入死循环.

(3) $B\,1\,0\,0 \vdash B\,1\,0\,0 \vdash B\,1\,0\,0 \vdash B\,1\,0\,0$

$q_1 \qquad q_1 \qquad q_1 \qquad q_1$

$\vdash B\,1\,0\,0\,B$

q_2

最后一个格局是停机格局. 这是关于100的接受计算，因此 $\mathscr{M}$ 接受100.

可以看出，$\mathscr{M}$ 接受的语言是

$$L(\mathscr{M}) = \{x0 | x \in \{0,1\}^*\}.$$

由于 DTM 是 NTM 的特殊情况，自然有：

定理6.10 每一个递归可枚举语言都被一台非确定型 Turing 机接受.

事实上，定理6.10的逆也成立，即 NTM 接受的语言都是 r.e.. 可以用 DTM 模拟 NTM，从而证明这个结论是正确的. 不过这里不去做这样的模拟. 这个结论是下一章几个定理合在一起的自然结果.

习　题

1. 设 TM $\mathscr{M}$ 的输入字母表 $A=\{0,1\}$，动作函数 δ 由下述4元组给出：$q_1BRq_1, q_10Rq_2, q_11Lq_1, q_20Lq_1, q_21Rq_1$，其中 q_1 是初始状态.

(1) 画出 $\mathscr{M}$ 的状态转移图.

(2) 给出 $\mathscr{M}$ 关于01010和0110的计算.

(3) 求 $\psi_{\mathscr{M}}^{(1)}(x)$.

(4) 求 $L(\mathscr{M})$.

2. 设字母表 $A=\{a,b\}$, $f(x)=x^R$(见第四章习题第3题). 试构造一台TM计算 $f(x)$.

3. 设计一台TM以一进制方式计算 $\lfloor\log_2(x-1)\rfloor$.

4. 设计接受下述语言的TM:

(1) $\{w|w\in\{a,b\}^*$ 且 w 至少含有2个连续的 $b\}$.

(2) $\{a^{2^n}|n\in N\}$.

(3) $\{0^n1^n|n\in N\}$.

(4) $\{w|w\in\{0,1\}^*$ 且 w 中0和1的个数相等$\}$.

5. 证明: A 上的语言 L 是r.e.,当且仅当存在DTM $\mathscr{M}$, $\mathscr{M}$ 有一个状态 q_y,使得对于所有的 $x\in A^*$, $x\in L$ 当且仅当 $\mathscr{M}$ 关于 x 的计算停机在 q_y. (当 $x\notin L$ 时, $\mathscr{M}$ 可能停机在其他状态,也可能永不停机.)

6. 证明: A 上的语言 L 是递归的,当且仅当存在DTM $\mathscr{M}$,使得对于所有的 $x\in A^*$,当 $x\in L$ 时 $\mathscr{M}$ 的计算停机在 q_y;当 $x\notin L$ 时 $\mathscr{M}$ 的计算停机在 q_N.

第七章　过程与文法

7.1　半 Thue 过 程

定义 7.1　设字母表 $A, g, h \in A^*$. 把形如

$$g \to h$$

的表达式称作**半 Thue 产生式**，简称作**产生式**. 半 Thue 产生式的有穷集合称作**半 Thue 过程**.

"Thue"一词来自挪威数学家 Axel Thue 的姓，读作"图厄".

产生式给出对 A 上字符串的一种替换规则.

设产生式 $g \to h$，把它记作 P. 又设 $u, v \in A^*$. 如果存在 $\alpha, \beta \in A^*$ 使得

$$u = \alpha g \beta, \qquad v = \alpha h \beta,$$

即把 u 中的子串 g 替换成 h 得到 v，则记作

$$u \underset{P}{\Rightarrow} v.$$

设半 Thue 过程 Π，如果存在 $P \in \Pi$ 使得 $u \underset{P}{\Rightarrow} v$，则记作

$$u \underset{\Pi}{\Rightarrow} v.$$

如果

$$u = u_1 \underset{\Pi}{\Rightarrow} u_2 \underset{\Pi}{\Rightarrow} \cdots \underset{\Pi}{\Rightarrow} u_n = v,$$

则记作

$$u \overset{*}{\underset{\Pi}{\Rightarrow}} v,$$

称序列 $u_1 = u, u_2, \cdots, u_n = v$ 是从 u 到 v 的**派生**. 特别地，取 $n = 1$，对任意的 $u \in A^*$，总有

$$u \overset{*}{\underset{\Pi}{\Rightarrow}} u.$$

当不会引起混淆时，常省去⇒号下的 P 或 Π，写成

$$u \Rightarrow v \qquad \text{和} \qquad u \overset{*}{\Rightarrow} v.$$

例如，设 $\Pi=\{ab\to aaa, ba\to bba\}$，则有

$aba \Rightarrow aaaa = a^4$，用一次 $ab\to aaa$.

$aba \Rightarrow abba \Rightarrow aaaba \Rightarrow a^6$，用一次 $ba\to bba$，再用二次 $ab\to aaa$.

$aba \Rightarrow abba \Rightarrow abbba \overset{*}{\Rightarrow} a^8$，用二次 $ba\to bba$，再用三次 $ab\to aaa$.

….

$h\to g$ 称作产生式 $g\to h$ 的**逆**. 如果半 Thue 过程包含它的每一个产生式的逆，则称这个过程是 **Thue 过程**.

7.2 用半 Thue 过程模拟 Turing 机

设 TM $\mathscr{M}=(Q, A, C, \delta, s_0, q_1)$，其中 $C=\{s_0, s_1, \cdots, s_m\}$，$Q=\{q_1, q_2, \cdots, q_n\}$. 要构造半 Thue 过程 $\Sigma(\mathscr{M})$ 模拟 TM $\mathscr{M}$. $\Sigma(\mathscr{M})$ 使用的字母表为

$$\check{C} = \{s_0, s_1, \cdots, s_m, q_0, q_1, \cdots, q_n, q_{n+1}, h\}.$$

$\mathscr{M}$ 的格局

$$\begin{array}{c} a_1 \cdots a_j \cdots a_k \\ \uparrow \\ q_i \end{array}$$

可以记作 $a_1 \cdots q_i a_j \cdots a_k$，用 $\check{C}$ 上的字符串

$$h a_1 \cdots q_i a_j \cdots a_k h$$

来表示. 这个字符串叫做 Post 字. 一个格局对应有多个 Post 字，在紧挨 h 的地方可以添加任意有穷个 s_0.

一般地，**Post 字**是一个字符串 huq_ivh，其中 $u, v\in C^*$，$0\leqslant i\leqslant n+1$. 当 $1\leqslant i\leqslant n$ 时，Post 字对应 $\mathscr{M}$ 的一个格局.

$\Sigma(\mathscr{M})$ 包括下述产生式：

(1) 对于 $\mathscr{M}$ 的每一个 $q_is_js_kq_l$，有

$$q_is_j \to q_ls_k.$$

(2) 对于 $\mathscr{M}$ 的每一个 $q_is_jRq_l$，有

$$q_i s_j s_k \to s_j q_i s_k, \quad k = 0,1,\cdots,m,$$

$$q_i s_j h \to s_j q_i s_0 h.$$

(3) 对于 $\mathscr{M}$ 的每一个 $q_i s_j L q_l$，有

$$s_k q_i s_j \to q_l s_k s_j, \quad k = 0,1,\cdots,m,$$

$$h q_i s_j \to h q_l s_0 s_j.$$

(4) 若 $\mathscr{M}$ 没有以 $q_i s_j$ 开头的 4 元组，则有

$$q_i s_j \to q_{n+1} s_j.$$

q_{n+1}相当于“停机”状态.

(5)
$$q_{n+1} s_j \to q_{n+1}, \quad j = 0,1,\cdots,m,$$

$$q_{n+1} h \to q_0 h,$$

$$s_j q_0 \to q_0, \quad j = 0,1,\cdots,m.$$

前三组产生式分别模拟对应的 4 元组. 于是，对于 $\mathscr{M}$ 的每一个有穷长度的计算，对应地有一个由 Post 字组成的派生，这些 Post 字恰好依次对应计算中的格局.

例如，$\mathscr{M}$ 有 4 元组：

$$q_2 s_2 s_1 q_3, q_3 s_1 R q_3,$$

则 $\Sigma(\mathscr{M})$含有产生式：

$$q_2 s_2 \to q_3 s_1,$$

$$q_3 s_1 s_k \to s_1 q_3 s_k, \quad k = 0,1,\cdots,m,$$

$$q_3 s_1 h \to s_1 q_3 s_0 h.$$

$\mathscr{M}$ 可以有下述计算：

$$\begin{array}{cccccccc} s_2 & s_2 s_1 & \vdash & s_2 & s_1 s_1 & \vdash & s_2 s_1 & s_1 \vdash s_2 s_1 s_1 \quad s_0 \\ & \uparrow & & & \uparrow & & & \uparrow \qquad\qquad \uparrow \\ & q_2 & & & q_3 & & & q_3 \qquad\quad q_3 \end{array}$$

对应的派生为：

$$h s_2 q_2 s_2 s_1 h \Rightarrow h s_2 q_3 s_1 s_1 h$$

$$\Rightarrow h s_2 s_1 q_3 s_1 h$$

$$\Rightarrow h s_2 s_1 s_1 q_3 s_0 h$$

注意，当 q_i 要移出字符串的两端时应添加 s_0；反之，如果派生中的

每一个 Post 字都对应 $\mathscr{M}$ 的一个格局，则这些格局构成一个计算.

第 4 组产生式保证如果 $\mathscr{M}$ 对输入 $x\in A^*$ 最终停机，则 $\Sigma(\mathscr{M})$ 从 hq_1s_0xh 派生出含有 q_{n+1} 的 Post 字. 事实上，也只有当 $\mathscr{M}$ 对输入 $x\in A^*$ 最终停机时 $\Sigma(\mathscr{M})$ 才能够从 hq_1s_0xh 派生出含有 q_{n+1} 的 Post 字. 因为前三组产生式都不可能派生出含有 q_{n+1} 的 Post 字，第 5 组产生式也不可能从不含 q_{n+1} 的 Post 字派生出含 q_{n+1} 的 Post 字.

第 5 组产生式保证 $\Sigma(\mathscr{M})$ 从含 q_{n+1} 的 Post 字能派生出 hq_0h. 事实上，也只能从含 q_{n+1} 的 Post 字派生出 hq_0h，因为其他 4 组产生式都不能派生出 hq_0h.

定理 7.1 设 NTM $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，$x\in A^*$，则 $\mathscr{M}$ 接受 x 当且仅当

$$hq_1s_0xh \overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} hq_0h.$$

证：设 $\mathscr{M}$ 接受 x，则

$$\begin{array}{ccc} s_0x & \overset{*}{\underset{\mathscr{M}}{\vdash}} & u\,s_jv \\ \uparrow & & \uparrow \\ q_1 & & q_j \end{array}$$

其中 $u,v\in C^*$，$q_j\in Q$，并且 $\mathscr{M}$ 没有以 q_js_j 开头的 4 元组. 于是，有

$$\begin{aligned} hq_1s_0xh &\overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} huq_js_jvh \\ &\underset{\Sigma(\mathscr{M})}{\Longrightarrow} huq_{n+1}s_jvh \\ &\overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} huq_{n+1}h \\ &\underset{\Sigma(\mathscr{M})}{\Longrightarrow} huq_0h \\ &\underset{\Sigma(\mathscr{M})}{\Longrightarrow} hq_0h \end{aligned}$$

反之，设 $hq_1s_0xh \overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} hq_0h$. 由于只有含 q_{n+1} 的 Post 字才能派生出含 q_0 的字，而含 q_{n+1} 的 Post 字又只能由含 q_js_j 的 Post 字派

生出来，这里 $\mathscr{M}$ 没有以 $q_i s_j$ 开头的 4 元组，因此必有

$$hq_1 s_0 xh \xRightarrow[\Sigma(\mathscr{M})]{*} huq_i s_j vh,$$

其中 $u,v\in C^*$. 相应地，有

$$\begin{array}{ccc} s_0x & \overset{*}{\underset{\mathscr{M}}{\vdash}} & u\,s_jv \\ \uparrow & & \uparrow \\ q_1 & & q_i \end{array}$$

而后一个格局是停机格局，所以 $\mathscr{M}$ 接受 x. □

由 $\Sigma(\mathscr{M})$ 中所有产生式的逆组成的半 Thue 过程记作 $\Omega(\mathscr{M})$. 显然，对于任意两个 Post 字 α,β，

$$\alpha \xRightarrow[\Sigma(\mathscr{M})]{*} \beta \text{ 当且仅当 } \beta \xRightarrow[\Omega(\mathscr{M})]{*} \alpha.$$

因此，有：

定理 7.2 设 NTM $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，$x\in A^*$，则 $\mathscr{M}$ 接受 x 当且仅当

$$hq_0h \xRightarrow[\Omega(\mathscr{M})]{*} hq_1s_0xh.$$

7.3 文　　法

定义 7.2 一个**短语结构文法** G 由下述 4 部分组成：

(1) 变元集 V，V 是非空有穷集合，它的元素称作**变元**或**非终极符**；

(2) 终极符集 T，T 是有穷集合，它的元素称作**终极符**，并且 $V\cap T=\varnothing$；

(3) $V\cup T$ 上产生式的有穷集合 Γ；

(4) **起始符** S，$S\in V$.

记作 $G=(V,T,\Gamma,S)$. 短语结构文法又称作**无限制文法**，简称文法.

文法实际上就是一个半 Thue 过程，把它的字母表分成两个不相交的部分，分别叫做变元集和终极符集，并指定一个特殊的变

元叫做起始符.

对于文法，今后总把 $\alpha \underset{\Gamma}{\Rightarrow} \beta$ 和 $\alpha \overset{*}{\underset{\Gamma}{\Rightarrow}} \beta$ 分别记作 $\alpha \underset{G}{\Rightarrow} \beta$ 和 $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$，或简记作 $\alpha \Rightarrow \beta$ 和 $\alpha \overset{*}{\Rightarrow} \beta$.

定义 7.3 文法 $G=(V,T,\Gamma,S)$**生成的语言**是

$$L(G) = \{u \mid u \in T^* \wedge S \overset{*}{\Rightarrow} u\}.$$

[**例 7.1**] 设文法 $G=(V,T,\Gamma,S)$，其中 $V=\{S,X\}$，$T=\{a,b\}$，$\Gamma=\{S\to aS, S\to aT, T\to bT, T\to b\}$. 不难看出

$$L(G) = \{a^n b^m \mid n,m > 0\}.$$

定理 7.3 设 L 是被非确定型 Turing 机接受的语言，则存在文法 G 使得 $L=L(G)$.

证：设 $L\subseteq A^*$，NTM $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$ 接受 L，其中 $Q=\{q_1,q_2,\cdots,q_n\}$. 构造文法 $G=(V,T,\Gamma,S)$ 如下：$T=A$，$V=C\cup Q\cup\{q_0,q_{n+1},h,p,S\}$. 这里 $q_0,q_{n+1},h,p,S\notin C\cup Q$，而 Γ 由 $\Omega(\mathscr{M})$ 的产生式以及下述产生式组成：

$$S\to hq_0h$$

$$hq_1s_0\to p$$

$$ps\to sp, \quad \forall s\in T,$$

$$ph\to\varepsilon.$$

设 $x\in L$，则 $x\in T^*$ 且 $\mathscr{M}$ 接受 x. 由定理 7.2，有

$$S \underset{G}{\Rightarrow} hq_0h \overset{*}{\underset{G}{\Rightarrow}} hq_1s_0xh \underset{G}{\Rightarrow} pxh \overset{*}{\underset{G}{\Rightarrow}} xph \underset{G}{\Rightarrow} x.$$

因此，$x\in L(G)$.

反之，设 $x\in L(G)$，则 $x\in T^*$ 且 $S \overset{*}{\underset{G}{\Rightarrow}} x$. 注意到 $S\to hq_0h$ 是 G 中唯一与 S 有关的产生式，故必有

$$S \underset{G}{\Rightarrow} hq_0h \overset{*}{\underset{G}{\Rightarrow}} x.$$

又因为 $ph\to\varepsilon$ 是 G 中唯一可以得到 T^* 中元素的产生式，故必有

$$S \underset{G}{\Rightarrow} hq_0h \overset{*}{\underset{G}{\Rightarrow}} yphw \underset{G}{\Rightarrow} yw = x$$

其中 $y,w\in T^*$. 而 $hq_1s_0\to p$ 是 G 中唯一产生 p 的产生式，并注意

到对任意的 $v\in T^*$ 有 $pv\overset{*}{\underset{G}{\Rightarrow}}vp$，故有

$$S\underset{G}{\Rightarrow}hq_0h\overset{*}{\underset{G}{\Rightarrow}}uhq_1s_0vhw\underset{G}{\Rightarrow}upvhw$$

$$\overset{*}{\underset{G}{\Rightarrow}}uvphw\Rightarrow uvw=x,$$

这里 $u,v\in T^*$ 且 $uv=y$. 在从 hq_0h 到 uhq_1s_0vhw 的派生过程中不会使用 $\Omega(\mathscr{M})$ 之外的产生式，即

$$hq_0h\overset{*}{\underset{\Omega(\mathscr{M})}{\Longrightarrow}}uhq_1s_0vhw.$$

但是 $\Omega(\mathscr{M})$ 的产生式从 Post 字只能派生出 Post 字，所以 $u=w=\varepsilon$，$x=uvw=v$. 于是，有

$$hq_0h\overset{*}{\underset{\Omega(\mathscr{M})}{\Longrightarrow}}hq_1s_0xh.$$

根据定理 7.2，$\mathscr{M}$ 接受 x. 从而，$x\in L$.

这就证明了对任意的 $x\in A^*$，$x\in L$ 当且仅当 $x\in L(G)$，即 $L=L(G)$. □

下面考虑文法接受的语言具有什么样的性质. 为此，我们再一次把字符串看成数的表示.

设文法 $G=(V,T,\Gamma,S)$，其中 $V=\{V_1,V_2,\cdots,V_k\}$，$T=\{s_1,s_2,\cdots,s_n\}$，$S=V_1$. 规定 $V\cup T$ 的元素的顺序如下：

$$s_1,s_2,\cdots,s_n,V_1,V_2,\cdots,V_k,$$

并且按照这个顺序把 $V\cup T$ 上的字符串看作 $n+k$ 进制数.

引理 7.4 谓词 $u\underset{G}{\Rightarrow}v$ 是原始递归的.

证：设 G 有 l 个产生式 $P_i:g_i\rightarrow h_i$，$1\leqslant i\leqslant l$. 对每一个 $i(1\leqslant i\leqslant l)$，

$$u\underset{P_i}{\Rightarrow}v\Leftrightarrow\exists\ \alpha\mathsf{E}\beta(u=\alpha g_i\beta\wedge v=\alpha h_i\beta)$$

$$\Leftrightarrow(\exists\ \alpha)_{\leqslant u}(\exists\ \beta)_{\leqslant u}\{u=\mathrm{CONCAT}_{n+k}(\alpha,g_i,\beta)$$

$$\wedge\ v=\mathrm{CONCAT}_{n+k}(\alpha,h_i,\beta)\},$$

因此谓词 $u\underset{P_i}{\Rightarrow}v$ 是原始递归的. 面

$$u\underset{G}{\Rightarrow}v\Leftrightarrow u\underset{P_1}{\Rightarrow}v\vee u\underset{P_2}{\Rightarrow}v\vee\cdots\vee u\underset{P_l}{\Rightarrow}v,$$

得证 $u\underset{G}{\Rightarrow}v$ 是原始递归的. □

定义谓词

$$\mathrm{DERIV}(u,y)\Leftrightarrow y=[u_1,u_2,\cdots,u_m,1]$$
$$\wedge\ u_1=S\ \wedge\ u_m=u\ \wedge\ u_1\underset{G}{\Rightarrow}u_2\underset{G}{\Rightarrow}\cdots\underset{G}{\Rightarrow}u_m.$$

在序列 $u_1,u_2,\cdots,u_m$ 后添加 1 是为避免当 $u=\epsilon$ 时可能产生的混乱.

引理 7.5 谓词 $\mathrm{DERIV}(u,y)$ 是原始递归的.

证:注意到 S 的值等于 $n+1$,有

$$\mathrm{DERIV}(u,y)\Leftrightarrow(y)_1=n+1\ \wedge\ (y)_m=u\ \wedge\ (y)_{m+1}=1$$
$$\wedge\ (\forall\ j)_{<m}\{j=0\ \vee\ (y)_j\underset{G}{\Rightarrow}(y)_{j+1}\},$$

其中 $m+1=\mathrm{Lt}(y)$. 由引理 7.4,得证 $\mathrm{DERIV}(u,y)$ 是原始递归的. □

定理 7.6 文法生成的语言是递归可枚举的.

证:设文法 $G=(V,T,\Gamma,S)$,对于 $x\in(V\cup T)^*$,

$$S\overset{*}{\underset{G}{\Rightarrow}}x\Leftrightarrow(\exists\ y)\mathrm{DERIV}(x,y)$$
$$\Leftrightarrow\min_y\mathrm{DERIV}(x,y)\downarrow$$

根据引理 7.5 和定理 2.13,$\min_y\mathrm{DERIV}(x,y)$ 是部分可计算函数,故集合

$$\{x\mid S\overset{*}{\underset{G}{\Rightarrow}}x\}$$

是 r.e.. 从而

$$L(G)=T^*\cap\{x\mid S\overset{*}{\underset{G}{\Rightarrow}}x\}$$

也是 r.e.. □

现在我们又得到一个关于语言的蕴涵圈图 7.1. 根据这个蕴涵圈有下述定理.

定理 7.7 设 L 是一个语言,下述命题是等价的:

(1) L 是递归可枚举的;

(2) L 被确定型 Turing 机接受;

(3) L 被非确定型 Turing 机接受；

(4) L 由文法生成.

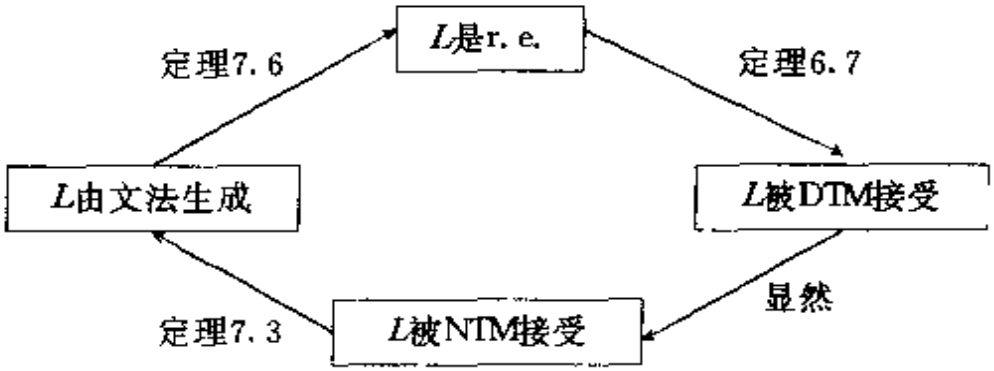


图 7.1

7.4 再论递归可枚举集

本节回到自然数集合上来，进一步讨论 r. e. 集与部分可计算函数的关系.

定理 7.8 设 $B\subseteq N$，若 B 是递归可枚举的，则存在原始递归谓词 $R(x,t)$使得

$$B=\{x\mid \exists t\ R(x,t)\}.$$

证：令 $L=\{s_1^x\mid x\in B\}$，则 L 是字母表 $T=\{s_1\}$上的 r. e. 语言. 由定理7. 7，存在文法 G 生成 L，即 $L=L(G)$. 于是，对于任意的 $x\in N$，

$$x\in B\Leftrightarrow \exists t\ \mathrm{DERIV}(s_1^x,t).$$

G 只有一个终极符 s_1，设有 k 个变元 $V_1,\cdots,V_k$. 在字母表$\{s_1,V_1,\cdots,V_k\}$上以 $k+1$为底 s_1^x表示的数为 $\mathrm{UPCHANGE}_{1,k+1}(x)$. 取

$$R(x,t)=\mathrm{DERIV}(\mathrm{UPCHANGE}_{1,k+1}(x),t),$$

$R(x,t)$是原始递归的，并且

$$B=\{x\mid \exists t\ R(x,t)\}.$$ □

定理7. 9 设 B 是一个非空递归可枚举集，则存在原始递归函数 $f(x)$使得

$$B = \{f(x) \mid x \in N\}.$$

证：根据定理7.8，存在原始递归谓词 $R(y,t)$ 使得

$$B = \{y \mid \exists t\ R(y,t)\}.$$

B 非空，设 $y_0 \in B$. 令

$$f(x) = \begin{cases} l(x) & \text{若 } R(l(x),r(x)), \\ y_0 & \text{否则,} \end{cases}$$

其中 $l(x),r(x)$ 的定义见2.4节. $f(x)$ 是原始递归的.

对任意的 x，如果 $\neg R(l(x),r(x))$，则 $f(x)=y_0 \in B$. 如果 $R(l(x),r(x))$，取 $y=l(x),t=r(x)$，则 $R(y,t)$ 为真. 于是，$f(x)=l(x)=y \in B$. 因此，总有 $f(x) \in B$.

反之，设 $y \in B$，则存在 t 使得 $R(y,t)$. 取 $x=\langle y,t\rangle$，有 $R(l(x),r(x))$，得 $f(x)=l(x)=y$. 这就证明了

$$B = \{f(x) \mid x \in N\}. \qquad \square$$

定理7.10 集合 B 是递归可枚举的当且仅当存在部分可计算函数 $f(x)$ 使得

$$B = \{f(x) \mid f(x)\downarrow\}.$$

证：必要性的证明和定理7.9的基本相同，只需要取

$$f(x) = \begin{cases} l(x) & \text{若 } R(l(x),r(x)) \\ \uparrow & \text{否则.} \end{cases}$$

下面证明充分性. 令

$$g(x) = \begin{cases} 0 & \text{若 } \exists z\{f(z)\downarrow \wedge\ x = f(z)\} \\ \uparrow & \text{否则.} \end{cases}$$

显然，

$$B = \{x \mid g(x)\downarrow\}.$$

下述 $\mathscr{S}$ 程序 $\mathscr{Q}$ 计算 $g(x)$，从而得证 B 是 r.e.. 设程序 $\mathscr{P}$ 计算 $f(x)$，$p=\#(\mathscr{P})$. 程序 $\mathscr{Q}$ 的清单如下：

$[A]$ IF $\neg$ STP$^{(1)}(Z,p,T)$ GOTO B
$V \leftarrow f(Z)$
IF $V = X$ GOTO E

$[B]$ $Z \leftarrow Z+1$

IF $Z \leqslant T$ GOTO A

$T \leftarrow T+1$

$Z \leftarrow 0$

GOTO A □

根据上述两个定理可以得到关于非空集合 B 的蕴涵圈图 7.2，从而得到下述定理：

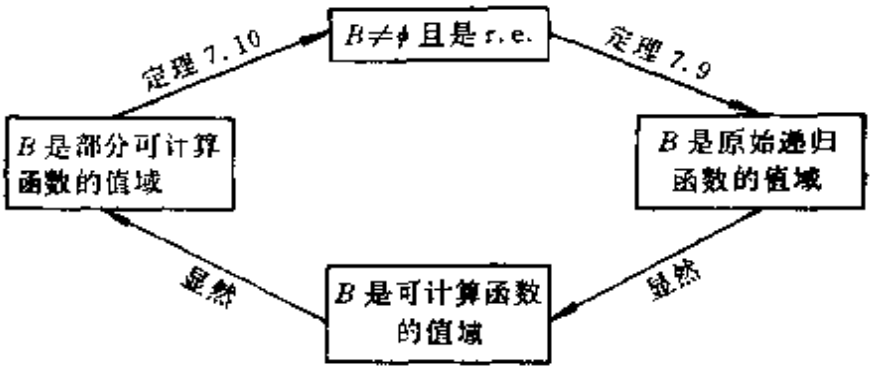


图 7.2

定理7.11 设 B 非空，则下述命题是等价的：

(1) B 是递归可枚举的，即是一个部分可计算函数的定义域；

(2) B 是一个原始递归函数的值域；

(3) B 是一个可计算函数的值域；

(4) B 是一个部分可计算函数的值域.

作为定理7.11的应用，我们证明一个非 r.e. 集.

[例7.2] 集合

$$T = \{t \in N \mid \Phi_t(x) \text{ 是全函数}\}$$

不是递归可枚举的.

证：假设 T 是 r.e.，由于 $T \neq \varnothing$，根据定理7.11，存在可计算函数 $g(x)$ 使得

$$T = \{g(x) \mid x \in N\}.$$

令

$$h(x) = \Phi(x, g(x)) + 1.$$

对每一个 $x\in N, g(x)\in T$. 由 T 的定义，$\Phi_{g(x)}$是一个全函数，当然有 $\Phi(x,g(x))\downarrow$，故 $h(x)$是一个全函数，所以 $h(x)$是可计算的. 于是，存在 $p\in T$ 使得 $h(x)=\Phi_p(x)$. 根据 $g(x)$的定义，存在 $x_0\in N$使得 $p=g(x_0)$，得到

$$h(x_0)=\Phi(x_0,g(x_0))+1=\Phi_p(x_0)+1=h(x_0)+1,$$

矛盾. □

7.5 部分递归函数

在2.3节给出了部分递归函数的定义，并证明部分递归函数都是部分可计算的. 本节将证明部分可计算函数都是部分递归的，从而部分递归函数类等同于部分可计算函数类，递归函数类等同于可计算函数类.

引理7.12 设 $g(x)$是部分可计算函数，则

$$B=\{\langle x,y\rangle \mid y=g(x)\}$$

是递归可枚举的.

证：令 $f(x)=\langle x,g(x)\rangle$，有

$$B=\{f(x)\mid f(x)\downarrow\}.$$

$f(x)$是部分可计算的，根据定理7.10得证 B 是 r.e.. □

定理7.13 设 $f(x)$是部分可计算函数，则存在原始递归谓词 $R(x,t)$使得

$$f(x)=l(\min_t R(x,t)).$$

证：令 $B=\{\langle x,y\rangle \mid y=f(x)\}$. 由引理7.12，$B$ 是 r.e.. 根据定理7.8，存在原始递归谓词 $Q(u,v)$使得

$$B=\{u\mid \exists\, v\ Q(u,v)\}.$$

于是，

$$\begin{aligned} y=f(x) &\Leftrightarrow \langle x,y\rangle\in B\\ &\Leftrightarrow \exists\, v\ Q(\langle x,y\rangle,v)\\ &\Leftrightarrow \exists\, t\ \{y=l(t)\wedge Q(\langle x,l(t)\rangle,r(t))\}. \end{aligned}$$

要证

$$f(x) = l(\min_t Q(\langle x, l(t)\rangle, r(t))).$$

事实上，若 $t_0 = \min_t Q(\langle x, l(t)\rangle, r(t))$ 有定义，令 $y = l(t_0)$，$v = r(t_0)$，则 $Q(\langle x, y\rangle, v)$，从而，$f(x) = y = l(t_0)$. 若 $\min_t Q(\langle x, l(t)\rangle, r(t))$ 没有定义，则对任意的 y, v，$Q(\langle x, y\rangle, v)$ 为假，从而 $f(x)\uparrow$.

令 $R(x,t) = Q(\langle x, l(t)\rangle, r(t))$，$R(x,t)$ 是原始递归谓词，且有

$$f(x) = l(\min_t R(x,t)). \qquad \square$$

利用 Gödel 编码，不难把上述结果推广到 n 元部分可计算函数.

定理7.14(范式定理) 设 $f(x_1, \cdots, x_n)$ 是 n 元部分可计算函数，则存在原始递归谓词 $R(x_1, \cdots, x_n, t)$ 使得

$$f(x_1, \cdots, x_n) = l(\min_t R(x_1, \cdots, x_n, t)).$$

证：令 $g(x) = f((x)_1, \cdots, (x)_n)$，$g(x)$ 是部分可计算的. 根据定理7.13，存在原始递归谓词 $Q(x,t)$ 使得

$$g(x) = l(\min_t Q(x,t)).$$

令 $R(x_1, \cdots, x_n, t) = Q([x_1, \cdots, x_n], t)$，$R$ 是一个原始递归谓词并且

$$\begin{aligned} f(x_1, \cdots, x_n) &= g([x_1, \cdots, x_n]) \\ &= l(\min_t Q([x_1, \cdots, x_n], t)) \\ &= l(\min_t R(x_1, \cdots, x_n, t)). \end{aligned} \qquad \square$$

定理7.15 一个函数是部分可计算函数当且仅当它是部分递归函数.

证：由定理2.15，已知任何部分递归函数都是部分可计算的.

反之，设 $f(x_1, \cdots, x_n)$ 是一个部分可计算函数，根据定理7.14，它可表示成

$$f(x_1, \cdots, x_n) = l(\min_t R(x_1, \cdots, x_n, t)),$$

其中 $R(x_1,\cdots,x_n,t)$是原始递归谓词. 函数 $l(z)$也是原始递归的. 根据定义,$R(x_1,\cdots,x_n,t)$和 $l(z)$都可以由初始函数经过有限次合成和原始递归运算得到. 从而,$f(x_1,\cdots,x_n)$可以由初始函数经过有限次合成、原始递归和极小化运算得到,即 $f(x_1,\cdots,x_n)$是部分递归函数.

推论7.16 一个全函数是可计算函数当且仅当它是递归函数,一个谓词是可计算谓词当且仅当它是递归谓词.

7.6 Church-Turing 论题

对于全函数 f,直观上说它是可计算的,是指任给 x,可以在有限步内得到 $f(x)$. 对于部分函数 f,直观上说它是部分可计算的,是指任给 x,当 $f(x)$有定义时可以在有限步内得到 $f(x)$. 但是,这种直观想法不能严格地判断函数是否是可计算的. 可计算性理论也称算法理论或能行性理论,它通过建立计算的数学模型,给出可计算的严格定义,从而精确地区分哪些是可计算的,哪些不是可计算的. 在本世纪30年代及其后,先后提出了若干计算的数学模型,包括前几章介绍的递归函数、Turing 机、半 Thue 过程等. 1936 年 A. Church 提出一个著名的论题:直观上可计算的函数就是递归函数. 同年 A. Turing 也提出一个论题:直观上可计算的函数就是 Turing 机可计算的函数. 其实它们是一回事. 现在通常把它们合称作 Church-Turing 论题,并且把它推广到部分函数.

Church-Turing 论题:通常所说的能行可计算函数等同于部分递归函数,也等同于可用 Turing 机计算的部分函数等.

论题中的"等"字把所有与部分递归函数等价的概念包括在内. 例如,$\mathscr{S}$ 部分可计算函数、$\mathscr{S}_n$ 部分可计算函数等. 由于直观上的能行可计算性没有严格的形式定义,因此 Church-Turing 论题是不能证明的. 但是,数十年来许多事实为论题提供了有力的支持. 在 A. Church 和 A. Turing 提出自己的论题之后,很快发现递

归性和 Turing 机可计算性是等价的. 以后又提出了多种计算数学模型，这些模型都没有提供更强的计算能力. 我们在前面已经清楚地看到这种情况. 此外，至今人们还没有发现一个直观上应该是可计算的、但不能用这些模型计算的函数. 因此，在数学和计算机科学界已经普遍接受 Church-Turing 论题，把(部分)递归函数、Turing机(部分)可计算的函数及其他等价的概念作为(部分)可计算函数的严格定义.

习　　题

1. 设 TM$\mathscr{M}$ 由下述4元组给出：q_1BRq_2，q_20Rq_2，q_21Lq_2.

(1) 写出 $\Sigma(\mathscr{M})$.

(2) 写出 $\mathscr{M}$ 关于 $w=000$的计算.

(3) 写出 $\Sigma(\mathscr{M})$对(b)中 $\mathscr{M}$ 计算的模拟.

2. 写出模拟上题中 TM$\mathscr{M}$ 的文法.

3. 设文法 $G=(V,T,\Gamma,S)$，其中 $V=\{A,B,C,D,E,S\}$，$T=\{a,b\}$，Γ 由下述产生式组成：

(1) $S\to ACaB$　　(2) $Ca\to aaC$

(3) $CB\to DB$　　(4) $CB\to E$

(5) $aD\to Da$　　(6) $AD\to AC$

(7) $aE\to Ea$　　(8) $AE\to\epsilon$

(1) 证明：对于每一个 $n>0$，$a^{2^n}\in L(G)$.

(2) 证明：$L(G)=\{a^{2^n}\mid n>0\}$.

4. 构造一个文法生成语言$\{ww\mid w\in\{0,1\}^*\}$.

5. 证明每一个无穷 r.e. 集都有无穷递归子集. (提示：利用第五章习题第1题(2))

第八章　不可判定的问题

8.1　判 定 问 题

如果问题的答案只有两种可能：是或否，则称这个问题是一个**判定问题**. 从考虑能行性的角度，我们感兴趣的是含有参数的判定问题. 例如，素数判定问题“任给一个数 x，x 是素数吗？”这里 x 是一个参数. 我们要讨论的是，是否能够能行地判定任给的 x 是否是素数.

设判定问题 Π 的所有实例的集合为 D_Π，所有答案为“是”的实例的集合为 Y_Π，记作 $\Pi=(D_\Pi,Y_\Pi)$. 例如，设 Π 是素数判定问题，则 $D_\Pi=N$，Y_Π 是所有素数. 又如，停机问题 H：“任给 $\mathscr{S}$ 程序 $\mathscr{P}$ 和数 x，$\mathscr{P}$ 对输入 x 最终停机吗？”，它的实例由一个 $\mathscr{S}$ 程序 $\mathscr{P}$ 和一个数 x 组成，记作$(\mathscr{P},x)$. 于是，

$D_H=\{(\mathscr{P},x)\mid \mathscr{P}$ 是 $\mathscr{S}$ 程序，$x\in N\}$，

$Y_H=\{(\mathscr{P},x)\mid (\mathscr{P},x)\in D_H$ 且 $\mathscr{P}$ 对输入 x 最终停机$\}$.

通过编码可以建立起判定问题与谓词之间的对应关系. 不失一般性，设编码 $e:D_\Pi\to N$，定义谓词

$$P_\Pi(x) \Leftrightarrow I\in Y_\Pi,\text{其中 } e(I)=x.$$

例如，在第三章通过 $\mathscr{S}$ 程序的编码，把停机问题 H 对应到谓词 $\mathrm{HALT}(x,y)$. 为了使谓词是一元的，只需使用一下配对函数. 定义编码 $e:D_H\to H$ 如下：对每一个 $I=(\mathscr{P},x)\in D_H$，

$$e(I)=\langle x,\#(\mathscr{P})\rangle.$$

在这个编码 e 下，$P_H(z)=\mathrm{HALT}(l(z),r(z))$.

我们要求编码 e 是能行的，即对于每一个实例 $I\in D_\Pi$，能够能行地给出 $e(I)$；反之，对每一个 $x\in N$，能够能行地判断 x 是否是

某个 $I\in D_\Pi$ 的代码并且当是的时候能够能行地给出这个 I. 对于同一个判定问题 Π，可以有不同的编码 e_1 和 e_2. 在这两个编码下，Π 分别对应到谓词 P_1 和 P_2. 根据 Church-Turing 论题，若 e_1 和 e_2 都是能行的，则存在可计算函数 $f_1:N\to N$ 和 $f_2:N\to N$ 使得

$$P_1(x)\Leftrightarrow P_2(f_1(x)) \text{ 和 } P_2(x)\Leftrightarrow P_1(f_2(x)).$$

从而，P_1 和 P_2 要么都是可计算的，要么都不是可计算的. 今后只使用能行的编码.

定义 8.1 如果谓词 P_Π 是可计算的，则称判定问题 Π 是**可判定的**或**可解的**；否则称 Π 是**不可判定的**或**不可解的**.

根据前面有关能行的编码的说明，这个定义与所用的编码(限制为能行的)无关，因此是有效的.

例如，素数判定问题是可判定的，因为谓词 Prime(x)是原始递归的. 而停机问题 H 是不可判定的，因为 HALT(x,y)不是可计算的，从而 $P_H(z)$也不是可计算的. 又如，设 $B\subseteq N$(或 $B\subseteq A^*$，A 是一个字母表)，B 的成员资格问题“任给 x，问 $x\in B$?”是可判定的当且仅当 B 是递归集.

实际上，每一个判定问题都可以看作某个集合的成员资格问题. 令

$$B_\Pi=\{x\mid P_\Pi(x)\},$$

则

$$I\in Y_\Pi \Leftrightarrow e(I)\in B_\Pi.$$

因此，Π 相当于 B_Π 的成员资格问题. 于是，Π 是可判定的当且仅当 B_Π 是递归的.

根据 Rice 定理(5.3 节)，我们有：

定理 8.1 设 Γ 是一元部分可计算函数的集合，并且存在一元部分可计算函数 g 和 h，使得 $g\in\Gamma$，$h\notin\Gamma$，则问题“任给一个 $\mathscr{S}$ 程序 $\mathscr{P}$，$\mathscr{P}$ 计算的一元函数属于 Γ 吗?”是不可判定的.

特别地，下述问题都是不可判定的：

(1) $\mathscr{P}$ 计算的函数是全函数吗?

(2) $\mathscr{P}$ 计算的函数是原始递归函数吗?

(3) $\mathscr{P}$ 计算的函数只在某个有穷集上没有定义吗?

(4) $\mathscr{P}$ 计算的函数与 $\mathscr{Q}$ 计算的函数相同吗? 这里 $\mathscr{Q}$ 是一个固定的 $\mathscr{S}$ 程序.

在后面的几节,我们不再给出编码,而直接讨论判定问题的可判定性. 相应地,用"算法"一词代替 $\mathscr{S}$ 程序或 Turing 机. 当然,我们是在 Church-Turing 论题意义下使用术语"算法". 算法是能机械地一步一步执行的操作规则. 我们能够说存在解判定问题 Π 的算法,当且仅当 Π 是可判定的. 更一般地,能够说存在计算函数 f 的算法,当且仅当 f 是部分可计算的.

例如,下面是解素数判定问题的算法:

$S1$. 若 $x\leqslant 1$ 则 x 不是素数,计算结束,否则执行 $S2$;

$S2$. 令 $i\leftarrow 2$;

$S3$. 若 $i=x$ 则 x 是素数,计算结束,否则执行 $S4$;

$S4$. 检查 $i|x$?

若 $i|x$ 则 x 不是素数,计算结束,否则执行 $S5$;

$S5$. 令 $i\leftarrow i+1$,执行 $S3$.

定义 8.2 设 Π_1 和 Π_2 是两个判定问题. 如果函数 $f: D_{\Pi_1}\to D_{\Pi_2}$ 满足下述条件:

(1) 存在计算 f 的算法,

(2) 对每一个 $I\in D_{\Pi_1}$, $I\in Y_{\Pi_1}\Leftrightarrow f(I)\in Y_{\Pi_2}$,

则称 f 是从 Π_1 到 Π_2 的**归约**.

如果存在从 Π_1 到 Π_2 的归约,则称 Π_1 **可归约到** Π_2.

定理 8.2 设判定问题 Π_1 可归约到 Π_2,则:

(1) Π_2 是可判定的蕴涵 Π_1 是可判定的.

(2) Π_1 是不可判定的蕴涵 Π_2 是不可判定的.

证: 只需证(1). 设 f 是 Π_1 到 Π_2 的归约. 如果 Π_2 是可判定的,则存在算法 A,对任给的 $I\in D_{\Pi_2}$,能够回答 I 是否属于 Y_{Π_2}. 如

下构造解 Π_1 的算法 A'：对任给的 $I\in D_{E_1}$，先计算 $f(I)$，然后把 A 运用于 $f(I)$. 当且仅当 A 对 $f(I)$ 回答“是”时，回答“是”. 根据归约的定义，A' 确实是解 Π_1 的算法，故 Π_1 是可判定的. □

实际上，定义 8.2 和定理 8.2 是对应于定义 3.1 和定理 3.6 (3.4 节)的非形式化叙述. 归约是证明不可判定性的有力工具，后面几节都是采用这个工具来证明不可判定性，即把一个已知的不可判定问题归约到要证的问题.

8.2 Turing 机的停机问题

由于 $\mathscr{S}$ 程序的停机问题是不可判定的，自然会想到 Turing 机的停机问题也是不可判定的. 关于 Turing 机的停机问题有多种提法，我们先给出它的一般提法和一种加强形式.

Turing 机的停机问题：任给 DTM$\mathscr{M}$ 和格局 σ，从格局 σ 开始，$\mathscr{M}$ 是否最终停机？

DTM$\mathscr{M}$ 的停机问题：任给格局 σ，从格局 σ 开始，$\mathscr{M}$ 是否最终停机？

我们从后一个问题开始.

定理 8.3 存在带字母表只含两个符号的 DTM$\mathscr{M}$ 使得它的停机问题是不可判定的.

证：取一个非递归的 r. e. 集 $B\subseteq N$(例如 K). 由推论 6.9，存在 DTM$\mathscr{M}$ 以 $\{s_0,s_1\}$ 为带字母表、接受语言 $\{s_1^x\mid x\in B\}$，其中 s_0 是空白符. 于是，对任意的 x，$x\in B$ 当且仅当从格局

$$\begin{array}{l} s_0x \\ \uparrow \\ q_1 \end{array}$$

开始，$\mathscr{M}$ 最终停机，其中 q_1 是 $\mathscr{M}$ 的初始状态. 因此，利用 $\mathscr{M}$ 停机问题的算法能解决判定问题“$x\in B$?”. 而 B 是非递归的，不存在这样的算法，所以 $\mathscr{M}$ 的停机问题是不可判定的. □

由于固定的 DTM $\mathscr{M}$ 的停机问题是 Turing 机的停机问题的特殊情况，根据定理 8.3 可以立即得到下述结论.

定理 8.4 Turing 机的停机问题是不可判定的.

设 DTM $\mathscr{M}$ 的带字母表为 $\{s_0, s_1\}$ 并且它的停机问题是不可判定的. 添加一个状态 $\tilde{q}$ 和若干 4 元组得到另一个 DTM $\tilde{\mathscr{M}}$：对 $\mathscr{M}$ 的每一个状态 q，如果 $\mathscr{M}$ 没有以 qs_0 开始的 4 元组，则添加 $qs_0s_0\tilde{q}$；如果 $\mathscr{M}$ 没有以 qs_1 开始的 4 元组，则添加 $qs_1s_1\tilde{q}$. 显然，从任一格局开始，$\mathscr{M}$ 最终停机当且仅当 $\tilde{\mathscr{M}}$ 能达到状态 $\tilde{q}$. 于是，我们又得到一个不可判定的 Turing 机停机问题.

定理 8.5 存在 DTM $\tilde{\mathscr{M}}$ 和状态 $\tilde{q}$，$\tilde{\mathscr{M}}$ 的带字母表只有两个符号，使得问题“任给格局 σ，从格局 σ 开始，$\tilde{\mathscr{M}}$ 是否能到达状态 $\tilde{q}$?”是不可判定的.

8.3 字问题和 Post 对应问题

8.3.1 字问题

半 Thue 过程 Π 的**字问题**：任给 Π 的字母表上的两个字符串 u 和 v，问是否 $u \overset{*}{\underset{\Pi}{\Rightarrow}} v$?

定理 8.6 存在 Turing 机 $\mathscr{M}$ 使得半 Thue 过程 $\Sigma(\mathscr{M})$ 和 $\Omega(\mathscr{M})$ 的字问题都是不可判定的.

证：设 DTM $\mathscr{M}$ 接受的语言 L 是非递归的，$\mathscr{M}$ 的输入字母表为 A. 先证 $\Sigma(\mathscr{M})$ 的字问题是不可判定的. 假设不然，存在一个算法能够判定任给的 $\Sigma(\mathscr{M})$ 的字母表上的字符串 u 和 v 是否有 $u \overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} v$. 当然，这个算法也能判定任给的 $x \in A^*$ 是否有 $hq_1s_0xh \overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} hq_0h$. 而

$$hq_1s_0xh \overset{*}{\underset{\Sigma(\mathscr{M})}{\Longrightarrow}} hq_0h \Leftrightarrow x \in L,$$

因此利用这个算法能够解决判定问题“$x \in L$?”，这与 L 是非递归的矛盾，故 $\Sigma(\mathscr{M})$的字问题是不可判定的.

由于对 $\Sigma(\mathscr{M})$的字母表上的任意两个字符串 u 和 v，

$$u \xrightarrow[\Sigma(\mathscr{M})]{*} v \quad \text{当且仅当} \quad v \xrightarrow[\Omega(\mathscr{M})]{*} u,$$

因此从 $\Sigma(\mathscr{M})$的字问题是不可判定的可以立即得到 $\Omega(\mathscr{M})$的字问题也是不可判定的. □

8.3.2 Post 对应问题

设字母表 A，A 上字符串有序对的有穷集合$\{(u_i, v_i) \mid u_i, v_i \in A^*, 1 \leqslant i \leqslant n\}$称作 **Post 对应系统**. 如果

$$w = u_{i_1} u_{i_2} \cdots u_{i_m} = v_{i_1} v_{i_2} \cdots v_{i_m},$$

则称 w 是这个系统的**解**，其中 $1 \leqslant i_1, i_2, \cdots, i_m \leqslant n$. 这里 $i_1, i_2, \cdots, i_m$ 中可以有重复.

Post 对应问题：任给一个 Post 对应系统，问它是否有解？

形象地说，Post 对应系统是一副骨牌，每张骨牌的上下半部各写一个 A 上的字符串. 玩的方法是把骨牌一张接一张地拼起来，使得上半部和下半部的字符串分别连起来之后得到相同的字符串. 在这里每一张牌可以重复使用多次(每一张牌有任意多枚复制品). 问题是，是否有这样的排列？

图 8.1 中(1)是一个 Post 对应系统；(2)是它的一个解.

(1) (2)

图 8.1 一个 Post 系统和它的解

定理 8.7 Post 对应问题是不可判定的.

证：设 Π 是一个半 Thue 过程，它的字问题是不可判定的. 定

理 8.6 表明确实存在这样的 Π，并且可以设 Π 不含形如 $p\to\epsilon$ 和 $\epsilon\to p$ 的产生式. 设 Π 的字母表为 $A=\{a_1,a_2,\cdots,a_n\}$. 又不妨设 Π 包含产生式 $a_i\to a_i(1\leqslant i\leqslant n)$（否则把这些产生式添加进来. 这样做不会影响任给两个字符串 $u,v\in A^*$，是否有 $u\overset{*}{\Rightarrow}v$). 这些产生式可以保证当 $u\overset{*}{\Rightarrow}v$ 时有奇长度的派生

$$u=u_1\Rightarrow u_2\Rightarrow\cdots\Rightarrow u_m=v,$$

其中 m 是奇数. 这是因为如果 m 是偶数，必有某个 $u_j\neq\epsilon$. 设 u_j 含有 a_i，利用 $a_i\to a_i$ 可得到 $u_j\Rightarrow u_j$，把它加入派生中就可得到所需的奇长度派生.

为了证明 Post 对应问题是不可判定的，我们把 Π 的字问题归约到 Post 对应问题. 任给 A 上的两个字符串 u 和 v，要构造一个 Post 对应系统 $P_{u,v}$ 使得

$$P_{u,v}\text{ 有解}\quad\text{当且仅当}\quad u\overset{*}{\Rightarrow}v.$$

构造 $P_{u,v}$ 的方法如下. 设 Π 的全部产生式为 $g_i\to h_i(1\leqslant i\leqslant k)$，其中 g_i 和 h_i 均不为空串 ϵ. $P_{u,v}$ 的字母表由 $2n+4$ 个符号组成：

$$[\ ,\ ]\ ,*\ ,\widetilde{*}\ ,a_i,\tilde{a}_i\quad(1\leqslant i\leqslant n).$$

$P_{u,v}$ 有 $2k+4$ 个有序对：

$$\left[\begin{matrix}[u*\\ [\end{matrix}\right]\quad\left[\begin{matrix}*\\ \widetilde{*}\end{matrix}\right]\quad\left[\begin{matrix}\widetilde{*}\\ *\end{matrix}\right]\quad\left[\begin{matrix}]\\ \widetilde{*}v]\end{matrix}\right]\quad\left[\begin{matrix}h_i\\ \tilde{g}_i\end{matrix}\right]\quad\left[\begin{matrix}\tilde{h}_i\\ g_i\end{matrix}\right]\qquad(1\leqslant i\leqslant k)$$

其中对于 $w\in A^*$，$\tilde{w}$ 是把 w 的每一个符号 a 换成 $\tilde{a}$ 所得到的字符串. 由于 Π 包含 $a_i\to a_i(1\leqslant i\leqslant n)$，故 $P_{u,v}$ 包含：

$$\left[\begin{matrix}a_i\\ \tilde{a}_i\end{matrix}\right]\quad\left[\begin{matrix}\widetilde{a}_i\\ a_i\end{matrix}\right]\qquad(1\leqslant i\leqslant n)$$

对任意的 $w\in A^*$ 且 $w\neq\epsilon$，总能用这些骨牌拼成：

$$\left[\begin{matrix}w\\ \tilde{w}\end{matrix}\right]\quad\text{和}\quad\left[\begin{matrix}\widetilde{w}\\ w\end{matrix}\right]$$

因此，下面可以使用这种形式的骨牌.

假设 $u\overset{*}{\Rightarrow}v$，则有派生

$$u = u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_m = v,$$

其中 m 是奇数. 对每一个 $i(1\leqslant i<m)$，设用 $g_{j_i}\to h_{j_i}$ 从 u_i 直接派生出 u_{i+1}，其中 $1\leqslant j_i\leqslant k$，于是存在 $r_i, s_i\in A^*$ 使得

$$u_i = r_i g_{j_i} s_i, \quad u_{i+1} = r_i h_{j_i} s_i.$$

注意到 m 是奇数，$r_i h_{j_i} s_i = r_{i+1} g_{j_{i+1}} s_{i+1}(1\leqslant i<m)$ 以及 $u = r_1 g_{j_1} s_1$，$v = r_{m-1} h_{j_{m-1}} s_{m-1}$，下述排列：

$\begin{matrix}[u *\\ [\end{matrix}$	$\begin{matrix}\tilde{r}_1\\ r_1\end{matrix}$	$\begin{matrix}\tilde{h}_{j_1}\\ g_{j_1}\end{matrix}$	$\begin{matrix}\tilde{s}_1\\ s_1\end{matrix}$	$\begin{matrix}\tilde{*}\\ *\end{matrix}$	$\begin{matrix}r_2\\ \tilde{r}_2\end{matrix}$	$\begin{matrix}h_{j_2}\\ \tilde{g}_{j_2}\end{matrix}$	$\begin{matrix}s_2\\ \tilde{s}_2\end{matrix}$	$\begin{matrix}*\\ \tilde{*}\end{matrix}$

$\cdots$

$\begin{matrix}\tilde{r}_{m-2}\\ r_{m-2}\end{matrix}$	$\begin{matrix}\tilde{h}_{j_{m-2}}\\ g_{j_{m-2}}\end{matrix}$	$\begin{matrix}\tilde{s}_{m-2}\\ s_{m-2}\end{matrix}$	$\begin{matrix}\tilde{*}\\ *\end{matrix}$	$\begin{matrix}r_{m-1}\\ \tilde{r}_{m-1}\end{matrix}$	$\begin{matrix}h_{j_{m-1}}\\ \tilde{g}_{j_{m-1}}\end{matrix}$	$\begin{matrix}s_{m-1}\\ \tilde{s}_{m-1}\end{matrix}$	$\begin{matrix}]\\ \tilde{*}v]\end{matrix}$

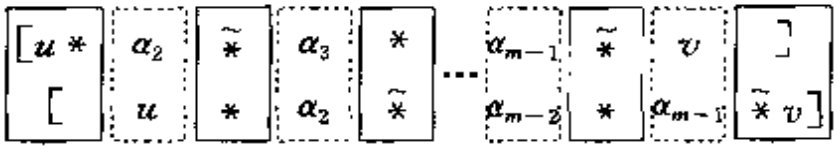
给出 $P_{u,v}$ 的解

$$w = [u_1 * \tilde{u}_2 \tilde{*}\ u_3 * \cdots * \tilde{u}_{m-1} \tilde{*}\ u_m].$$

在上面的排列中，$\begin{matrix}\tilde{\varepsilon}\\ \varepsilon\end{matrix}$ 和 $\begin{matrix}\varepsilon\\ \tilde{\varepsilon}\end{matrix}$ 都是“白板” $\begin{matrix}\\ \end{matrix}$，应该把它删去.

反之，假设 $P_{u,v}$ 有解 w，显然 $w\neq\varepsilon$. 由于 $P_{u,v}$ 中只有 $\begin{matrix}[u *\\ [\end{matrix}$ 和 $\begin{matrix}]\\ \tilde{*}v]\end{matrix}$ 的上下部字符串分别以相同的符号 [开头和以] 结束，故 w 必以 [开头、以] 结束. 若 $w = y[x]z$，其中 x 不含 [和]，由于同样的理由，$[x]$ 也是 $P_{u,v}$ 的解. 因此，我们可以设 $w=[x]$，其中 x 不含[和]. 得到 w 的骨牌排列必以 $\begin{matrix}[u *\\ [\end{matrix}$ 开头、以 $\begin{matrix}]\\ \tilde{*}v]\end{matrix}$ 结束，并且

中间不再出现这两张骨牌. 在上半部已出现一个 $*$，在下半部已出现一个 $\tilde{*}$. 在 $P_{u,v}$ 中除这两张骨牌外，只有 $\left[\begin{matrix}\tilde{*}\\ *\end{matrix}\right]$ 和 $\left[\begin{matrix}*\\ \tilde{*}\end{matrix}\right]$ 上有 $*$ 和 $\tilde{*}$，因此骨牌的排列必形如：



其中 m 是奇数，$\left[\begin{matrix}\\ \\ \end{matrix}\right]$ 由若干张骨牌拼成且不含 $\left[\begin{matrix}*\\ \tilde{*}\end{matrix}\right]$ 和 $\left[\begin{matrix}\tilde{*}\\ *\end{matrix}\right]$. 相应地，有

$$w = [u * \alpha_2 \tilde{*}\ \alpha_3 * \cdots * \alpha_{m-1} \tilde{*}\ v].$$

注意到 $P_{u,v}$ 中除 4 张含有 [,]，$*$ 和 $\tilde{*}$ 的特殊骨牌外，其余的骨牌均形如：

$$\left[\begin{matrix}\tilde{h}_i\\ g_i\end{matrix}\right] \quad 和 \quad \left[\begin{matrix}h_i\\ \tilde{g}_i\end{matrix}\right]$$

这里 $g_i \to h_i$ 是 Π 的产生式，所以

$$\alpha_2 = \tilde{u}_2 \quad 且 \quad u \overset{\cdot}{\Rightarrow} u_2,$$

$$\alpha_3 = u_3 \quad 且 \quad u_2 \overset{*}{\Rightarrow} u_3,$$

$$\vdots$$

$$\alpha_{m-1} = \tilde{u}_{m-1} \quad 且 \quad u_{m-1} \overset{\cdot}{\Rightarrow} v,$$

得证 $u \overset{*}{\Rightarrow} v$.

于是，我们得到

$$P_{u,v} \text{ 有解当且仅当 } \quad u \overset{\cdot}{\Rightarrow} v.$$

构造 $P_{u,v}$ 的方法显然是能行的，因此这就把 Π 的字问题归约到 Post 对应问题. □

8.4 有关文法的不可判定问题

设 Turing 机 $\mathscr{M}=(Q,A,C,\delta,s_0,q_1)$，$u\in A^*$. 文法 $G_{\mathscr{M},u}=(T,V,\Gamma,S)$定义如下：$T=\{a\}$，$V=C\cup Q\cup\{q_0,q_{n+1},h,S,X\}$，即 V 等于$\Sigma(\mathscr{M})$的字母表加两个新变元 S 和 X，其中 $a\notin V$. Γ 等于 $\Sigma(\mathscr{M})$加上产生式

$$S\to hq_1s_0uh,$$
$$hq_0h\to X,$$
$$X\to aX,$$
$$X\to a.$$

显然，

$$u\in L(\mathscr{M})\quad\Leftrightarrow\quad S\overset{*}{\underset{G_{\mathscr{M},u}}{\Longrightarrow}}X.$$

于是得到下述引理.

引理 8.8 如果$u\in L(\mathscr{M})$，则 $L(G_{\mathscr{M},u})=\{a^i\mid i>0\}$；如果 $u\notin L(\mathscr{M})$，则 $L(G_{\mathscr{M},u})=\varnothing$.

定理 8.9 下述问题是不可判定的：

(1) 任给一个文法 G，是否 $L(G)=\varnothing$？

(2) 任给一个文法 G，是否 $L(G)$是无穷的？

(3) 任给一个文法 G 和字符串 u，是否 $u\in L(G)$？

证：取一台 Turing 机 $\mathscr{M}$ 使得 $L(\mathscr{M})$是非递归的(这样的 $\mathscr{M}$ 确实存在). 由引理 8.8，

$$\begin{aligned}u\in L(\mathscr{M})\quad&\Leftrightarrow\quad L(G_{\mathscr{M},u})\neq\varnothing\\&\Leftrightarrow\quad L(G_{\mathscr{M},u})\text{ 是无穷的}\\&\Leftrightarrow\quad a\in L(G_{\mathscr{M},u}).\end{aligned}$$

由于“$u\in L(\mathscr{M})$?”是不可判定的，故定理中的 3 个问题都是不可判定的. □

8.5 一阶逻辑中的判定问题

考虑一阶逻辑中的3个判定问题：

公式可满足性问题：任给谓词演算形式系统 $K_{\mathscr{L}}$ 和公式 u，问 u 是否是可满足的？

公式永真性问题：任给谓词演算形式系统 $K_{\mathscr{L}}$ 和公式 u，问 u 是否是永真的？

公式可证性问题：任给谓词演算形式系统 $K_{\mathscr{L}}$、公式集 Γ 和公式 u，问是否 $\Gamma \underset{K_{\mathscr{L}}}{\vdash} u$？

在数理逻辑中知道，对于任意的谓词演算形式系统 $K_{\mathscr{L}}$、公式集 Γ 和公式 u，有

$$u \text{ 是可满足的} \quad \Leftrightarrow \quad \neg u \text{ 不是永真的},$$

$$\Gamma \underset{K_{\mathscr{L}}}{\vdash} u \quad \Leftrightarrow \quad \Gamma \models u.$$

因此，上述3个问题要么都是可判定的、要么都是不可判定的. D. Hilbert 曾称寻找这些问题的算法是“数理逻辑的主要问题”(1928年). 因为经验表明任何数学命题都可用数理逻辑的符号表达，每一条定理都可以表示成一阶逻辑的可证公式. 如果找到了解决上述问题的算法，那么任何定理的证明都可以化为运用这个算法的机械验证，从而使得数学家们创造性的定理证明成为多余的，至少在理论上是多余的. 但是上述问题都是不可判定的，因而根本不存在这样的算法.

设半 Thue 过程 Π，字母表 $A=\{a_1,a_2,\cdots,a_n\}$，产生式 $g_i \to h_i$ $(1\leqslant i\leqslant k)$，这里假设对所有的 $i(1\leqslant i\leqslant k)$，$g_i\neq\varepsilon$，$h_i\neq\varepsilon$. 构造谓词演算形式系统 K_{Π} 如下：

常量符号 $a_1,a_2,\cdots,a_n$.

二元函数变元符号 $\circ$，记 $x\circ y=\circ(x,y)$.

二元谓词变元符号 Q.

对于每一个 $w\in A^*-\{\varepsilon\}$，定义项 w^* 如下：

$$a_i^{\#}=a_i,$$

$$(ua_i)^{\#}=u^{\#}\circ a_i,\quad i=1,2,\cdots,n,u\in A^*-\{\varepsilon\}.$$

公式集 Γ 包括下述公式：

$(\gamma.1)\ \forall\ xQ(x,x)$，

$(\gamma.2)\ \forall\ x\forall\ y\forall\ z\ (Q(x,y)\wedge Q(y,z)\rightarrow Q(x,z))$，

$(\gamma.3)\ \forall\ x\forall\ y\forall\ r\forall\ s\ (Q(x,y)\wedge Q(r,s)\rightarrow Q(x\circ r,y\circ s))$，

$(\gamma.4)\ \forall\ x\forall\ y\forall\ z\ (Q((x\circ y)\circ z,x\circ(y\circ z)))$，

$(\gamma.5)\ \forall\ x\forall\ y\forall\ z\ (Q(x\circ(y\circ z),(x\circ y)\circ z))$，

$(\gamma.5+i)\ Q(g_i^{\#},h_i^{\#}),\quad i=1,2,\cdots,k.$

记 γ 为 Γ 中所有公式的合取.

解释 I 规定如下：论域为 $A^*-\{\varepsilon\}$.

$$a_i^I=a_i,\quad i=1,2,\cdots,n,$$

$$\circ^I(u,v)=uv,$$

$$Q^I(u,v)\quad \text{当且仅当}\quad u\overset{*}{\underset{\Pi}{\Rightarrow}}v.$$

对任意的 $u,v,w\in A^*-\{\varepsilon\}$，有下述性质：

性质1 $(w^{\#})^I=w.$

证：对 $|w|$ 作归纳证明，根据 $w^{\#}$ 和 I 的定义容易得到所需的结论. □

性质2 如果 $I\models\gamma\rightarrow Q(u^{\#},v^{\#})$，则 $u\overset{*}{\underset{\Pi}{\Rightarrow}}v.$

证：显然在解释 I 下 γ 为真，因此 $(Q(u^{\#},v^{\#}))^I$ 为真. 而

$$(Q(u^{\#},v^{\#}))^I=Q^I(u,v),$$

得证 $u\overset{*}{\underset{\Pi}{\Rightarrow}}v.$ □

性质3 如果 $w=uv$，则 $\Gamma\underset{K_E}{\vdash}Q(w^{\#},u^{\#}\circ v^{\#}).$

证：对 $|v|$ 作归纳证明. 当 $|v|=1$ 时，设 $v=a_i$，有 $w^{\#}=(ua_i)^{\#}=u^{\#}\circ a_i^{\#}$. 结论显然成立.

设当 $|v|=k(k\geqslant 1)$ 时结论成立. 考虑 $|v|=k+1$ 的情况，设 v

$=v_1a_i$，其中$|v_1|=k$. 于是，$w^{\#}=(uv)^{\#}=(uv_1)^{\#}\circ a_i^{\#}$，$v^{\#}=v_1^{\#}\circ a_i^{\#}$. 下面是由前提$\Gamma$推出$Q(w^{\#},u^{\#}\circ v^{\#})$的证明：

$Q(a_i^{\#},a_i^{\#})$ $(\gamma.1)$

$Q((uv_1)^{\#},u^{\#}\circ v_1^{\#})$ 归纳假设

$Q((uv_1)^{\#},u^{\#}\circ v_1^{\#})\wedge Q(a_i^{\#},a_i^{\#})\rightarrow Q((uv_1)^{\#}\circ a_i^{\#},(u^{\#}\circ v_1^{\#})\circ a_i^{\#})$ $(\gamma.3)$

$Q(w^{\#},(u^{\#}\circ v_1^{\#})\circ a_i^{\#})$

$Q((u^{\#}\circ v_1^{\#})\circ a_i^{\#},u^{\#}\circ(v_1^{\#}\circ a_i^{\#}))$ $(\gamma.4)$

$Q(w^{\#},(u^{\#}\circ v_1^{\#})\circ a_1^{\#})\wedge Q((u^{\#}\circ v_1^{\#})\circ a_i^{\#},u^{\#}\circ(v_1^{\#}\circ a_i^{\#}))\rightarrow Q(w^{\#},u^{\#}\circ(v_1^{\#}\circ a_i^{\#}))$ $(\gamma.2)$

$Q(w^{\#},u^{\#}\circ(v_1^{\#}\circ a_i^{\#}))$，即$Q(w^{\#},u^{\#}\circ v^{\#})$.

得证当$|v|=k+1$时结论也成立. □

性质4 如果$w=uv$，则$\Gamma\underset{K_{\Pi}}{\vdash}Q(u^{\#}\circ v^{\#},w^{\#})$.

证：与性质3的证明类似. □

性质5 如果$u\underset{\Pi}{\Rightarrow}v$，则$\Gamma\underset{K_{\Pi}}{\vdash}Q(u^{\#},v^{\#})$.

证：设$u=rg_is$，$v=rh_is$. 分情况讨论如下：

情况1 $r=s=\varepsilon$. 此时$Q(u^{\#},v^{\#})$就是$(\gamma.5+i)$.

情况2 $r=\varepsilon$，$s\neq\varepsilon$.

$Q(g_i^{\#},h_i^{\#})$ $(\gamma.5+i)$

$Q(s^{\#},s^{\#})$ $(\gamma.1)$

$Q(g_i^{\#},h_i^{\#})\wedge Q(s^{\#},s^{\#})\rightarrow Q(g_i^{\#}\circ s^{\#},h_i^{\#}\circ s^{\#})$ $(\gamma.3)$

$Q(g_i^{\#}\circ s^{\#},h_i^{\#}\circ s^{\#})$

$Q(u^{\#},g_i^{\#}\circ s^{\#})$ $u=g_is$ 及性质3

$Q(u^{\#},g_i^{\#}\circ s^{\#})\wedge Q(g_i^{\#}\circ s^{\#},h_i^{\#}\circ s^{\#})$

$\rightarrow Q(u^{\#},h_i^{\#}\circ s^{\#})$ $(\gamma.2)$

$Q(u^{\#},h_i^{\#}\circ s^{\#})$

$Q(h_i^{\#}\circ s^{\#},v^{\#})$ $v=h_is$ 及性质4

$$Q(u^{\#},h_i^{\#}\circ s^{\#})\wedge Q(h_i^{\#}\circ s^{\#},v^{\#})\to Q(u^{\#},v^{\#}) \qquad (\gamma.2)$$

$Q(u^{\#},v^{\#})$

情况3　$r\neq\varepsilon,s=\varepsilon$,与情况2类似可证.

情况4　$r\neq\varepsilon,s\neq\varepsilon$,由性质3,有

$$Q(u^{\#},(rg_i)^{\#}\circ s^{\#}).$$

由情况3,有

$$Q((rg_i)^{\#},(rh_i)^{\#}).$$

可推出

$$Q((rg_i)^{\#}\circ s^{\#},(rh_i)^{\#}\circ s^{\#}),$$

$$Q(u^{\#},(rh_i)^{\#}\circ s^{\#}).$$

由性质4,有

$$Q((rh_i)^{\#}\circ s^{\#},v^{\#}).$$

推得

$$Q(u^{\#},v^{\#}).$$ □

性质6　如果 $u\overset{*}{\underset{\Pi}{\Rightarrow}}v$,则 $\Gamma\underset{K_\Pi}{\vdash}Q(u^{\#},v^{\#})$.

证:利用性质5,对从 u 到 v 的派生长度作归纳证明. □

性质7　$u\overset{*}{\underset{\Pi}{\Rightarrow}}v$ 当且仅当 $\models\gamma\to Q(u^{\#},v^{\#})$.

证:首先,

$u\overset{*}{\underset{\Pi}{\Rightarrow}}v$　蕴涵　$\Gamma\underset{K_\Pi}{\vdash}Q(u^{\#},v^{\#})$　性质6

蕴涵　$\Gamma\models Q(u^{\#},v^{\#})$

蕴涵　$\models\gamma\to Q(u^{\#},v^{\#})$.

其次,

$\Gamma\underset{K_\Pi}{\vdash}Q(u^{\#},v^{\#})$　蕴涵　$\models\gamma\to Q(u^{\#},v^{\#})$

蕴涵　$I\models\gamma\to Q(u^{\#},v^{\#})$

蕴涵　$u\overset{*}{\underset{\Pi}{\Rightarrow}}v$.　性质2 □

定理8.10　存在谓词演算形式系统 $K_\mathscr{L}$ 使得它的公式永真性

问题"任给公式 u，是否$\models u$?"是不可判定的.

证：取半 Thue 过程 Π 使得它的字问题是不可判定的，并且不包含形如 $g\to\varepsilon$ 和 $\varepsilon\to h$ 的产生式. 定理8.6表明确实存在这样的半 Thue 过程 Π. 根据性质7，可以把 Π 的字问题归约到 K_Π 的公式永真性问题，从而得证 K_Π 的公式永真性问题是不可判定的. □

由上述定理立即得到本节开头提出的3个问题的不可判定性.

定理8.11 在一阶逻辑中，公式可满足性问题、公式永真性问题、公式可证性问题都是不可判定的.

习　题

1. 证明存在 TM$\mathscr{M}$ 使得下述问题是不可解的：任给一个格局 σ，$\mathscr{M}$ 从 σ 开始最终是否会以完全空白的带停机?

2. 证明存在以$\{s_1,s_2,B\}$为带字母表的 TM$\mathscr{M}$ 使得下述问题是不可解的：任给一个格局 σ，$\mathscr{M}$ 从 σ 开始是否会打印 s_2?

3. 设 Π 是字母表 A 上的半 Thue 过程，$u_0\in A^*$. (u_0,Π)称作**半 Thue 系统**. 又设 $w\in A^*$. 如果 $u_0\overset{*}{\underset{\Pi}{\Rightarrow}}w$，则称 w 是(u_0,Π)的**定理**.

证明存在半 Thue 系统(u_0,Π)使得下述问题是不可解的：任给 $w\in A^*$，问 w 是否是(u_0,Π)的定理?

4. 证明：仅包含一个产生式的半 Thue 过程的字问题是可解的.

5. 给出字母表仅含一个符号的 Post 对应问题的算法.

6. A^*对连接运算构成半群，称作字母表 A 上的**自由半群**. 证明下述问题是不可解的：

任给两个自由半群 A_1^* 和 A_2^* 及从 A_1^* 到 A_2^* 的同态映射 ϕ_1和 ϕ_2，问是否存在 $x\in A_1^*$ 使得 $\phi_1(x)=\phi_2(x)$?

7. 证明下述关于文法的问题是不可解的：

(1) 任给一对文法 G_1和 G_2，问是否 $L(G_1)\subseteq L(G_2)$?

(2) 任给一对文法 G_1和 G_2，问是否 $L(G_1)=L(G_2)$?

第九章　正则语言

9.1　Chomsky 谱系

本章和下面两章讨论文法及其生成的语言. 和人们在日常交往中使用的文法和语言不同，我们研究的是形式文法和形式语言.

语言学家 N. Chomsky 根据对产生式附加的限制条件，把文法分成 4 类：0 型文法、1 型文法、2 型文法和 3 型文法. 现称作 Chomsky 谱系.

设文法 $G=(V,T,\Gamma,S)$. 约定：今后变元用大写字母 A,B,C,X,Y,Z 等表示，S 表示起始符，终极符用小写字母 a,b,c 等表示，终极符串用 x,y,z,u,v,w 等表示，变元和终极符的字符串用小写希腊字母 $\alpha,\beta,\gamma,\omega$ 等表示. 当然还会使用带下标的这些字母.

0 型文法就是一般的文法，对产生式不附加任何条件. 0 型文法又叫做短词结构文法或无限制文法，简称文法. 0 型文法生成的语言称作 **0 型语言**. 0 型语言就是 r.e. 语言.

定义 9.1　如果文法 G 的每一个产生式 $\alpha\to\beta$ 都满足条件

$$|\alpha|\leqslant|\beta|,$$

则称 G 是 **1 型文法**. 1 型文法又叫做**上下文有关文法**，简记作 CSG. 如果存在 1 型文法 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$，则称 L 是 **1 型语言**. 1 型语言又叫做**上下文有关语言**，简记作 CSL.

定义 9.2　如果文法 G 的每一个产生式都形如

$$A\to\alpha,$$

其中 $A\in V,\alpha\in(T\cup V)^*$，则称 G 是 **2 型文法**. 2 型文法又叫做**上下文无关文法**，简记作 CFG. 2 型文法生成的语言称作 **2 型语言**. 2 型语言又叫做**上下文无关语言**，简记作 CFL.

定义 9.3 如果文法 G 的每一个产生式都形如

$$A \to wB \quad 或 \quad A \to w,$$

其中 $A,B \in V, w \in T^*$，则称 G 是**右线性文法**. 如果每一个产生式都形如

$$A \to Bw \quad 或 \quad A \to w,$$

其中 $A,B \in V, w \in T^*$，则称 G 是**左线性文法**. 左线性文法和右线性文法统称作 **3 型文法**. 3 型文法又叫做**正则文法**. 3 型文法生成的语言称作 **3 型语言**. 3 型语言又叫做**正则语言**.

在上述定义中，对 1 型语言的定义做了特殊处理. 这是因为任何 1 型文法都不会生成空串 ε，即对任何 1 型文法 G 有 $\varepsilon \notin L(G)$，而其他型的文法生成的语言都可能包含 ε. 经过这样处理之后，1 型语言也和其他型语言一样，不把 ε 排除在外.

[**例 9.1**] 文法 $G: S \to aA, A \to aA, A \to bB, B \to bB, A \to b, B \to b$.

这是右线性文法，它生成语言

$$L = \{a^n b^m \mid n,m > 0\}.$$

[**例 9.2**] 文法 $G: S \to aSb, S \to ab$.

这是上下文无关文法，它生成语言

$$L = \{a^n b^n \mid n > 0\}.$$

[**例 9.3**] 文法 G:

① $S \to aSBC$，　② $S \to aBC$，

③ $CB \to BC$，　④ $aB \to ab$，

⑤ $bB \to bb$，　⑥ $bC \to bc$，

⑦ $cC \to cc$.

这是上下文有关文法. 下面证明它生成语言

$$L = \{a^n b^n c^n \mid n > 0\}.$$

首先，对任意的 $n>0$，

$$S \overset{*}{\Rightarrow} a^{n-1} S (BC)^{n-1} \qquad n-1 \text{ 次 } ①$$

$$\Rightarrow a^n(BC)^n \qquad ②$$

$$\overset{*}{\Rightarrow} a^nB^nC^n \qquad \frac{1}{2}n(n-1)\text{ 次 } ③$$

$$\Rightarrow a^nbB^{n-1}C^n \qquad ④$$

$$\overset{*}{\Rightarrow} a^nb^nC^n \qquad n-1\text{ 次 } ⑤$$

$$\Rightarrow a^nb^ncC^{n-1} \qquad ⑥$$

$$\overset{*}{\Rightarrow} a^nb^nc^n \qquad n-1\text{ 次 } ⑦$$

得证 $L\subseteq L(G)$.

其次，从 S 开始只能用①和②得到含 a,B,C 的字符串，且字符串中这 3 个字母的个数相同. 而 b 和 c 分别只能用 B 和 C 换取，因此 $L(G)$的字符串中 a,b,c 的个数相同. 又 a 始终出现在其他符号的左边，B 只有紧挨在 a 或 b 的右边时才能被替换成 b，C 只有紧挨在 b 或 c 的右边时才能被替换成 c，因此 b 必在 a 的右边，c 又必在 b 的右边. 得证 $L(G)\subseteq L$.

定理 9.1 对每一个右线性文法 G，都存在左线性文法 G' 使得 $L(G)=L(G')$；反之亦然.

证：设右线性文法 $G=(V,T,\Gamma,S)$，如下构造左线性文法 $G'=(V\cup\{S'\},T,\Gamma',S')$，其中 $S'\notin T\cup V$ 是添加的新变元，作为 G' 的起始符.

$$\Gamma'=\{B\to Aw\mid A\to wB\}$$
$$\cup\{S'\to Aw\mid A\to w\}\cup\{S\to\varepsilon\}.$$

要证 $L(G)=L(G')$. 设 $w\in L(G)$，有两种可能：

(1) G 有产生式 $S\to w$，则 G' 有产生式 $S'\to Sw$ 和 $S\to\varepsilon$. 于是，

$$S'\underset{G'}{\Rightarrow}Sw\underset{G'}{\Rightarrow}w,$$

得 $w\in L(G')$.

(2) 对某个 $n\geqslant 1$，有派生

$$S\underset{G}{\Rightarrow}u_1A_1\underset{G}{\Rightarrow}u_1u_2A_2\underset{G}{\Rightarrow}\cdots$$
$$\underset{G}{\Rightarrow}u_1\cdots u_nA_n\underset{G}{\Rightarrow}u_1\cdots u_nu_{n+1}=w,$$

其中 $S\to u_1A_1$，$A_i\to u_{i+1}A_{i+1}(1\leqslant i<n)$和 $A_n\to u_{n+1}$均是 G 的产生式. 相应地，G'有产生式 $A_1\to Su_1$，$A_{i+1}\to A_iu_{i+1}(1\leqslant i<n)$，$S'\to A_nu_{n+1}$及 $S\to\varepsilon$. 于是，

$$S'\underset{G'}{\Rightarrow}A_nu_{n+1}\underset{G'}{\Rightarrow}A_{n-1}u_nu_{n+1}\underset{G'}{\Rightarrow}\cdots$$

$$\underset{G'}{\Rightarrow}A_1u_2\cdots u_{n+1}\underset{G}{\Rightarrow}Su_1\cdots u_{n+1}\underset{G}{\Rightarrow}w,$$

也得 $w\in L(G')$.

反之，设 $w\in L(G')$，注意到 $S\to\varepsilon$ 是 G' 中唯一右端不含变元的产生式，必有派生

$$S'\underset{G'}{\Rightarrow}A_1v_1\underset{G'}{\Rightarrow}A_2v_2v_1\underset{G'}{\Rightarrow}\cdots$$

$$\underset{G'}{\Rightarrow}A_nv_n\cdots v_1\underset{G}{\Rightarrow}v_n\cdots v_1=w,$$

其中 $n\geqslant 1$，$A_n=S$，G'有产生式 $S'\to A_1v_1$，$A_i\to A_{i+1}v_{i+1}(1\leqslant i<n)$及 $S\to\varepsilon$. 相应地，G 有产生式 $A_1\to v_1$，$A_{i+1}\to v_{i+1}A_i(1\leqslant i<n)$. 于是，

$$S=A_n\underset{G}{\Rightarrow}v_nA_{n-1}\underset{G}{\Rightarrow}\cdots\underset{G}{\Rightarrow}v_n\cdots v_2A_1\underset{G}{\Rightarrow}v_n\cdots v_2v_1=w,$$

得 $w\in L(G)$. 这就证明了 $L(G)=L(G')$.

类似可证，对每一个左线性文法 G，都存在右线性文法 G' 使得 $L(G)=L(G')$. □

推论 9.2 每一个正则语言既可以用右线性文法生成，也可以用左线性文法生成.

在第七章已经证明 0 型文法等价于 Turing 机，即 0 型文法生成的语言类和 Turing 机接受的语言类相同，都是递归可枚举语言. 下面将会看到，1 型文法、2 型文法和 3 型文法也都分别有等价的自动机模型，它们是线性界限自动机，下推自动机和有穷自动机.

9.2 有穷自动机

有穷自动机是一种能力很有限的计算装置，设想它有一条带，

用来存放输入字符串. 控制器有有穷个状态. 带头只能读、不能写，并且只能往一个方向(例如，从左向右)运动，因而它只能顺序检查字符串的每一个符号. 在每一步，根据当前所处的状态和带头读到的符号转移到另一个状态. 最后，根据读完字符串时机器所处的状态决定是否接受这个字符串. 见图 9.1.

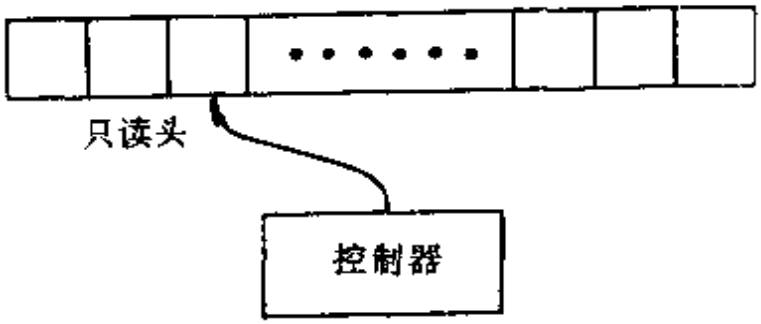


图 9.1 FA 示意图

定义 9.4 一台**有穷自动机**(缩写作 FA)$\mathscr{M}$ 由 5 部分组成：

(1) 字母表 A；

(2) 状态集 Q，Q 是非空有穷集合；

(3) 转移函数 $\delta: Q\times A\to Q$；

(4) 初始状态 $q_1\in Q$；

(5) 接受状态集 $F\subseteq Q$；

记作 $\mathscr{M}=(Q,A,\delta,q_1,F)$.

[**例 9.4**] 设 FA $\mathscr{M}=(Q,A,\delta,q,F)$ 由表 9-1 给出，给 q_3 加一个圈表示它是接受状态. 和 TM 类似，也可以用有向图表示一台 FA，称作 FA 的**状态转移图**. 在弧 (q,q') 旁标 a 表示 $\delta(q,a)=q'$. 双圈的节点表示接受状态. $\mathscr{M}$ 的状态转移图如图 9.2 所示.

表 9-1 FA $\mathscr{M}$

Q \ δ \ A	a	b
q_1	q_2	q_4
q_2	q_2	q_3
ⓠ$_3$	q_4	q_2
q_4	q_4	q_4

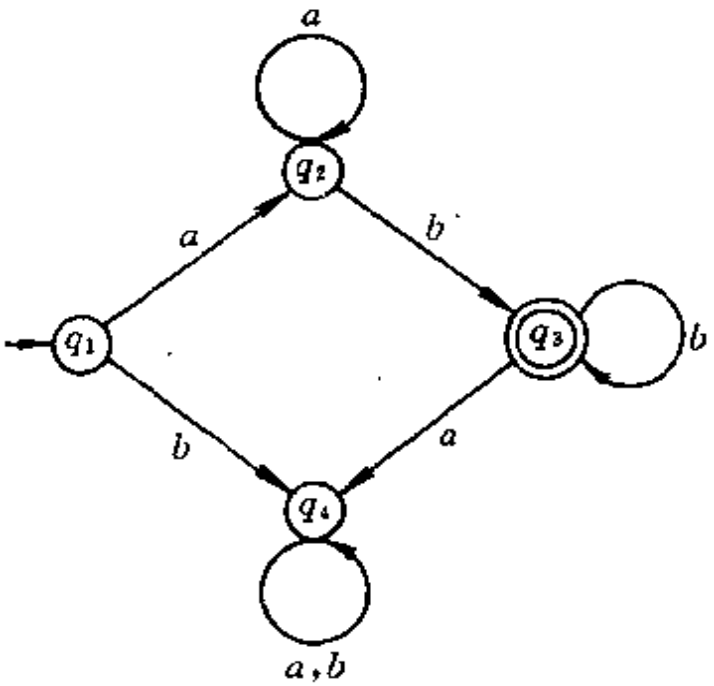


图 9.2　FA $\mathscr{M}$ 的状态转移图

给定字符串 $w\in A^*$，FA $\mathscr{M}$ 计算开始时处于初始状态，只读头指向 w 的第一个符号. 在计算的每一步，设 $\mathscr{M}$ 处于状态 q，只读头指向符号 a，则 $\mathscr{M}$ 转移到状态 $\delta(q,a)$，同时只读头右移一格. 直至读完(只读头移出)w 为止. 若读完 w 时 $\mathscr{M}$ 处于接受状态，则 $\mathscr{M}$ 接受 w；否则 $\mathscr{M}$ 拒绝 w. $\mathscr{M}$ 接受的字符串的全体称作 $\mathscr{M}$ 接受的语言，记作 $L(\mathscr{M})$.

为了更形象化地描述 $L(\mathscr{M})$，把 δ 扩张到 $Q\times A^*$ 上. 任给 $q\in Q$ 和 $u\in A^*$，$\mathscr{M}$ 从状态 q 开始，检查完 u 的每一个符号后所处的状态记作 $\delta^*(q,u)$. δ^* 递归地定义如下：对所有的 $q\in Q$，$u\in A^*$ 和 $a\in A$，

$$\delta^*(q,\varepsilon)=q,$$

$$\delta^*(q,ua)=\delta(\delta^*(q,u),a).$$

定义 9.5　设 FA $\mathscr{M}=(Q,A,\delta,q,F)$，$w\in A^*$. 如果 $\delta^*(q_1,w)\in F$，则称 $\mathscr{M}$ **接受** w；否则称 $\mathscr{M}$ **拒绝** w 或**不接受** w.

$\mathscr{M}$ 接受的语言

$$L(\mathscr{M})=\{w\in A^*\mid \mathscr{M}\text{ 接受 }w\}.$$

不难看出，例 9.4 中的 FA $\mathscr{M}$ 接受语言

$$L(\mathscr{M}) = \{a^n b^m \mid n, m > 0\}.$$

在定义 9.4 中，转移函数 δ 是从 $Q \times A$ 到 Q 的函数，$\mathscr{M}$ 的每一步是完全确定的. 这种有穷自动机是确定型的. 与 Turing 机类似，也有非确定型有穷自动机，它的动作是不确定的，定义如下.

定义 9.6 一台**非确定型有穷自动机**(缩写作 NFA) $\mathscr{M}$ 由 5 部分组成，记作 $\mathscr{M} = (Q, A, \delta, q_1, F)$，其中 Q, A, q_1 和 F 与定义 9.4中的相同，δ 是 $Q \times A$ 到 Q 的二元关系.

当需要强调确定型时，把确定型有穷自动机缩写成 DFA. *DFA* 是 NFA 的特殊情况. 今后用 FA 泛指有穷自动机.

NFA $\mathscr{M}$ 在计算的每一步，如果 $\mathscr{M}$ 当前处于状态 q，只读头指向符号 a，则当 $\delta(q,a) \neq \varnothing$ 时 $\mathscr{M}$ 可以转移到 $\delta(q,a)$ 中的任一状态；当 $\delta(q,a) = \varnothing$ 时 $\mathscr{M}$ 停止计算. 给定字符串 $w \in A^*$，从初始状态开始，$\mathscr{M}$ 读完 w 时可能处于若干状态. 当且仅当这些状态中有接受状态时 $\mathscr{M}$ 接受 w.

把 δ 扩张成 $Q \times A^*$ 到 Q 的二元关系 δ^*：$((q,u),q') \in \delta^*$ 当且仅当从状态 q 开始，读完 u 时 $\mathscr{M}$ 可能处于状态 q'.

递归地定义 δ^* 如下：对每一个 $q \in Q, u \in A^*$ 和 $a \in A$，

$$\delta^*(q,\varepsilon) = \{q\},$$

$$\delta^*(q,ua) = \bigcup_{p \in \delta^*(q,u)} \delta(p,a).$$

定义 9.7 设 NFA $\mathscr{M} = (Q, A, \delta, q_1, F)$，$w \in A^*$. 如果 $\delta^*(q_1,w) \cap F \neq \varnothing$，则称 $\mathscr{M}$ **接受** w；否则称 $\mathscr{M}$ **拒绝** w 或**不接受** w.

$\mathscr{M}$ 接受的语言

$$L(\mathscr{M}) = \{w \in A^* \mid \mathscr{M} \text{ 接受 } w\}.$$

[**例 9.5**] 设 NFA $\mathscr{M}$ 由表 9-2 给出，它的状态转移图如图 9.3 所示. 不难看出，它接受语言

$$L(\mathscr{M}) = \{x00 \mid x \in \{0,1\}^*\}.$$

如果把{0,1}上的字符串看作普通的二进制数，$\mathscr{M}$ 恰好接受所有可以被 4 整除的数.

表 9-2　NFA $\mathscr{M}$

Q \ δ \ A	0	1
q_1	$\{q_1,q_2\}$	$\{q_1\}$
q_2	$\{q_3\}$	$\varnothing$
ⓠ$_3$	$\varnothing$	$\varnothing$

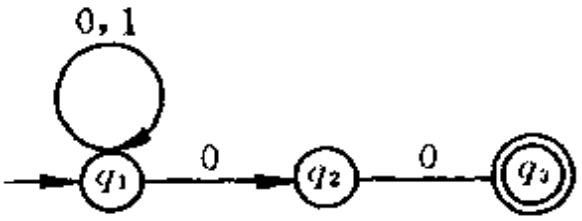


图 9.3　NFA $\mathscr{M}$ 的状态转移图

定理 9.3　语言 L 被 DFA 接受当且仅当它被 NFA 接受.

证：DFA 是 NFA 的特殊情况，因此若 L 被 DFA 接受，当然也被 NFA 接受.

设 NFA $\mathscr{M}=(Q,A,\delta,q_1,F)$，$L=L(\mathscr{M})$. 构造 DFA $\tilde{\mathscr{M}}=(\tilde{Q},\tilde{A},\tilde{\delta},Q_1,\tilde{F})$ 如下：$\tilde{Q}=P(Q)$，$\tilde{A}=A$，$Q_1=\{q_1\}$，

$$\tilde{F}=\{Q' \mid Q'\subseteq Q \wedge Q'\cap F\neq\varnothing\},$$

$$\tilde{\delta}(Q',a)=\bigcup_{q\in Q'}\delta(q,a),$$

其中 $P(Q)$ 是 Q 的幂集.

先证对任意的 $u\in A^*$，

$$\tilde{\delta}^*(Q_1,u)=\delta^*(q_1,u). \qquad (\#)$$

对 $|u|$ 进行归纳证明. 当 $|u|=0$ 时，$u=\varepsilon$，

$$\tilde{\delta}^*(Q_1,\varepsilon)=Q_1=\{q_1\}=\delta^*(q_1,\varepsilon),$$

(#)成立. 设当 $|u|=l(l\geqslant 0)$ 时(#)成立. 考虑 $|u|=l+1$ 的情况，设 $u=va$，其中 $|v|=l$，$a\in A$. 于是

$$\tilde{\delta}^*(Q_1,u)=\tilde{\delta}^*(Q_1,va)$$

$$
\begin{aligned}
&=\tilde{\delta}(\tilde{\delta}^*(Q_1,v),a) && \tilde{\delta}^* \text{ 的定义}\\
&=\tilde{\delta}(\delta^*(q_1,v),a) && \text{归纳假设}\\
&=\bigcup_{q\in\delta^*(q_1,v)}\delta(q,a) && \tilde{\delta} \text{ 的定义}\\
&=\delta^*(q_1,va) && \delta^* \text{ 的定义}\\
&=\delta^*(q_1,u).
\end{aligned}
$$

得证(#)成立.

于是,

$$
\begin{aligned}
u\in L(\tilde{\mathscr{M}}) &\Leftrightarrow \tilde{\delta}^*(Q_1,u)\in\tilde{F}\\
&\Leftrightarrow \delta^*(q_1,u)\in\tilde{F}\\
&\Leftrightarrow \delta^*(q_1,u)\cap F\neq\varnothing\\
&\Leftrightarrow u\in L(\mathscr{M}),
\end{aligned}
$$

得证 $L(\tilde{\mathscr{M}})=L(\mathscr{M})$. □

定理 9.4 有穷自动机接受的语言是正则语言.

证:由定理 9.3,只需考虑 DFA. 设 DFA $\mathscr{M}=(Q,A,\delta,q_1,F)$,构造右线性文法 $G=(Q,A,\Gamma,q_1)$,其中 Γ 包括下述产生式:

对每一个 $q\in Q$ 和 $a\in A$,

如果 $\delta(q,a)=q'$,则有产生式 $q\to aq'$,

如果 $q\in F$,则有产生式 $q\to\varepsilon$.

对任意的 $w\in A^*$,分两种情况讨论:

(1) $w=\varepsilon$.

$$
\begin{aligned}
\varepsilon\in L(\mathscr{M}) &\Leftrightarrow q_1\in F\\
&\Leftrightarrow G \text{ 有产生式 } q_1\to\varepsilon\\
&\Leftrightarrow \varepsilon\in L(G).
\end{aligned}
$$

(2) $w\neq\varepsilon$. 设 $w=a_1a_2\cdots a_k$,其中 $a_1,a_2,\cdots,a_k\in A$,$k\geqslant 1$.

若 $w\in L(\mathscr{M})$,设 $\delta(q_1,a_1)=q_{i_1}$,$\delta(q_{i_1},a_2)=q_{i_2}$,$\cdots$,$\delta(q_{i_{k-1}},a_k)=q_{i_k}$,则 $q_{i_k}\in F$. 于是 G 有产生式 $q_1\to a_1q_{i_1}$,$q_{i_1}\to a_2q_{i_2}$,$\cdots$,$q_{i_{k-1}}\to a_kq_{i_k}$及 $q_{i_k}\to\varepsilon$,从而有

$$
q_1\underset{G}{\Rightarrow}a_1q_{i_1}\underset{G}{\Rightarrow}a_1a_2q_{i_2}\underset{G}{\Rightarrow}\cdots
$$

$$\underset{G}{\Rightarrow}a_1\cdots a_kq_{i_k}\underset{G}{\Rightarrow}a_1\cdots a_k=w,$$

得 $w\in L(G)$.

反之，若 $w\in L(G)$，根据 G 的构造，必有产生式 $q_1\to a_1q_{i_1}$，$q_{i_1}\to a_2q_{i_2}$，$\cdots$，$q_{i_{k-1}}\to a_kq_{i_k}$及 $q_{i_k}\to\varepsilon$ 使

$$q_1\underset{G}{\Rightarrow}a_1q_{i_1}\underset{G}{\Rightarrow}a_1a_2q_{i_2}\underset{G}{\Rightarrow}\cdots$$

$$\underset{G}{\Rightarrow}a_1\cdots a_kq_{i_k}\underset{G}{\Rightarrow}a_1\cdots a_k=w.$$

于是，$\delta(q_1,a_1)=q_{i_1}$，$\delta(q_{i_1},a_2)=q_{i_2}$，$\cdots$，$\delta(q_{i_{k-1}},a_k)=q_{i_k}$. 以及 $q_{i_k}\in F$. 从而，$\delta^*(q_1,w)=q_{i_k}\in F$. 得证 $w\in L(\mathscr{M})$.

所以，$L(\mathscr{M})=L(G)$. □

后面将会看到，任何正则语言都被 FA 接受，从而 FA 接受的语言类恰好是正则语言类.

9.3 封 闭 性

定义 9.8 设字母表 A，L_1，L_2，$L\subseteq A^*$. 记

$$L_1\cdot L_2=\{uv\mid u\in L_1\text{ 且 }v\in L_2\},$$

称作 L_1 与 L_2 的**连结**. $L_1\cdot L_2$ 可简写作 L_1L_2. 又记

$$L^0=\{\varepsilon\},$$

$$L^i=L\cdot L^{i-1},(i\geqslant 1)$$

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

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

L^* 称作 L 的 Kleene **闭包**，简称**闭包**. L^+ 称作 L 的**正闭包**.

显然，

$$L^*=\{u_1u_2\cdots u_n\mid n\geqslant 0,u_1,u_2,\cdots,u_n\in L\},$$

$$L^+=\{u_1u_2\cdots u_n\mid n\geqslant 1,u_1,u_2,\cdots,u_n\in L\}.$$

定义 9.9 设 DFA $\mathscr{M}=(Q,A,\delta,q_1,F)$，如果不存在 $q\in Q$ 和 $a\in A$ 使得 $\delta(q,a)=q_1$，则称 $\mathscr{M}$ 是**非重新启动的**.

引理 9.5 设 $\mathscr{M}$ 是一台 DFA，则存在非重新启动的 DFA $\tilde{\mathscr{M}}$ 使得 $L(\mathscr{M})=L(\tilde{\mathscr{M}})$.

证：设 $\mathscr{M}=(Q,A,\delta,q_1,F)$，添加一个新的状态 q_0 作为 $\tilde{\mathscr{M}}$ 的初始状态，并对所有的 $q\in Q$ 和 $a\in A$，令

$$\tilde{\delta}(q,a)=\delta(q,a),$$
$$\tilde{\delta}(q_0,a)=\delta(q_1,a).$$

令

$$\tilde{F}=\begin{cases}F\cup\{q_0\} & \text{若 } q_1\in F\\ F & \text{否则}\end{cases}$$

这样构造出来的 DFA $\tilde{\mathscr{M}}=(Q\cup\{q_0\},A,\tilde{\delta},q_0,\tilde{F})$ 是非重新启动的. 除了第一步之外，$\tilde{\mathscr{M}}$ 和 $\mathscr{M}$ 的计算完全相同. 检查第一步的各种可能（输入串 $u=\varepsilon$ 和 $u\neq\varepsilon$，$q_1\in F$ 和 $q_1\notin F$），不难证明 $L(\tilde{\mathscr{M}})=L(\mathscr{M})$. □

引理 9.6 设 $A\subseteq B$，$L\subseteq A^*$. 如果存在以 A 为字母表的 DFA 接受 L，则存在以 B 为字母表的 DFA 接受 L.

证：设 DFA $\mathscr{M}=(Q,A,\delta,q_1,F)$ 使得 $L=L(\mathscr{M})$. 添加一个新的状态 q^*，接受状态集和初始状态不变，把 δ 扩张到 $(Q\cup\{q^*\})\times B$ 上：

$$\tilde{\delta}(q,b)=\begin{cases}\delta(q,b) & \text{若 } q\in Q \text{ 且 } b\in A\\ q^* & \text{若 } q=q^* \text{ 或 } b\in B-A,\end{cases}$$

得到 DFA $\tilde{\mathscr{M}}=(Q\cup\{q^*\},B,\tilde{\delta},q_1,F)$.

当 $w\in A^*$ 时，$\tilde{\mathscr{M}}$ 关于 w 的计算和 $\mathscr{M}$ 的完全一样；当 w 中含有 $B-A$ 中的符号时，$\tilde{\mathscr{M}}$ 将进入状态 q^*，并且永远保持在这个状态上，$q^*\notin F$，$\tilde{\mathscr{M}}$ 不接受 w. 所以，$L(\tilde{\mathscr{M}})=L(\mathscr{M})$. □

定理 9.7 设 L_1,L_2,L 是被有穷自动机接受的语言，则下述语言均被有穷自动机接受：

(1) $L_1\cup L_2$；

(2) A^*-L，其中 A 是 L 的字母表；

(3) $L_1 \cap L_2$;

(4) $L_1 \cdot L_2$;

(5) L^*.

证:根据引理 9.6,不妨假设下面所论及的语言都是字母表 A 上的语言.

(1) 根据引理 9.5,可设非重新启动的 DFA $\mathscr{M}_1=(Q_1,A,\delta_1,q_1,F_1)$ 和 $\mathscr{M}_2=(Q_2,A,\delta_2,q_2,F_2)$ 分别接受 L_1 和 L_2,这里不妨设 $Q_1 \cap Q_2=\varnothing$. 构造 NFA $\tilde{\mathscr{M}}=(\tilde{Q},A,\tilde{\delta},\tilde{q},\tilde{F})$,其中

$$\tilde{Q}=Q_1 \cup Q_2 \cup \{\tilde{q}\}-\{q_1,q_2\},\text{这里 } \tilde{q} \notin Q_1 \cup Q_2,$$

$$\tilde{F}=\begin{cases} F_1 \cup F_2 \cup \{\tilde{q}\}-\{q_1,q_2\}, & \text{若 } q_1 \in F_1 \text{ 或 } q_2 \in F_2 \\ F_1 \cup F_2, & \text{否则} \end{cases}$$

$$\tilde{\delta}(q,a)=\begin{cases} \{\delta_1(q,a)\} & \text{若 } q \in Q_1-\{q_1\} \\ \{\delta_2(q,a)\} & \text{若 } q \in Q_2-\{q_2\} \\ \{\delta_1(q_1,a),\delta_2(q_2,a)\} & \text{若 } q=\tilde{q}. \end{cases}$$

当输入串 $w \neq \varepsilon$ 时,注意到 $\mathscr{M}_1$ 和 $\mathscr{M}_2$ 是非重新启动的并且 $Q_1 \cap Q_2=\varnothing$,$\tilde{\mathscr{M}}$ 的第一步或者按照 $\mathscr{M}_1$ 的方式进入 $\mathscr{M}_1$ 的状态;或者按照 $\mathscr{M}_2$ 的方式进入 $\mathscr{M}_2$ 的状态. 如果进入 $\mathscr{M}_1$ 的状态,则以后的计算完全按照 $\mathscr{M}_1$ 的方式进行;同样地,如果进入 $\mathscr{M}_2$ 的状态,则以后的计算完全按照 $\mathscr{M}_2$ 的方式进行. 如图 9.4 所示.

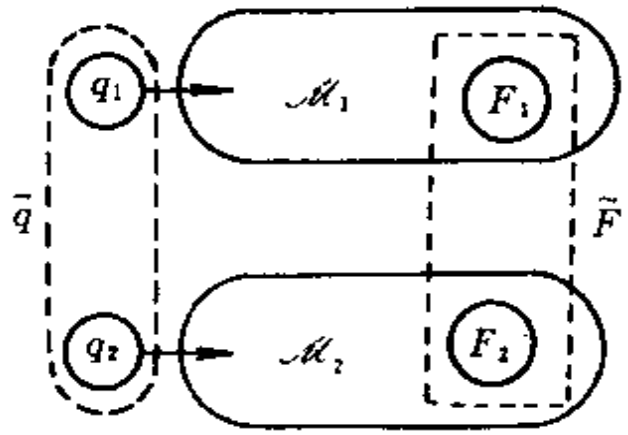


图 9.4

所以，$\tilde{\mathscr{M}}$接受w当且仅当$\mathscr{M}_1$接受w或者$\mathscr{M}_2$接受w. 当输入串$w=\varepsilon$时，根据$\tilde{F}$的定义同样有$\tilde{\mathscr{M}}$接受ε当且仅当$\mathscr{M}_1$接受ε或者$\mathscr{M}_2$接受ε. 得证$L(\tilde{\mathscr{M}})=L_1\cup L_2$.

(2) 设DFA $\mathscr{M}=(Q,A,\delta,q_1,F)$接受$L$，则$\tilde{\mathscr{M}}=(Q,A,\delta,q_1,Q-F)$接受$A^*-L$.

(3) $L_1\cap L_2=A^*-((A^*-L_1)\cup(A^*-L_2))$，根据(1)和(2)可证$L_1\cap L_2$被FA接受.

(4) 设DFA $\mathscr{M}_1=(Q_1,A,\delta_1,q_1,F_1)$和$\mathscr{M}_2=(Q_2,A,\delta_2,q_2,F_2)$分别接受语言$L_1$和$L_2$，且设$Q_1\cap Q_2=\varnothing$. 把$\mathscr{M}_1$和$\mathscr{M}_2$粘接起来，构造出接受$L_1L_2$的FA $\tilde{\mathscr{M}}=(\tilde{Q},A,\tilde{\delta},q_1,\tilde{F})$，它以$\mathscr{M}_1$的初始状态$q_1$作为初始状态，$\tilde{Q}=Q_1\cup Q_2$，

$$\tilde{\delta}(q,a)=\begin{cases}\{\delta_1(q,a)\}, & \text{若 } q\in Q_1-F_1,\\ \{\delta_1(q,a),\delta_2(q_2,a)\}, & \text{若 } q\in F_1,\\ \{\delta_2(q,a)\}, & \text{若 } q\in Q_2,\end{cases}$$

$$\tilde{F}=\begin{cases}F_2\cup F_1, & \text{若 } q_2\in F_2\\ F_2, & \text{否则.}\end{cases}$$

$\tilde{\mathscr{M}}$的工作方式如图9.5所示.

图 9.5

$\tilde{\mathscr{M}}$的计算的前一半按照$\mathscr{M}_1$的方式进行，$\tilde{\mathscr{M}}$可以在$\mathscr{M}_1$的任何接受状态进入$\mathscr{M}_2$(从q_2开始). $\tilde{\mathscr{M}}$一旦进入$\mathscr{M}_2$，以后的计算将按$\mathscr{M}_2$的方式进行. 因此，对任意的$w\in A^*$，$\tilde{\mathscr{M}}$接受w当且仅当存在$u,v\in A^*$使$w=uv$并且$\mathscr{M}_1$接受u、$\mathscr{M}_2$接受v，即$w\in L_1L_2$.

(5) 设 DFA $\mathscr{M}=(Q,A,\delta,q_1,F)$接受 L，并且 $\mathscr{M}$ 是非重新启动的. 接受 L^* 的 FA $\tilde{\mathscr{M}}$ 与(4)中的有些类似，这次是把 q_1 与接受状态粘接起来. 在 $\mathscr{M}$ 的接受状态，$\tilde{\mathscr{M}}$ 可以按照 $\mathscr{M}$ 的方式进行计算，或者重新从 q_1 开始计算($\mathscr{M}$ 自己不可能回到 q_1)，如图 9.6 所示. 具体构造如下：

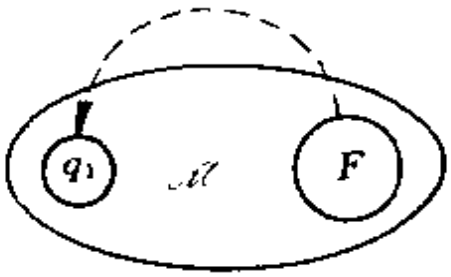


图 9.6

$$\tilde{\mathscr{M}}=(Q,A,\tilde{\delta},q_1,\tilde{F}),$$

其中

$$\tilde{\delta}(q,a)=\begin{cases}\{\delta(q,a)\}, & \text{若 } q\notin F,\\ \{\delta(q,a),\delta(q_1,a)\}, & \text{若 } q\in F,\end{cases}$$

$$\tilde{F}=F\cup\{q_1\}.$$

任给 $w\in A^*$，当 $w=\varepsilon$ 时，$\tilde{\mathscr{M}}$ 接受 ε，同时也有 $\varepsilon\in L^*$. 当 $w\neq\varepsilon$ 时，$\tilde{\mathscr{M}}$ 接受 w 当且仅当可以把 w 分成 k 段 $w=u_1u_2\cdots u_k$ 使得 $\mathscr{M}$ 接受 $u_1,u_2,\cdots,u_k$，即 $w\in L^*$. □

9.4 正则表达式

定义 9.10 字母表 A 上的**正则表达式**及其所表示的语言递归定义如下：

(1) $\varnothing$是正则表达式，它表示空集；

(2) ε 是正则表达式，它表示$\{\varepsilon\}$；

(3) 对于每一个 $a\in A$，a 是正则表达式，它表示$\{a\}$；

(4) 如果 r 和 s 分别是表示语言 R 和 S 的正则表达式，则 $(r+s)$，$(r\cdot s)$ 和 (r^*) 也是正则表达式，它们分别表示 $R\cup S$，$R\cdot S$ 和 R^*.

正则表达式 α 表示的语言记作 $\langle\alpha\rangle$.

规定运算的优先等级：* 优先于 ·，· 优先于 +. 按照这个规定，可以省去正则表达式中不必要的括号. 此外，常把 $r\cdot s$ 简写成 rs. 把 rr^* 缩写成 r^+，它表示语言 R^+.

[**例 9.6**] $\langle a^+b^+\rangle=\{a^nb^m\mid n,m>0\}$，

$$\langle(0+1)^*00\rangle=\{x00\mid x\in\{0,1\}^*\},$$

它们分别是例 9.4 和例 9.5 中 FA 接受的语言. 又如

$$\langle(01^*+10^*)(01)^*\rangle$$
$$=\{01^n(01)^m\mid n,m\geqslant 0\}\cup\{10^n(01)^m\mid n,m\geqslant 0\}.$$

[**例 9.7**] 设字母表 A，L 是 A^* 的有穷子集，则存在正则表达式 α 使得 $\langle\alpha\rangle=L$.

证：设 $u\in A^*$，若 $u=\varepsilon$，则 $\langle\varepsilon\rangle=\{u\}$；若 $u\neq\varepsilon$，设 $u=a_1a_2\cdots a_k$，其中 $a_1,a_2,\cdots,a_k\in A$，$k\geqslant 1$，则 $\langle a_1a_2\cdots a_k\rangle=\{u\}$. 总之，存在正则表达式 α 使得 $\langle\alpha\rangle=\{u\}$.

若 $L=\varnothing$，则 $\langle\varnothing\rangle=L$；若 $L\neq\varnothing$，设 $L=\{u_1,u_2,\cdots,u_m\}$，其中 $u_1,u_2,\cdots,u_m\in A^*$，$m\geqslant 1$，则存在正则表达式 α_i 使得 $\langle\alpha_i\rangle=\{u_i\}$，$i=1,2,\cdots,m$. 令 $\alpha=\alpha_1+\alpha_2+\cdots+\alpha_m$，

$$\begin{aligned}\langle\alpha\rangle&=\langle\alpha_1\rangle\cup\langle\alpha_2\rangle\cup\cdots\cup\langle\alpha_m\rangle\\&=\{u_1\}\cup\{u_2\}\cup\cdots\cup\{u_m\}\\&=L.\end{aligned}$$

下面给出正则表达式与有穷自动机和正则文法的关系.

定理 9.8 正则表达式表示的语言都可以被有穷自动机接受.

证：根据定义 9.10 和定理 9.7，只需证明 $\varnothing$，$\{\varepsilon\}$ 和 $\{a\}(a\in A)$ 可以被 FA 接受. 接受它们的 FA 如图 9.7 所示. □

定理 9.9 每一个正则语言都可以用正则表达式表示.

(1) (2) (3)

图 9.7

证：设右线性文法 $G=(V,T,\Gamma,A_1)$，$L=L(G)$，其中 $T=\{a_1,a_2,\cdots,a_m\}$，$V=\{A_1,A_2,\cdots,A_n\}$. G 的产生式只有两种类型：

(1) $A_i\to wA_j$，(2) $A_i\to w$，

其中 $1\leqslant i,j\leqslant n$，$w\in T^*$. 把 $A_i\to w$ 记作 $A_i\to wA_{n+1}$，于是 G 的产生可以统一成一种形式：

$$A_i\to wA_j,$$

其中 $1\leqslant i\leqslant n$，$1\leqslant j\leqslant n+1$，$w\in T^*$.

记

$$R_{ij}^k=\{w\in T^*\mid A_i\overset{*}{\Rightarrow}wA_j \text{ 且在派生过程中不出现下标大于 } k \text{ 的变元(不计 } A_i \text{ 和 } A_j)\},$$

$1\leqslant i\leqslant n$，$1\leqslant j\leqslant n+1$，$0\leqslant k\leqslant n$.

显然，

$$R_{1n+1}^n=L.$$

"$A_i\overset{*}{\Rightarrow}wA_j$ 且在派生过程中不出现下标大于 $k(k\geqslant 1)$ 的变元"有两种可能：(1)在派生过程中不出现下标大于 $k-1$ 的变元. 此时，$w\in R_{ij}^{k-1}$；(2)在派生过程中出现 l 个 A_k，$l\geqslant 1$. 此时，w 可分成 $l+1$ 段，首尾两段分别属于 R_{ik}^{k-1} 和 R_{kj}^{k-1}，其余 $l-1$ 段都属于 R_{kk}^{k-1}. 于是，有下述递推关系：

$$R_{ij}^0=\begin{cases}\{u\mid G \text{ 有产生式 } A_i\to uA_j\}, & \text{若 } i\neq j,\\ \{u\mid G \text{ 有产生式 } A_i\to uA_j\}\cup\{\varepsilon\}, & \text{若 } i=j,\end{cases}$$

$$R_{ij}^k=R_{ij}^{k-1}\cup R_{ik}^{k-1}(R_{kk}^{k-1})^*R_{kj}^{k-1},$$

$1\leqslant i\leqslant n$，$1\leqslant j\leqslant n+1$，$1\leqslant k\leqslant n$.

当 $k=0$ 时，对所有的 $i,j(1\leqslant i\leqslant n,1\leqslant j\leqslant n+1)$，$R_{ij}^0$ 是 T^* 的有穷子集. 根据例 9.7，存在正则表达式 r_{ij}^0 使得 $R_{ij}^0=\langle r_{ij}^0\rangle$.

假设当 $k-1(1\leqslant k\leqslant n)$ 时，对所有的 $i,j(1\leqslant i\leqslant n,1\leqslant j\leqslant n+1)$ 存在正则表达式 r_{ij}^{k-1} 使得 $R_{ij}^{k-1}=\langle r_{ij}^{k-1}\rangle$. 取 $r_{ij}^k=r_{ij}^{k-1}+r_{ik}^{k-1}(r_{kk}^{k-1})^* r_{kj}^{k-1}$，由前面给出的递推关系，有 $R_{ij}^k=\langle r_{ij}^k\rangle$. 这就证明了，所有的 $R_{ij}^k(1\leqslant i\leqslant n,1\leqslant j\leqslant n+1,0\leqslant k\leqslant n)$ 都可以用正则表达式表示. 特别地，$L=R_{1n+1}^n$ 可以用正则表达式表示. □

这样又得到一个蕴涵圈，如图 9.8 所示. 它表明正则文法、有穷自动机和正则表达式是 3 个等价的计算模型. 更详细地，把前面的有关结果综合在下述定理中.

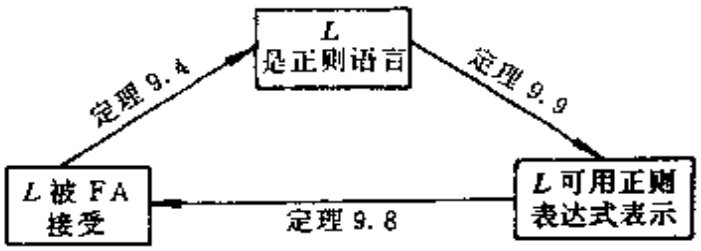


图 9.8

定理 9.10 下述命题是等价的：

(1) L 由右线性文法生成.

(2) L 由左线性文法生成.

(3) L 被 DFA 接受.

(4) L 被 NFA 接受.

(5) L 被非重新启动的 DFA 接受.

(6) L 可以用正则表达式表示.

9.5 泵 引 理

定理 9.11(泵引理) 设 DFA $\mathscr{M}$ 具有 n 个状态，$L=L(\mathscr{M})$，$x\in L$ 且 $|x|\geqslant n$，则 $x=uvw$ 使得 $v\neq\varepsilon$ 并且对所有的 $i\geqslant 0$，$uv^iw\in L$.

证：$\mathscr{M}$ 检查 x 共要转换 $|x|$ 次状态. 包括开始时所处的初始状态 q_1 在内，$\mathscr{M}$ 共经过 $|x|+1$ 个状态. 由于 $|x|\geqslant n$，根据鸽巢原理，这 $|x|+1$ 个状态中至少有两个是相同的，设为 q. 因此，可以把 x 表示成 $x=uvw$ 使得 $v\neq\epsilon$，并且

$$\delta^*(q_1,u)=q,$$
$$\delta^*(q,v)=q,$$
$$\delta^*(q,w)\in F.$$

从状态 q 开始扫描完 v 又回到状态 q. 删去这个过程或者重复若干次这个过程，都不会影响从 q_1 开始扫描 u 到达状态 q、最后又从 q 开始扫描 w 到达某个接受状态. 所以，对所有的 $i\geqslant 0$，$uv^iw\in L$.

□

[**例 9.8**] $L=\{a^mb^m\mid m>0\}$ 不是正则语言.

证：假设 L 是正则语言，则存在 DFA $\mathscr{M}$ 接受 L. 设 $\mathscr{M}$ 有 n 个状态，取 $x=a^kb^k$，其中 $2k\geqslant n$. 由泵引理，x 可写成 $x=uvw$ 使得 $v\neq\epsilon$ 并且对所有的 $i\geqslant 0$，$uv^iw\in L$. 分情况讨论如下：

(1) $v=a^r$，$0<r\leqslant k$. 此时，$u=a^s$，$w=a^tb^k$，其中 $s+r+t=k$，$s,t\geqslant 0$. 但是，$uv^2w=a^{k+r}b^k\notin L$，矛盾.

(2) $v=b^r$，$0<r\leqslant k$. 与(1)类似.

(3) $v=a^sb^r$，$0<s,r\leqslant k$. 此时，$uv^2w=a^kb^ra^sb^k\notin L$，矛盾.

所以，L 不是正则语言.

由例 9.2，已知 $\{a^mb^m\mid m>0\}$ 是上下文无关语言，从而它是非正则的上下文无关语言.

习　题

1. 构造生成下述语言的正则文法：

(1) $\{a^{3k+1}\mid k\geqslant 0\}$.

(2) $\{x\mid x\in\{0,1\}^*$ 且 x 含有子串 000$\}$.

2. 证明：设 L 是一个正则语言，则存在右线性文法 G 使得 $L(G)=$

$L-\{\varepsilon\}$且每一个产生式都形如

$$A \to aB \quad 或 \quad A \to a,$$

其中 A 和 B 是变元，a 是终极符.

3. 构造接受第 1 题中语言的 FA.

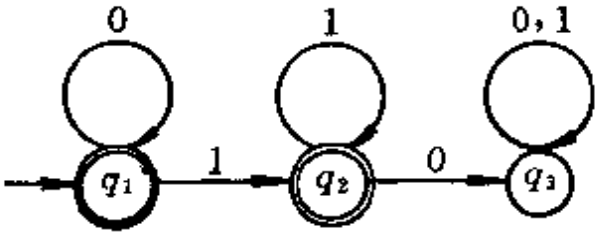
4. 构造接受下述语言的 FA：

(1) $\{a,b\}$上所有以 $abba$ 结尾的字符串集合.

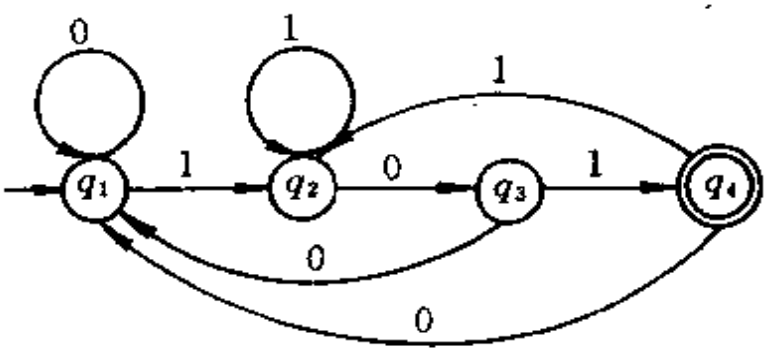
(2) $\{0,1\}$上所有 5 的整数倍的二进制表示组成的集合.

(3) $\{a,b\}$上所有不含子串 bb 的字符串集合.

5. 图 9.9 给出两个 FA 的状态转移图，求这些 FA 接受的语言.



(a)



(b)

图 9.9　第 5 题

6. 写出模拟上题 FA 的正则文法.

7. 构造接受下述语言的 NFA：

(1) 第 1 题(b).　(2) 第 4 题(a).

8. 证明：对于任意的 NFA $\mathscr{M}$，存在恰好有一个接受状态的 NFA $\mathscr{M}'$ 使得 $L(\mathscr{M}')=L(\mathscr{M})-\{\varepsilon\}$.

9. 对于正则表达式 α,β，用 $\alpha\equiv\beta$ 表示 $\langle\alpha\rangle=\langle\beta\rangle$. 设 α,β,γ 是正则表达式，证明下述等式：

(1) $\alpha+\beta\equiv\beta+\alpha$.

(2) $(\alpha+\beta)+\gamma\equiv\alpha+(\beta+\gamma)$.

(3) $(\alpha\beta)\gamma\equiv\alpha(\beta\gamma)$.

(4) $\gamma(\alpha+\beta)\equiv\gamma\alpha+\gamma\beta$.

(5) $(\alpha+\beta)\gamma\equiv\alpha\gamma+\beta\gamma$.

(6) $\varnothing^*\equiv\varepsilon$.

(7) $(\alpha^*)^*\equiv\alpha^*$.

(8) $(\varepsilon+\alpha)^*\equiv\alpha^*$.

(9) $(\alpha^*+\beta^*)^*\equiv(\alpha^*\beta^*)^*\equiv(\alpha+\beta)^*$.

10. 设 α,β 是正则表达式且 $\varepsilon\notin\langle\alpha\rangle$. 试证明：$\xi=\beta\alpha^*$ 满足方程

$$\xi\equiv\beta+\xi\alpha,$$

并且如果 ξ' 也满足方程，则 $\xi'\equiv\xi$.

11. 证明下述语言不是正则的：

(1) $\{a^nb^{2n}\mid n>0\}$.

(2) $\{a^nb^m\mid 0<n\leqslant m\}$.

(3) $\{ww^R\mid w\in\{a,b\}^*\}$.

(4) $\{a^mb^nc^n\mid m,n\geqslant 0\}$.

12. 设 L 是正则语言. 令

$$L'=\{x\mid(\exists\ y)xy\in L\},$$

证明 L' 是正则的.

第十章　上下文无关语言

10.1　上下文无关文法

10.1.1　正上下文无关文法

定义 10.1　形如 $A\to\varepsilon$ 的产生式称作**空产生式**，其中 A 是一个变元.

不含空产生式的上下文无关文法称作**正上下文无关文法**.

设 G 是一个上下文无关文法，如果 $\varepsilon\in L(G)$，显然 G 中一定含有空产生式. 下面将要证明，如果 $\varepsilon\notin L(G)$，则存在正上下文无关文法 $\bar{G}$ 使得 $L(G)=L(\bar{G})$.

定义 10.2　设上下文无关文法 $G=(V,T,\Gamma,S)$，称

$$\ker(G)=\{A\in V\mid A\overset{*}{\underset{G}{\Rightarrow}}\varepsilon\},$$

为 G 的**核**.

[**例 10.1**]　G: $S\to aA, A\to XYX, X\to b, Y\to c, X\to\varepsilon, Y\to\varepsilon$. 它的核

$$\ker(G)=\{X,Y,A\}.$$

令

$K_1=\{A\in V\mid A\to\varepsilon\in\Gamma\}$,

$K_{i+1}=K_i\cup\{A\in V\mid A\to\alpha\in\Gamma$，其中 $\alpha\in K_i^*\}$，$i=1,2,\cdots$.

由于 G 只有有穷个变元，$K_i\subseteq K_{i+1}$，必存在 t 使得 $K_{t+1}=K_t$，则 $\ker(G)=K_t$.

事实上，对所有的 $i>t$，$K_i=K_t$. 显然，$K_t\subseteq\ker(G)$. 反之，设 $A\overset{*}{\Rightarrow}\varepsilon$ 的派生长度为 i，要证 $A\in K_t$. 对 i 作归纳证明. 当 $i=1$ 时，$A\to$

ε 是 G 的产生式，故 $A\in K_1$. 假设当 $i\leqslant r$ 时结论成立，考虑 $i=r+1$ 时的情况. 设 $A\overset{*}{\Rightarrow}\varepsilon$ 的派生长度为 $r+1$，则有产生式 $A\to X_1X_2\cdots X_s$，其中 $X_j\overset{*}{\Rightarrow}\varepsilon$ 且有长度不超过 r 的派生 $(1\leqslant j\leqslant s)$. 根据归纳假设，$X_1,X_2,\cdots,X_s\in K_r$，从而 $A\in K_{r+1}$. 得证 $\ker(G)=K_r$.

根据上述递推关系，不难给出求 $\ker(G)$ 的算法.

引理 10.1 设 G 是上下文无关文法，则存在正上下文无关文法 $\tilde{G}$ 使得 $L(G)=L(\tilde{G})$ 或 $L(G)=L(\tilde{G})\cup\{\varepsilon\}$.

证：$\tilde{G}$ 与 G 有相同的终极符集 T，变元集 V 及起始符 S. 如下构造出 $\tilde{G}$ 的产生式：首先把 G 中的空产生式删掉，然后对剩下的每一个(非空)产生式，如果它的右端含有 $\ker(G)$ 中的变元，则把所有从右端删去若干个 $\ker(G)$ 中的变元后得到的产生式(不包括空产生式)添加进来.

对于例 10.1 中的 $G,\tilde{G}$ 的产生式包括：$S\to aA, A\to XYX, X\to b, Y\to c, S\to a, A\to YX, A\to XX, A\to XY, A\to X, A\to Y$. 前 4 个是 G 中原有的非空产生式，后面 6 个是新添加进来的.

要证 $L(G)=L(\tilde{G})$ 或 $L(G)=L(\tilde{G})\cup\{\varepsilon\}$.

设 $A\to\alpha$ 是 $\tilde{G}$ 的产生式，但不是 G 的产生式，它是由 G 的产生式

$$A\to\beta_0\alpha_1\beta_1\alpha_2\cdots\alpha_r\beta_r$$

删去右端的 $\beta_0,\beta_1,\cdots,\beta_r$ 后得到的，其中 $\beta_0,\beta_1,\cdots,\beta_r\in(\ker(G))^*$，$\beta_0$ 和 β_r 可能是 ε，而 $\alpha_1,\alpha_2,\cdots,\alpha_r\in(V\cup T)^*$ 且 $\alpha=\alpha_1\alpha_2\cdots\alpha_r$. 由于

$$\beta_i\overset{*}{\underset{G}{\Rightarrow}}\varepsilon,\qquad 0\leqslant i\leqslant r,$$

故有

$$A\underset{G}{\Rightarrow}\beta_0\alpha_1\beta_1\alpha_2\cdots\alpha_r\beta_r\overset{*}{\underset{G}{\Rightarrow}}\alpha_1\alpha_2\cdots\alpha_r=\alpha,$$

即，可以用 G 模拟 $A\to\alpha$. 从而得证 $L(\tilde{G})\subseteq L(G)$.

反之，设 $w\in L(G)$ 且 $w\neq\varepsilon$，要证 $w\in L(\tilde{G})$. 为此，我们证明更一般性的结论：对任意的 $A\in V, w\in T^*$，若 $A\overset{*}{\underset{G}{\Rightarrow}}w$ 且 $w\neq\varepsilon$，则

$A\overset{*}{\underset{\tilde{G}}{\Rightarrow}}w$.

对 $A\overset{*}{\underset{G}{\Rightarrow}}w$ 的派生长度 i 作归纳证明. 当 $i=1$ 时，$A\to w$ 是 G 的产生式，也是 $\tilde{G}$ 的产生式. 结论成立. 设当 $i\leqslant r$ 时结论成立. 考虑 $i=r+1$，设

$$A\underset{G}{\Rightarrow}u_0X_1u_1X_2\cdots X_su_s\overset{*}{\underset{G}{\Rightarrow}}w,$$

其中 $X_j\overset{*}{\underset{G}{\Rightarrow}}v_j$ 并且有长度不超过 r 的派生 $(1\leqslant j\leqslant s)$，$u_j\in T^*$ $(0\leqslant j\leqslant s)$，并且 $w=u_0v_1u_1v_2\cdots v_su_s$. 由归纳假设，若 $v_j\neq\varepsilon$，则有 $X_j\overset{*}{\underset{\tilde{G}}{\Rightarrow}}v_j$；若 $v_j=\varepsilon$，则 $X_j\in\ker(G)$. 把这样的 X_j 从 $A\to u_0X_1u_1X_2\cdots X_su_s$ 中删去后仍是 $\tilde{G}$ 的产生式. 于是，记

$$\tilde{X}_j=\begin{cases}X_j & \text{若 } v_j\neq\varepsilon\\ \varepsilon & \text{若 } v_j=\varepsilon,\end{cases}$$

则 $A\to u_0\tilde{X}_1u_1\tilde{X}_2\cdots\tilde{X}_su_s$ 是 $\tilde{G}$ 的产生式. 从而，

$$A\underset{\tilde{G}}{\Rightarrow}u_0\tilde{X}_1u_1\tilde{X}_2\cdots\tilde{X}_su_s\overset{*}{\underset{\tilde{G}}{\Rightarrow}}u_0v_1u_1v_2\cdots v_su_s=w.$$

得证当 $i=r+1$ 时结论也成立. □

定理 10.2 语言 L 是上下文无关的，当且仅当存在正上下文无关文法 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$.

证：必要性. 由引理 10.1 即可得到.

充分性. 设 G 是正上下文无关文法. 若 $L=L(G)$，因为 G 也是上下文无关文法，所以 L 是上下文无关语言. 若 $L=L(G)\cup\{\varepsilon\}$，如下构造文法 $\tilde{G}$：添加一个新的变元 $\tilde{S}$ 作为 $\tilde{G}$ 的起始符，$\tilde{G}$ 包括 G 的全部产生式和

$$\tilde{S}\to S,\quad \tilde{S}\to\varepsilon,$$

其中 S 是 G 的起始符. 显然，$L(\tilde{G})=L(G)\cup\{\varepsilon\}=L$，并且 $\tilde{G}$ 是上下文无关的. □

10.1.2 Chomsky 范式

定义 10.3 如果上下文无关文法 $G=(V,T,\Gamma,S)$ 的产生式

都形如

$$X \to YZ \quad 或 \quad X \to a,$$

其中 $X,Y,Z \in V, a \in T$，则称 G 是 **Chomsky 范式**.

定义 10.4 设正上下文无关文法 $G=(V,T,\Gamma,S)$，形如 $X \to Y$ 的产生式称作**单枝的**，其中 $X,Y \in V$.

如果 G 不含单枝的产生式，则称它是**分枝的**.

引理 10.3 设 G 是正上下文无关文法，则存在分枝的正上下文无关文法 $\tilde{G}$ 使得 $L(G)=L(\tilde{G})$.

证：设 $G=(V,T,\Gamma,S)$不是分枝的. 如下构造 $\tilde{G}$：

若 G 含有产生式

$$X_1 \to X_2,\ X_2 \to X_3,\ \cdots,\ X_k \to X_1, \qquad (*)$$

其中 $k \geqslant 1, X_1, X_2, \cdots, X_k \in V$，则删去这些产生式. 引入新变元 X，用它替换其余产生式中的 $X_1, X_2, \cdots, X_k$. 若 $X_1, X_2, \cdots, X_k$ 中的某一个是起始符，则以 X 作为起始符，否则原起始符保持不变. 这样做不会改变文法生成的语言. 重复这个做法，直到不含这种构成变元循环生成的$(*)$形式的产生式为止.

若仍含有单枝的产生式，则必有产生式 $X \to Y$ 使得文法不含 $Y \to Z$ 形式的单枝产生式. 删去 $X \to Y$. 如果有产生式 $Y \to \alpha$，则添加一个新的产生式 $X \to \alpha$. 这样做也不会改变文法生成的语言，但减少一个单枝产生式. 重复这样做，直到不再含有单枝产生式为止. 最后得到的文法就是所需要的 $\tilde{G}$. □

定理 10.4 设 G 是正上下文无关文法，则存在 Chomsky 范式文法 $\tilde{G}$ 使得 $L(G)=L(\tilde{G})$.

证：由引理 10.3，不妨设 $G=(V,T,\Gamma,S)$是分枝的. 对每一个 $a \in T$，引入新变元 X_a 和产生式 $X_a \to a$. 对 G 的每一个产生式 $A \to \alpha$，若 $\alpha \in T$ 或 $\alpha \in V^*$，则保留此产生式；否则把 α 中的每一个符号 $a \in T$ 替换成 X_a，得到 α'. 用 $A \to \alpha'$ 替换产生式 $A \to \alpha$. 这样得到的文法与 G 生成相同的语言，它的产生式只有下述两种形式：

(1) $X \to X_1 X_2 \cdots X_k, \quad k \geqslant 2,$

(2) $X\to a$,

其中 $X, X_1, \cdots, X_k$ 是变元，$a\in T$. 为了得到 Chomsky 范式文法，只需删去(1)中 $k>2$ 的产生式.

设有 t 个这样的产生式 $X^i\to X_1^iX_2^i\cdots X_{k_i}^i, k_i>2, i=1,2,\cdots,t$. 对每一个 i，引入 k_i-2 个新变元 $Z_1^i, Z_2^i, \cdots, Z_{k_i-2}^i$，并用下述产生式替换这 t 个中的第 i 个：

$$X^i\to X_1^iZ_1^i,$$
$$Z_1^i\to X_2^iZ_2^i,$$
$$\vdots$$
$$Z_{k_i-3}^i\to X_{k_i-2}^iZ_{k_i-2}^i,$$
$$Z_{k_i-2}^i\to X_{k_i-1}^iX_{k_i}^i.$$

这样获得的文法 $\bar{G}$ 是 Chomsky 范式文法，且 $L(G)=L(\bar{G})$. □

推论 10.5 语言 L 是上下文无关的当且仅当存在 Chomsky 范式文式 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$.

10.2 Bar-Hillel 泵引理

设上下文无关文法 $G=(V,T,\Gamma,S)$，满足下述条件的有序树称作 G-树：

(1) 每一个顶点有一个标记，标记取自 $V\cup T\cup\{\varepsilon\}$；

(2) 根的标记是变元；

(3) 内点的标记是变元；

(4) 如果标记为 A 的顶点的儿子从左到右的标记是 $\alpha_1, \alpha_2, \cdots, \alpha_k$，则 $A\to\alpha_1\alpha_2\cdots\alpha_k$ 是 G 的产生式；

(5) 标记为 ε 的顶点必是树叶，并且它是它父亲的唯一儿子.

根的标记为起始符 S 的 G-树称作 G 的**派生树**或**语法分析树**.

G-树 Λ 的全部树叶的标记从左到右(如果顶点 u 在 v 的左边，则 u 和 u 的后继在 v 和 v 的后继的左边)排列得到的字符串称

作 Λ **的结果**，记作〈Λ〉.

设 G-树 Λ 的根的标号为 A，则 Λ 描述了从 A 到〈Λ〉的派生，这样的派生通常不只一个. 反之，对从某个变元开始的派生，总可以画出它的 G-树.

[**例 10.2**] CFG G: $S\to aXS, S\to\varepsilon, X\to bXc, X\to bc$. 图 10.1 给出一棵派生树 Λ，〈Λ〉$=ab^2c^2abc$. 下述派生都对应这棵派生树：

$$\begin{aligned}S &\Rightarrow aXS\Rightarrow aXaXS\Rightarrow aXaX\Rightarrow aXabc\\ &\Rightarrow abXcabc\Rightarrow ab^2c^2abc,\\ S &\Rightarrow aXS\Rightarrow abXcS\Rightarrow ab^2c^2S\Rightarrow ab^2c^2aXS\\ &\Rightarrow ab^2c^2abcS\Rightarrow ab^2c^2abc,\end{aligned}$$

第一个是最右派生，即总对最右边的变元使用产生式. 第二个是最左派生，即总对最左边的变元使用产生式. 当然，还可以有其他的派生.

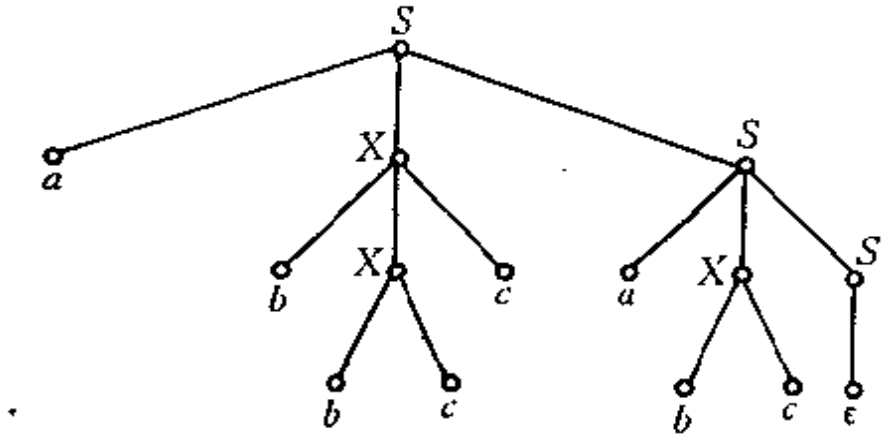


图 10.1 一棵派生树

可以证明：任给 $\alpha\in(V\cup T)^*$，$A\overset{*}{\underset{G}{\Rightarrow}}\alpha$ 当且仅当存在 G-树 Λ 使得〈Λ〉$=\alpha$，这里 Λ 的根标记为 A.

特别地，任给 $w\in T^*$，$w\in L(G)$ 当且仅当存在 G 的派生树 Λ 使得〈Λ〉$=w$. 这个 Λ 称作 G 关于 w 的派生树.

定理 10.6 (Bar-Hillel 泵引理) 设 Chomsky 范式文法 G 有 n 个变元，$L=L(G)$，$x\in L$. 若 $|x|\geqslant 2^n$，则 x 可表示成 $x=u_1v_1uv_2u_2$ 使得

(1) $|v_1uv_2|\leqslant 2^n$,

(2) $v_1v_2\neq\varepsilon$,

(3) 对所有的 $k\geqslant 0$, $u_1v_1^kuv_2^ku_2\in L$.

证：考虑 G 关于 x 的派生树 Λ. 设 Λ 的高度为 t(约定：树根的高度为 1)，由于 G 是 Chomsky 范式文法，故 $|x|\leqslant 2^{t-2}$. 从而，$t\geqslant n+2$. 取 Λ 的一条最长通路，顶点自上而下依次为 $\gamma_1,\gamma_2,\cdots,\gamma_t$. 除 γ_t 的标记是终极符外，其余 $t-1$ 个顶点的标记都是变元. 由于 $t-1>n$，所以必存在两个顶点标记同一个变元. 设 γ_i 和 $\gamma_j(i<j)$是最低的这样两个顶点，即 γ_i 和 γ_j 具有相同的标记 A 且 $\gamma_{i+1},\cdots,\gamma_{t-1}$没有相同的标记.

把以 γ_i 和 γ_j 为根的子树分别记作 Λ_i 和 Λ_j. 又记 $u=\langle\Lambda_j\rangle$，$\langle\Lambda_i\rangle=v_1uv_2$，$x=u_1v_1uv_2u_2$，如图 10.2 所示，有

$$A\overset{*}{\Rightarrow}u,$$

$$A\overset{*}{\Rightarrow}v_1uv_2,$$

$$S\overset{*}{\Rightarrow}u_1Au_2.$$

于是，对任意的 $k\geqslant 0$，$S\overset{*}{\Rightarrow}u_1v_1^kuv_2^ku_2$，即(3)成立.

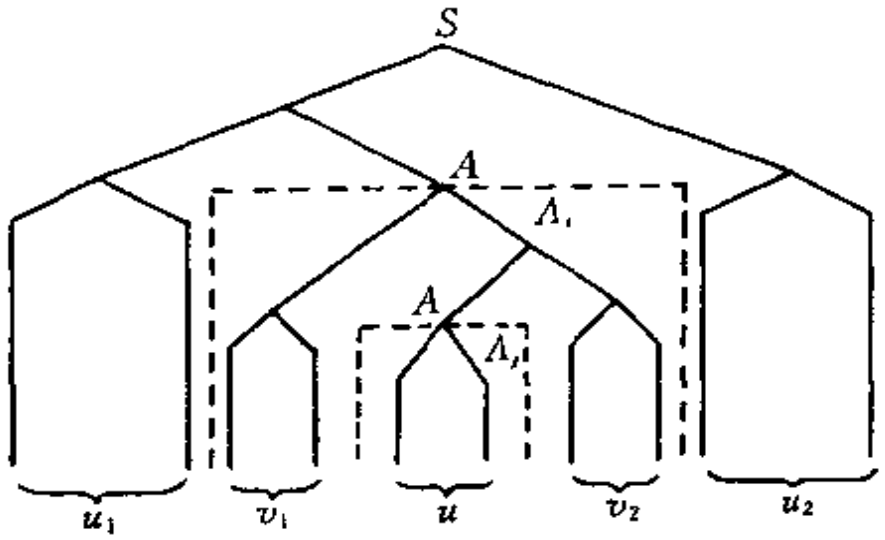


图 10.2

根据 γ_i 和 γ_j 的定义，Λ_i 的高度$\leqslant n+2$，从而 $|v_1uv_2|\leqslant 2^n$，即(1)成立. 由于顶点 γ_i 是二叉的并且没有树叶以 ε 为标记，故 v_1 和

v_2 不能同时为 ε，即(2)成立. □

推论 10.7 设 L 是上下文无关语言，则存在正整数 m 使得所有长度$\geqslant m$ 的 $x\in L$ 都可表示成 $x=u_1v_1uv_2u_2$，其中

(1) $|v_1uv_2|\leqslant m$，

(2) $v_1v_2\neq\varepsilon$，

(3) 对所有的 $k\geqslant 0$，$u_1v_1^kuv_2^ku_2\in L$.

证：由推论 10.5，存在 Chomsky 范式文法 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$. 设 G 有 n 个变元，根据定理 10.6，取 $m=2^n$ 即可. □

[例 10.3] $L=\{a^nb^nb^n\mid n>0\}$不是上下文无关语言.

证：假若 L 是CFL，根据推论 10.7，对充分大的 n，有 $a^nb^nc^n=u_1v_1uv_2u_2$，其中 $v_1v_2\neq\varepsilon$ 并且对所有的 $k\geqslant 0$，$u_1v_1^kuv_2^ku_2\in L$.

取 $k=2$，$u_1v_1^2uv_2^2u_2\in L$. 可见，v_1 只能取 a^i，b^i 和 c^i 3 种形式中的一种，否则 v_1^2 中会出现 a,b,c 中某两个的逆序排列，这是不可能的. v_2 也是一样.

不妨设 $v_1=a^i$，$v_2=b^j$，i 和 j 不同时为 0. 取 $k=0$，u_1uu_2 中有 $n-i$个 a，$n-j$ 个 b，n 个 c，不属于 L，矛盾. 所以，L 不是CFL.

由例 9.3，已知$\{a^nb^nc^n\mid n>0\}$是上下文有关的，故它是非上下文无关的上下文有关语言.

10.3 下推自动机

在有穷自动机上加一个“先进后出”的栈就得到一台下推自动机(简记作 PDA). 它有一个控制器、一条输入带和一个栈. 控制器有有穷个状态. 带头和有穷自动机的一样，只读不写，从左到右逐个扫描输入字符串. 栈头永远指向栈顶，它能够读到栈顶的符号、又能用一个字符串替换栈顶的符号. 栈的内容是一个字符串，左端第一个符号放在栈顶位置. 在栈顶写一个字符串时，栈中的原有符号顺序向下移动. 用空串替换栈顶符号的结果是移去栈顶符号，此

时栈中其余的符号顺序上移. 如图 10.3 所示.

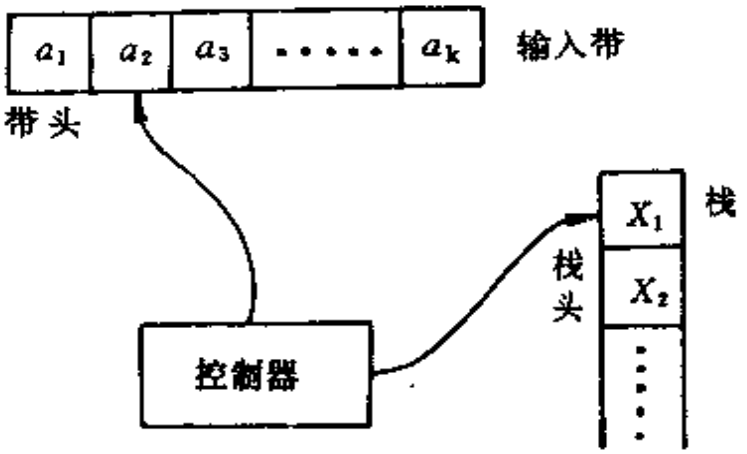


图 10.3 PDA 的示意图

PDA 是非确定型的，在每一步有若干个(当然是有穷个)动作可供选择. 它的动作有两种类型. 第一种类型的动作与输入字符串有关，根据当前的状态、带头扫描的符号和栈顶符号，决定可供选择的几个动作. 每一个动作是转移到指定的下一个状态，用给定的字符串替换栈顶符号，并且将带头右移一格. 第二种类型的动作与输入字符串无关，叫做 **ε 动作**. 它和第一种类型的动作的区别是，不必考虑带头扫描的符号，且带头保持不动. ε 动作使 PDA 能够不读输入符号就可以处理栈，而第一种类型的动作每一步要“消耗”一个输入符号.

定义 10.5 一台**下推自动机** $\mathscr{M}$ 由 7 部分组成：

(1) 状态集 Q，Q 是一个有穷集合；

(2) 输入字母表 A；

(3) 栈字母表 Ω；

(4) 动作函数 δ，δ 是从 $Q\times(A\cup\{\varepsilon\})\times\Omega$ 到 $Q\times\Omega^*$ 的二元关系；

(5) 初始状态 q_1，$q_1\in Q$；

(6) 栈起始符 $X_0\in\Omega$；

(7) 接受状态集 F，$F\subseteq Q$；

记作 $\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$.

约定：用 p,q,r（可带下标，下同）等表示状态；a,b,c 等表示输入字符；u,v,w,x,y,z 等表示输入字符串；X,Y,Z 等表示栈符号；α,β,γ 等表示栈字符串或 $A\cup\Omega$ 上的字符串.

δ 有两种情况：

(1) 对 $q\in Q$，$a\in A$ 和 $X\in\Omega$，

$$\delta(q,a,X)=\{(p_i,\gamma_i)\mid i=1,2,\cdots,m\},$$

其中 $p_i\in Q,\gamma_i\in\Omega^*$ $(1\leqslant i\leqslant m)$. 它给出第一种类型的动作：当 $\mathscr{M}$ 处于状态 q、带头扫描符号 a、栈顶符号为 X 时，有 m 个可能的动作. 第 i 个可能的动作是移去栈顶符号 X，把 γ_i 推入栈内，转移到状态 p_i，同时带头右移一格.

(2) 对 $q\in Q$ 和 $X\in\Omega$，

$$\delta(q,\varepsilon,X)=\{(p_i,\gamma_i)\mid i=1,2,\cdots,m\},$$

其中 $p_i\in Q,\gamma_i\in\Omega^*$ $(1\leqslant i\leqslant m)$. 它给出 ε 动作，与输入串无关，带头保持不动.

3 元组 (q,u,γ) 称作 PDA $\mathscr{M}$ 的**瞬时描述或格局**，其中 $q\in Q$，$u\in A^*,\gamma\in\Omega^*$. 它的含义是：$\mathscr{M}$ 处于状态 q，输入字符串尚未读过的部分（即从带头扫描的符号开始右边的子字符串）为 u，栈的内容为 γ. 由于 PDA 的带头只能从左向右移动，读过的输入符号在后面的计算不会再被使用，所以 $\mathscr{M}$ 的瞬时描述完全确定了此后可能得到的结果.

设 σ 和 τ 是 $\mathscr{M}$ 的两个格局，如果可以用一个动作从 σ 得到 τ，则记作

$$\sigma\vdash_{\mathscr{M}}\tau.$$

如果有 $\sigma=\sigma_0\vdash_{\mathscr{M}}\sigma_1\vdash_{\mathscr{M}}\cdots\vdash_{\mathscr{M}}\sigma_k=\tau$，这里 $k\geqslant 0$，则记作

$$\sigma\overset{*}{\vdash}_{\mathscr{M}}\tau.$$

特别地，总有

$$\sigma\overset{*}{\vdash}_{\mathscr{M}}\sigma.$$

当不会引起混淆时，可以省去 $\vdash_{\mathscr{M}}$ 和 $\overset{*}{\vdash}_{\mathscr{M}}$ 的下标 $\mathscr{M}$，分别记作 $\vdash$ 和 $\overset{*}{\vdash}$.

给定输入字符串 $w\in A^*$，PDA $\mathscr{M}$ 的初始格局为 (q_1,w,X_0)，即在计算开始时 $\mathscr{M}$ 处于初始状态 q_1，带头扫描 w 的第一个符号（若 $w=\varepsilon$，带头将不起作用），栈内只有一个符号 X_0. 计算一步一步地进行. 有两种方式定义 PDA $\mathscr{M}$ 接受 w. 第一种方式和 FA 的一样，如果存在从初始格局开始的计算使得读完输入串 w 并进入某个接受状态，即

$$(q_1,w,X_0)\overset{*}{\vdash}_{\mathscr{M}}(p,\varepsilon,\gamma),$$

其中 $p\in F,\gamma\in\Omega^*$，则称 $\mathscr{M}$ **接受** w. 第二种方式称作**空栈方式**，如果存在从初始格局开始的计算使得读完输入串 w 并倒空栈，即

$$(q_1,w,X_0)\overset{*}{\vdash}_{\mathscr{M}}(p,\varepsilon,\varepsilon),$$

其中 $p\in Q$，则称 $\mathscr{M}$ **接受** w. 当采用空栈方式时，接受状态集 F 不起作用，通常取 $F=\varnothing$.

定义10.6 设 PDA $\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$，$\mathscr{M}$ **以接受状态方式接受的语言**记作 $L(\mathscr{M})$，

$$L(\mathscr{M})=\{w\in A^*\ \text{存在}\ p\in F\ \text{和}\ \gamma\in\Omega^*\ \text{使得}$$
$$(q_1,w,X_0)\overset{*}{\vdash}_{\mathscr{M}}(p,\varepsilon,\gamma)\}.$$

$\mathscr{M}$ **以空栈方式接受的语言**记作 $N(\mathscr{M})$，

$$N(\mathscr{M})=\{w\in A^*\mid\text{存在}\ p\in Q\ \text{使得}$$
$$(q_1,w,X_0)\overset{*}{\vdash}_{\mathscr{M}}(p,\varepsilon,\varepsilon)\}.$$

[例10.4] 构造 PDA 分别以接受状态方式和空栈方式接受语言

$$L_1=\{u\#u^R\mid u\in\{0,1\}^*\},$$

其中 u^R 是 u 的反转.

先考虑以空栈方式接受 L_1 的 PDA $\mathscr{M}_1$. $\mathscr{M}_1$ 有3个栈符号 R,B 和 G，R 是栈起始符，B 和 G 分别对应于0和1. 有2个状态 q_1 和 q_2. q_1 是初始状态. 计算分成两个阶段：记录阶段和检验阶段，以读到

#为界. 计算开始时是记录阶段. 在 q_1 下，读到0把 B 推入栈，读到1把 G 推入栈. 当读到#时，从栈顶往下栈内的符号恰好对应输入串前半部分的反转. 此时 $\mathscr{M}_1$ 转移到状态 q_2，计算进入检验阶段. 在 q_2 下，如果读到0时栈顶为 B，则把这个 B 托出；如果读到1时栈顶为 G，则把这个 G 托出. 显然，输入串属于 L_1 当且仅当读完输入串时处于状态 q_2，并且栈内只剩下一个符号 R. 如果得到这样的格局，只需再用一个 ε 动作把 R 托出栈，使得栈排空.

$\mathscr{M}_1=(Q,A,\Omega,\delta,q_1,R,\varnothing)$，其中 $Q=\{q_1,q_2\}$，$A=\{0,1,\#\}$，$\Omega=\{R,B,G\}$，δ 如表10-1所示.

表10-1 PDA $\mathscr{M}_1$ 的动作函数 δ

q,a \ X	R	B	G
$q_1,0$	$\{(q_1,BR)\}$	$\{(q_1,BB)\}$	$\{(q_1,BG)\}$
$q_1,1$	$\{(q_1,GR)\}$	$\{(q_1,GB)\}$	$\{(q_1,GG)\}$
$q_1,\#$	$\{(q_2,R)\}$	$\{(q_2,B)\}$	$\{(q_2,G)\}$
q_1,ε	$\varnothing$	$\varnothing$	$\varnothing$
$q_2,0$	$\varnothing$	$\{(q_2,\varepsilon)\}$	$\varnothing$
$q_2,1$	$\varnothing$	$\varnothing$	$\{(q_2,\varepsilon)\}$
$q_2,\#$	$\varnothing$	$\varnothing$	$\varnothing$
q_2,ε	$\{(q_2,\varepsilon)\}$	$\varnothing$	$\varnothing$

只需对 $\mathscr{M}_1$ 稍加修改即可得到以接受状态方式接受 L_1 的PDA $\mathscr{M}_2$. $\mathscr{M}_2$ 需增添一个接受状态 q_3，令 $F=\{q_3\}$. 把 $\delta(q_2,\varepsilon,R)$ 改为 $\{(q_3,R)\}$，并且令 $\delta(q_3,a,X)=\varnothing$，这里 $a\in\{0,1,\#,\varepsilon\}$，$X\in\{R,B,G\}$，即在状态 q_3 下 $\mathscr{M}_2$ 没有动作. 当读完输入串，处于 q_2，栈内恰好只有一个符号 R 时，$\mathscr{M}_2$ 用一个 ε 动作把状态转移到 q_3.

[例10.5] 构造PDA分别以接受状态方式和以空栈方式接受语言

$$L_2=\{uu^R\mid u\in\{0,1\}^*\}.$$

L_2 与例10.4中的 L_1 非常相似，唯一的区别是 L_1 中的字符串有

一个分界符#把字符串分成互为反转的两段，而L_2中字符串没有这样的分界符. 这样一来，在计算过程中无法像例10.4中那样知道在什么时候从q_1转移到q_2. 为了克服这个困难，需要利用PDA的非确定性. 对例10.4中的$\mathscr{M}_1$修改如下：在状态q_1下，当读到0且栈顶为B时有两个动作可供选择. 一个动作和$\mathscr{M}_1$的一样，把一个B推入栈内；另一个动作是把栈顶的B托出并且转移到状态q_2. 同样，在状态q_1下，当读到1且栈顶为G时也有两个动作可供选择. 一个是把一个G推入栈内；另一个是把栈顶的G托出并且转移到状态q_2. 于是，在状态q_1下，当遇到上述两种情况时机器有两种选择，继续在状态q_1下把读到的符号记录到栈中，或者转移到q_2并开始检查后半段是否是前半段的反转. 在计算中要不断地猜想是否该开始检查，这样的猜想是完全任意的. 但是，显然当且仅当输入串属于L_2时存在"正确的猜想"，使得恰好在把输入串分成互为反转的两段的分界处开始进入检查阶段，从而在读完输入串后能将栈排空. 此外，由于$\varepsilon\in L_2$，还需有$\delta(q_1,\varepsilon,R)=\{(q_2,\varepsilon)\}$.

以空栈方式接受L_2的PDA $\mathscr{M}_3=(Q,A,\Omega,\delta,q_1,R,\varnothing)$，其中$Q=\{q_1,q_2\}$，$A=\{0,1\}$，$\Omega=\{R,B,G\}$，$\delta$如表10-2所示. 对$\mathscr{M}_3$稍加修改不难得到以接受状态方式接受$L_2$的PDA.

表10-2　PDA$\mathscr{M}_3$的动作函数δ

q,a \ X	R	B	G
$q_1,0$	$\{(q_1,BR)\}$	$\{(q_1,BB),(q_2,\varepsilon)\}$	$\{(q_1,BG)\}$
$q_1,1$	$\{(q_1,GR)\}$	$\{(q_1,GB)\}$	$\{(q_1,GG),(q_2,\varepsilon)\}$
q_1,ε	$\{(q_2,\varepsilon)\}$	$\varnothing$	$\varnothing$
$q_2,0$	$\varnothing$	$\{(q_2,\varepsilon)\}$	$\varnothing$
$q_2,1$	$\varnothing$	$\varnothing$	$\{(q_2,\varepsilon)\}$
q_2,ε	$\{(q_2,\varepsilon)\}$	$\varnothing$	$\varnothing$

定义10.6给出PDA接受语言的两种方式，其实这两种方式是等价的，即PDA以这两种方式接受的语言类是相同的. 下述定

理给出这个结果.

定理10.8 设语言 L，存在 PDA $\mathscr{M}_1$ 使得 $L(\mathscr{M}_1)=L$ 当且仅当存在 PDA $\mathscr{M}_2$ 使得 $N(\mathscr{M}_2)=L$.

证：设 $\mathscr{M}_1=(Q_1,A,\Omega_1,\delta_1,q_1,X_1,F)$，$L(\mathscr{M}_1)=L$. 要用 $\mathscr{M}_2$ 模拟 $\mathscr{M}_1$，当 $\mathscr{M}_1$ 进入接受状态时 $\mathscr{M}_2$ 能用一个特殊的状态 q_e 将栈排空. 为了保证若 $\mathscr{M}_2$ 排空栈则 $\mathscr{M}_1$ 必进入某个接受状态，添加一个栈底标志 X_2（它同时也是 $\mathscr{M}_2$ 的栈起始符），只有在 $\mathscr{M}_1$ 的接受状态和 q_e 下能把它清除出栈. 令

$$\mathscr{M}_2=(Q_1\cup\{q_2,q_e\},A,\Omega_1\cup\{X_2\},\delta_2,q_2,X_2,\varnothing),$$

其中 $q_2,q_e\notin Q_1$，$X_2\notin\Omega_1$，δ_2 规定如下：

(1) $\delta_2(q_2,\varepsilon,X_2)=\{(q_1,X_1X_2)\}$；

(2) 对所有 $p\in Q_1$，$a\in A\cup\{\varepsilon\}$，$Z\in\Omega_1$，

(2.1) 若 $p\in F$ 且 $a=\varepsilon$，则

$$\delta_2(p,a,Z)=\delta_1(p,a,Z)\cup\{(q_e,\varepsilon)\};$$

(2.2) 否则 $\delta_2(p,a,Z)=\delta_1(p,a,Z)$；

(3) 对所有 $p\in F$，$\delta_2(p,\varepsilon,X_2)=\{(q_e,\varepsilon)\}$；

(4) 对所有 $Z\in\Omega_1\cup\{X_2\}$，$\delta_2(q_e,\varepsilon,Z)=\{(q_e,\varepsilon)\}$；

(5) 其他情况，$\delta_2(p,a,Z)=\varnothing$.

规则(1)使 $\mathscr{M}_2$ 一开始就进入 $\mathscr{M}_1$ 的初始格局，只是在 X_1 下面多一个栈底标志 X_2. 规则(2)使 $\mathscr{M}_2$ 能够模拟 $\mathscr{M}_1$. 一旦 $\mathscr{M}_1$ 进入接受状态，规则(2)还使 $\mathscr{M}_2$ 有两种选择，继续模拟 $\mathscr{M}_1$ 或者转入状态 q_e 并开始清除栈. 规则(2.1)，(3)和(4)保证只要 $\mathscr{M}_1$ 进入接受状态，$\mathscr{M}_2$ 就能将栈排空. 同时，由于只能在 $\mathscr{M}_1$ 的接受状态和 q_e 下才能将 X_2 清除出栈，并且只有从 $\mathscr{M}_1$ 的接受状态才能转移到 q_e，所以又只有 $\mathscr{M}_1$ 进入接受状态后 $\mathscr{M}_2$ 才能将栈排空.

任给 $x\in L(\mathscr{M}_1)$，则存在 $p\in F$ 和 $\gamma\in\Omega_1^*$ 使得

$$(q_1,x,X_1)\overset{*}{\underset{\mathscr{M}_1}{\vdash}}(p,\varepsilon,\gamma).$$

于是，

$$(q_2,x,X_2)\underset{\mathscr{M}_2}{\vdash}(q_1,x,X_1X_2)\qquad\text{规则(1)}$$

$$\underset{\mathscr{M}_2}{\overset{*}{\vdash}}(p,\varepsilon,\gamma X_2)\qquad\text{规则(2)}$$

$$\underset{\mathscr{M}_2}{\overset{*}{\vdash}}(q_e,\varepsilon,\varepsilon)\qquad\text{规则(2.1)、(3)、(4)}$$

得 $x\in N(\mathscr{M}_2)$.

反之，任给 $x\in N(\mathscr{M}_2)$，由于规则(1)给出唯一与 q_2 有关的动作，必有

$$(q_2,x,X_2)\underset{\mathscr{M}_2}{\vdash}(q_1,x,X_1X_2).$$

根据规则(3)和(4)，把 X_2 清除出栈后的状态是 q_e，故

$$(q_2,x,X_2)\underset{\mathscr{M}_2}{\vdash}(q_1,x,X_1X_2)$$

$$\underset{\mathscr{M}_2}{\overset{*}{\vdash}}(q_e,\varepsilon,\varepsilon).$$

又由于第一次出现 q_e 只能是从 $\mathscr{M}_1$ 的某个接受状态转移来的，故存在 $p\in F$ 和 $\gamma\in\Omega^*$ 使

$$(q_2,x,X_2)\underset{\mathscr{M}_2}{\vdash}(q_1,x,X_1X_2)$$

$$\underset{\mathscr{M}_2}{\overset{*}{\vdash}}(p,\varepsilon,\gamma X_2)$$

$$\underset{\mathscr{M}_2}{\overset{*}{\vdash}}(q_e,\varepsilon,\varepsilon).$$

从 q_e 不能转移到其他状态，故从 q_1 转移到 p 的过程中 $\mathscr{M}_2$ 一直在模拟 $\mathscr{M}_1$，从而有

$$(q_1,x,X_1)\underset{\mathscr{M}_1}{\overset{*}{\vdash}}(p,\varepsilon,\gamma).$$

得 $x\in L(\mathscr{M}_1)$. 所以，$N(\mathscr{M}_2)=L(\mathscr{M}_1)=L$.

再设 $\mathscr{M}_2=(Q_2,A,\Omega_2,\delta_2,q_2,X_2,\varnothing)$，$N(\mathscr{M}_2)=L$. 要用 $\mathscr{M}_1$ 模拟 $\mathscr{M}_2$. $\mathscr{M}_1$ 有一个栈底符号 X_1(它也是 $\mathscr{M}_1$ 的栈起始符)，当且仅当 $\mathscr{M}_2$ 把它的所有栈符号排出栈时 X_1 再次出现在栈顶. 这时 $\mathscr{M}_1$ 转移到接受状态 q_a. 令

$$\mathscr{M}_1=(Q_2\cup\{q_1,q_a\},A,\Omega_2\cup\{X_1\},\delta_1,q_1,X_1,\{q_a\}),$$

其中 $q_1,q_a\notin Q_2$，$X_1\notin\Omega_2$，δ_1 规定如下：

(1) $\delta_1(q_1,\varepsilon,X_1)=\{(q_2,X_2X_1)\}$;

(2) 对所有的 $p\in Q_2, a\in A\cup\{\varepsilon\}, Z\in\Omega_2$,

$$\delta_1(p,a,Z)=\delta_2(p,a,Z);$$

(3) 对所有的 $p\in Q_2, \delta_1(p,\varepsilon,X_1)=\{(q_a,\varepsilon)\}$;

(4) 其他情况,$\delta_1(p,a,Z)=\varnothing$.

证明 $L(\mathscr{M}_1)=L$ 的细节留给读者完成. □

10.4 上下文无关文法与下推自动机的等价性

定理10.9 设 L 是一上下文无关语言,则存在 PDA $\mathscr{M}$ 使得 $L=N(\mathscr{M})$.

证:由推论10.5,存在 Chomsky 范式文法 $G=(V,T,\Gamma,S)$ 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$. 构造 PDA $\mathscr{M}=(\{q\},T,V,\delta,q,S,\varnothing)$,其中 δ 规定如下:对所有 $a\in T, X,Y\in V$,

$$(q,\varepsilon)\in\delta(q,a,X) \quad\Leftrightarrow\quad X\to a\in\Gamma,$$

$$(q,YZ)\in\delta(q,\varepsilon,X) \quad\Leftrightarrow\quad X\to YZ\in\Gamma.$$

要证 $L(G)=N(\mathscr{M})$. 对 G 采用最左派生,即每一次总是对最左边的那个变元采用产生式. 由于 G 是 Chomsky 范式文法,在最左派生过程中总形如 $S\overset{*}{\Rightarrow}w\alpha$,其中 $w\in T^*, \alpha\in V^*$. 只需证对任意的 $w\in T^*$ 和 $\alpha\in V^*$,有

$$S\overset{*}{\Rightarrow}w\alpha \quad\Leftrightarrow\quad (q,w,S)\overset{*}{\vdash}(q,\varepsilon,\alpha).$$

"⇒"对派生长度 i 进行归纳证明. 当 $i=0$ 时,$w=\varepsilon, \alpha=S$. 结论显然成立.

假设当 $i=r$ 时结论成立. 考虑 $i=r+1$ 时的情况,有两种可能:

(1) 最后使用的产生式是 $X\to a$. 于是,有

$$S\overset{*}{\Rightarrow}uX\alpha\Rightarrow ua\alpha, \quad \text{这里 } w=ua.$$

由归纳假设,有

$$(q,u,S)\overset{*}{\vdash}(q,\varepsilon,X\alpha).$$

从而，

$$(q,w,S) \overset{*}{\vdash} (q,a,X\alpha) \vdash (q,\varepsilon,\alpha),$$

最后一步使用$(q,\varepsilon)\in\delta(q,a,X)$.

(2) 最后使用的产生式是$X\to YZ$. 于是，有

$$S\overset{*}{\Rightarrow}wX\beta\Rightarrow wYZ\beta,\text{这里 }\alpha=YZ\beta.$$

由归纳假设并使用$(q,YZ)\in\delta(q,\varepsilon,X)$，得

$$(q,w,S) \overset{*}{\vdash} (q,\varepsilon,X\beta) \vdash (q,\varepsilon,YZ\beta) = (q,\varepsilon,\alpha).$$

得证当$i=r+1$时结论亦成立.

"$\Leftarrow$"可类似地对$\mathscr{M}$的计算长度作归纳证明.

最后，若$L=L(G)$，则$\mathscr{M}$就是所要的 PDA. 若$L=L(G)\cup\{\varepsilon\}$，可构造出 PDA$\tilde{\mathscr{M}}$使得$N(\tilde{\mathscr{M}})=N(\mathscr{M})\cup\{\varepsilon\}$，从而有$N(\tilde{\mathscr{M}})=L$. $\tilde{\mathscr{M}}$的构造如下：添加一个新的栈符号$\tilde{S}$作为栈起始符，并令

$$\delta(q,\varepsilon,\tilde{S})=\{(q,\varepsilon),(q,S)\},$$
$$\delta(q,a,\tilde{S})=\varnothing,\text{对所有 }a\in T.$$

$\tilde{\mathscr{M}}$在第一步有两个动作可供选择. 若取(q,ε)，则仅当输入串为ε时$\tilde{\mathscr{M}}$接受；若取(q,S)，则此后的计算完全按照$\mathscr{M}$进行. □

定理10.10 设$\mathscr{M}$是一台 PDA，则$L=N(\mathscr{M})$是上下文无关语言.

证：设$\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,\varnothing)$，构造 CFG $G=(V,A,\Gamma,S)$，其中

$$V=\{S\}\cup\{[q,X,p]\mid q,p\in Q,X\in\Omega\},$$

Γ由下述产生式组成：

(1) 对每一个$p\in Q$，$S\to[q_1,X_0,p]$；

(2) 若$\delta(q,a,X)$含有$(p_1,Y_1Y_2\cdots Y_m)$，其中$q,p_1\in Q,X,Y_1,Y_2,\cdots,Y_m\in\Omega,a\in A\cup\{\varepsilon\}$，则

当$m>0$时，对所有$p_2,\cdots,p_{m+1}\in Q$，

$$[q,X,p_{m+1}]\to a[p_1,Y_1,p_2][p_2,Y_2,p_3]$$

$$\cdots[p_m,Y_m,p_{m+1}];$$

当 $m=0$时，$[q,X,p_1]\to a$.

要证 $L(G)=N(\mathscr{M})$，即对所有的 $w\in A^*$，

$$S\overset{*}{\Rightarrow}w \quad\Leftrightarrow\quad \exists\, p\in Q \quad (q_1,w,X_0)\overset{*}{\vdash}(p,\varepsilon,\varepsilon).$$

注意到

$$S\overset{*}{\Rightarrow}w \quad\Leftrightarrow\quad \exists\, p\in Q \quad [q_1,X_0,p]\overset{*}{\Rightarrow}w,$$

只需证明下述更一般性的结论：对所有的 $p,q\in Q,X\in\Omega,w\in A^*$ 有

$$[q,X,p]\overset{*}{\Rightarrow}w \quad\Leftrightarrow\quad (q,w,X)\overset{*}{\vdash}(p,\varepsilon,\varepsilon).$$

可以这样解释上式：G 从$[q,X,p]$派生出 w，当且仅当 $\mathscr{M}$ 从 q 转移到 p、从栈顶擦去 X 恰好需要消耗(读入)w.

“⇒”对派生长度 i 作归纳证明. 当 $i=1$时，$w=a\in A\cup\{\varepsilon\}$且 G 有产生式$[q,X,p]\to a$. 于是，$\delta(q,a,X)$中含有(p,ε)，故$(q,w,X)\vdash(p,\varepsilon,\varepsilon)$. 结论成立.

假设当 $i\leqslant r$ 时结论成立，考虑 $i=r+1$. 设

$$[q,X,p]\Rightarrow a[p_1,Y_1,p_2][p_2,Y_2,p_3]\cdots[p_m,Y_m,p_{m+1}]$$
$$\overset{*}{\Rightarrow}w,$$

其中 $a\in A\cup\{\varepsilon\},Y_1,\cdots,Y_m\in\Omega,p=p_{m+1},p_1,\cdots,p_m\in Q,m\geqslant 1$，则必有

$$w=aw_1w_2\cdots w_m$$

$$[p_j,Y_j,p_{j+1}]\overset{*}{\Rightarrow}w_j,\quad j=1,2,\cdots,m,$$

且这些派生的长度不超过 r. 由 Γ 的构造和归纳假设，$\delta(q,a,X)$含有$(p_1,Y_1Y_2\cdots Y_m)$以及

$$(p_j,w_j,Y_j)\overset{*}{\vdash}(p_{j+1},\varepsilon,\varepsilon),\quad j=1,2,\cdots,m.$$

于是，

$$(q,w,X)\vdash(p_1,w_1w_2\cdots w_m,Y_1Y_2\cdots Y_m)$$
$$\overset{*}{\vdash}(p_2,w_2\cdots w_m,Y_2\cdots Y_m)$$

$$\overset{*}{\vdash} \cdots$$

$$\overset{*}{\vdash} (p_m, w_m, Y_m)$$

$$\overset{*}{\vdash} (p, \varepsilon, \varepsilon),$$

即当 $i=r+1$时结论也成立.

"⇐"对 $\mathscr{M}$ 的计算步数 i 作归纳证明. 当 $i=1$时，$w=a\in A\cup\{\varepsilon\}$，$\delta(q,a,X)$含有$(p,\varepsilon)$，所以 G 有产生式$[q,X,p]\to a$，从而$[q,X,p]\Rightarrow a=w$，结论成立.

假设当 $i\leqslant r$ 时结论成立，考虑 $i=r+1$. 设 $w=au$，$a\in A\cup\{\varepsilon\}$，$u\in A^*$，并且

$$(q,w,X)\vdash (p_1,u,Y_1Y_2\cdots Y_m)\overset{*}{\vdash}(p,\varepsilon,\varepsilon),$$

则 u 一定可以写成 $u=u_1u_2\cdots u_m$ 使得

$$(p_j,u_j,Y_j)\overset{*}{\vdash}(p_{j+1},\varepsilon,\varepsilon),\quad j=1,2,\cdots,m,$$

其中 $p_{m-1}=p$. 即 $\mathscr{M}$ 从 p_1开始读完 u_1时，栈的高度第一次变成 $m-1$(在此之前，栈的高度可能多次变化，但总不低于 m，即 Y_2及其下面的符号始终在栈内)，状态转移到 p_2；从 p_2开始，读完 u_2时栈的高度第一次降低到 $m-2$，转移到 p_3；……，最后读完 u_m(也读完整个 u)时整个栈被排空，到达状态 p. 如图10.4所示. 这些计算的步数都不超过 r. 由归纳假设，有

$$[p_j,Y_j,p_{j+1}]\overset{*}{\Rightarrow}u_j,\quad j=1,2,\cdots,m.$$

又 $\delta(q,a,X)$中含有$(p_1,Y_1Y_2\cdots Y_m)$，故 G 有产生式

$$[q,X,p_{m+1}]\to a[p_1,Y_1,p_2][p_2,Y_2,p_3]\cdots[p_m,Y_m,p_{m+1}].$$

从而，有

$$[q,X,p]\Rightarrow a[p_1,Y_1,p_2][p_2,Y_2,p_3]\cdots[p_m,Y_m,p_{m+1}]$$

$$\overset{*}{\Rightarrow}au_1u_2\cdots u_m=w.$$

得证当 $i=r+1$时结论也成立. □

将前面的一些结果综合如下：

定理10.11 下述命题是等价的：

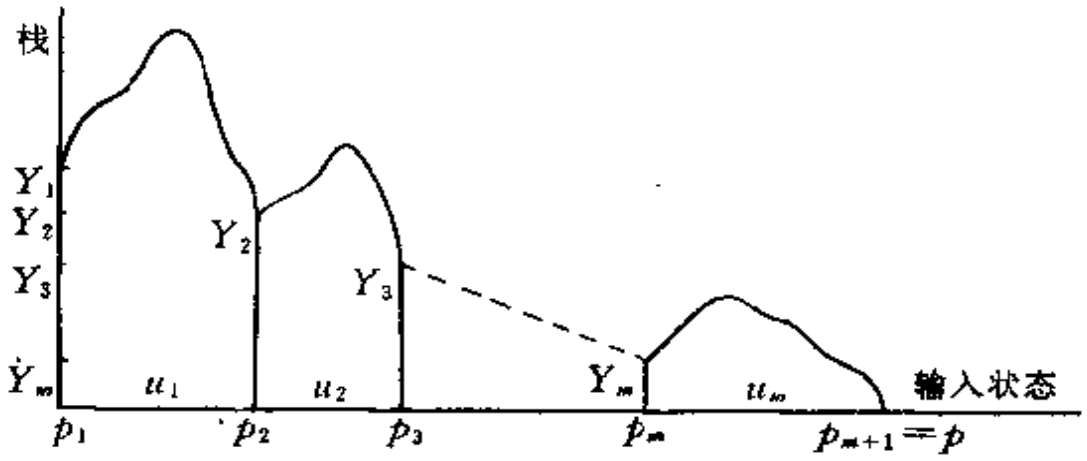


图 10.4

(1) L 是上下文无关语言，即存在上下文无关文法 G 使得 $L=L(G)$；

(2) 存在正上下文无关文法 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$；

(3) 存在 Chomsky 范式文法 G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$；

(4) 存在下推自动机 $\mathscr{M}$ 使得 $L=L(\mathscr{M})$；

(5) 存在下推自动机 $\mathscr{M}$ 使得 $L=N(\mathscr{M})$.

10.5 确定型上下文无关语言

如无特别声明，PDA 都是非确定型的. PDA 接受的语言恰好是 CFL. 本节介绍确定型 PDA，并且证明它接受的语言类是 CFL 的真子集.

定义10.7 设 PDA $\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$，如果：

(1) 对所有的 $q\in Q$ 和 $X\in\Omega$，若 $\delta(q,\varepsilon,X)\neq\varnothing$ 则对所有的 $a\in A,\delta(q,a,X)=\varnothing$，

(2) 对所有的 $q\in Q,a\in A\cup\{\varepsilon\}$ 和 $X\in\Omega,|\delta(q,a,X)|\leqslant 1$，

则称 $\mathscr{M}$ 是**确定型下推自动机**，缩写为 DPDA.

条件(1)保证 $\mathscr{M}$ 不可能对某个 $a\in A$，既有读入 a 的动作、又

有 ε 动作；条件(2)保证 $\mathscr{M}$ 对所有的输入($a\in A$ 或 ε)不可能有两个和两个以上的动作. 给定输入 $w\in A^*$，DPDA $\mathscr{M}$ 的计算是唯一的. 例如，例10.4中的 PDA 是 DPDA. 对于 DPDA，常把 $\delta(q,a,X)=\{(p,\alpha)\}$ 写作 $\delta(q,a,X)=(p,\alpha)$.

定义10.8 确定型下推自动机(以接受状态方式)接受的语言称作**确定型上下文无关语言**，缩写为 DCFL.

下面要证明 DCFL 在补运算下是封闭的，即设 DPDA $\mathscr{M}$，$L=L(\mathscr{M})$，则存在 DPDA $\mathscr{M}'$ 使 $\overline{L}=L(\mathscr{M}')$. 非常自然的想法是用 $\mathscr{M}'$ 模拟 $\mathscr{M}$，并且当 $\mathscr{M}$ 进入接受状态时 $\mathscr{M}'$ 进入非接受状态；当 $\mathscr{M}$ 进入非接受状态时 $\mathscr{M}'$ 进入接受状态. 但是，这里必须先克服一个困难：$\mathscr{M}$ 可能不能读完输入串 w. 如果 $\mathscr{M}$ 不读完输入串 w，$\mathscr{M}$ 当然不接受 w. $\mathscr{M}'$ 模拟 $\mathscr{M}$，也不读完 w，也不接受 w. 这是不对的. 因此，我们要强迫 $\mathscr{M}$ 读完每一个输入串，即对每一个 $w\in A^*$，存在 $p\in Q$ 和 $\alpha\in\Omega^*$ 使得

$$(q_1,w,X_0)\overset{*}{\underset{\mathscr{M}}{\vdash}}(p,\varepsilon,\alpha).$$

$\mathscr{M}$ 不能读完 w 有下述3种情况：

情况1 $w=uav$，$(q_1,w,X_0)\overset{*}{\vdash}(p,av,Z\beta)$ 并且 $\delta(p,a,Z)=\delta(p,\varepsilon,Z)=\varnothing$. $\mathscr{M}$ 在格局 $(p,av,Z\beta)$ 没有动作可做，计算结束.

为了保证不出现这种情况，只需对所有的 $q\in Q$ 和 $X\in\Omega$，若 $\delta(q,\varepsilon,X)=\varnothing$ 则对每一个 $a\in A$，$\delta(q,a,X)\neq\varnothing$.

情况2 $(q_1,w,X_0)\overset{*}{\vdash}(p,u,\varepsilon)$，其中 $u\neq\varepsilon$. 即，$\mathscr{M}$ 尚未读完 w 已把栈排空，计算结束.

情况3 $(q_1,w,X_0)\overset{*}{\vdash}(p,u,\beta)$，$u\neq\varepsilon$，并且

$$(p,u,\beta)\vdash(p_1,u,\beta_1)\vdash(p_2,u,\beta_2)\vdash\cdots.$$

即 $\mathscr{M}$ 从格局 (p,u,β) 开始，不再读入输入符号，而无休止地做 ε 动作.

此时必有

$$(p,\varepsilon,\beta)\vdash(p_1,\varepsilon,\beta_1)\vdash(p_2,\varepsilon,\beta_2)\vdash\cdots.$$

显然，一定存在 $k\geqslant 0$使得所有的 $j>k$，$|\beta_j|\geqslant|\beta_k|$，这里 $\beta=\beta_0$. 不妨设 $k=0$，则 $\beta=Z\alpha$，$\beta_i=\alpha_i\alpha$，$i=1,2,\cdots$. 于是，有

$$(p,\varepsilon,Z)\vdash(p_1,\varepsilon,\alpha_1)\vdash(p_2,\varepsilon,\alpha_2)\vdash\cdots.$$

称这样的格局(p,ε,Z)是循环格局.

不难证明：对输入串 w 出现情况3，当且仅当存在循环格局(p,ε,Z)使得

$$(q_1,w,X_0)\overset{*}{\vdash}(p,u,Z\alpha),\text{其中 } u\neq\varepsilon.$$

引理10.12 设 DPDA$\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$，则存在 DPDA$\mathscr{M}'=(Q',A,\Omega',\delta',q_1',X_0',F')$使得 $L(\mathscr{M}')=L(\mathscr{M})$，并且

(1) 对所有的 $q\in Q'$和 $X\in\Omega'$，若 $\delta'(q,\varepsilon,X)=\varnothing$则对每一个 $a\in A$，$\delta'(q,a,X)\neq\varnothing$，

(2) 对每一个 $w\in A^*$，若$(q_1',w,X_0')\overset{*}{\underset{\mathscr{M}'}{\vdash}}(p,u,\alpha)$则 $\alpha\neq\varepsilon$.

证：取 $Q'=Q\cup\{q_1',d\}$，$\Omega'=\Omega\cup\{X_0'\}$，$F=F'$. 当 $\mathscr{M}$ 的计算出现情况1时，$\mathscr{M}'$进入新添加的状态 d，然后在状态 d 下读完剩余的输入符号并且保持栈符号串不变. 新添加的栈符号 X_0'是栈底符(也是 $\mathscr{M}'$的栈起始符)，从一开始就被放在栈底的位置上. 它不会被清除出栈，从而保证 $\mathscr{M}'$的计算不会出现情况2.

δ'规定如下：

(1) $\delta(q_1',\varepsilon,X_0')=(q_1,X_0,X_0')$.

(2) 对所有的 $p\in Q$，$a\in A$ 和 $Z\in\Omega$，若 $\delta(p,a,Z)=\varnothing$ 且 $\delta(p,\varepsilon,Z)=\varnothing$，则 $\delta'(p,a,Z)=(d,Z)$.

(3) 对所有的 $p\in Q$ 和 $a\in A$，$\delta'(p,a,X_0')=(d,X_0')$.

(4) 对所有的 $a\in A$ 和 $Z\in\Omega'$，$\delta'(d,a,Z)=(d,Z)$.

(5) 对所有的 $p\in Q$，$a\in A\cup\{\varepsilon\}$和 $Z\in\Omega$，若 $\delta'(p,a,Z)$未经(2)定义，则 $\delta'(p,a,Z)=\delta(p,a,Z)$.

规定(1)使 $\mathscr{M}'$的第一步是把 X_0'放在栈底的位置上并开始模拟 $\mathscr{M}$. 规定(5)使 $\mathscr{M}'$模拟 $\mathscr{M}$，直到 $\mathscr{M}$ 读完输入串 w 或出现情况(1)和情况(2)为止. 若 $\mathscr{M}$ 的计算出现情况(1)，则规定(2)使

$\mathscr{M}'$进入状态 d. 若 $\mathscr{M}$ 的计算出现情况(2),此时栈顶符号是 X_0',则规定(3)使 $\mathscr{M}'$也进入状态 d. 在这两种情况下,$\mathscr{M}'$都会在状态 d 下继续读完输入,且保持状态和栈符号串不变. d 是非接受状态,$\mathscr{M}'$不接受 w. 由此可见,$L(\mathscr{M}')=L(\mathscr{M})$.

最后,根据规定(2),$\mathscr{M}'$满足条件(1). 又由前面的分析不难看到,对任何 $w\in A^*$,若$(q_1',w,X_0')\overset{+}{\underset{\mathscr{M}'}{\vdash}}(p,u,\alpha)$,则必有 $\alpha=\beta X_0'$,其中 $\beta\in\Omega^*$. 条件(2)也成立. □

引理10.13 设 DPDA$\mathscr{M}$,则存在 DPDA$\mathscr{M}'$使得 $L(\mathscr{M}')=L(\mathscr{M})$并且对每一个输入,$\mathscr{M}'$最终停机在某个形如$(p,\varepsilon,\alpha)$的格局,其中 $\alpha\neq\varepsilon$.

证: 不妨设 $\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$满足引理10.12中的条件(1)和(2),因此 $\mathscr{M}$ 的计算不会出现情况(1)和(2). 现在要用 $\mathscr{M}'$模拟 $\mathscr{M}$ 并且保证 $\mathscr{M}'$的计算不会出现情况(3). 新添加两个状态 f 和 d,f 是接受状态、而 d 不是接受状态. 即令

$$\mathscr{M}'=(Q\cup\{f,d\},A,\Omega,\delta',q_1,X_0,F\cup\{f\}),$$

其中 δ'规定如下:

(1) 对所有的 $p\in Q,a\in A$ 和 $Z\in\Omega,\delta'(p,a,Z)=\delta(p,a,Z)$.

(2) 对所有的 $p\in Q$ 和 $Z\in\Omega$,若(p,ε,Z)不是循环格局,则 $\delta'(p,\varepsilon,Z)=\delta(p,\varepsilon,Z)$.

(3) 对所有的 $p\in Q$ 和 $Z\in\Omega$,若(p,ε,Z)是循环格局,设

$$(p,\varepsilon,Z)\underset{\mathscr{M}}{\vdash}(p_1,\varepsilon,\alpha_1)\underset{\mathscr{M}}{\vdash}(p_2,\varepsilon,\alpha_2)\underset{\mathscr{M}}{\vdash}\cdots.$$

(3.1) 若 $p,p_1,p_2,\cdots$中没有接受状态,则 $\delta'(p,\varepsilon,Z)=(d,Z)$.

(3.2) 若 $p,p_1,p_2,\cdots$中有接受状态,则 $\delta'(p,\varepsilon,Z)=(f,Z)$.

(4) 对所有的 $a\in A$ 和 $Z\in\Omega,\delta'(f,a,Z)=(d,Z)$.

(5) 对所有的 $a\in A$ 和 $Z\in\Omega,\delta'(d,a,Z)=(d,Z)$.

对任何输入 $w\in A^*$,$\mathscr{M}'$从一开始就模拟 $\mathscr{M}$,直到下述几种情况:

(a) $\mathscr{M}$ 停机在某个格局(p,ε,α). $\mathscr{M}'$对 w 的计算与 $\mathscr{M}$ 的完

全一样. $\mathscr{M}'$ 接受 w 当且仅当 $\mathscr{M}$ 接受 w. 又根据引理10.12, $\alpha\neq\varepsilon$.

(b) $\mathscr{M}$ 到达格局 $(p,u,Z\alpha)$, 其中 (p,ε,Z) 是循环格局. 设

$$(p,\varepsilon,Z)\underset{\mathscr{M}}{\vdash}(p_1,\varepsilon,\alpha_1)\underset{\mathscr{M}}{\vdash}(p_2,\varepsilon,\alpha_2)\underset{\mathscr{M}}{\vdash}\cdots.$$

(b.1) $u=\varepsilon$ 且 $p,p_1,p_2,\cdots$ 中有接受状态. 根据规定(3.2), $\mathscr{M}'$ 从格局 $(p,\varepsilon,Z\alpha)$ 转变成 $(f,\varepsilon,Z\alpha)$, 计算结束.

(b.2) $u=\varepsilon$ 且 $p,p_1,p_2,\cdots$ 中没有接受状态. 根据规定(3.1), $\mathscr{M}'$ 从格局 $(p,\varepsilon,Z\alpha)$ 转变成 $(d,\varepsilon,Z\alpha)$, 计算结束.

(b.3) $u\neq\varepsilon$ 且 $p,p_1,p_2,\cdots$ 中有接受状态. 根据规定(3.2), $\mathscr{M}'$ 从格局 $(p,u,Z\alpha)$ 转变成 $(f,u,Z\alpha)$. 接着按照规定(4), 读入一个输入符号转移到状态 d. 然后按照规定(5), 在状态 d 下读完输入后停机在格局 $(d,\varepsilon,Z\alpha)$.

(b.4) $u\neq\varepsilon$ 且 $p,p_1,p_2,\cdots$, 中没有接受状态. 与(b.3)类似, $\mathscr{M}'$ 从格局 $(p,u,Z\alpha)$ 转变成 $(d,u,Z\alpha)$, 然后在状态 d 下读完输入后停机在格局 $(d,\varepsilon,Z\alpha)$.

$\mathscr{M}$ 和 $\mathscr{M}'$ 都仅在情况(b.1)接受 w.

根据上述分析, $\mathscr{M}'$ 满足引理的要求. □

定理10.4 确定型上下文无关语言的补集也是确定型上下文无关的.

证: 设 DPDA $\mathscr{M}=(Q,A,\Omega,\delta,q_1,X_0,F)$. 根据引理10.13, 不妨设对任何输入 $w\in A^*$, $\mathscr{M}$ 都最终停机在某个格局 (p,ε,α). 要构造 DPDA $\mathscr{M}'$ 使得 $L(\mathscr{M}')=A^*-L(\mathscr{M})$. 现在的困难是, $\mathscr{M}$ 在读完 w 之后可能再做几个 ε 动作, 然后才停机:

$$(p,\varepsilon,\alpha)\underset{\mathscr{M}}{\vdash}(p_1,\varepsilon,\alpha_1)\underset{\mathscr{M}}{\vdash}\cdots\underset{\mathscr{M}}{\vdash}(p_t,\varepsilon,\alpha_t).$$

这使得我们不能用简单地交换接受状态和非接受状态(即令 $F'=\Omega-F$)来达到接受 $L(\mathscr{M})$ 的补集的目的. 因为如果 $p,p_1,\cdots,p_t$ 中既有接受状态、又有非接受状态, 则 $\mathscr{M}$ 接受 w, 交换接受状态和非接受状态之后也接受 w. 解决的办法是在状态中引入第二个分量, 用来记录 $\mathscr{M}$ 自读入上一个输入符号以来是否进入过接受状态. 如果没有进入过接受状态, 则在读下一个输入符号之前 $\mathscr{M}'$ 转

移到自己的接受状态.

令 $\mathscr{M}'=(Q',A,\Omega,\delta',q_1',X_0,F')$,

其中

$$Q'=\{[q,k]\mid q\in Q,k=1,2,3\},$$

$$F'=\{[q,3]\mid q\in Q\},$$

$$q_1'=\begin{cases}[q_1,1] & \text{若 } q_1\in F,\\ [q_1,2] & \text{否则.}\end{cases}$$

$[q,k]$中的 k 用来记录 $\mathscr{M}$ 在读入两个输入符号之间是否进入过接受状态. 如果自读入上一个输入符号以来,$\mathscr{M}$ 进入过接受状态,则 $k=1$;如果自读入上一个输入符号以来,$\mathscr{M}$ 尚未进入过接受状态,则 $k=2$. $\mathscr{M}'$ 模拟 $\mathscr{M}$ 的动作且根据各种情况改变 k 的值. 当 $k=2$时,$\mathscr{M}'$ 先将 k 的值变成3,然后才模拟 $\mathscr{M}$ 读入输入符号的动作.

δ' 规定如下:对所有的 $q\in Q,a\in A$ 和 $Z\in\Omega$,

(1) 若 $\delta(q,\varepsilon,Z)=(p,\alpha)$,则对于 $k=1$和2,

$$\delta'([q,k],\varepsilon,Z)=([p,k'],\alpha),$$

其中当 $k=1$或 $p\in F$ 时 $k'=1$,否则 $k'=2$.

(2) 若 $\delta(q,a,Z)=(p,\alpha)$,则

$$\delta'([q,2],\varepsilon,Z)=([q,3],Z),$$

$$\delta'([q,1],a,Z)=\delta'([q,3],a,Z)=([p,k],\alpha),$$

其中当 $p\in F$ 时 $k=1$,当 $p\notin F$ 时 $k=2$.

设 $w\in L(\mathscr{M})$,则 $\mathscr{M}$ 对 w 的计算为

$$(q_1,w,X_0)\overset{*}{\underset{\mathscr{M}}{\vdash}}(p_1,\varepsilon,\alpha_1)\underset{\mathscr{M}}{\vdash}\cdots\underset{\mathscr{M}}{\vdash}(p_t,\varepsilon,\alpha_t),\quad(*)$$

其中 p_1是 $\mathscr{M}$ 读完 w 时进入的状态(若 $w=\varepsilon$,则 $p_1=q_1$),$p_1,p_2,\cdots,p_t$ 中有接受状态. 设 $p_1,\cdots,p_{i-1}\notin F,p_i\in F$,则 $\mathscr{M}'$ 对 w 的计算为

$$(q_1',w,X_0)\overset{*}{\underset{\mathscr{M}'}{\vdash}}([p_1,2],\varepsilon,\alpha_1)\underset{\mathscr{M}'}{\vdash}\cdots$$

$$\underset{\mathscr{M}'}{\vdash}([p_{i-1},2],\varepsilon,\alpha_{i-1})$$

$$\vdash_{\mathscr{M}'} ([p_t,1],\varepsilon,\alpha_t) \vdash_{\mathscr{M}'} \cdots$$
$$\vdash_{\mathscr{M}'} ([p_t,1],\varepsilon,\alpha_t).$$

$\mathscr{M}'$不接受 w.

设 $w\in A^*-L(\mathscr{M})$，$\mathscr{M}$ 对 w 的计算为(*)，和前面不同的是，$p_1,p_2,\cdots,p_t$ 都是非接受状态.

于是，

$$(q_1',w,X_0) \overset{*}{\vdash}_{\mathscr{M}'} ([p_t,2],\varepsilon,\alpha_t),$$

其中 $\alpha_t\neq\varepsilon$，设 $\alpha_t=Z\beta$. 因为 $\mathscr{M}$ 停机在格局$(p_t,\varepsilon,Z\beta)$，必有 $\delta(p_t,\varepsilon,Z)=\varnothing$. 根据引理10.12，对所有的 $a\in A$，$\delta(p_t,a,Z)\neq\varnothing$. 由规定(2)，得

$$(q_1',w,X_0) \overset{*}{\vdash}_{\mathscr{M}'} ([p_t,2],\varepsilon,\alpha_t) \vdash_{\mathscr{M}'} ([p_t,3],\varepsilon,\alpha_t),$$

$\mathscr{M}'$接受 w. □

[**例10.6**] $L=\{a,b,c\}^*-\{a^nb^nc^n\mid n\geqslant 1\}$是 CFL、但不是 DCFL.

证：先证 L 不是 DCFL. 假若 L 是 DCFL，由定理10.14，$\overline{L}=\{a^nb^nc^n\mid n\geqslant 1\}$也是 DCFL，当然是 CFL. 但是我们已经知道 $\overline{L}$ 不是 CFL(例10.3)，故 L 不是 DCFL.

可以把 L 划分成3部分 $L=L_1\cup L_2\cup L_3$，其中

$$L_1=\{a,b,c\}^*-\{a^ib^jc^k\mid i,j,k\geqslant 1\},$$
$$L_2=\{a^ib^jc^k\mid i\neq j\},$$
$$L_3=\{a^ib^jc^k\mid j\neq k\}.$$

不难构造出接受 L_1的 FA、生成 L_2和 L_3的 CFG，证明两个 CFL 的并仍是CFL，从而证得 L 是 CFL.

这个例子表明，DCFL 类是 CFL 类的真子集.

习　题

1. 给出生成下述语言的 CFG：

(1) $\{a^ib^j \mid i \geqslant j > 0\}$.

(2) $\{a^ib^{2i} \mid i > 0\}$.

(3) $\{a^ib^jc^k \mid i \neq j$ 或 $j \neq k\}$.

2. 给出生成上题(a)，(b)中语言的 Chomsky 范式文法.

3. 给出生成字母表$\{a,b\}$上所有正则表达式的 CFG.

4. 设文法 G 的产生式为：

$$S \to aS, \quad S \to aSbS, \quad S \to \varepsilon.$$

证明：$L(G)$为$\{a,b\}$上所有满足下述条件的字符串 x 的集合，在 x 的每个前缀中 a 的个数不少于 b 的个数.

5. 证明：如果 L_1, L_2是 CFL，则 $L_1 \cup L_2, L_1 \cdot L_2$和 L_1^* 也是 CFL.

6. 举例说明 CFL 类在交运算和补运算下不是封闭的.

7. 证明下述语言不是 CFL：

(1) $\{a^i \mid i$ 是素数$\}$.

(2) $\{a^{i^2} \mid i > 0\}$.

(3) $\{a^ib^jc^k \mid 0 < i < j < k\}$.

(4) $\{a^ib^jc^id^j \mid i, j > 0\}$.

(5) $\{ww \mid w \in \{a,b\}^*\}$.

8. 证明：仅含一个符号的字母表上的 CFL 是正则的.

9. 构造接受下述语言的 PDA：

(1) $\{a,b\}$上所有 a 的个数是 b 的个数的2倍的字符串的集合.

(2) $\{w \mid w \in \{a,b\}^*$且 $w = w^R\}$.

第十一章　上下文有关语言

11.1　上下文有关文法

根据定义上下文有关文法都是0型文法，故上下文有关语言都是递归可枚举语言. 实际上，能进一步证明上下文有关语言都是递归的.

引理 11.1　设 CSG $G=(V,T,\Gamma,S)$，则

$$\{u \mid S\overset{*}{\Rightarrow}u\}$$

是递归的.

证：设 $|V\cup T|=n$，在 7.3 节已知

$$S\overset{*}{\Rightarrow}u \quad \Leftrightarrow \quad (\exists\ y)\mathrm{DERIV}(u,y).$$

现在要证明可以把右端的量词加强为有界存在量词，从而 $S\overset{*}{\Rightarrow}u$ 是原始递归的.

设 $S\Rightarrow u_1\Rightarrow u_2\Rightarrow\cdots\Rightarrow u_m=u$，因为 G 是 CSG，故

$$1\leqslant|u_1|\leqslant|u_2|\leqslant\cdots\leqslant|u_m|=|u|.$$

以 n 为底、长度为 $|u|+1$ 的最小自然数是

$$g(u)=\sum_{i=0}^{|u|}n^i.$$

$g(u)$是原始递归函数. 显然，$u_j<g(u)$，$1\leqslant j\leqslant m$. 还可以假设 u_1，$u_2,\cdots,u_m$ 是不相同的. 否则，若 $u_i=u_j(i<j)$，则可以删去 $u_{i+1},\cdots,u_j$，仍然是 S 到 u 的派生. 在 $V\cup T$ 上长度不超过 $|u|$ 的不同的字符串的个数为 $g(u)$，因此 $m\leqslant g(u)$. 从而

$$[u_1,\cdots,u_m,1]=\prod_{i=1}^{m}p_i^{u_i}\cdot p_{m+1}\leqslant h(u),$$

其中

$$h(u)=\prod_{i=1}^{g(u)}\prod p_i^{g(u)}\cdot p_{g(u)+1},$$

$h(u)$也是原始递归函数. 由此可知

$$S\dot{\Rightarrow}u \quad\Leftrightarrow\quad (\exists\ y)_{\leqslant h(u)}\mathrm{DERIV}(u,y).$$

得证 $S\dot{\Rightarrow}u$ 是原始递归的. □

定理 11.2 上下文有关语言是递归的.

证：设 CSG $G=(V,T,\Gamma,S)$,

$$L(G)=T^*\cap\{u\mid S\dot{\Rightarrow}u\}.$$

T^* 显然是递归的，由引理 11.1，得证 $L(G)$是递归的. □

引理 11.3 存在非上下文有关的递归语言.

证：考虑字母表{1}上的语言. 令

$$\Sigma=\{1,V,b,\rightarrow,/\}.$$

用 Σ 上的字符串给以{1}为终极符集的 CSG 编码，方法如下：设文法有 k 个变元，记作 $S=V_1,V_2,\cdots,V_k$，分别用 $Vb,Vbb,\cdots,Vb^k$ 作为它们的代码. 把产生式中每一个变元换成它的代码得到这个产生式的代码. 将产生式的代码任意排列并用/分隔开相邻的两个产生式就得到文法的代码. 例如，CSG G：

$$S\rightarrow 1SA,1S\rightarrow 1A1,1AA\rightarrow 111$$

的代码是

$$Vb\rightarrow 1VbVbb/1Vb\rightarrow 1Vbb1/1VbbVbb\rightarrow 111,$$

这里 S 的代码是 Vb，A 的代码是 Vbb. 当然，G 的代码不唯一.

把 Σ 上的字符串看作 5 进制数，按照从小到大的顺序排列所有可以成为 CSG 的代码的字符串. 第 i 个字符串所表示的 CSG 记作 G_i. 令 $L_i=L(G_i)$. $L_1,L_2,\cdots$枚举出所有的 CSL. 令

$$L=\{1^i\mid 1^i\notin L_i,i>0\}.$$

L 不是 CSL. 假若不然，则存在 $k>0$ 使 $L=L_k$. 于是，

$1^k\in L$ 当且仅当 $1^k\notin L_k$ （L 的定义）

当且仅当 $1^k\in L$， （$L=L_k$）

矛盾.

最后要证明 L 是递归的. 关于 L 的成员资格问题的算法如下：对于任给的 $i \geqslant 1$，首先逐个把 1，2，3，…表示成 Σ 上的字符串并检查它是否是一个 CSG 的代码，直至找到 G_i 为止. 然后判断 G_i 是否接受 1^i. 由 CSL 的递归性，存在这样的算法. $1^i \in L$ 当且仅当 G_i 不接受 1^i. □

现在将本书讨论过的语言类之间的包含关系总结如下.

定理 11.4 按下述顺序每一个语言类真包含它后面的语言类，它们是：

(1) 递归可枚举语言，即 0 型语言；

(2) 递归语言；

(3) 上下文有关语言，即 1 型语言；

(4) 上下文无关语言，即 2 型语言；

(5) 确定型上下文无关语言；

(6) 正则语言，即 3 型语言.

证：设 L 是 CFL，则存在正 CFG G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$. 正 CFG 是 CSG，故 L 是 CSL. 得证(3)包含(4).

每一台 DFA 都可以看作一台 DPDA，它只有一个栈符号并且始终放在栈顶不动，故(5)包含(6). 其余的包含关系或已经证明，或是显然的.

定理 5.10、引理 11.3、例 10.3、例 10.6 和例 9.8 分别给出这些语言类不相等的例子或证明，其中例 9.8 中的语言 $\{a^nb^n \mid n>0\}$ 不仅是 CFL、也是 DCFL. 不难构造出接受这个语言的 DPDA. □

11.2 线性界限自动机

线性界限自动机(缩写为 LBA)是一种带的使用范围受到严格限制的非确定型 Turing 机，它有两个特殊的带符号 λ 和 ρ，分别叫做**左端标志符**和**右端标志符**，满足下述条件：

(1) 扫描 λ 和 ρ 时，分别只可能右移读写头和左移读写头；

(2) 不打印 λ 和 ρ.

一台 LBA $\mathscr{M}$ 可以写成 $\mathscr{M}=(Q,A,C,\delta,B,q_1,\lambda,\rho,F)$，其中 $B.\lambda,\rho\in C-A,F\subseteq Q$ 是接受状态集.

设输入串 $w\in A^*$，$\mathscr{M}$ 的初始格局为

$$\begin{array}{c}\lambda w\rho\\ \uparrow\\ q_1\end{array}$$

如果从初始格局开始 $\mathscr{M}$ 能够进入某个接受状态 $f\in F$ 则 $\mathscr{M}$ **接受** w，否则不接受 w. $\mathscr{M}$ **接受的语言**是它接受的 A 上字符串的全体，记作 $L(\mathscr{M})$.

线性界限自动机在计算过程中只使用输入串占用的带方格. 可以证明，它的能力与限制使用的带方格数为输入长度的线性函数(即空间复杂度为线性函数)的非确定型 Turing 机的能力相同. 这就是线性界限自动机的名字的来由. 下面证明这种受限制的 NDTM 与 CSG 是等价的.

引理 11.5 设 G 是一个 CSG，则存在 LBA $\mathscr{M}$ 使得 $L(\mathscr{M})=L(G)$.

证：设 CSG $G=(V,T,\Gamma,S)$，它有 m 个产生式 $g_i\to h_i$，其中

$$g_i=\alpha_1^{(i)}\cdots\alpha_{k_i}^{(i)},\quad h_i=\beta_1^{(i)}\cdots\beta_{l_i}^{(i)},$$

$\alpha_1^{(i)},\cdots,\alpha_{k_i}^{(i)},\beta_1^{(i)},\cdots,\beta_{l_i}^{(i)}\in T\cup V$，并且 $k_i\leqslant l_i,i=1,2,\cdots,m$. 记空白符 $s_0,s_0\notin T\cup V$. 令

$$\alpha_j^{(i)}=s_0,\quad k_i+1\leqslant j\leqslant l_i,\quad i=1,2,\cdots,m.$$

于是，每个 g_i 都变得和 h_i 一样长.

取 LBA $\mathscr{M}=(Q,T,C,\delta,s_0,q_1,\lambda,\rho,F)$，

其中

$$\begin{aligned}C&=V\cup T\cup\{s_0,\lambda,\rho\},\\ Q&=\{q_1,\sigma,\bar{\sigma},\tau,\bar{\tau},f\}\\ &\quad\cup\{p_j^{(i)},q_j^{(i)}\mid i=1,2,\cdots,m,j=1,2,\cdots,l_i\},\\ F&=\{f\},\end{aligned}$$

Q 中的元素按照其职能叫做初始状态 q_1，检查状态 σ，模拟状态 $p_j^{(i)}$ 和 $q_j^{(i)}$，返回状态 $\bar{\sigma}$，结束状态 τ 和 $\bar{\tau}$，接受状态 f.

$\mathscr{M}$ 反复地从 λ 扫描到 ρ，再返回到 λ. 每一次模拟一个产生式 $g_i \to h_i$ 的作用，逐个把 $\beta_j^{(i)}$ 改写成 $\alpha_j^{(i)}$ $(j=1,2,\cdots,l_i)$，从而把 h_i 改写成 g_i. 最后，如果在 λ 和 ρ 之间除 s_0 之外恰好有一个 S，则 $\mathscr{M}$ 进入接受状态 f 并停机. 每一次巡查(从 λ 向右检查到 ρ，再从 ρ 向左返回到 λ)分为 4 个阶段：初始阶段、检查阶段、模拟阶段和返回阶段. 最后还有一个结束阶段. δ 由下述 4 元组给出：

初始阶段，对每一个 $a \in V \cup T \cup \{s_0\}$，

$$q_1 a a \sigma$$

$$q_1 a a \tau$$

以及

$$q_1 \lambda R q_1$$

$\mathscr{M}$ 从状态 q_1 非确定地进入检查状态 σ 或结束状态 τ.

检查阶段

$\sigma a R \sigma$，对每一个 $a \in V \cup T \cup \{s_0\}$

$\sigma \rho L \bar{\sigma}$，

$\sigma \beta_1^{(i)} \beta_1^{(i)} p_1^{(i)}$，$1 \leqslant i \leqslant m$.

$\mathscr{M}$ 或者向右继续检查，或者在遇到 $\beta_1^{(i)}$ 时转入模拟产生式 $g_i \to h_i$ 的状态. 如果向右一直检查到 ρ，则进入返回状态 $\bar{\sigma}$.

模拟阶段，对 $i=1,2,\cdots,m$，

$$\left.\begin{matrix} p_j^{(i)} \beta_j^{(i)} \alpha_j^{(i)} q_j^{(i)} \\ q_j^{(i)} \alpha_j^{(i)} R p_{j+1}^{(i)} \end{matrix}\right\} 1 \leqslant j < l_i$$

$$p_{l_i}^{(i)} \beta_{l_i}^{(i)} \alpha_{l_i}^{(i)} \bar{\sigma}$$

以及

$$p_j^{(i)} s_0 R p_j^{(i)}, \quad 1 \leqslant j \leqslant l_i$$

$\mathscr{M}$ 将某个 h_i 改写成 g_i(忽略所有的 s_0)后进入返回状态 $\bar{\sigma}$.

返回阶段

$$\bar{\sigma}aL\bar{\sigma}, \text{对每一个 } a \in V \cup T \cup \{s_0\}$$

$$\bar{\sigma}\lambda Rq_1.$$

$\mathscr{M}$ 返回到左端并重新进入初始状态 q_1，准备做下一轮巡查或准备结束.

结束阶段

$$\tau s_0 R\tau$$

$$\tau S R\bar{\tau}$$

$$\bar{\tau}s_0 R\bar{\tau}$$

$$\bar{\tau}\rho L f$$

$\mathscr{M}$ 能够进入接受状态 f 当且仅当 $\mathscr{M}$ 进入结束状态 τ 时带的内容为

$$\lambda s_0' S s_0' \rho, \quad i, j \geqslant 0.$$

根据前面的叙述和分析，不难证明

$$\mathscr{M} \text{ 接受 } w \quad \Leftrightarrow \quad S \overset{*}{\underset{G}{\Rightarrow}} w. \qquad \square$$

现在考虑相反的问题：用 CSG 模拟 LBA. 在第七章已经成功地用半 Thue 过程模拟 Turing 机. 现在不同的是要求每一个产生式的右端的长度不小于左端的长度. 在第七章，用 Post 字描述 Turing 机的格局，Post 字中需要两个端标志和一个状态，并在模拟的最后消去这些符号. 这势必要减少字符串的长度. 为了保证派生过程字符串的长度不减小，必须设法用和带内容长度相同的字符串描述 LBA 的格局.

设 LBA $\mathscr{M}=(Q, A, C, \delta, q_1, s_0, \lambda, \rho, F)$，这里不妨设只有一个接受状态，$F=\{f\}$. 构造半 Thue 过程 Σ 如下：记

$$\bar{C} = \{a, {}^{[}a, a^{]}, \overleftarrow{a}, \overrightarrow{a} \mid a \in C'\},$$

其中 $C'=C-\{\lambda, \rho\}$. 这些符号的含义是

${}^{[}a$： a 在字符串的左端；

$a^{]}$： a 在字符串的右端；

$\overleftarrow{a}$： a 在字符串的左端并且正在扫描 λ；

$\vec{a}$：　a 在字符串的右端并且正在扫描 ρ.

Σ 的字母表

$$\mathscr{C}=\{x_q|x\in\tilde{C},q\in Q\}\cup\tilde{C}\cup\{S\}.$$

当 $x=a,[a,a]$ 时，x_q 表示当前状态为 q，扫描该符号；当 $x=\overleftarrow{a},\vec{a}$ 时，x_q 表示当前状态为 q，扫描 λ 或 ρ.

当 $|w|\geqslant 2$ 时，可以用 $\mathscr{C}$ 上的字符串描述 $\mathscr{M}$ 的格局，并且其长度为 $|w|$. 例如

格局	代码
$\lambda abab\rho$ ↑ q	$[aba_qb]$
$\lambda abab\rho$ ↑ q	$[a_qbab]$
$\lambda abab\rho$ ↑ q	$\overleftarrow{a}_qbab]$
$\lambda abab\rho$ ↑ q	$[abab_q]$
$\lambda abab\rho$ ↑ q	$[ab\vec{a}b_q$

Σ 的产生式如下，它们和 $\mathscr{M}$ 的4元组相对应(除第(6)组之外)：

$\mathscr{M}$ 的4元组	Σ 的产生式
(1) $qabp,a,b\in C'$	$a_q\to b_p$ $[a_q\to[b_p$ $a_q]\to b_p]$

续表

$\mathscr{M}$ 的4元组	Σ 的产生式
(2) $qaRp, a\in C'$	$a_qb\to ab_p$ $[a_qb\to[ab_p$ $a_qb]\to ab_p]$ $[a_qb]\to[ab_p]$ 对所有的 $b\in C'$ $a_q]\to\vec{a}_p$
(3) $q\lambda Rp$	$\overleftarrow{a}_q\to[a_p$ 对所有的 $a\in C'$
(4) $qaLp, a\in C'$	$ba_q\to b_pa$ $ba_q]\to b_pa]$ $[ba_q\to[b_pa$ $[ba_q]\to[b_pa]$ 对所有的 $b\in C'$ $[a_q\to\overleftarrow{a}_p$
(5) $q\rho Lp$	$\vec{a}_q\to a_p]$ 对所有的 $a\in C'$
(6)	$x_f\to S$ $xS\to S$ $Sx\to S$ 对所有的 $x\in\bar{C}$

引理11.6 对于所有的 $a,b\in A$，$u\in A^*$，

$$\mathscr{M}\text{ 接受 }aub \iff \overleftarrow{a}_{q_1}ub]\overset{*}{\underset{\Sigma}{\Rightarrow}}S.$$

证：根据前5组产生式，不难证明从初始格局

$$\begin{array}{c}\lambda aub\rho\\ \uparrow\\ q_1\end{array}$$

开始，$\mathscr{M}$ 能够进入接受状态 f，当且仅当存在 $x\in\bar{C},\xi,\eta\in\bar{C}^*$ 使得

$$\overleftarrow{a}_{q_1}ub]\overset{*}{\underset{\Sigma}{\Rightarrow}}\xi x_f\eta.$$

第(6)组产生式使得 Σ 可以从 $\xi x_f\eta$ 派生出 S. 又注意到只有第(6)组产生式与 S 有关并且第一次出现 S 一定是从某个 x_f 变换来的，故结论成立. □

定理11.7 L 是上下文有关语言当且仅当存在LBA $\mathscr{M}$ 使得 $L=L(\mathscr{M})$.

证：设 L 是CSL，则存在CSG G 使得 $L=L(G)$ 或 $L=L(G)\cup\{\varepsilon\}$. 由引理11.5，存在LBA $\mathscr{M}$ 使得 $L(\mathscr{M})=L(G)$. 若 $L=L(G)$，则有 $L=L(\mathscr{M})$. 若 $L=L(G)\cup\{\varepsilon\}$，则对引理11.5证明中的LBA $\mathscr{M}$ 添加一个4元组

$$q_1\lambda Rf.$$

修改后的LBA接受 ε 和 $\mathscr{M}$ 接受的字符串，从而接受 L.

设LBA $\mathscr{M}=(Q,A,C,\delta,q_1,s_0,\lambda,\rho,F)$，这里不妨设只有一个接受状态，$F=\{f\}$. 引理11.6中的半Thue过程 Σ 的每一个产生式 $g\to h$ 均满足 $|g|\geqslant|h|$.

构造CSG $G=(V,A,\Gamma,\overleftarrow{S})$，其中

$$V=(\mathscr{C}-A)\cup\{a^0\mid a\in A\}\cup\{\overleftarrow{S}\},$$

Γ 包括 Σ 中所有产生式的逆以及

(1) $\left.\begin{array}{l}\overleftarrow{a}_{q_1}\to a^0\\ a^0b\to ab^0\\ a^0b^1\to ab\end{array}\right\}$对所有的 $a,b\in A$.

(2) $\overleftarrow{S}\to S$.

(3) $\overleftarrow{S}\to a$，对所有的 $a\in A\cap L(\mathscr{M})$.

根据引理11.6，对所有的 $a,b\in A$ 和 $u\in A^*$，有

$$\mathscr{M}\text{ 接受 }aub\quad\Leftrightarrow\quad\overleftarrow{S}\underset{G}{\Rightarrow}S\overset{*}{\underset{G}{\Rightarrow}}\overleftarrow{a}_{q_1}ub^1\overset{*}{\underset{G}{\Rightarrow}}aub.$$

即，对所有的 $w\in A^*$，若 $|w|\geqslant 2$ 则

$$w\in L(\mathscr{M})\quad\Leftrightarrow\quad w\in L(G).$$

而第(3)组产生式保证，对所有的 $a\in A$，

$$a\in L(\mathscr{M})\quad\Leftrightarrow\quad a\in L(G).$$

所以有 $L(\mathscr{M})=L(G)$. □

至此我们已经介绍了4种类型的文法：短语结构文法、上下文有关文法、上下文无关文法和正则文法，以及与它们等价的4种自

动机：Turing 机、线性界限自动机、下推自动机和有穷自动机. 每一种自动机都可以有非确定型和确定型. 前面已经证明，对于 Turing 机和有穷自动机，确定型和非确定型是等价的. 而确定型下推自动机接受的语言是非确定型下推自动机接受的语言的真子集. 对于线性界限自动机，如无特别声明都是指非确定型的. 当然也有确定型线性界限自动机. 人们自然也要问：确定型线性界限自动机与非确定型线性界限自动机等价吗？即，它们接受的语言类相同吗？这就是所谓的**"线性界限自动机问题"**，它是一个重要而又非常困难的问题，至今尚未解决.

最后说明一下"上下文有关"和"上下文无关"的来源. 可以证明任一上下文有关文法都等价于这样一个文法，它的产生式都形如

$$\varphi A\psi \to \varphi\alpha\psi,$$

其中 $A\in V$，$\varphi,\psi,\alpha\in(V\cup T)^*$ 且 $\alpha\neq\varepsilon$.

产生式 $\varphi A\psi\to\varphi\alpha\psi$ 表示只有当变元 A 的上下文分别是 φ 和 ψ 时才能被改写成 α. 这种改写规则是和上下文有关的. 而在上下文无关文法中，产生式都形如

$$A \to \alpha,$$

运用它把 A 改写成 α 时不需要考虑 A 的上下文，这种改写规则是和上下文无关的.

习　题

1. 证明：对于每一个 CSG $G=(V,T,\Gamma,S)$ 都存在 CSG $G'=(V',T,\Gamma',S')$ 使得 $L(G)=L(G')$ 并且 G' 的产生式都形如

$$\varphi A\psi \to \varphi\alpha\psi,$$

其中 $A\in V'$，$\varphi,\psi,\alpha\in(V'\cup T)^*$ 且 $\alpha\neq\varepsilon$.

2. 证明：对于每一个文法 G 都存在文法 G' 使得 $L(G)=L(G')$ 并且 G' 的产生式的左端不含终极符.

3. 设 L_1, L_2 是 CSL，证明 $L_1 \cup L_2, L_1 \cap L_2, L_1 \cdot L_2, L_1^*$ 也是 CSL.（提示：利用上题.）

4. 设 $L \subseteq A^*$ 是 r. e. 语言，证明存在 CSL $L' \subseteq (A \cup \{c\})^*$ 使得对所有的 $w \in A^*$，有

$$w \in L \quad \Leftrightarrow \quad (\exists\, i \in N) wc^i \in L',$$

这里 $c \notin A$.

5. 证明对于每一个 r. e. 语言 L 存在 CSG G，使得 G 加一个形式为 $V \to \epsilon$ 的产生式后得到的文法生成 L.（提示：利用上题并且把 c 作为变量.）

6. 证明每一个 CFL 被确定型 LBA 接受.

第十二章　计算复杂性

12.1　Turing 机的运行时间和工作空间

先给出一个关于数量级的记号. 设 f 和 g 是 N 到 N 的函数，如果存在实数 $c>0$ 使得除有限个 n 外都有 $g(n)\leqslant cf(n)$，则记作 $g=O(f)$. 如果 $g=O(f)$ 且 $f=O(g)$，则记作 $g=\Theta(f)$.

本章使用离线多带 Turing 机作为标准的计算模型(见 6.4.4 小节)，并且对模型做如下规定：

(1) 工作带都是单向无穷的，有最左单元，向右是无穷的. 在最左单元内固定存放着左端标志符 #.

(2) 指定一个状态叫做接受状态，记作 q_Y. 接受字符串 w 当且仅当 Turing 机关于 w 的计算停机在 q_Y. 在下述两种情况下都不接受 w：关于 w 的计算永不停机或者停机在非接受状态. 这种接受方式称作**状态接受方式**. 状态接受方式和标准的**停机接受方式**是等价的(见第六章习题第 5 题). 但是，按照状态接受方式，对于每一个递归语言 L 存在总停机(对所有输入都停机)的 Turing 机接受 L(见第六章习题第 6 题).

(3) 当 Turing 机用来计算函数时，再给它附加一条只写输出带. 输出带的带头只写不读并且每写一个符号后自动右移一格. 当 Turing 机停机时(不必停在 q_Y)，输出带上的内容即是它计算出来的函数值.

确定型和非确定型的离线多带 Turing 机也分别缩写为 DTM 和 NTM.

Turing 机的计算步数称作它使用的时间，在工作带上使用的方格数称作它使用的空间. 时间和空间是 Turing 机计算(接受字

符串或计算函数值)时必需的两种最重要的资源. 当然还可以讨论其他资源,如带头来回的次数. 本章将要讨论时间和空间受限制的 Turing 机所接受的语言类.

定义 12.1 设 DTM $\mathscr{M}$,输入字母表为 A.

(1) 设 $w\in A^*$,如果 $\mathscr{M}$ 对 w 的计算停机,则把计算中使用的总步数记作 $t_{\mathscr{M}}(w)$;否则 $t_{\mathscr{M}}(w)$没有定义,且记作 $t_{\mathscr{M}}(w)=\infty$. $t_{\mathscr{M}}(w)$称作 $\mathscr{M}$ 关于 w 的计算使用的**时间**.

(2) $\mathscr{M}$ 的**时间复杂度**定义为

$$t_{\mathscr{M}}(n)=\max\{t_{\mathscr{M}}(w)\mid w\in A^* \text{ 且 } |w|\leqslant n\},$$

这里规定对任意的 $k\in N$,$\max\{\infty,k\}=\infty$.

(3) 设 $w\in A^*$,如果 $\mathscr{M}$ 对 w 的计算在每一条工作带上只使用有限个方格,则把每一条工作带上被使用的方格数(不计左端存放 # 的方格在内)的最大值记作 $s_{\mathscr{M}}(w)$;否则 $s_{\mathscr{M}}(w)$没有定义,且记作 $s_{\mathscr{M}}(w)=\infty$. $s_{\mathscr{M}}(w)$称作 $\mathscr{M}$ 关于 w 的计算使用的**空间**.

(4) $\mathscr{M}$ 的**空间复杂度**定义为

$$s_{\mathscr{M}}(n)=\max\{s_{\mathscr{M}}(w)\mid w\in A^* \text{ 且 } |w|\leqslant n\}.$$

定义 12.2 设 NTM $\mathscr{M}$,输入字母表为 A.

(1) 设 $w\in A^*$,如果 $\mathscr{M}$ 对 w 的每一个计算都停机,则把这些计算使用的步数的最大值记作 $t_{\mathscr{M}}(w)$;否则 $t_{\mathscr{M}}(w)$没有定义,且记作 $t_{\mathscr{M}}(w)=\infty$. $t_{\mathscr{M}}(w)$称作 $\mathscr{M}$ 关于 w 的计算使用的**时间**.

(2) $\mathscr{M}$ 的**时间复杂度**定义为

$$t_{\mathscr{M}}(n)=\max\{t_{\mathscr{M}}(w)\mid w\in A^* \text{ 且 } |w|\leqslant n\}.$$

(3) 设 $w\in A^*$,如果 $\mathscr{M}$ 对 w 的一个计算在每一条工作带上只使用有限个方格,则把这个计算在每一条工作带上使用的方格数(不计左端存放 # 的方格在内)的最大值称作这个计算使用的空间. 如果 $\mathscr{M}$ 关于 w 的每一个计算都使用有限的空间,则把这些计算使用的最大空间记作 $s_{\mathscr{M}}(w)$;否则 $s_{\mathscr{M}}(w)$没有定义,且记作 $s_{\mathscr{M}}(w)=\infty$. $s_{\mathscr{M}}(w)$称作 $\mathscr{M}$ 关于 w 的计算使用的**空间**.

(4) $\mathscr{M}$ 的**空间复杂度**定义为

$$s_{\mathscr{M}}(n)=\max\{s_{\mathscr{M}}(w)\mid w\in A^{*}\text{ 且 }|w|\leqslant n\}.$$

定义 12.3 设 $\mathscr{M}$ 是一台(确定型或非确定型)Turing 机，$f(n)$ 是 N 到 N 的全函数. 如果对所有的 n，$t_{\mathscr{M}}(n)\leqslant f(n)$，则称 $\mathscr{M}$ 是 **$f(n)$时间界限的**，又称 $f(n)$是 $\mathscr{M}$ 的**时间复杂度上界**. 如果对所有的 n，$s_{\mathscr{M}}(n)\leqslant f(n)$，则称 $\mathscr{M}$ 是 **$f(n)$空间界限的**，又称 $f(n)$是 $\mathscr{M}$ 的**空间复杂度上界**.

下面对上述定义作一些说明，并对复杂度上界作一些特殊规定.

(1) 这里定义的时间复杂度和空间复杂度都是“最坏情况”的复杂度. 若 $\mathscr{M}$ 是 $t(n)$时间界限的 DTM，则对所有长度不超过 n 的输入串，$\mathscr{M}$ 都在 $t(n)$步内停机.

(2) 为方便起见，可以说复杂度上界是某个 N 上的实函数 $f(n)$. 约定：这时实际上是说复杂度上界为$\lceil f(n)\rceil$. 例如，说 $\mathscr{M}$ 是 $\frac{1}{2}n^2$ 时间界限的，实际上是说 $\mathscr{M}$ 的时间复杂度上界为$\lceil\frac{1}{2}n^2\rceil$.

(3) 根据定义，$t_{\mathscr{M}}(n)$和 $s_{\mathscr{M}}(n)$都是非降的. 因此，以后也总假设时间复杂度上界和空间复杂度上界是非降的.

(4) 通常 $\mathscr{M}$ 总要读完整个输入串 w，这至少需要 $|w|+1$ 步. 因此，以后总假设时间复杂度上界 $t(n)\geqslant n+1$，并且约定：说时间复杂度上界为 $t(n)$，实际上是说为 $\max\{n+1,t(n)\}$. 例如，说时间复杂度上界为 n，实际上是说上界为 $n+1$. 说时间复杂度上界为 $\frac{1}{2}n^2$，实际上是说上界为 $\max\left\{n+1,\lceil\frac{1}{2}n^2\rceil\right\}$.

(5) 由于采用离线 Turing 机作为标准的计算模型，区分开输入、输出占用的空间和中间数据占用的空间. 空间复杂度只计算中间数据占用的空间，这使得有可能 $s_{\mathscr{M}}(n)<n$.

显然 $\mathscr{M}$ 至少要使用一个工作单元，因此以后总假设空间复杂度上界 $s(n)\geqslant 1$，并且约定：说空间复杂度上界为 $s(n)$，实际上是说上界为 $\max\{1,s(n)\}$. 特别地，说空间复杂度上界为 $\log n$ 时，

空间复杂度上界实际上是下述函数 $s(n)$：当 $n=0$ 时 $s(0)=1$，当 $n\geqslant 1$ 时 $s(n)=\max\{1,[\log n]\}$.

(6) 对于各种在线 Turing 机也可以类似地定义计算使用的时间和空间、时间复杂度和空间复杂度. 由于在线 Turing 机无法区分输入串占用的空间和计算过程中中间数据占用的空间，它的空间复杂度 $s_{\mathscr{M}}(n)\geqslant n$.

定义 12.4 如果存在 $t(n)$时间界限的 DTM(NTM)接受语言 L，则称 L **具有(非确定型)时间复杂度上界** $t(n)$，或 L 是**(非确定型)** $t(n)$**时间可接受的.**

如果存在 $s(n)$空间界限的 DTM(NTM)接受语言 L，则称 L **具有(非确定型)空间复杂度上界** $s(n)$，或 L 是**(非确定型)** $s(n)$**空间可接受的.**

定义 12.5 设全函数 $f: N\to N$. 如果存在 $t(n)$时间界限的 DTM 计算 f，则称 f 是 $t(n)$**时间可计算的.**

如果存在 $s(n)$空间界限的 DTM 计算 f，则称 f 是 $s(n)$**空间可计算的.**

[**例 12.1**] 考虑语言

$$L=\{wcw^R\mid w\in\{0,1\}^*\}.$$

构造 DTM $\mathscr{M}_1$ 如下：$\mathscr{M}_1$ 有一条工作带. 输入带和工作带的带头从左向右移动，把输入串的符号依次复写到工作带上，直至在输入带上读到 c 为止(不复写 c). 然后输入带带头继续向右，而工作带带头向左同时移动且比较它们扫描的符号. 如果两个带头同时读完它们扫描的字符串并且每一步扫描的符号都相同，则接受输入串，否则拒绝输入串. $\mathscr{M}_1$ 接受语言 L，其时间复杂度为 $O(n)$，空间复杂度也为 $O(n)$.

再构造一个接受 L 的 DTM $\mathscr{M}_2$，其空间复杂度为 $O(\log n)$. $\mathscr{M}_2$ 有两条工作带，用来记录二进制数. $\mathscr{M}_2$ 首先检查输入串是否恰好有一个 c 以及在 c 的两边的符号数是否相等. 如果回答是肯定的，则对每一个 $i=1,2,\cdots$，检查输入串左右两端第 i 个符号是

否相同，直至发现一对符号不相同或扫描到 c 为止. 利用工作带计数可以找到左端和右端第 i 个符号. 由于正整数 k 的二进制表示有 $\lceil \log_2(k+1) \rceil$ 位，故 $\mathscr{M}_2$ 的空间复杂度为 $O(\log n)$. 比较 c 两边的一对符号、输入带的带头要来回往返一次，故 $\mathscr{M}_2$ 的时间复杂度为 $O(n^2)$.

因此，语言 L 具有时间复杂度上界 $O(n)$ 和空间复杂度上界 $O(\log n)$.

[例 12.2] 把一进制数转化成二进制数，即对每一个输入 1^n，n 是一个自然数，输出 $b_0 b_1 \cdots b_m$，这里 $b_i \in \{0,1\}$，$0 \leqslant i \leqslant m$，$\sum_{i=0}^{m} b_i \cdot 2^i = n$ 并且当 $m \neq 0$ 时 $b_m = 1$.

DTM $\mathscr{M}$ 有一条工作带. 首先在工作带上写一个 0. 然后输入带头从左向右，每移一格，$\mathscr{M}$ 在工作带上以二进制方式加 1，直到读完输入为止. 最后把工作带上的内容复写到输出带上.

显然，$\mathscr{M}$ 把一进制数转化成二进制数. 它的空间复杂度为 $\log n$. 对 i 位二进制数加 1 可在 $O(i)$ 步内完成，做 2^{i-1} 次 i 位二进制数加 1 进位成 $i+1$ 位二进制数. 由于 $\sum_{i=1}^{k} i \cdot 2^{i-1} = (k-1)2^k + 1$，故 $\mathscr{M}$ 的时间复杂度为 $O(n\log n)$.

12.2 线性加速、带压缩和带数目的减少

本节介绍线性加速、带压缩和减少带的数目对复杂度的影响. 将若干个符号合并成一个“大符号”可以使使用的方格数减少若干倍，这就是带压缩. 类似地可以将计算的若干步合并成一步，从而加速计算. 因此，对于语言的时间复杂度上界和空间复杂度上界可以忽略一个常数因子. 例如，说时间复杂度上界 n^2 或 $O(n^2)$，而不必讨论它的系数，是 $2n^2$ 还是 $3n^2$. 又例如，说空间复杂度上界 $\log n$ 或 $O(\log n)$，而不必指明对数的底，因为对于不同的底 a 和 $b(a,b$

$>1)$，$\log_a n$ 和 $\log_b n$ 只差一个常数因子.

定理 12.1(带压缩定理)　设 $\mathscr{M}$ 是一台 $s(n)$空间界限的 DTM(NTM)，则对任意的 $c(0<c<1)$，存在 $cs(n)$空间界限的 DTM(NTM)$\mathscr{M}'$和 $\mathscr{M}$ 接受相同的语言.

证：设 $\mathscr{M}$ 是一台离线 DTM，有 k 条工作带，输入字母表 A，带字母表 C，状态集 Q. 取正整数 r 使得 $rc\geqslant 2$. $\mathscr{M}'$也是一台有 k 条工作带的离线 DTM，输入字母表仍为 A，带字母表 $C'=C^r$，即工作带上的每一个符号是 C 中符号的 r 元组$(s_1,\cdots,s_r)$. $\mathscr{M}'$第 m 条工作带上第 j 个方格内的符号为$(s_1,\cdots,s_r)$表示 $\mathscr{M}$ 第 m 条工作带上第$(j-1)r+1,\cdots,jr$ 个方格内的符号依次是 $s_1,\cdots,s_r$. $\mathscr{M}'$的状态集 $Q'=Q\times\{1,2,\cdots,r\}^k$. 当 $\mathscr{M}'$的状态为$(q,i_1,i_2,\cdots,i_k)$时，表示 $\mathscr{M}$ 的状态为 q 并且第 m 条工作带的带头正在扫描 $\mathscr{M}'$第 m 条工作带上被扫描的符号的第 i_m 个分量$(m=1,2,\cdots,k)$. 这样 $\mathscr{M}'$的格局描述了 $\mathscr{M}$ 的格局. 根据 $\mathscr{M}$ 的动作函数可以构造出 $\mathscr{M}'$的动作函数，使得 $\mathscr{M}'$模拟 $\mathscr{M}$ 的计算，即它们接受相同的语言. 这里不再给出模拟的细节.

$\mathscr{M}'$在每一条工作带上使用的方格数不超过$\lceil s(n)/r\rceil$. 当$s(n)\leqslant r$ 时，$\lceil s(n)/r\rceil=1$. 当 $s(n)>r$ 时，$\lceil s(n)/r\rceil<s(n)/r+1<2s(n)/r\leqslant cs(n)$. 因此，$\mathscr{M}'$是 $cs(n)$空间界限的. 注意，根据前面的说明，$\mathscr{M}'$的空间复杂度上界实际上是 $\max\{1,\lceil cs(n)\rceil\}$.

当 $\mathscr{M}$ 是一台 NTM 时，$\mathscr{M}'$也是 NTM，上述证明仍然有效.

□

定理 12.2(线性加速定理)　设 $\mathscr{M}$ 是一台 $t(n)$时间界限的 DTM(NTM). 如果$\lim\limits_{n\to\infty}\dfrac{n}{t(n)}=0$，则对任意的 $c(0<c<1)$，存在 $ct(n)$时间界限的 DTM(NTM)$\mathscr{M}'$和 $\mathscr{M}$ 接受相同的语言.

证：设 $\mathscr{M}$ 是一台有 k 条工作带的离线 Turing 机，输入字母表为 A，带字母表为 C. 又设 r 是一个正整数. $\mathscr{M}'$有 $k+1$ 条工作带，它的输入字母表也为 A，带字母表 $C'=C^r$. 和定理 12.1 中的一

样，$\mathscr{M}'$工作带上的每一个符号对应 $\mathscr{M}$ 带上 r 个连续的符号.把 $\mathscr{M}$ 的这样一段子字符串叫做一个字段.$\mathscr{M}$ 的每一个字段被压缩成 $\mathscr{M}'$的一个符号.$\mathscr{M}'$首先把输入写到一条工作带上且把 r 个符号合并成一个符号.此后 $\mathscr{M}'$把这条工作带当作输入带使用，用来模拟 $\mathscr{M}$ 的输入带.

$\mathscr{M}'$的动作分成组，每组由 8 步组成，叫做一个组合步.每一个组合步分 2 个阶段，模拟 $\mathscr{M}$ 的 r 步.

在第一阶段，$\mathscr{M}'$让每个带头先左移一格检查左邻方格内的符号，然后右移 2 格检查右邻方格内的符号，最后再左移一格回到原来的位置.$\mathscr{M}'$的每个带头访问的 3 个方格对应 $\mathscr{M}$ 的带上的 3 个字段：带头开始时所在字段、左边紧邻的字段和右边紧邻的字段.把这 3 个字段分别记作 B、L 和 R.由于每个字段的长为 r，$\mathscr{M}$ 计算 r 步每个带头不可能移出这 3 个字段.因此，$\mathscr{M}'$掌握这些信息之后就能确定 $\mathscr{M}$ 经过 r 步后的格局.$\mathscr{M}$ 的每个带头在这 r 步中的活动范围有 3 种可能：字段 B、字段 B 和 L、字段 B 和 R.为了模拟 $\mathscr{M}$ 的 r 步，$\mathscr{M}'$只需要改写每条带上对应于 $\mathscr{M}$ 在这 r 步中访问过的字段的方格，即带头所扫描的方格、带头所扫描的方格和左邻方格、或者带头所扫描的方格和右邻方格.在第 2 阶段，$\mathscr{M}'$完成这样的模拟且把每个带头放到正确的位置上.第一阶段有 4 步.第 2 阶段最多需要改写 2 个方格的内容和移动 2 次(左移一格再右移回来，或者右移一格再左移回来)，至多有 4 步.因此，$\mathscr{M}'$的每个组合步至多包括 8 步.模拟 $\mathscr{M}$ 的 $t(n)$步，$\mathscr{M}'$至多需要 $8\lceil t(n)/r\rceil$步.除此之外，在计算开始时 $\mathscr{M}'$要把输入串复写到一条工作带上，把 r 个符号合并成一个符号，且在完成复写后把带头恢复到初始位置.这需要 $n+\lceil n/r\rceil$步.因此，$\mathscr{M}'$的时间复杂度不超过

$$n+\lceil n/r\rceil+8\lceil t(n)/r\rceil. \tag{12.1}$$

为了实现上述模拟，$\mathscr{M}'$的状态不但要像定理 12.1 中的那样记录 $\mathscr{M}$ 的状态和每个带头的位置，而且要记录 $\mathscr{M}'$每个带头在

组合步第一阶段读到的信息，即 $\mathscr{M}$ 相应的 3 个字段.

显然，(12.1)式不超过

$$2n + 8t(n)/r + 8 \tag{12.2}$$

由于$\lim_{n\to\infty}\frac{n}{t(n)}=0$，存在 n_0 使得当 $n\geqslant n_0$ 时$\frac{n}{t(n)}\leqslant\frac{c}{8}$. 取 r 使得 $rc\geqslant 16$，$n_1=\max\{4,n_0\}$. 于是，当 $n\geqslant n_1$ 时(12.2)式不超过 $ct(n)$.

剩下 A^* 中长度小于 n_1 的字符串需要再处理一下. 由于只有有限个这样的字符串，可以如下修改 $\mathscr{M}'$：对于每一个长度小于 n_1 的字符串 w，读完 w 后就决定是否接受(若 $w\in L(\mathscr{M})$则接受，否则不接受). 这只需要 $|w|+1$ 步. 根据约定，定理要求 $\mathscr{M}'$的时间复杂度不超过 $\max\{n+1,ct(n)\}$. 上面给出的 $\mathscr{M}'$满足这个要求，证毕. □

在上述定理 12.2 中，$\mathscr{M}$ 是离线 Turing 机. 为了模拟 $\mathscr{M}$ 且将运行时间缩短常数倍，$\mathscr{M}'$需要增加一条工作带. 当 $\mathscr{M}$ 是 k 带在线 Turing 机时，其中 $k>1$，则上述两个定理仍然成立，并且 $\mathscr{M}'$也是 k 带在线 Turing 机. $\mathscr{M}'$在把输入串复写到另一条带且把若干个符号合并成一个后，原来存放输入的带可以改用来存放中间数据，从而不需要增加带数，但至少要有 2 条带才行.

根据线性加速定理和带压缩定理，用 $O(\cdot)$表述时间和空间复杂度上界是合适的. 设 $g(n)=O(f(n))$，$\mathscr{M}$ 是 $g(n)$时间(空间)界限的，那么存在 $f(n)$时间(空间)界限的 $\mathscr{M}'$模拟 $\mathscr{M}$. 因此只需说 $\mathscr{M}$ 是 $O(f(n))$时间(空间)界限的就足够了.

下面两个定理给出减少带的数目对时间和空间复杂性的影响.

定理 12.3 设 $\mathscr{M}$ 是一台具有 k 条工作带的、$s(n)$空间界限的确定型(非确定型)离线 Turing 机，其中 $k>1$，则存在只有一条工作带的确定型(非确定型)离线 Turing 机 $\mathscr{M}'$，使得 $\mathscr{M}'$和 $\mathscr{M}$ 接受相同的语言并且具有相同的空间复杂度上界 $s(n)$.

证：像 6.4.3 小节中那样，采用多道技术 $\mathscr{M}'$用一条工作带

模拟 $\mathscr{M}$ 的 k 条工作带. $\mathscr{M}'$ 在工作带上使用的方格数不超过 $s(n)$. □

根据上述证明，不难看到：当 $s(n)\geqslant n$ 时，不论 $\mathscr{M}$ 是具有 k 条工作带的离线 Turing 机、还是 k 带在线 Turing 机，都可以用一台单带(在线)Turing 机模拟 $\mathscr{M}$ 并且具有相同的空间复杂度上界.

定理 12.4 设 $\mathscr{M}$ 是一台确定型(非确定型)$t(n)$时间界限的多带 Turing 机(离线的或在线的)，则存在确定型(非确定型)单带 Turing 机 $\mathscr{M}'$，使得 $\mathscr{M}'$ 和 $\mathscr{M}$ 接受相同的语言并且是 $t^2(n)$时间界限的.

证：$\mathscr{M}'$ 仍采用多道技术模拟 $\mathscr{M}$. 模拟 $\mathscr{M}$ 的一步，$\mathscr{M}'$ 的带头需从左向右、再从右向左来回运动一次. 运动的距离不超过 $t(n)$，故 $\mathscr{M}'$ 使用的时间不超过 $ct^2(n)$，其中 c 是一个正整数. 为了消去 c，根据线性加速定理先用一台 $t(n)/\sqrt{c}$ 时间界限的多带 Turing 机 $\mathscr{M}_1$ 模拟 $\mathscr{M}$，再用 $\mathscr{M}'$ 模拟 $\mathscr{M}_1$. 这样得到的 $\mathscr{M}'$ 具有时间复杂度上界 $t^2(n)$. □

12.3 时间谱系和空间谱系

定义 12.6 设全函数 $t(n)\geqslant n+1$ 和 $s(n)\geqslant 1$. 定义下述复杂性类：

DTIME($t(n)$)是所有 $t(n)$时间可接受的语言组成的语言类；

DSPACE($s(n)$)是所有 $s(n)$空间可接受的语言组成的语言类；

NTIME($t(n)$)是所有非确定型 $t(n)$时间可接受的语言组成的语言类；

NSPACE($s(n)$)是所有非确定型 $s(n)$空间可接受的语言组成的语言类.

今后在使用上述复杂性类的记号时，前面关于 $t(n)$和 $s(n)$的

特殊约定继续有效，即 $t(n)$实际上是指 $\max\{n+1,\lceil t(n)\rceil\}$，$s(n)$实际上是指 $\max\{1,\lceil s(n)\rceil\}$. 例如，DTIME$(0.1n^2)$应理解为 DTIME$(\max\{n+1,\lceil 0.1n^2\rceil\})$.

设 $t(n)<t'(n)$，显然 DTIME$(t(n))\subseteq$DTIME$(t'(n))$. 问题是这个包含关系是真包含吗？即，是否存在 L 使得 $L\in$DTIME$(t'(n))$且 $L\notin$DTIME$(t(n))$？根据线性加速定理，当$\lim\limits_{n\to\infty}\frac{n}{t(n)}=0$时，如果 $t'(n)$仅是 $t(n)$的常数倍，则这两个语言类相等. 那么 $t'(n)$要比 $t(n)$大多少才能使 DTIME$(t(n))\subset$DTIME$(t'(n))$？对于其余 3 种复杂性类也有同样的问题.

人们发现，当允许用任意的递归函数作复杂度上界时会出现一些奇怪的现象. 例如，任给递归函数 f 都存在一个时间复杂度上界$t(n)$，使得 DTIME$(t(n))=$DTIME$(f(t(n))$. 若取 $f(n)=2^n$，则有 DTIME$(t(n))=$DTIME$(2^{t(n)})$，即在时间复杂度上界$t(n)$和 $2^{t(n)}$之间有一个“间隙”. 在这个“间隙”内没有任何语言. 一般地，在上界 $t(n)$和 $f(t(n))$之间有这样一个“间隙”. f 可以是一个增长非常迅速的函数(如 Ackermann 函数 $A(n,n)$)，从而得到一个巨大的“间隙”. 又如，存在这样的语言 L，对于每一个接受 L 的 DTM $\mathscr{M}$ 都有另一个 DTM $\mathscr{M}'$，$\mathscr{M}'$也接受 L 但比 $\mathscr{M}$ 快得多，从而不存在接受 L 的最快的 DTM. 对于空间也有这些奇怪的现象. 为了避开这些反常的现象，下面有时把复杂度上界限制为“性质良好”的函数. 这种“性质良好”的函数就是时间可构造的和空间可构造的函数.

定义 12.7 设递归函数 $f:N\to N$. 如果存在 DTM 对每一个长度为 n 的输入都恰好在计算 $f(n)$步后停机，则称 f 是**时间可构造的**.

如果存在 f 空间界限的 DTM 对每一个长度为 n 的输入都停机在这样的格局，它的每条工作带上恰好有 $f(n)$个非空白单元，则称 f 是**空间可构造的**.

定理 12.5 (1) 函数 f 是空间可构造的当且仅当 f 是 $O(f)$ 空间可计算的.

(2) 如果存在 $\epsilon>0$ 使得除有限个 n 之外都有 $f(n)\geqslant(1+\epsilon)n$，则 f 是时间可构造的当且仅当 f 是 $O(f)$ 时间可计算的.

定理的证明可以在参考文献[4]中找到，这里略去. 这个定理表明时间可构造和空间可构造的函数是非常广泛的. 例如，n^k，2^{cn}，$n!$，2^{n^k} 都是时间可构造和空间可构造的. $\log n$，$\log^k n$ 是空间可构造的.

先证明两个引理.

引理 12.6 设 L 是 $s(n)$ 空间可接受的，其中 $s(n)\geqslant\log_2 n$ 且是空间可构造的，则存在 $s(n)$ 空间界限且对所有输入都停机的 DTM 接受 L.

证：设离线 Turing 机 $\mathscr{M}$ 接受 L. 它是 $s(n)$ 空间界限的，有 r 个状态和 t 个带符号. 根据定理 12.3，不妨设 $\mathscr{M}$ 只有一条工作带. 对任何长度为 n 的输入，$\mathscr{M}$ 至多有 $T(n)=(n+2)r(s(n)+1)t^{s(n)}$ 个不同的格局，其中输入带带头至多有 $n+2$ 个不同的位置，工作带带头至多有 $s(n)+1$ 个不同的位置，工作带上至多有 $t^{s(n)}$ 个不同的字符串，$\mathscr{M}$ 有 r 个状态. 由于 $s(n)\geqslant\log_2 n$，存在正整数 a 使得 $a^{s(n)}\geqslant T(n)$. 构造所需要的 DTM $\mathscr{M}'$ 的基本思想是增加一条工作带用来记录 $\mathscr{M}$ 计算的步数. 若步数已超过 $T(n)$，则 $\mathscr{M}$ 必进入死循环，从而拒绝输入串. 以 a 为底进行计数，计算占用的空间不超过 $s(n)$.

$\mathscr{M}'$ 有两条工作带，一条用来模拟 $\mathscr{M}$ 的工作带，另一条用来以 a 为底进行计数. 任给输入 w，$n=|w|$，$\mathscr{M}'$ 首先在用来计数的带上标明 $s(n)$ 个方格(因为 $s(n)$ 是空间可构造的)，然后模拟 $\mathscr{M}$ 关于 w 的计算，同时记录 $\mathscr{M}$ 运行的步数. 如果在计数的带上企图使用所标范围以外的方格，则 $\mathscr{M}'$ 停止计算并且拒绝 w. 否则 $\mathscr{M}'$ 模拟到 $\mathscr{M}$ 的计算停机为止，并且 $\mathscr{M}'$ 接受 w 当且仅当 $\mathscr{M}$ 接受 w. 根据前面的分析，不难看到 $\mathscr{M}'$ 满足引理的要求. □

引理 12.7 对于每一个递归函数 $f:N\to N$，存在时间(空间)可构造的函数 g，使得对所有的 n 有 $g(n)\geqslant f(n)$.

证：设 $\mathscr{M}$ 以一进制方式计算 f：对于输入 1^n，$\mathscr{M}$ 输出 $1^{f(n)}$. $\mathscr{M}'$ 工作如下：任给输入 w，$|w|=n$，$\mathscr{M}'$ 在工作带上写 1^n，然后模拟 $\mathscr{M}$ 关于 1^n 的计算.

显然，对于所有长度为 n 的输入 $\mathscr{M}'$ 的计算步数是相同的，记作 $g(n)$. 函数 g 是时间可构造的，它恰好是 $\mathscr{M}'$ 的运行时间. 为了输出 $1^{f(n)}$，$\mathscr{M}'$ 需要 $f(n)$步，故 $g(n)\geqslant f(n)$.

同样，对于所有长度为 n 的输入 $\mathscr{M}'$ 使用的空间是相同的，记作 $g'(n)$. 函数 g' 是空间可构造的. 在这里 $\mathscr{M}'$ 用一条工作带模拟 $\mathscr{M}$ 的输出带，从而至少要使用 $f(n)$空间，故 $g'(n)\geqslant f(n)$. □

在给出本节的主要结果之前，先叙述关于 Turing 机的编码方法. 考虑以{0,1}为输入字母表，具有一条工作带的离线 Turing 机 $\mathscr{M}$. 设 $\mathscr{M}$ 的带符号为 $s_1,s_2,\cdots,s_t$，状态为 $q_1,q_2,\cdots,q_r$，其中 $s_1=0$，$s_2=1$，$s_3=\#$ 是左端标志符，$s_4=\$$ 是右端标志符，$s_5=B$ 是空白符，q_1 是初始状态，q_2 是唯一的接受状态. $\mathscr{M}$ 的动作函数是 6 元组

$$q_a s_b s_c \Delta_d \Delta_e q_f \qquad (*)$$

的有穷集合，其中 $1\leqslant a,f\leqslant r$，$1\leqslant b\leqslant 4$，$1\leqslant c\leqslant l$ 且 $c\neq 4$，$1\leqslant d\leqslant 2$，$1\leqslant e\leqslant l+2$，$\Delta_1=R$，$\Delta_2=L$，$\Delta_{j+2}=s_j$. 6 元组(*)表示当 $\mathscr{M}$ 处于状态 q_a、在输入带和工作带上分别读到符号 s_b 和 s_c 时，下一个动作是将状态转移到 q_f 同时输入带头和工作带头分别按 Δ_d 和 Δ_e 动作. 当 Δ_d 和 Δ_e 是 R 时右移一格，是 L 时左移一格，是 s_j 时打印 s_j.

6 元组(*)的代码为

$$0^a10^b10^c10^d10^e10^f$$

将 $\mathscr{M}$ 的 6 元组按任意的顺序排列，其代码依次是 $D_1,D_2,\cdots,D_m$，则 $\mathscr{M}$ 的代码为

$$D_1 11 D_2 11 \cdots 11 D_m,$$

并且前面可以添加任意有穷个 1. 于是，$\mathscr{M}$ 的代码是{0,1}上的一

个字符串. $\mathscr{M}$ 有任意多个代码，并且代码可以任意的长. 反之，任给{0,1}上的一个字符串 w，w 不一定是某个 Turing 机的代码. 当 w 是某个 Turing 机的代码时，把这台 Turing 机记作 $\mathscr{M}_w$.

对有 k 条工作带的离线 Turing 机可类似编码，它的动作函数是 $4+2k$ 元组的有穷集合. 每个 $4+2k$ 元组的代码与 6 元组的代码类似，有 $4+2k$ 串 0，每两串之间用一个 1 隔开.

定理 12.8 设 $s_1(n)$和 $s_2(n)$是两个空间可构造的函数，$s_1(n)\geqslant\log_2 n$，$s_2(n)\geqslant\log_2 n$ 并且

$$\lim_{n\to\infty}\frac{s_1(n)}{s_2(n)}=0,$$

则存在一个语言，它在 DSPACE($s_2(n)$)中，但不在 DSPACE($s_1(n)$)中.

证：要构造一台 $s_2(n)$空间界限的 DTM $\mathscr{M}$，它与任何一台 $s_1(n)$空间界限的 DTM 都不接受相同的语言，从而 $\mathscr{M}$ 接受的语言在 DSPACE($s_2(n)$)中，但不在 DSPACE($s_1(n)$)中.

$\mathscr{M}$ 以{0,1}为输入字母表，有 3 条工作带，分别叫做带 1、带 2、带 3. 任给 $w\in\{0,1\}^*$，$\mathscr{M}$ 首先在带 1 上对 $s_2(n)$个方格做出标记，这里 $n=|w|$. 这是可以做到的，因为 $s_2(n)$是空间可构造的. 此后，如果带 1 的带头企图离开标定的范围，$\mathscr{M}$ 就停机并且不接受 w.

$\mathscr{M}$ 检查 w 是否是一台具有一条工作带的离线 Turing 机的代码. 如果不是，则 $\mathscr{M}$ 停机并且不接受 w；如果是，则 $\mathscr{M}$ 模拟 $\mathscr{M}_w$ 关于 w 的计算.

$\mathscr{M}$ 用带 1 模拟 $\mathscr{M}_w$ 的工作带. 设 $\mathscr{M}_w$ 有 l 个带符号，$\mathscr{M}$ 用二进制的 $0,1,\cdots,l-1$ 分别表示 $s_1,s_2,\cdots,s_l$，每个二进制数占$\lceil\log_2(l+1)\rceil$位. $\mathscr{M}$ 在带 2 上用二进制数记录 $\mathscr{M}_w$ 输入带带头的位置. $\mathscr{M}$ 在带 3 上记录 $\mathscr{M}_w$ 的状态，用二进制数 i 表示状态 q_i，$1\leqslant i\leqslant r$，r 是 $\mathscr{M}_w$ 的状态数. $\mathscr{M}$ 首先初始化 3 条工作带，表示 $\mathscr{M}_w$ 处于初始格局. $\mathscr{M}$ 以下述方式模拟 $\mathscr{M}_w$ 的一步：$\mathscr{M}$ 根据 $\mathscr{M}_w$ 当

前的状态和两个带头读到的符号在 w 中寻找对应的 6 元组. 如果不存在这样的 6 元组，则停机. 如果找到了这个 6 元组，则 $\mathscr{M}$ 按照这个 6 元组的含义动作. $\mathscr{M}$ 接受 w 当且仅当 $\mathscr{M}$ 在带 1 标定的空间内完成模拟，并且 $\mathscr{M}_w$ 停机在非接受状态.

注意到 $r<n$，$\mathscr{M}$ 在带 2 上至多使用 $\log_2(n+3)$个方格，在带 3 上至多使用 $\log_2(n+1)$个方格. 而 $s_2(n)\geqslant\log_2 n$，根据带压缩定理可以设 $\mathscr{M}$ 在带 2 和带 3 上使用的方格数不超过 $s_2(n)$，从而 $\mathscr{M}$ 是 $s_2(n)$空间界限的.

设 $\mathscr{M}'$是一台 $s_1(n)$空间界限的离线 Turing 机. 根据定理 12.3和引理 12.6，不妨设 $\mathscr{M}'$只有一条工作带且对所有的输入都停机. 由于$\lim\limits_{n\to\infty}\dfrac{s_1(n)}{s_2(n)}=0$，$\mathscr{M}'$有一个足够长的代码 $w\in\{0,1\}^*$使得 $s_2(n)\geqslant\lceil\log_2(l+1)\rceil s_1(n)$，其中 $n=|w|$，l 是 $\mathscr{M}'$的带符号数. 注意到 $\mathscr{M}$ 模拟 $\mathscr{M}'$关于 w 的计算时在带 1 上至多使用$\lceil\log_2(l+1)\rceil s_1(n)$个方格. 于是 $\mathscr{M}$ 有足够的空间模拟 $\mathscr{M}'$关于 w 的计算，并且 $\mathscr{M}$ 接受 w 当且仅当 $\mathscr{M}'$不接受 w. 因此，$L(\mathscr{M})\neq L(\mathscr{M}')$. 从而，$L(\mathscr{M})\notin\mathrm{DSPACE}(s_1(n))$. □

定理 12.9 对于任何递归函数 $f:N\to N$，存在递归语言 $L\notin\mathrm{DSPACE}(f(n))$.

证：根据引理 12.7，存在空间可构造的函数 $s_1(n)$和 $s_2(n)$使得 $s_1(n)\geqslant f(n)$，$s_2(n)\geqslant ns_1(n)$. 不妨设 $s_1(n)\geqslant\log_2 n$，$s_2(n)\geqslant\log_2 n$. 于是，$s_1(n)$和 $s_2(n)$满足定理 12.8 的条件，故存在 $L\in\mathrm{DSPACE}(s_2(n))$并且 $L\notin\mathrm{DSPACE}(s_1(n))$. 显然，$L$ 是递归的并且不属于 $\mathrm{DSPACE}(f(n))$. □

根据定理 12.8，存在空间谱系：

$$\mathrm{DSPACE}(s_1)\subset\mathrm{DSPACE}(s_2)\subset\cdots\subset\mathrm{DSPACE}(s_i)\subset\cdots,$$

这里对每一个 i，s_i 和 s_{i+1}满足定理 12.8 的条件.

存在类似的时间谱系. 由于用一条带模拟多条带时时间可能增加到平方值，因此时间谱系不能像空间谱系那样稠密.

定理 12.10 设 $t_2(n)$是时间可构造的函数，并且

$$\lim_{n\to\infty}\frac{t_1^2(n)}{t_2(n)}=0,$$

则存在一个语言，它在 DTIME($t_2(n)$)中、但不在 DTIME($t_1(n)$)中.

证：设 DTM$\mathscr{T}$ 对每一个长度为 n 的输入恰好运行 $t_2(n)$步. DTM$\mathscr{M}$ 在进行下面描述的计算的同时，并行地进行对 $\mathscr{T}$ 的模拟，并且当 $\mathscr{M}$ 企图运行 $t_2(n)+1$ 步时立即停机在非接受状态，从而保证 $\mathscr{M}$ 是 $t_2(n)$时间界限的.

任给输入 $w\in\{0,1\}^*$，$n=|w|$. $\mathscr{M}$ 首先检查 w 是否是一台具有 k 条工作带的离线 Turing 机的代码，这里 k 是某个正整数. 若不是，则 $\mathscr{M}$ 停机并拒绝 w；若是，则模拟 $\mathscr{M}_w$ 关于 w 的计算.

$\mathscr{M}$ 把 w 复制到带 1 上，模拟 $\mathscr{M}_w$ 的输入带. 而输入带上的 w 继续作为 $\mathscr{M}_w$ 的代码在计算中起作用. $\mathscr{M}$ 用带 2 模拟 $\mathscr{M}_w$ 的 k 条工作带. 这次不使用 6.4.3 小节中的多道技术，而是将 k 条工作带的内容按下述顺序排成一行：带 1 的方格 0，带 2 的方格 0，…，带 k 的方格 0；带 1 的方格 1，带 2 的方格 1，…，带 k 的方格 1；带 1 的方格 2，…，并且在每个带头扫描的符号左边插入一个特殊符号 $\triangleright$. 例如，$\mathscr{M}_w$ 有 3 条工作带，带的内容和带头位置分别为

$$\begin{array}{l} a_0\underset{\uparrow}{a_1}a_2a_3\cdots \\ b_0\underset{\uparrow}{b_1}b_2b_3\cdots \\ c_0c_1\underset{\uparrow}{c_2}c_3\cdots \end{array}$$

$\mathscr{M}$ 的带 2 的内容为

$$a_0\triangleright b_0c_0a_1b_1c_1\triangleright a_2b_2c_2a_3b_3\triangleright c_3\cdots.$$

和定理 12.8 证明中的一样，这些符号都以二进制数的方式作为代码. $\mathscr{M}$ 用带 3 以二进制数方式记录 $\mathscr{M}_w$ 的当前状态.

$\mathscr{M}$ 如下模拟 $\mathscr{M}_w$ 的一步：带 2 的带头从左向右扫描，直至检

查完 $\mathscr{M}_w$ 的 k 个带头扫描的所有符号为止. 在扫描过程中，把 $\mathscr{M}_w$ 的工作带上被扫描的 k 个符号复制到带 4 上. 然后根据带 1 ($\mathscr{M}_w$ 的输入带)、带 3($\mathscr{M}_w$ 的状态)和带 4 的内容，在输入带上寻找 w 中对应的 $4+2k$ 元组的代码. 若不存在所要的代码则 $\mathscr{M}$ 停机；否则按照这个代码模拟 $\mathscr{M}_w$ 的动作——移动带 1 的带头位置，修改带 2 和带 3 的内容. 最后，将带 2 的带头移到左端，将带 4 清成空白带并将带头移到左端.

$\mathscr{M}$ 接受 w 当且仅当 $\mathscr{M}$ 在 $t_2(n)$步内模拟完 $\mathscr{M}_w$ 关于 w 的计算并且 $\mathscr{M}_w$ 停机在非接受状态.

设 $\mathscr{M}'$是一台 $t_1(n)$时间界限的 DTM，它有 k 条工作带. 设 w 是 $\mathscr{M}'$的代码，$n=|w|$. $\mathscr{M}$ 关于 w 的计算过程中，带 2 的内容的长度不超过 $k\lceil\log_2(l+1)\rceil t_1(n)+k+1$，其中 l 是 $\mathscr{M}'$的带符号数. 于是存在常数 d 使得 $\mathscr{M}$ 能在 $dt_1^2(n)$步内完成模拟. 由定理的条件，总能取到 $\mathscr{M}'$足够长的代码 w 使得 $dt_1^2(n)<t_2(n)$，从而使 $\mathscr{M}$ 有充分的时间完成对 $\mathscr{M}'$关于 w 的计算的模拟. 于是，$\mathscr{M}$ 接受 w 当且仅当 $\mathscr{M}'$不接受 w. 因此，$L(\mathscr{M})\neq L(\mathscr{M}')$.

得证 $L(\mathscr{M})\in\mathrm{DTIME}(t_2(n))-\mathrm{DTIME}(t_1(n))$. □

类似定理 12.9，有：

定理 12.11 对任何递归函数 $f:N\to N$，存在递归语言 $L\notin\mathrm{DTIME}(f(n))$.

定理 12.8 和定理 12.10 的条件可以减弱，特别是可以把定理 12.10 中的 $t_1^2(n)$减弱为 $t_1(n)\log_2 t_1(n)$. 请参阅参考文献[2].

12.4 复杂性度量之间的关系

定理 12.12 设函数 $s(n)\geqslant 1$，$s'(n)\geqslant 1$，$t(n)\geqslant n+1$，$t'(n)\geqslant n+1$. 那么：

(1) $\mathrm{DTIME}(t)\subseteq\mathrm{NTIME}(t)$，$\mathrm{DSPACE}(s)\subseteq\mathrm{NSPACE}(s)$.

(2) $\mathrm{DTIME}(t)\subseteq\mathrm{DSPACE}(t)$，$\mathrm{NTIME}(t)\subseteq\mathrm{NSPACE}(t)$.

(3) 如果 $s'=O(s)$，则

$\mathrm{DSPACE}(s')\subseteq\mathrm{DSPACE}(s)$，$\mathrm{NSPACE}(s')\subseteq\mathrm{NSPACE}(s)$.

(4) 如果 $s'=\Theta(s)$，则

$\mathrm{DSPACE}(s')=\mathrm{DSPACE}(s)$，$\mathrm{NSPACE}(s')=\mathrm{NSPACE}(s)$.

(5) 如果 $t'=O(t)$且$\lim\limits_{n\to\infty}\frac{n}{t(n)}=0$，则

$\mathrm{DTIME}(t')\subseteq\mathrm{DTIME}(t)$，$\mathrm{NTIME}(t')\subseteq\mathrm{NTIME}(t)$.

(6) 如果 $t'=\Theta(t)$且$\lim\limits_{n\to\infty}\frac{n}{t(n)}=0$，则

$\mathrm{DTIME}(t')=\mathrm{DTIME}(t)$，$\mathrm{NTIME}(t')=\mathrm{NTIME}(t)$.

(7) 如果 t 是时间可构造的，则

$\mathrm{NTIME}(t)\subseteq\mathrm{DSPACE}(t)$.

记：(1)是显然的. 根据带压缩定理和线性加速定理，分别有(3)和(5). (4)和(6)分别是(3)和(5)的直接推论.

(2)的 $t(n)$ 步在每条带上至多使用 $t(n)+1$ 个方格，故 $\mathrm{DTIME}(t(n))\subseteq\mathrm{DSPACE}(t(n)+1)$. 而 $t(n)+1\leqslant 2t(n)$. 由(3)，得 $\mathrm{DTIME}(t)\subseteq\mathrm{DSPACE}(t)$.

同理可证 $\mathrm{DTIME}(t)\subseteq\mathrm{NSPACE}(t)$.

(7) 中，设 NTM $\mathscr{M}$ 是 $t(n)$时间界限的. 构造 DTM $\mathscr{M}'$模拟 $\mathscr{M}$：任给输入 w，$\mathscr{M}'$逐个模拟 $\mathscr{M}$ 关于 w 的每一个可能的计算. 利用 t 的时间可构造性，对每一个可能的计算 $\mathscr{M}'$至多模拟 $t(n)$ 步，这里 $n=|w|$. $\mathscr{M}'$接受 w 当且仅当它找到一个 $\mathscr{M}$ 关于 w 的接受计算. $\mathscr{M}'$在模拟每一个可能的计算时重复使用自己的空间. 这至多需要 $t(n)$空间. 设 $\mathscr{M}$ 在每一步至多有 r 个动作可供选择，则可用长为 $t(n)$的 r 进制数表示 $\mathscr{M}$ 关于 w 的一个计算. 于是，$\mathscr{M}'$可以用这样一个 r 进制数来控制模拟. 这只需添加一条工作带用来记录 r 进制数，至多使用 $t(n)$个方格. 故 $\mathscr{M}'$是 $t(n)$空间界限的. □

引理 12.13 设 s 是空间可构造的且 $s(n)\geqslant\log_2 n$，则存在 DTM 对每一个长度为 n 的输入恰好使用 $s(n)$空间并且在 $c^{s(n)}$步

内停机，其中 $c>1$ 是一个常数.

证：由于 s 是空间可构造的，有 DTM $\mathscr{M}$ 对每一个长度为 n 的输入恰好使用 $s(n)$ 空间. 不妨设 $\mathscr{M}$ 有一条工作带. 存在常数 $a>1$，对每一个长度为 n 的输入 w，如果在工作带上使用 $k\geqslant\log_2 n$ 个方格，则 $\mathscr{M}$ 至多有 a^k 个不同的方格.（见引理 12.6 的证明）. 从而，当在工作带上使用 $k\geqslant\log_2 n$ 个方格且超过 a^k 步时，$\mathscr{M}$ 必进入死循环、重复进行前面已经进行过的一段计算，因此不再会使用新的方格. 这时可以强迫它停机.

DTM $\mathscr{M}'$ 有 3 条工作带，带 1 用来模拟 $\mathscr{M}$ 的工作带，带 2 用来以 a 进制方式记录模拟 $\mathscr{M}$ 的步数，带 3 用来记录带 1 和带 2 上被使用的方格数之差. $\mathscr{M}'$ 首先在带 2 上标明 $\log_2 n$ 个方格（这只需在带 2 上写出 n 的二进制表示），然后模拟 $\mathscr{M}$，同时记录模拟的步数和带 1 与带 2 上使用的方格数之差. $\mathscr{M}'$ 停机当且仅当 $\mathscr{M}$ 停机或者 $\mathscr{M}'$ 企图在带 2 上使用多于 $\log_2 n$ 和多于带 1 上使用的方格. 如果 $\mathscr{M}'$ 是在第二种情况下停机的，则 $\mathscr{M}$ 必进入死循环，它已恰好使用了 $s(n)$ 个工作带上的方格. $\mathscr{M}'$ 在带 2 和带 3 上使用的方格数不超过带 1 上使用的方格数，故 $\mathscr{M}'$ 恰好使用 $s(n)$ 空间.

在带 2 上标明 $\log_2 n$ 个方格可以在 $O(n\log_2 n)$ 步内完成（见例 12.2）. 模拟 $\mathscr{M}$ 不超过 $a^{s(n)}$ 步，计录模拟步数可以在 $O(a^{s(n)}\log_2 a^{s(n)})$ 步内完成. 用带 3 记录带 1 和带 2 上被使用的方格数之差至多需要 $s(n)$ 步. 注意到 $s(n)\geqslant\log_2 n$，存在 $c>1$ 使得 $\mathscr{M}'$ 在 $c^{s(n)}$ 步内停机. □

定理 12.14 设 s 是空间可构造的且 $s(n)\geqslant\log_2 n$，$L\in\mathrm{NSPACE}(s(n))$，则存在常数 $c>0$ 使得 $L\in\mathrm{DTIME}(2^{cs(n)})$.

证：设 $L=L(\mathscr{M})$，$\mathscr{M}$ 是一台 $s(n)$ 空间界限的 NTM. 不失一般性，不妨设 $\mathscr{M}$ 停机在接受状态时已把所有的工作带清成空白带并且把带头都移到左端，从而 $\mathscr{M}$ 有唯一的接受格局. 对每一个长度为 n 的输入 w，$\mathscr{M}$ 至多有 $2^{as(n)}$ 个不同的格局，其中 $a>0$ 是一

个常数.

有向图 $G_w=(V_w,A_w)$的定义如下：顶点集 V_w 由 $\mathscr{M}$ 关于 w 的所有格局组成. 对任意的两个格局 σ 和 τ，$(\sigma,\tau)\in A_w$ 当且仅当 $\sigma \underset{\mathscr{M}}{\vdash} \tau$. 记初始格局 σ_0，接受格局 σ_f. 显然，$\mathscr{M}$ 接受 w 当且仅当 G_w 中有一条从 σ_0到 σ_f 的通路.

DTM $\mathscr{M}'$以下述方式工作：对每一个输入 w，首先构造有向图 G_w，然后检查 G_w 中是否有从 σ_0到 σ_f 的通路. $\mathscr{M}'$接受 w 当且仅当 G_w 中有这样的通路. 显然，$L(\mathscr{M}')=L$.

$\mathscr{M}'$构造 G_w 要用到 $s(n)$的空间可构造性. 根据引理12.13，$\mathscr{M}'$可以在$2^{as(n)}$步内标出 $s(n)$个方格，其中 $a>0$是一个常数. 利用标定的 $s(n)$个方格生成 $\mathscr{M}$ 的格局. 对每一对格局 σ 和 τ，可以在 n 和 $s(n)$的多项式步内判断是否 $\sigma \underset{\mathscr{M}}{\vdash} \tau$. 由于 $s(n)\geqslant\log_2 n$ 和 $|V_w|\leqslant 2^{as(n)}$，$\mathscr{M}'$可以在$2^{bs(n)}$步内构造出 G_w，其中 $b>0$是一个常数. 判断 G_w 中是否有从 σ_0到 σ_f 的通路可在$|V_w|$的多项式步内完成，从而存在常数 $c>0$使得 $\mathscr{M}'$的时间复杂度不超过$2^{cs(n)}$. □

推论12.15 (1) 设 s 是空间可构造的且 $s(n)\geqslant\log_2 n$，$L\in$ DSPACE$(s(n))$，则存在常数 $c>0$使得 $L\in$DTIME$(2^{cs(n)})$.

(2) 设 t 是时间可构造的，$L\in$NTIME$(t(n))$，则存在常数 $c>0$使得 $L\in$DTIME$(2^{ct(n)})$.

证：由定理12.12(1)、(2)和定理12.14，可推出所需结论. 这里要用到任何时间可构造的函数 t 都是空间可构造的. 事实上，只需要 DTM 添加一条工作带并且每一步都使用这条带的一个新单元，则它恰好使用 $t(n)$空间. □

注意：定理12.14和推论12.15中的常数 c 都与 L 有关. 根据上式定理和推论，有

$$\begin{aligned}\text{DSPACE}(s(n)) &\subseteq \text{NSPACE}(s(n))\\ &\subseteq \bigcup_{c>0}\text{DTIME}(2^{cs(n)}),\\ \text{NTIME}(t(n)) &\subseteq \bigcup_{c>0}\text{DTIME}(2^{ct(n)}),\end{aligned}$$

其中 s 是空间可构造的且 $s(n)\geqslant\log_2 n$，t 是时间可构造的.

定理12.16 设 s 是空间可构造的且 $s(n)\geqslant\log_2 n$，则

$$\mathrm{NSPACE}(s(n))\subseteq\mathrm{DSPACE}(s^2(n)).$$

证：设 $\mathscr{M}$ 是一台 $s(n)$空间界限的 NTM，它有一条工作带. 和定理12.14的证明一样，不妨设 $\mathscr{M}$ 有唯一的接受格局 σ_f. 对每一个长度为 n 的输入 w，$\mathscr{M}$ 至多有 $2^{as(n)}$ 个不同的格局，这里 $a>0$ 是一个常数. 从而，$\mathscr{M}$ 接受 w 当且仅当从初始格局 σ_0 至多经过 $2^{as(n)}$ 步可以到达接受格局 σ_f.

设 σ_1 和 σ_2 是 $\mathscr{M}$ 的两个格局，$\sigma_1\overset{i}{\vdash}\sigma_2$ 表示 $\mathscr{M}$ 从 σ_1 至多经过 2^i 步可以到达 σ_2. 于是

$$\mathscr{M}\text{ 接受 }w,\text{当且仅当 }\sigma_0\overset{\lceil as(n)\rceil}{\vdash}\sigma_f.$$

显然有下述递推关系：对任意的格局 σ_1,σ_2，

$$\sigma_1\overset{0}{\vdash}\sigma_2\quad\Leftrightarrow\quad\sigma_1=\sigma_2\text{或}\sigma_1\vdash\sigma_2,$$

$$\sigma_1\overset{i}{\vdash}\sigma_2\quad\Leftrightarrow\quad\text{存在}\sigma\text{使得}\sigma_1\overset{i-1}{\vdash}\sigma\text{且}\sigma\overset{i-1}{\vdash}\sigma_2,i\geqslant 1.$$

图12.1给出按上述思想模拟 $\mathscr{M}$ 的算法. 这个算法可以用一台 DTM $\mathscr{M}'$ 来实现.

由于 $s(n)\geqslant\log_2 n$，$\mathscr{M}'$ 可以用 $O(s(n))$ 个方格存放一个 $\mathscr{M}$ 关于 w 的格局. TEST(σ_1,σ_2,i)的计算是一个递归过程. 当 $i>0$时，TEST(σ_1,σ_2,i)归结为对 $\mathscr{M}$ 关于 w 的每一个格局 σ 计算 TEST$(\sigma_1,\sigma,i-1)$和 TEST$(\sigma,\sigma_2,i-1)$. $\mathscr{M}'$ 对每一个 σ 的计算重复使用工作单元，并且在 TEST$(\sigma_1,\sigma,i-1)$计算结束后开始计算 TEST$(\sigma,\sigma_2,i-1)$，使它们可以使用同一组工作单元. 为了控制递归计算的进程，$\mathscr{M}'$ 用一条工作带作为记录调用 TEST 的运行记录的栈. 每个调用需存储它的参数 σ_1,σ_2 和 i. 由于 $i\leqslant\lceil as(n)\rceil$，存储这3个参数需 $O(s(n))$ 个方格. 开始计算时 i 的值为 $\lceil as(n)\rceil$，以后每次调用 i 的值减1，当 $i=0$时不产生调用，故栈中最多有 $\lceil as(n)\rceil$ 个记录. 因此，栈使用的方格数为 $O(s^2(n))$. 其他运算都可以在 $O(s(n))$ 空间内完成. 所以，$\mathscr{M}'$ 是 $O(s^2(n))$ 空间界限的. 由定

理12.12(3)，得证 $L(\mathscr{M})\subseteq\mathrm{DSPACE}(s^2(n))$.

```
begin
输入 w;
设 |w|=n, M 关于 w 的初始格局为 σ0;
if TEST (σ0,σf,⌈as(n)⌉)then 接受
else 拒绝
end;
procedure TEST(σ1,σ2,i);
begin
if i=0 then
    if σ1=σ2 ∨ σ1 ⊢ σ2 then return true
     else return false
else begin
     for M 关于 w 的每一个格局 σ do
     if TEST (σ1,σ,i−1) ∧ TEST (σ,σ2,
i−1)
     then return true;
     reture false
     end
end;
```

图12.1 模拟 $\mathscr{M}$ 的算法

最后，利用 $s(n)$ 是空间可构造的，$\mathscr{M}'$ 可以在工作带上标明 $s(n)$ 个方格，从而使 $\mathscr{M}'$ 能够枚举 $\mathscr{M}$ 关于 w 的格局和给出参数 i 的初值 $\lceil as(n)\rceil$. □

下面给出最常用的复杂性类：

$$\mathrm{L}=\mathrm{DSPACE}(\log_2 n)$$

$$\mathrm{NL}=\mathrm{NSPACE}(\log_2 n)$$

$$\mathrm{P}=\bigcup_{i>0}\mathrm{DTIME}(n^i)$$

$$\mathrm{NP}=\bigcup_{i>0}\mathrm{NTIME}(n^i)$$

$$\mathrm{PSPACE}=\bigcup_{i>0}\mathrm{DSPACE}(n^i)$$

$$\mathrm{NPSPACE}=\bigcup_{i>0}\mathrm{NSPACE}(n^i)$$

$$\mathrm{EXPTIME}=\bigcup_{i>0}\mathrm{DTIME}(2^{n^i})$$

$$\mathrm{NEXPTIME}=\bigcup_{i>0}\mathrm{NTIME}(2^{n^i})$$

$$\mathrm{EXPSPACE}=\bigcup_{i>0}\mathrm{DSPACE}(2^{n^i})$$

根据前面的有关定理，它们之间有下述关系：

$$L\subseteq NL\subseteq P\subseteq NP\subseteq PSPACE=NPSPACE$$
$$\subseteq EXPTIME\subseteq NEXPTIME\subseteq EXPSPACE,$$

以及

$$L(NL)\subset PSPACE\subset EXPSPACE,$$
$$P\subset EXPTIME.$$

后面两个真包含式表明第一个包含式中的包含关系一定有真包含关系. 遗憾的是，这些包含关系中哪些是真包含、哪些是等于，至今没有解决. 实际上，这是计算复杂性理论研究的核心问题之一. 其中人们特别关注“P=NP?”，它是理论计算机科学中尚未解决的最重要的问题之一。下一章将专用来介绍这方面的内容。

第十三章　NP 完全性

13.1　P　与　NP

为什么“P＝NP?”会成为理论计算机科学界关注的一个焦点问题？让我们先看看多项式时间算法和指数时间算法的区别.

表 13-1 给出在一台每秒做 1 亿次运算的计算机上，用几个多项式时间算法和指数时间算法解不同规模的问题实例所需的时间. 随着规模 n 的变大，两个指数时间算法所需的时间以爆炸性的速率迅速增加. 当 n 比较大时，它们所需的时间是无法承受的.

表 13-1　多项式时间算法与指数时间算法的运行时间对比

时间复杂度	规模 n					
	10	20	30	40	50	60
n	10^{-7}秒	2×10^{-7}秒	3×10^{-7}秒	4×10^{-7}秒	5×10^{-7}秒	6×10^{-7}秒
n^2	10^{-6}秒	4×10^{-6}秒	9×10^{-6}秒	1.6×10^{-5}秒	2.5×10^{-5}秒	3.6×10^{-5}秒
n^3	10^{-5}秒	8×10^{-5}秒	2.7×10^{-4}秒	6.4×10^{-4}秒	1.25×10^{-3}秒	2.16×10^{-3}秒
n^5	.001 秒	.032 秒	.243 秒	1.02 秒	3.12 秒	7.80 秒
2^n	10^{-6}秒	.001 秒	10.74 秒	3.05 小时	130.3 天	366 年
3^n	5.9×10^{-4}秒	34.8 秒	23.7 天	3855 年	2×10^6 世纪	1.3×10^{11}世纪

可能有人会说，只要提高计算机的速度就能解决这个问题. 表 13-2 给出提高计算机的速度对这几个算法在单位时间内可解问题的规模的影响. 从表中看到，把计算机的速度提高 1000 倍，n^3 算法在单位时间内可解问题的规模是原来的 10 倍. 而对于 2^n 算法，只增加不到 10!

表 13-2　提高计算机的速度对单位时间内可解的问题规模的影响

时间复杂度	单位时间内可解的问题规模		
	现在的计算机	快 100 倍	快 1000 倍
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_5	$N_5+6.64$	$N_5+9.97$
3^n	N_6	$N_6+4.19$	$N_6+6.29$

现在已经普遍接受这样的看法，多项式时间算法是“好的算法”，因为它是“足够有效的”. 如果一个问题有了多项式时间算法，就可以认为这个问题已经“很好地解决了”. 有多项式时间算法的问题是**易解的**. 如果一个问题不存在多项式时间算法，则称它是**难解的**.

这就是 **Cook-Karp** 论题：L 是易解的当且仅当 $L\in$P. 根据这个论题和前面的结果，EXPTIME 中有难解的语言. 要问：NP 中有难解的语言吗？ 即，P=NP?

下面给出 NP 的另一种描述，这将有利于了解这个复杂性类中的语言的特性.

设字母表 A，把 A^* 上的有序对(x,y)表示成 $x\#y$，其中 $\#\notin A$. 于是，A^* 上的二元关系 R 可以看作语言

$$L_R=\{x\#y\mid(x,y)\in R\}.$$

如果这个语言是多项式时间可判定的，则称 R **是多项式时间可判定的**. 如果对于每一对(x,y)，$(x,y)\in R$ 蕴涵 $|y|\leqslant|x|^k$，这里 $k\geqslant1$是一个常数，则称 R 是**多项式平衡的**.

定理 13.1　$L\in$NP 的充分必要条件是存在多项式时间可判定和多项式平衡的二元关系 R，使得

$$L=\{x\mid\exists\ y\quad(x,y)\in R\}.$$

证：充分性. 设 DTM$\mathscr{M}$ 接受 L_R 且具有时间复杂度上界 n^t. 又 R 是多项式平衡的，存在常数 k 使得$(x,y)\in R$ 蕴涵 $|y|\leqslant n^k$. NTM$\mathscr{M}'$ 的计算分成两个阶段：猜想阶段和验证阶段. 对输入 x，$\mathscr{M}'$ 首先在猜想阶段任意给出(猜想)一个长度不超过 n^k 的字符串 y，这里 $n=|x|$. 由于 n^k 是时间可构造的，$\mathscr{M}'$ 可以控制 y 的长度. 然后进入验证阶段，模拟 $\mathscr{M}$ 关于 $x\#y$ 的计算. $\mathscr{M}'$ 的这个计算接受 x 当且仅当 $\mathscr{M}$ 接受 $x\#y$. 显然，$\mathscr{M}'$ 接受 L. 它的运行时间不超过 $n^k+(n+n^k+1)^t=O(n^{kt})$，故 $L\in$NP.

必要性. 设 n^k 时间界限的 NTM$\mathscr{M}$ 接受 L，在每一步至多有 r 个可能的动作. 不妨设 L 的字母表 A 包含 $T=\{1,2,\cdots,r\}$. 对于每一个输入 x，$\mathscr{M}$ 关于 x 的计算可以用 T 上的字符串 y 来表示：计算的第 i 步选择第 y_i 个动作，这里 y_i 是 y 的第 i 个符号. 显然，$|y|\leqslant|x|^k$. 令

$$R=\{(x,y)\mid x\in L,y\text{ 表示 }\mathscr{M}\text{ 接受 }x\text{ 的计算}\}.$$

显然，R 是多项式平衡的，并且

$$L=\{x\mid\exists\ y\ \ (x,y)\in R\}.$$

判定 R 的 DTM$\mathscr{M}'$ 如下工作：对每一个输入 w，首先检查 w 是否形如 $x\#y$，其中 $x\in A^*$，$y\in T^*$. 若不是，则 $\mathscr{M}'$ 拒绝 w；若是，则 $\mathscr{M}'$ 按照 y 的含义模拟 $\mathscr{M}$ 关于 x 的计算. $\mathscr{M}'$ 接受 $x\#y$ 当且仅当这个计算是 $\mathscr{M}$ 接受 x 的计算. $\mathscr{M}'$ 的计算时间为 $O(|w|)$. □

设 $x\in L$，把使得$(x,y)\in R$ 的 y 称作 x 的**证明**. 对于 NP 中的语言 L，当 $x\in L$ 时并不要求在多项式时间内找到它的证明，而只要求能在多项式时间内验证一个证明. 很多问题都具有这种多项式时间可验证性. 例如：

Hamilton 回路问题(HC)：任给一个无向图 G，问 G 中有 Hamilton 回路吗？所谓 Hamilton 回路是图中经过每一个顶点且每一个顶点恰好经过一次的回路.

还没有找到判断一个无向图是否有 Hamilton 回路的多项式时间算法. 现有的算法在本质上都是穷举搜索. n 个顶点有 $n!$ 种排

列. 这类算法都是指数时间的. 但是,HC 显然是多项式时间可验证的. 任给顶点的一个排列,容易验证它是否是一条 Hamilton 回路. 如果图 G 有 Hamilton 回路,则存在这样的顶点排列.

每一个最优化问题都有对应的判定形式的问题. 例如:

货郎问题(最优化形式):任给 n 个"城市"$C=\{c_1,c_2,\cdots,c_n\}$,每一对"城市"c_i 和 $c_j(i\neq j)$的"距离"为 $d_{ij}\in Z^+$,这里 $d_{ij}=d_{ji}$,求一条最短的"环游",即 $1,2,\cdots,n$ 的排列 π 使得

$$\sum_{i=1}^{n-1}d_{\pi(i)\pi(i+1)}+d_{\pi(n)\pi(1)}$$

最小. (Z^+是正整数集).

对应的判定问题是

货郎问题(TSP):任给 n 个"城市"$C=\{c_1,c_2,\cdots,c_n\}$,每一对"城市"c_i 和 $c_j(i\neq j)$的"距离"为 $d_{ij}\in Z^+$以及界限 $K\in Z^+$,这里 $d_{ij}=d_{ji}$,问:是否存在长度不超过 K 的"环游"?即,是否存在 $1,2,\cdots,n$ 的排列 π 使得

$$\sum_{i=1}^{n-1}d_{\pi(i)\pi(i+1)}+d_{\pi(n)\pi(1)}\leqslant K?$$

借助货郎问题(最优化形式)的算法容易给出货郎判定问题的算法:求出最短的"环游",若"环游"的长$\leqslant K$ 则回答"是";否则回答"否". 不难看到,如果这个最优化问题算法是多项式时间的,则关于判定问题的算法也是多项式时间的. 反过来,若这个判定问题是难解的,则对应的最优化问题也是难解的. 以后将只讨论判定问题的复杂性.

一般地,对于最小化问题,对应的判定问题的实例中要添加一个数 K,并问:"存在…$\leqslant K$?". 对于最大化问题,对应的判定问题的实例中也要添加一个数 K,并问:"存在…$\geqslant K$?".

TSP 也具有多项式时间可验证性. 任给一条"环游",计算它的长度并与 K 比较,就能知道这条"环游"是否是一个证明. 这些都能在多项式时间内完成. 但是,与 Hamilton 回路问题一样,至今

没有找到它的多项式时间算法、又没能证明它不是多项式时间可解的.

在逻辑、图论、网络、代数、数论、形式语言与自动机、程序设计、排序与调度、数学规划、游戏等各种领域内有许多各式各样的问题,它们都和上面两个问题一样,具有多项式时间可验证性,但至今不知道有没有多项式时间算法. 也就是说,这些问题都在 NP 中,但不知道是否在 P 中. 直觉上,寻找证明要比验证证明困难得多,即 P⊂NP. 但至今不知道这是否是真的. 因此,"P=NP?"不仅在理论上、而且在实际中有重大意义,成为理论计算机科学中的一个著名问题.

说一个判定问题在 NP 或 P 中是指它对应的语言在 NP 或 P 中. 如 8.1 节中那样,判定问题与语言的对应是通过编码实现的.

设判定问题 $\pi=(D_\pi,Y_\pi)$,编码 $e:D_\pi\to A^*$. 与 π 对应的语言为

$$L[\pi,e]=\{e(I)\mid I\in Y_\pi\}.$$

严格地说,只有在给定的编码下谈判定问题的复杂性才有意义. 但是,实际中常用的编码往往不会影响判定问题对应的语言在要讨论的复杂性类中的位置. 于是,在这个意义下可以讨论判定问题本身的复杂性,而不涉及具体的编码.

例如,当判定问题 π 的实例是一个无向图时(如 Hamilton 回路问题),可以用顶点表和边表、邻点表、相邻矩阵 3 种方式给简单无向图编码. 按这 3 种编码方式,图 13.1 的代码如下:

顶点表和边表　$V1V10V11V100V101/V1V10$
$/V10V11/V11V101/V10V101/V1V101$

邻点表　$/V10V101/V1V11V101/V10V101$
$//V1V10V11$

相邻矩阵　01001/10101/01001/00000/11100

设函数 $f,g:S\to N$. 如果存在多项式 p 和 q 使得对所有的 $x\in S$,有 $f(x)\leqslant p(g(x))$ 和 $g(x)\leqslant q(f(x))$,则称 f 和 g **是多项式相关的.**

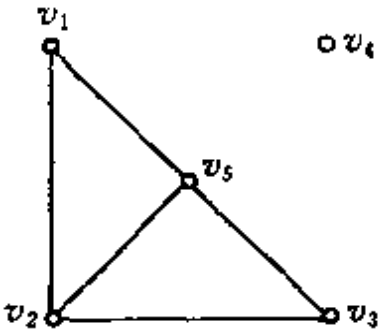


图 13.1

设图 G 有 v 个顶点、e 条边，则 G 在这 3 种编码下的代码长度的上下界在表 13-3 中给出. 注意到 $e\leqslant\frac{1}{2}v(v-1)$，这些代码的长度都是多项式相关的. 因此，如果在一个编码下 π 对应的语言在 P(NP. PSPACE)中，则在另外二个编码下 π 对应的语言也在 P(NP, PSPACE)中.

表 13-3 在不同的编码下无向图(顶点数 v、边数 e)的代码长度

编码方式	下 界	上 界
顶点表和边表	$2v+5e$	$2v+5e+(v+2e)\lceil\log_2 v\rceil$
邻点表	$v+4e$	$v+4e+2e\lceil\log_2 v\rceil$
相邻矩阵	v^2+v-1	v^2+v-1

以后总假定使用的编码都具有这种多项式相关性，即对于判定问题 π 的任意两个编码 e 和 e'，存在多项式 p 和 q 使得对所有的实例 $I\in D_\pi$，有 $e(I)|\leqslant p(|e'(I)|)$和$|e'(I)|\leqslant q(|e(I)|)$. 需要指出的是，在这样的编码中只能采用二进制(或底大于 2 的进制)表示，而不能用一进制表示作为自然数的代码. 一进制表示的长度和二进制表示的长度不是多项式相关的. 在这样的假定下，以后将采用下述说法：

设复杂性类 $\mathscr{C}$，$P\subseteq\mathscr{C}$. 如果判定问题 π 在编码 e 下对应的语言 $L[\pi,e]\in\mathscr{C}$，则称 π 在 $\mathscr{C}$ 中，也记作 $\pi\in\mathscr{C}$.

更进一步地，对判定问题 π 的每一个实例 $I\in D_\pi$ 规定一个正整数，记作$|I|$，称作 I 的**规模**. $|I|$反映了描述 I 所需要的符号量，可把 π 的复杂度(严格地说，应该是 $L[\pi,e]$的复杂度)表示成$|I|$的函数. 当然，这里也要求$|I|$与$|e(I)|$多项式相关. 除此之外，规定$|I|$时有相当大的自由，通常都用问题中的某些参数自然地表示

出来. 例如，当 I 是一个 v 个顶点 e 条边的简单无向图时，可以取 $|I|=v+e$ 或 $|I|=v$. 它们都与表 13-3 中 3 种编码下的代码长度多项式相关.

对于编码的另一个要求是可译码性. 当考虑的复杂性类包含 P 时，要求在多项式时间内能从代码中识别出所需要的分量. 今后同样也假设使用的编码都满足这个要求，实际上通常使用的编码都满足这个要求. 于是，这就允许我们像通常一样地用实例中的自然成份(如图的顶点和边)作为运算对象来描述算法，而不必针对代码设计 Turing 机. 只要这样的算法能用 $|I|$ 的多项式来操作完成，就可以用一台多项式时间界限的 Turing 机实现. 称这样的算法是**多项式时间的**. 表 13-4 给出这些形式的和非形式的术语之间的对应. 今后的定义和关于一般性质的定理仍以形式的方式给出，而对具体的判定问题通常都采用非形式的方式.

表 13-4　形式的和非形式的术语对照

形式的	非形式的
语言 L	判定问题 π
字符串 x	实例 I
字符串长度 $\|x\|$	实例的规模 $\|I\|$
DTM	算法
NTM	非确定型算法
多项式时间的	多项式时间的
语言类 $\mathscr{C}$	判定问题类 $\mathscr{C}$

13.2　多项式时间变换和完全性

如果 P≠NP，那么 NP 中“最难的”语言必不属于 P. 如果 NP 中“最难的”语言属于 P，那么 NP 中所有的语言都属于 P，得到 P=NP. 从而把“P=NP?”归结为“NP 中‘最难的’语言是否属于 P?”为了描述一个复杂性类中“最难的”语言，我们首先引入下述

定义.

定义 13.1 设字母表 $A, L_1, L_2 \subseteq A^*$. 如果函数 $f: A^* \to A^*$ 满足条件：

(1) f 是多项式时间可计算的，

(2) $\forall\ x \in A^*, x \in L_1 \Leftrightarrow f(x) \in L_2$，

则称 f 是从 L_1 到 L_2 的**多项式时间变换**，或**多项式时间的多一归约**.

如果存在 L_1 到 L_2 的多项式时间变换，则称 L_1 **可多项式时间变换(多一归约)到** L_2，记作 $L_1 \leqslant_m L_2$.

对应地，有下述非形式的定义：

设判定问题 $\pi_1 = (D_1, Y_1), \pi_2 = (D_2, Y_2)$. 如果函数 $f: D_1 \to D_2$ 满足条件：

(1) f 是多项式时间可计算的，

(2) $\forall\ I \in D_1, I \in Y_1 \Leftrightarrow f(I) \in Y_2$，

则称 f 是从 π_1 到 π_2 的**多项式时间变换**，或**多项式时间的多一归约**.

如果存在 π_1 到 π_2 的多项式时间变换，则称 π_1 **可多项式时间变换(多一归约)到** π_2，记作 $\pi_1 \leqslant_m \pi_2$.

[**例 13.1**] $\mathrm{HC} \leqslant_m \mathrm{TSP}$.

多项式时间变换 f 规定如下：对 HC 的每一个实例 I，I 是一个无向图 $G=(V,E)$，对应的 $f(I)$ 为“城市”集 V，界限 $|V|$ 以及对所有的 $u, v \in V$，

$$d(u,v) = \begin{cases} 1 & \text{若} \{u,v\} \in E \\ 2 & \text{否则} \end{cases}$$

显然，f 是多项式时间可计算的. 在 $f(I)$ 中一条“环游”的长度不超过 $|V|$ (实际上恰好等于 $|V|$) 当且仅当它是 G 的一条 Hamilton 回路，故 $I \in Y_{\mathrm{HC}} \Leftrightarrow f(I) \in Y_{\mathrm{TSP}}$.

与定义 13.1 类似，下面的定理和定义都有对应的非形式叙述，不再一一给出. 但是，可能会使用它们.

定理 13.2 设 $L_1 \leqslant_m L_2$，$L_2 \leqslant_m L_3$，则 $L_1 \leqslant_m L_3$.

证：设 $L_1, L_2, L_3 \subseteq A^*$，$f, g$ 分别是从 L_1 到 L_2、L_2 到 L_3 的多项式时间变换. DTM $\mathscr{M}_1$ 和 $\mathscr{M}_2$ 分别计算 f 和 g，它们分别是多项式 p_1 和 p_2 时间界限的. 令 $h(x)=g(f(x))$，$\forall\, x \in A^*$.

首先，$\forall\, x \in A^*$，$x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow h(x) \in L_3$.

其次，设 DTM $\mathscr{M}$ 是 $\mathscr{M}_1$ 和 $\mathscr{M}_2$ 的合成：任给输入 x，$\mathscr{M}$ 模拟 $\mathscr{M}_1$ 关于 x 的计算得到 $f(x)$，接着模拟 $\mathscr{M}_2$ 以 $f(x)$作为输入的计算，$\mathscr{M}$ 把 $\mathscr{M}_2$ 的输出作为自己的输出. 显然，$\mathscr{M}$ 计算函数 h. 不妨设 p_1 和 p_2 都是单调增加的. 注意到 $|f(x)| \leqslant p_1(|x|)$，$\mathscr{M}$ 的运行时间不超过

$$p_1(|x|) + p_2(|f(x)|) \leqslant p_1(|x|) + p_2(p_1(|x|)),$$

这也是 $n=|x|$ 的多项式.

因此，h 是从 L_1 到 L_3 的多项式时间变换. □

定理 13.3 设 $L_1 \leqslant_m L_2$，那么

(1) 若 $L_2 \in \mathrm{P}$，则 $L_1 \in \mathrm{P}$.

(2) 若 $L_2 \in \mathrm{NP}$，则 $L_1 \in \mathrm{NP}$.

证：(1) 设 f 是 L_1 到 L_2 的多项式时间变换，DTM $\mathscr{M}_1$ 计算 f，它是多项式 p 时间界限的. 又设 DTM $\mathscr{M}_2$ 接受 L_2，它是多项式 q 时间界限的. DTM $\mathscr{M}$ 是 $\mathscr{M}_1$ 和 $\mathscr{M}_2$ 的合成：任给输入 x，$\mathscr{M}$ 模拟 $\mathscr{M}_1$ 关于 x 的计算得到 $f(x)$，然后模拟 $\mathscr{M}_2$ 以 $f(x)$作为输入的计算. $\mathscr{M}$ 接受 x 当且仅当 $\mathscr{M}_2$ 接受 $f(x)$.

根据多项式时间变换的定义，$L(\mathscr{M})=L_1$. 又类似定理 13.2，可证 $\mathscr{M}$ 是多项式时间界限的. 从而，$L_1 \in \mathrm{P}$.

(2) 类似可证. □

$\leqslant_m$ 是任何语言类上的偏序关系，它反映了语言之间的难易. 若 $L_1 \leqslant_m L_2$，则在下述意义下 L_2 的难度不低于 L_1：$L_2 \in \mathrm{P}$ 蕴涵 $L_1 \in \mathrm{P}$；或者反过来说，$L_1 \notin \mathrm{P}$ 蕴涵 $L_2 \notin \mathrm{P}$. 于是，可以利用 $\leqslant_m$ 给出语言类 $\mathscr{C}$ 中“最难的”语言的定义，即 $\mathscr{C}$ 完全的语言.

定义 13.2 设语言类 $\mathscr{C}$，如果 $\forall\, L' \in \mathscr{C}$，$L' \leqslant_m L$，则称 L 是 $\mathscr{C}$

难的.

如果 L 是 $\mathscr{C}$ 难的且 $L\in\mathscr{C}$，则称 L 是 $\mathscr{C}$ **完全的.**

下面几个结论都很容易证明，留作练习.

定理 13.4 设语言类 $\mathscr{C}$，L 是 $\mathscr{C}$ 难的. 那么

(1) 若 $L\in$P，则 $\mathscr{C}\subseteq$P.

(2) 若 $\mathscr{C}-$P$\neq\varnothing$，则 $L\notin$P.

推论 13.5 如果存在 NP 完全的语言 $L\in$P，则 P=NP.

定理 13.6 设语言类 $\mathscr{C}$，如果存在 $\mathscr{C}$ 难的语言 L' 使得 $L'\leqslant_m L$，则 L 也是 $\mathscr{C}$ 难的.

推论 13.7 设 $L\in$NP，如果存在 NP 完全的语言 L' 使得 $L'\leqslant_m L$，则 L 是 NP 完全的.

根据推论 13.5，只要能找到一个 NP 完全的语言 $L\in$P 或 $\notin$P，问题"P=NP?"就解决了. 这样就把原来要考虑 NP 中所有的语言转化成只需要考虑一个语言. 尽管至今尚未找到这样的语言，至少在理论上进了一大步.

推论 13.7 提供了证明 NP 完全语言的一条途径. 为了证明 L 是 NP 完全的，只要

(1) 证明 $L\in$NP；

(2) 把一个已知的 NP 完全语言 L' 多项式时间变换到 L.

13.3 Cook 定 理

S. A. Cook 于 1971 年给出第一个 NP 完全问题. 这是数理逻辑中的一个问题.

取全功能集$\{\wedge,\vee,\neg\}$，其中$\wedge$是合取联结词、$\vee$是析取联结词、$\neg$是否定联结词. 由变元、圆括号和联结词$\wedge$、$\vee$、$\neg$组成的表达式称作**合式公式**. 变元和它的否定称作**文字**. 有限个文字的析取称作**简单析取式**. 有限个简单析取式的合取称作**合取范式**. 例如，

$$\neg x_1\vee(x_2\wedge x_3\wedge(\neg x_2\vee x_4))$$

是一个合式公式.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4)$$

是一个合取范式，其中 $x_1 \vee \neg x_2 \vee x_3$，$\neg x_1 \vee x_2$，$x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4$ 都是简单析取式.

设 F 是关于变元 $x_1, x_2, \cdots, x_n$ 的合式公式，如果真值赋值 $t: \{x_1, x_2, \cdots, x_n\} \to \{0,1\}$ 使得 $t(F)=1$，则称 t 是 F 的一个**成真赋值**. 这里 1 表示真值，0 表示假值. 如果合式公式 F 有成真赋值，则称 F 是**可满足的**.

可满足性问题(SAT)：任给一个合取范式 F，问 F 是否是可满足的?

定理 13.8(Cook 定理)　可满足性问题是 NP 完全的.

证：判断 SAT 的非确定型多项式时间的算法如下：对任给的合取范式 F，先猜想一个赋值，然后检查它是否使 F 成真. 故 SAT $\in$NP.

任给 $L \in$ NP，存在多项式时间界限的 NTM $\mathscr{M}$ 接受 L. 根据定理 12.4，不妨设 $\mathscr{M}$ 是一台单向无穷的单带在线 Turing 机. 它的字母表为 A，有 r 个状态 $q_1, q_2, \cdots, q_r$，$l+1$ 个带符号 $s_0, s_1, \cdots, s_l$，其中 q_1 是初始状态，q_r 是唯一的接受状态，s_0 是空白符，s_1 是左端标志符 #. 多项式 p 为 $\mathscr{M}$ 的时间复杂度上界.

对每一个 $x \in A^*$，要构造一个合取范式 F_x 满足下述条件，从而给出从 L 到 SAT 的多项式时间变换.

(1) F_x 可以在 $|x|$ 的多项式时间内构造出来；

(2) F_x 是可满足的当且仅当 $x \in L$，即 F_x 存在成真赋值当且仅当 $\mathscr{M}$ 存在接受 x 的计算.

设 $|x|=n$，计算至多使用 $p(n)+1$ 个方格，从左至右依次编号为 $0, 1, \cdots, p(n)$. 用“时刻 t”表示“在完成第 t 步时”，这里 $0 \leqslant t \leqslant p(n)$. $\mathscr{M}$ 关于 x 的计算可用 $p(n)+1$ 个格局来描述，每个格局

只需考虑 $p(n)+1$ 个方格的内容. 这使得可以用合取范式来描述 $\mathscr{M}$ 关于 x 的计算. 约定：$\mathscr{M}$ 停机后格局保持不变. 从而可以认为 $\mathscr{M}$ 关于 x 的计算恰好有 $p(n)+1$ 个格局.

变量的含义在表 13-5 中给出.

表 13-5

变 量	范 围	含 义
$q[t,i]$	$0\leqslant t\leqslant p(n)$ $1\leqslant i\leqslant r$	在时刻 t，$\mathscr{M}$ 处于状态 q_i
$h[t,j]$	$0\leqslant t\leqslant p(n)$ $0\leqslant j\leqslant p(n)$	在时刻 t，带头扫描方格 j
$s[t,j,k]$	$0\leqslant t\leqslant p(n)$ $0\leqslant j\leqslant p(n)$ $0\leqslant k\leqslant l$	在时刻 t，方格 j 内的符号为 s_k

F_x 分成 6 部分：$F_x=F_1\wedge F_2\wedge F_3\wedge F_4\wedge F_5\wedge F_6$. 每一部分为真的含义是：

F_1：在每一时刻，$\mathscr{M}$ 恰好处于一个状态；

F_2：在每一时刻，带头恰好扫描一个方格；

F_3：在每一时刻，每个方格内恰好有一个符号；

F_4：在时刻 0，$\mathscr{M}$ 处于关于 x 的初始格局；

F_5：在时刻 $p(n)$，$\mathscr{M}$ 处于状态 q_r；

F_6：对每一个 $t(0\leqslant t<p(n))$，$\sigma_t\vdash\sigma_{t+1}$.

构造如下：

$$F_1=\bigwedge_{0\leqslant t\leqslant p(n)}\Big(\Big(\bigvee_{1\leqslant i\leqslant r}q[t,i]\Big)\wedge\Big(\bigwedge_{1\leqslant i<i'\leqslant r}(\neg q[t,i]\vee\neg q[t,i'])\Big)\Big).$$

$$F_2=\bigwedge_{0\leqslant t\leqslant p(n)}\Big(\Big(\bigvee_{0\leqslant j\leqslant p(n)}h[t,j]\Big)\wedge\Big(\bigwedge_{0\leqslant j<j'\leqslant p(n)}(\neg h[t,j]\vee\neg h[t,j'])\Big)\Big).$$

$$F_3=\bigwedge_{\substack{0\leqslant t\leqslant p(n)\\0\leqslant j\leqslant p(n)}}\Big(\Big(\bigvee_{0\leqslant k\leqslant l}s[t,j,k]\Big)$$

$$\wedge \Big(\bigwedge_{0\leqslant k<k'\leqslant l}(\neg s[t,j,k] \vee \neg s[t,j,k'])\Big)\Big).$$

$$F_4 = q[0,1] \wedge h[0,0] \wedge s[0,0,1]$$
$$\wedge \Big(\bigwedge_{1\leqslant j\leqslant n} s[0,j,k_j]\Big)$$
$$\wedge \Big(\bigwedge_{n+1\leqslant j\leqslant p(n)} s[0,j,0]\Big),$$

其中 $x=s_{k_1}s_{k_2}\cdots s_{k_n}$.

$F_5=q[p(n),r]$.

F_6 又可分解成两部分 $F_6=F_6'\wedge F_6''$,其中

F_6':对每一对 t 和 $j(0\leqslant t<p(n),0\leqslant j\leqslant p(n))$,若带头在时刻 t 不扫描方格 j,则方格 j 的内容在时刻 $t+1$ 和在时刻 t 的相同.

F_6'':从时刻 t 到时刻 $t+1(0\leqslant t<p(n))$,$\mathscr{M}$ 的状态、带头位置以及方格的内容按照动作函数 δ 改变.

注意到

$$\neg h[t,j] \wedge s[t,j,k] \to s[t+1,j,k]$$
$$\Leftrightarrow \neg(\neg h[t,j] \wedge s[t,j,k]) \vee s[t+1,j,k]$$
$$\Leftrightarrow h[t,j] \vee \neg s[t,j,k] \vee s[t+1,j,k],$$

故

$$F_6' = \bigwedge_{\substack{0\leqslant t<p(n)\\0\leqslant j\leqslant p(n)\\0\leqslant k\leqslant l}} (h[t,j] \vee \neg s[t,j,k] \vee s[t+1,j,k]).$$

对每一对 i 和 $k(0\leqslant i\leqslant r,0\leqslant k\leqslant l)$分两种情况:

若 $\delta(q_i,s_k)=\varnothing$,则对所有的 t 和 $j(0\leqslant t<p(n),0\leqslant j\leqslant p(n))$,

$$q[t,i] \wedge h[t,j] \wedge s[t,j,k]$$
$$\to q[t+1,i] \wedge h[t+1,j] \wedge s[t+1,j,k]$$
$$\Leftrightarrow \neg(q[t,i] \wedge h[t,j] \wedge s[t,j,k])$$
$$\vee (q[t+1,i] \wedge h[t+1,j] \wedge s[t+1,j,k])$$
$$\Leftrightarrow (\neg q[t,i] \vee \neg h[t,j]$$
$$\vee \neg s[t,j,k] \vee q[t+1,i])$$

$$\wedge\ (\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k] \vee h[t+1,j])$$
$$\wedge\ (\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k] \vee s[t+1,j,k]).$$

于是，F_6'' 含有

$$\bigwedge_{\substack{0\leqslant t<p(n)\\0\leqslant j\leqslant p(n)}} ((\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k] \vee q[t+1,i])$$
$$\wedge\ (\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k] \vee h[t+1,j])$$
$$\wedge\ (\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k] \vee s[t+1,j,k])).$$

若 $\delta(q_i,s_k)=\{(q_{i_1},\Delta_1),\cdots,(q_{i_m},\Delta_m)\}$，这里 $m\geqslant 1$，每个 $\Delta_v=s_{k_v}$，R 或 L，则对所有的 t 和 $j(0\leqslant t<p(n),0\leqslant j\leqslant p(n))$，

$$q[t,i] \wedge h[t,j] \wedge s[t,j,k]$$
$$\rightarrow \bigvee_{1\leqslant v\leqslant m} (q[t+1,i_v] \wedge h[t+1,j_v] \wedge s[t+1,j,k_v'])$$
$$\Leftrightarrow (\neg q[t,i] \vee \neg h[t,j] \vee \neg s[t,j,k]) \vee \left(\bigvee_{1\leqslant v\leqslant m} (q[t+1,i_v] \wedge h[t+1,j_v] \wedge s[t+1,j,k_v'])\right),$$

其中，当 $\Delta_v=s_{k_v}$ 时，$k_v'=k_v$ 且 $j_v=j$；当 $\Delta_v=R$ 时，$j_v=j+1$ 且 $k_v'=k$；当 $\Delta_v=L$ 时，$j_v=j-1$ 且 $k_v'=k$.

利用分配律可将上式化成合取范式，记作 $C[i,k]$. $C[i,k]$ 是 3^m 个简单析取式的合取，每一个简单析取式有 $3+m$ 个文字. 于是，F_6'' 含有

$$\bigwedge_{\substack{0\leqslant t<p(n)\\0\leqslant j\leqslant p(n)}} C[i,k].$$

最后，注意到 r,l 以及每一个 $|\delta(q_i,s_k)|(1\leqslant i\leqslant r,0\leqslant k\leqslant l)$ 都

是常数(对固定的 $\mathscr{M}$ 而言),F_x 是 $O(p^3(n))$ 个简单析取式的合取.根据 $\mathscr{M}$ 和 x 生成这些简单析取式是很直接的,故构造 F_x 可在 $|x|$ 的多项式时间内完成. □

合取范式是合式公式的特殊情况,下述问题是可满足性问题的推广.

合式公式的可满足性问题:任给一个合式公式 F,问 F 是否是可满足的?

由于 SAT 是这个问题的子问题,显然这个问题是 NP 难的.实际上,把 SAT 的实例 I 映射到自身就是从 SAT 到这个问题的多项式时间变换.而定理 13.8 证明中的非确定型多项式时间算法也适用于这个问题,故有:

推论 13.9 合式公式的可满足性问题是 NP 完全的.

下面考虑 SAT 的一种特殊情况:合取范式的每一个简单析取式恰好有 3 个文字.这样的合取范式叫做三元合取范式.SAT 的这个子问题称作**三元可满足性问题**(3SAT).

定理 13.10 三元可满足性问题是 NP 完全的.

证:由于 3SAT 是 SAT 的子问题,故 3SAT$\in$NP.

为了证明 3SAT 是 NP 难的,要证 SAT$\leqslant_m$3SAT.任给一个合取范式 F,要构造一个三元合取范式 F',使得 F 是可满足的当且仅当 F' 是可满足的.

设 $F=\bigwedge\limits_{1\leqslant j\leqslant m} C_j$,其中 $C_j(1\leqslant j\leqslant m)$ 是简单析取式.而 $F'=\bigwedge\limits_{1\leqslant j\leqslant m} F'_j$,对每一个 $j(1\leqslant j\leqslant m)$,$F'_j$ 是一个合取范式,并且

$$F'_j \text{ 是可满足的} \Leftrightarrow C_j \text{ 是可满足的} \qquad (*)$$

分情况如下构造 F'_j.在下面,诸 z_i 表示文字,即 z_i 是一个变元 x,或 $\neg x$.

(1) $C_j=z_1$.引入新变元 y_{j1}, y_{j2},

$$F'_j=(z_1 \vee y_{j1} \vee y_{j2}) \wedge (z_1 \vee \neg y_{j1} \vee y_{j2}) \wedge (z_1 \vee y_{j1} \vee \neg y_{j2})$$

$$\wedge (z_1 \vee \neg y_{j1} \vee \neg y_{j2})$$

由于 $y_{j1} \vee y_{j2}, \neg y_{j1} \vee y_{j2}, y_{j1} \vee \neg y_{j2}, \neg y_{j1} \vee \neg y_{j2}$ 不可能同时为真. 故 F'_j 为真当且仅当 z_1 为真. 从而,(*)成立.

(2) $C_j = z_1 \vee z_2$. 引入新变元 y_j,

$$F'_j = (z_1 \vee z_2 \vee y_j) \wedge (z_1 \vee z_2 \vee \neg y_j).$$

(3) $C_j = z_1 \vee z_2 \vee z_3$. 此时取 $F'_j = C_j$.

对于这两种情况,(*)显然成立.

(4) $C_j = z_1 \vee z_2 \vee \cdots \vee z_k, k \geqslant 4$. 引入 $k-3$ 个新变元 $y_{j1}, y_{j2}, \cdots, y_{jk-3}$,

$$\begin{aligned} F'_j = &(z_1 \vee z_2 \vee y_{j1}) \wedge (\neg y_{j1} \vee z_3 \vee y_{j2}) \\ &\wedge (\neg y_{j2} \vee z_4 \vee y_{j3}) \\ &\wedge \cdots \wedge (\neg y_{jk-4} \vee z_{k-2} \vee y_{jk-3}) \\ &\wedge (\neg y_{jk-3} \vee z_{k-1} \vee z_k). \end{aligned}$$

设赋值 t 使得 $t(C_j)=1$,则存在 $i(1 \leqslant i \leqslant k)$ 使 $t(z_i)=1$. 把 t 扩张到 $\{y_{j1}, y_{j2}, \cdots, y_{jk-3}\}$ 上. 当 $i=1$ 或 2 时,令 $t(y_{js})=0(1 \leqslant s \leqslant k-3)$;当 $i=k-1$ 或 k 时,令 $t(y_{js})=1(1 \leqslant s \leqslant k-3)$;当 $3 \leqslant i \leqslant k-2$ 时,令

$$t(y_{js}) = \begin{cases} 1 & 1 \leqslant s \leqslant i-2 \\ 0 & i-1 \leqslant s \leqslant k-3 \end{cases}$$

这里 1 表示真值,0 表示假值. 不难验证,$t(F'_j)=1$. 反之,设赋值 t 使得 $t(F'_j)=1$. 若 $t(y_{j1})=0$,则 $t(z_1 \vee z_2)=1$;若 $t(y_{jk-3})=1$,则 $t(z_{k-1} \vee z_k)=1$;否则必有 $s(1 \leqslant s \leqslant k-4)$ 使得 $t(y_{js})=1$ 且 $t(y_{js+1})=0$,从而 $t(z_{s+2})=1$. 总之,都有 $t(C_j)=1$. 因此,(*)成立.

F'_j 中简单析取式的个数不超过 C_j 中文字个数的 4 倍,因此可在 $|F|$ 的多项式时间内构造出 F'. 得证这是从 SAT 到 3SAT 的多项式时间变换. □

由于确实存在 NP 完全的语言,推论 13.5 可进一步地改写成:

定理 13.11 $P=\mathrm{NP}$ 当且仅当存在 NP 完全的语言 $L\in\mathrm{P}$.

13.4 NP 完全问题

13.4.1 顶点覆盖、团和独立集问题

设无向图 $G=(V,E)$，$V'\subseteq V$. 如果对每一条边 $e\in E$ 都有 $e\cap V'\neq\varnothing$，则称 V' 是 G 的一个**顶点覆盖**. 如果对 V' 中的任意两点 u 和 $v(u\neq v)$ 都有 $\{u,v\}\in E$，即 V' 导出的子图是一个完全子图，则称 V' 是 G 的一个**团**. 如果对 V' 中的任意两点 u 和 v 都有 $\{u,v\}\notin E$，则称 V' 是 G 的一个**独立集**. 下述图论中的引理表明这三个概念是密切相关的.

引理 13.12 对任意的无向图 $G=(V,E)$ 和子集 $V'\subseteq V$，下述命题是等价的：

(1) V' 是 G 的顶点覆盖.

(2) $V-V'$ 是 G 的独立集.

(3) $V-V'$ 是补图 $\overline{G}=(V,\overline{E})$ 的团，这里 $\overline{E}=\{\{u,v\}\mid u,v\in V,u\neq v$ 且 $\{u,v\}\notin E\}$.

考虑下述 3 个问题：

顶点覆盖问题(VC)：任给一个无向图 $G=(V,E)$ 和非负整数 $K\leqslant|V|$，问 G 是否有大小(即，顶点数)不超过 K 的顶点覆盖？

团问题：任给一个无向图 $G=(V,E)$ 和非负整数 $J\leqslant|V|$，问 G 是否有大小不小于 J 的团？

独立集问题：任给一个无向图 $G=(V,E)$ 和非负整数 $J\leqslant|V|$，问 G 是否有大小不小于 J 的独立集？

根据引理 13.12，很容易把其中的一个问题多项式时间变换到另一个问题. 例如，把顶点覆盖问题多项式时间变换到团问题. 设 $G=(V,E)$ 和 $K\leqslant|V|$ 构成顶点覆盖问题的一个实例，对应的团问题的实例由补图 $\overline{G}=(V,\overline{E})$ 和 $J=|V|-K$ 组成. 因此，只要证明这 3 个问题中的一个是 NP 完全的，就立即得到另外两个的

NP 完全性.

定理 13.13 顶点覆盖问题是 NP 完全的.

证：VC 的非确定型多项式时间算法如下：猜想一个顶点数不超过 K 的顶点子集 V'，然后检查每一条边是否至少有一个顶点在 V' 中. 所以，VC$\in$NP.

要证 3SAT$\leqslant_m$VC. 设变元 $x_1, x_2, \cdots, x_n$，三元合取范式 $F=\bigwedge\limits_{1\leqslant j\leqslant m} C_j$，其中 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$，每个 z_{jk} 为某个 x_i 或 $\neg x_i$. 这构成 3SAT 的一个实例 I，把它映射到 VC 的实例 $f(I)$. $f(I)$ 由图 $G=(V,E)$ 和 $K=n+2m$ 构成，其中

$$
\begin{aligned}
V &= V_1 \cup V_2, E = E_1 \cup E_2 \cup E_3,\\
V_1 &= \{x_i, \bar{x}_i \mid 1 \leqslant i \leqslant n\},\\
V_2 &= \{(z'_{jk}, j) \mid 1 \leqslant j \leqslant m, k = 1,2,3\},\\
E_1 &= \{\{x_i, \bar{x}_i\} \mid 1 \leqslant i \leqslant n\},\\
E_2 &= \{\{(z'_{j1}, j), (z'_{j2}, j)\}, \{(z'_{j2}, j), (z'_{j3}, j)\},\\
&\quad \{(z'_{j1}, j), (z'_{j3}, j)\}\ 1 \leqslant j \leqslant m\},\\
E_3 &= \{\{z'_{jk}, (z'_{jk}, j)\} \mid 1 \leqslant j \leqslant m, k = 1,2,3\},
\end{aligned}
$$

这里当 $z_{jk}=x_i$ 时，$z'_{jk}=x_i$；当 $z_{jk}=\neg x_i$ 时，$z'_{jk}=\bar{x}_i$.

图 G 可分成三部分，每一部分都有自己的功能. 第一部分，对每一个变元 x_i，G 有 2 个顶点 $x_i, \bar{x}_i$ 和 E_1 的一条边 $\{x_i, \bar{x}_i\}$. 为了覆盖 $\{x_i, \bar{x}_i\}$，顶点覆盖集 V' 必须包含 x_i 或 $\bar{x}_i$，分别对应于赋值 $t(x_i)=1$ 或 $t(x_i)=0$.

第二部分，对每一个简单析取式 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$，$G$ 有 3 个顶点 $(z'_{j1}, j), (z'_{j2}, j), (z'_{j3}, j)$ 和 E_2 中连接这 3 个顶点的 3 条边，它们恰好组成一个三角形. 为了覆盖这 3 条边，V' 至少包含这 3 个顶点中的 2 个. 因此，为了覆盖 E_1 和 E_2 中的边，V' 至少包含 V_1 中的 n 个顶点和 V_2 中的 $2m$ 个顶点，共 $n+2m$ 个顶点. 由于限制 $|V'|\leqslant K=n+2m$，故 V' 恰好包含每一对 x_i 和 $\bar{x}_i$ 中的一个，每一个三角形的 2 个顶点.

第三部分是连接 V_1 和 V_2 中顶点的边 E_3. 这些边与 F 有关. 对于每一个 C_j，把对应于 C_j 中的文字 z_{jk} 的顶点 z'_{jk} 与对应于 C_j 的三角形中的顶点 (z'_{jk}, j) 连接起来 $(k=1,2,3)$. V' 能否覆盖 E_3 取决于对应的赋值 t 是否使 F 为真. 图 13.2 给出一个例子.

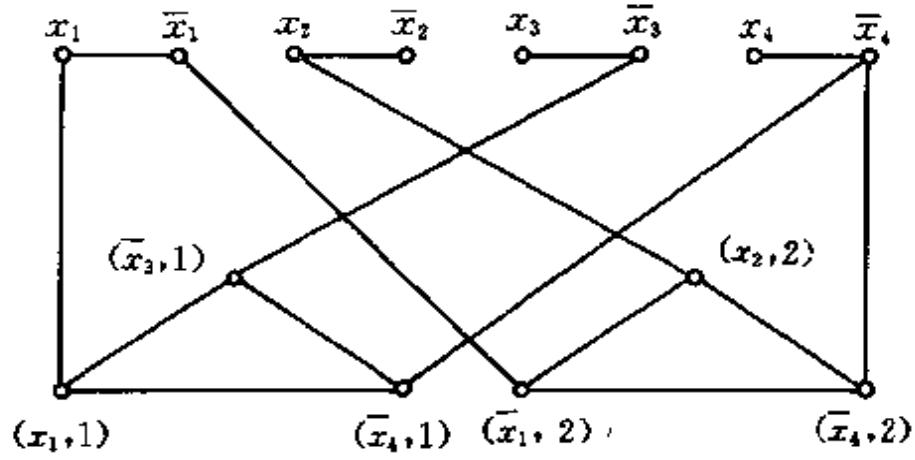


图 13.2　对应于 $F=(x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$ 的图 G，这里 $n=4, m=2, K=8$

设 t 是 F 的成真赋值. 对每一个 $i(1 \leqslant i \leqslant n)$，若 $t(x_i)=1$，则取顶点 x_i；若 $t(x_i)=0$，则取顶点 $\bar{x}_i$. 这 n 个顶点覆盖 E'. 对每一个 $j(1 \leqslant j \leqslant m)$，由于 $t(C_j)=1$，C_j 中至少有一个文字 z_{jk} 的值为 1. 于是，从对应的三角形的顶点 (z'_{jk}, j) 引出的边已被顶点 z'_{jk} 覆盖，取该三角形的另外两个顶点. 这就覆盖这个三角形的 3 条边和引出的另外两条边. 这样得到的 $n+2m$ 个顶点是 G 的一个顶点覆盖.

反之，设 $V' \subseteq V$ 是 G 的一个顶点覆盖且 $|V| \leqslant K=n+2m$. 根据前面的分析，每一对 x_i 和 $\bar{x}_i$ 中恰好有一个属于 V'，每一个三角形恰好有两个顶点属于 V'. 对每一个 $i(1 \leqslant i \leqslant n)$，若 $x_i \in V'$，则令 $t(x_i)=1$；若 $\bar{x}_i \in V'$，则令 $t(x_i)=0$. 对任一个 C_j，不妨设 (z'_{j2}, j)，$(z'_{j3}, j) \in V'$，边 $\{z'_{j1}, (z'_{j1}, j)\}$ 只能被 z'_{j1} 覆盖，即 $z'_{j1} \in V'$，从而 $t(z_{j1})=1, t(C_j)=1$. 因此，t 是 F 的成真赋值. 得证，F 是可满足的当且仅当 G 有大小不超过 $K=n+2m$ 的顶点覆盖.

G 有 $2n+3m$ 个顶点、$n+6m$ 条边. G 和 K 都能在多项式时间

内构造出来.从而,这是一个从 3SAT 到 VC 的多项式时间变换.

□

推论 13.14 团问题是 NP 完全的.

推论 13.15 独立集问题是 NP 完全的.

13.4.2 三维匹配问题

三维匹配问题(3DM):任给集合 $M\subseteq W\times U\times V$,其中 W,U,V 互不相交且 $|W|=|U|=|V|=q$,问 M 是否包含一个匹配,即是否有子集 $M'\subseteq M$ 使得 $|M'|=q$ 且对所有的 $(w_1,u_1,v_1),(w_2,u_2,v_2)\in M'$,$(w_1,u_1,v_1)\neq(w_2,u_2,v_2)$ 蕴涵 $w_1\neq w_2,u_1\neq u_2,v_1\neq v_2$?

这个问题是偶图的完美匹配问题的推广.偶图的完美匹配问题是问:任给的一个偶图是否有完美匹配?这是二维情况下的匹配问题,它是多项式时间可解的(见参考文献[5]).

定理 13.16 三维匹配问题是 NP 完全的.

证:猜想 M 的一个由 q 个元素组成的子集,检查它是否是任意两个不同的元素的坐标都不相同.检查工作可以在多项式时间内完成,从而这是 3DM 的一个非确定型多项式时间算法.所以,3DM$\in$NP.

要证 3SAT$\leqslant_m$3DM.任给 3SAT 的一个实例,它由变元 $x_1,x_2,\cdots,x_n$ 和三元合取范式 $F=\bigwedge\limits_{1\leqslant j\leqslant m}C_j$ 组成,对应的 3DM 的实例为:

$$W=\{x_{ij},\bar{x}_{ij}\mid 1\leqslant i\leqslant n,1\leqslant j\leqslant m\},$$

$$U=A\cup S_1\cup G_1,$$

$$A=\{a_{ij}\mid 1\leqslant i\leqslant n,1\leqslant j\leqslant m\},$$

$$S_1=\{s_{1j}\mid 1\leqslant j\leqslant m\},$$

$$G_1=\{g_{1k}\mid 1\leqslant k\leqslant m(n-1)\},$$

$$V=B\cup S_2\cup G_2,$$

$$B = \{b_{ij} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n\},$$
$$S_2 = \{s_{2j} \mid 1 \leqslant j \leqslant m\},$$
$$G_2 = \{g_{2k} \mid 1 \leqslant k \leqslant m(n-1)\},$$

这里 $q = |W| = |U| = |V| = 2nm$. M 的元素按其功能可分为三部分：

$$M = \left(\bigcup_{1 \leqslant i \leqslant n} T_i\right) \cup \left(\bigcup_{1 \leqslant j \leqslant m} H_j\right) \cup G.$$

第一部分是 $T_i (1 \leqslant i \leqslant n)$. 每一个 T_i 对应一个变量 x_i，它的构造与 F 中的简单析取式的个数 m 有关. T_i 包含“内部”元素 a_{ij} 和 $b_{ij} (1 \leqslant j \leqslant m)$ 及“外部”元素 x_{ij} 和 $\bar{x}_{ij} (1 \leqslant j \leqslant m)$. “内部”元素 a_{ij} 和 b_{ij} 不出现在 T_i 以外的 3 元组中，而“外部”元素 x_{ij} 和 $\bar{x}_{ij}$ 会出现在其他 3 元组中. T_i 又可分成两部分 $T_i = T_i^t \cup T_i^f$，其中

$$T_i^t = \{(\bar{x}_{ij}, a_{ij}, b_{ij}) \mid 1 \leqslant j \leqslant m\},$$
$$T_i^f = \{(x_{ij}, a_{ij+1}, b_{ij}) \mid 1 \leqslant j < m\} \cup \{(x_{im}, a_{i1}, b_{im})\}.$$

当 $m = 4$ 时，T_i 如图 13.3 所示.

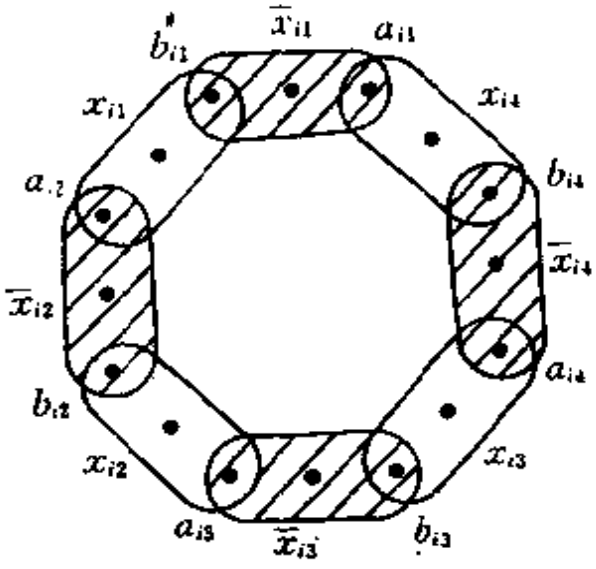


图 13.3　当 $m=4$ 时的 T_i. 带阴影的部分为 T_i^t，不带阴影的部分为 T_i^f

由于 a_{ij} 和 $b_{ij} (1 \leqslant j \leqslant m)$ 只出现在 T_i 中，任何匹配 M' 将恰好包含 T_i 中的 m 个 3 元组，并且可以看出 M' 一定是要么包含 T_i^t 的

所有 3 元组、要么包含 T_i^f 的所有 3 元组. 于是，M' 导出一个真值赋值 t，$t(x_i)=1$ 当且仅当 $M' \cap T_i = T_i^t$.

第二部分是 $H_j(1 \leqslant j \leqslant m)$. 每一个 H_j 对应一个简单析取式 C_j，它包含两个“内部”元素 s_{1j} 和 s_{2j} 以及 W 中的外部元素. H_j 包含 W 中的哪些元素由 C_j 中的文字决定. 若 $C_j = z_1 \vee z_2 \vee z_3$，则

$$H_j = \{(z_{lj}, s_{1j}, s_{2j}) \mid l = 1,2,3\},$$

这里当 $z_l = x_i$ 时 $z_{lj} = x_{ij}$；当 $z_l = \neg x_i$ 时 $z_{lj} = \bar{x}_{ij}$，$l=1,2,3$. 于是，任何匹配 M' 将恰好包含 H_j 中的一个 3 元组. 不妨设这个 3 元组是 (z_{1j}, s_{1j}, s_{2j})，其中 $z_{1j} = x_{ij}$（或 $\bar{x}_{ij}$），则 z_{1j} 不出现在 $T_i \cap M'$ 的 3 元组中. 能做到这一点当且仅当 M' 导出的真值赋值 t 使 C_j 为真.

第三部分 G 是一堆“填料”. 前两部分已经完成了对 3 元合取范式 F 可满足性的刻划，但是 M' 还不是一个完整的匹配. M' 包含每一个 T_i 中的 T_i^t 或 T_i^f、包含每一个 H_j 中的一个 3 元组，共有 $(n+1)m$ 个 3 元组，还缺 $q-(n+1)m=(n-1)m$ 个. 这 $(n+1)m$ 个 3 元组覆盖了 A, B, S_1, S_2 的全部元素以及 W 中的 $(n+1)m$ 个元素. W 中尚有 $(n-1)m$ 个元素未被覆盖. 为了提供将 M' 补充成一个完整的匹配的“原料”，添加 $(n-1)m$ 个 g_{1k} 和 $(n-1)m$ 个 g_{2k}，它们都是“内部”元素. 将它们与每一个 x_{ij} 和 $\bar{x}_{ij}$ 配成 3 元组，

$$G = \{(x_{ij}, g_{1k}, g_{2k}),$$
$$(\bar{x}_{ij}, g_{1k}, g_{2k}) \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m, 1 \leqslant k \leqslant (n-1)m\}.$$

只要 M' 在第一、二部分中取到所需要的 $(n+1)m$ 个 3 元组，就能从 G 中取到 $(n-1)m$ 个元素把 M'“填充”成一个匹配.

根据上面的分析，如果 M 包含一个匹配 M'，则 M' 导出的真值赋值必使 F 成真. 反之，设 t 是 F 的成真赋值，我们如下构造一个匹配 $M' \subseteq M$：对每一个 $i(1 \leqslant i \leqslant n)$，若 $t(x_i)=1$ 则 $M' \cap T_i = T_i^t$；若 $t(x_i)=0$ 则 $M' \cap T_i = T_i^f$. 对每一个 $j(1 \leqslant j \leqslant m)$，设 $C_j = z_1 \vee z_2 \vee z_3$. 由于 t 使 C_j 为真，C_j 中的 3 个文字至少有一个在 t 下的值为 1，不妨设 $t(z_1)=1$. 若 $z_1 = x_i$ 则 M' 包含 (x_{ij}, s_{1j}, s_{2j})；若 $z_1 = \neg x_i$ 则 M' 包含 $(\bar{x}_{ij}, s_{1j}, s_{2j})$. 最后在 G 中选取那些含有未被覆

盖的 x_i 或 $\bar{x}_i$ 的 3 元组，这样的 3 元组共有 $(n-1)m$ 个. 因此，M 包含一个匹配当且仅当 F 是可满足.

根据 F 构造 M 是直截了当的，显然能够在多项式时间内完成. 所以，这是从 3SAT 到 3DM 的多项式时间变换. □

13.4.3 Hamilton 回路问题

定理 13.17 Hamilton 回路问题是 NP 完全的.

证：已经知道 HC∈NP. 要把 VC 多项式时间变换到 HC. 设图 $G=(V,E)$ 和非负整数 $K\leqslant|V|$ 是 VC 的一个实例. 要构造一个图 $G'=(V',E')$ 使得 G' 有一条 Hamilton 回路当且仅当 G 有大小不超过 K 的顶点覆盖.

图 G' 有 K 个“选择器”顶点 $a_1,a_2,\cdots,a_K$，用它们从 G 的顶点集 V 中挑选出 K 个顶点. 对 G 的每一条边 $e\in E$，G' 包含一个“覆盖检验”子图，用它来保证这条边至少有一个端点在被挑选出来的 K 个顶点之中. 图 13.4 给出关于边 $e=\{u,v\}$ 的这种子图 $G'_e=(V'_e,E'_e)$，它有 12 个顶点和 14 条边. V'_e 中只有 $(u,e,1)$，$(v,e,1)$，$(u,e,6)$ 和 $(v,e,6)$ 可以与其他添加的边相关联. 不难验证，G' 中任意一条 Hamilton 回路只能按照图 13.5 所示的 3 种方式经过 E'_e 的边.

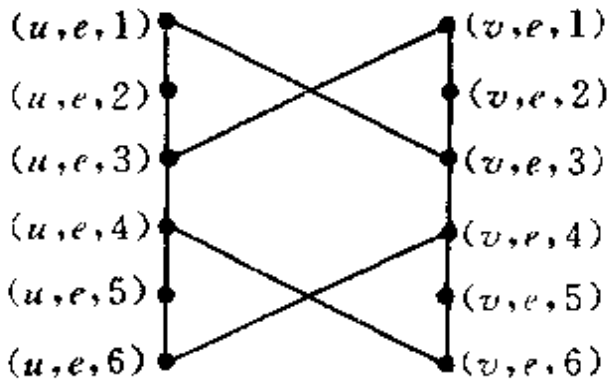


图 13.4 G' 中关于 $e=\{u,v\}$ 的覆盖检验子图 G'_e

添加一些边用来连接两个覆盖检验子图，或一个覆盖检验子

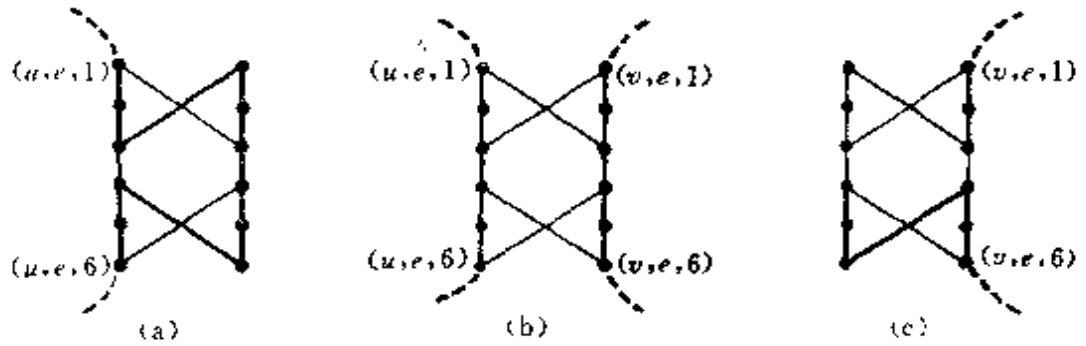


图 13.5　Hamilton 回路通过 G'_e 的 3 种可能的方式：
(a) u 属于但 v 不属于这个覆盖，(b) u 和 v 都属于这个覆盖，
(c) v 属于但 u 不属于这个覆盖.

图与一个选择器顶点. 对 G 的每一个顶点 $v\in V$，把与 v 关联的边(任意地)排列成 $e_v[1],e_v[2],\cdots,e_v[\deg(v)]$，这里 $\deg(v)$ 是 v 的度数，即与 v 关联的边数. 关于这些以 v 为端点的边的覆盖检验子图用下述边连接在一起：

$$E'_v=\{\{(v,e_v[i],6),(v,e_v[i+1],1)\}\mid 1\leqslant i<\deg(v)\}.$$

如图 13.6 所示，它构成 G' 中包含所有形如 (v,e,i) 的顶点且只包含这些顶点的唯一路径.

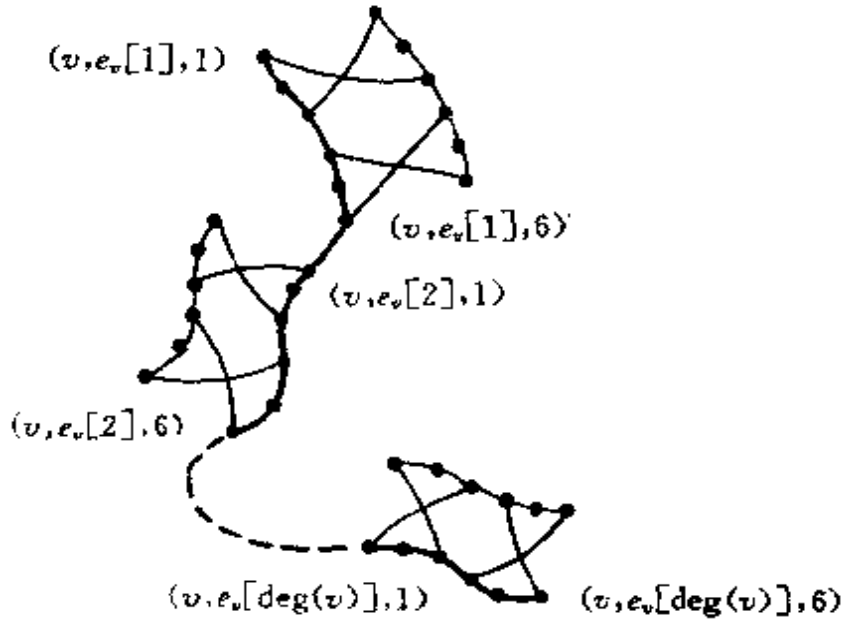


图 13.6　与 E 中以 v 为端点的边对应的覆盖检验子图的连接方式

最后再用边把每一条这样的路径的两个端点与每一个选择器顶点 $a_i(1\leqslant i\leqslant k)$连接起来. 这些边为：

$$E''=\{\{a_i,(v,e_v[1],1)\},\\ \{a_i,(v,e_v[\deg(v)],6)\}\\ |1\leqslant i\leqslant K,v\in V\}$$

整个图 $G'=(V',E')$，其中

$$V'=\{a_i|1\leqslant i\leqslant K\}\cup(\bigcup_{e\in E}V'_e),$$

$$E'=(\bigcup_{e\in E}E'_e)\cup(\bigcup_{v\in V}E'_v)\cup E''.$$

不难看到能够在多项式时间内从 G 和 K 构造出 G'.

下面证明 G'有 Hamilton 回路当且仅当 G 有大小不超过 K 的顶点覆盖. 假设 C 是 G'的一条 Hamilton 回路. 考虑 C 中任意一段从某个选择器顶点 a_i 到另一个选择器顶点 a_j、而中间不再有这种顶点的路径. 根据覆盖检验子图的连接方式，这段路径所经过的覆盖检验子图都恰好对应 G 中与某个顶点 v 关联的边，且以图 13.5 中的方式经过每一个覆盖检验子图. 于是，K 个选择器顶点把 C 分成 K 段 $C_1,C_2,\cdots,C_K$，分别对应 V 中的 K 个顶点，不妨设依次为 $v_1,v_2,\cdots,v_K$. 对于 E 中的任一条边 e，设 C_i 经过子图 G'_e. 而 C_i 经过的覆盖检验子图都对应于与 v_i 关联的边，故 e 与 v_i 关联. 因此，$\{v_1,v_2,\cdots,v_K\}$是 G 的一个顶点覆盖.

反之，设 $V^*\subseteq V$ 是 G 的顶点覆盖且 $|V^*|\leqslant K$. 不妨设 $|V^*|=K$，因为在 V^* 中增加一些顶点之后仍是 G 的顶点覆盖. 设 $V^*=\{v_1,v_2,\cdots,v_k\}$. 如下选取 G'中的边，它们恰好构成一条 Hamilton 回路. 对于 E 中的每一条边 $e=\{u,v\}$，根据 $\{u,v\}\cap V^*$ 等于 $\{u\}$，$\{u,v\}$，或 $\{v\}$，从关于 e 的覆盖检验子图 G'_e 中分别选择图 13.5 (a)，(b)，或 (c)中指明的边，然后选择 $E'_{v_i}(1\leqslant i\leqslant K)$中所有的边. 用这些边把从各个覆盖检验子图中选取出来的边连成 K 段路段，每一段对应 V^* 中的一个顶点. 最后，选择边

$$\{a_i,(v_i,e_{v_i}[1],1)\},\quad 1\leqslant i\leqslant K,$$

$$\{a_{i+1},(v_i,e_{v_i}[\deg(v_i)],6)\},\quad 1\leqslant i<K,$$

以及

$$\{a_1,(v_K,e_{v_K}[\deg(v_K)],6)\}.$$

把 K 个选择器顶点插入这 K 段路径之间，并用这些边把它们连成一条 Hamilton 回路. □

根据例 13.1 可得到

推论 13.18 货郎问题是 NP 完全的.

13.4.4 划分问题

划分问题：任给 n 个正整数 $a_1,a_2,\cdots,a_n$，问是否能把这 n 个数均匀的分成两部分，即是否存在 $I'\subseteq I=\{1,2,\cdots,n\}$ 使得

$$\sum_{i\in I'}a_i=\sum_{i\in I-I'}a_i?$$

定理 13.19 划分问题是 NP 完全的.

证：把给定的数任意地分成两组，可以在多项式时间内计算出每一组的和并且比较这两个和是否相等，因此该问题在 NP 中.

要把 3DM 多项式时间变换到划分问题. 设集合 W,U,V 以及 $M\subseteq W\times U\times V$ 是 3DM 的任意一个实例，其中

$$W=\{w_1,w_2,\cdots,w_q\},$$
$$U=\{u_1,u_2,\cdots,u_q\},$$
$$V=\{v_1,v_2,\cdots,v_q\},$$
$$M=\{m_1,m_2,\cdots,m_k\}.$$

对应的划分问题的实例由 $k+2$ 个数 $a_1,a_2,\cdots,a_k,b_1,b_2$ 组成. a_i 和 3 元组 m_i 相关联. a_i 用二进制表示给出，把这个二进制表示分成 $3q$ 段. 每一段有 $p=\lceil\log_2(k+1)\rceil$ 位，标着 $W\cup U\cup V$ 中的一个元素，如图 13.7 所示. 设 $m_i=(w_{f(i)},u_{g(i)},v_{h(i)})$，则 a_i 的 $w_{f(i)},u_{g(i)},v_{h(i)}$ 段的最右一位为 1，其余为 0，即

$$a_i=2^{p(3q-f(i))}+2^{p(2q-g(i))}+2^{p(q-h(i))}.$$

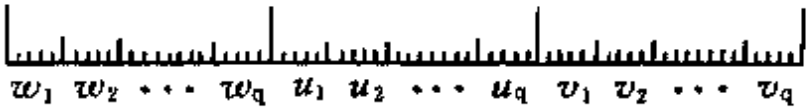


图 13.7　a_i 的二进制表示

注意到 $\sum_{i=1}^{k} a_i$ 在每一段至多是 $k \leqslant 2^p-1$，故在计算这个和的过程中不会出现从一段到另一段的进位. 于是若干 a_i 的和中某一段(如 w_1 段)的值恰好是对应的几个 m_i 中含有该段标记(如 w_1)的个数. 例如，若 $a_1+a_7+a_8$ 的 w_1 段的值为 2，则 m_1, m_7, m_8 中共含有 2 个 w_1.

令

$$B = \sum_{j=0}^{3q-1} 2^{pj}.$$

它的二进制表示恰好是每一段的最右一位为 1，其余均为 0. 于是，对任意的 $I' \subseteq I = \{1, 2, \cdots, k\}$，$\sum_{i \in I'} a_i = B$ 当且仅当 $M' = \{m_i \mid i \in I'\}$ 是一个匹配.

但是，当 $\sum_{i \in I'} a_i = B$ 时，$\sum_{i \in I-I'} a_i = A - B$ 不一定恰好等于 B，其中 $A = \sum_{i=1}^{k} a_i$. 因而引入 b_1 和 b_2 使得

$$B + b_1 = A - B + b_2$$

取 $b_1 = 2A - B$，$b_2 = A + B$. 不难看到，任给 $M \subseteq W \times U \times V$，可以在多项式时间内构造出这 $k+2$ 个数 $a_1, a_2, \cdots, a_k, b_1, b_2$.

设 $M' \subseteq M$ 是一个匹配，取

$$I' = \{i \mid m_i \in M'\},$$

则有

$$\sum_{i \in I'} a_i + b_1 = B + 2A - B = 2A,$$

$$\sum_{i \in I-I'} a_i + b_2 = A - B + A + B = 2A,$$

两者相等.

反之，设能把 $a_1,a_2,\cdots,a_k,b_1,b_2$ 均分成两部分. 注意到 $b_1+b_2=3A>\frac{1}{2}\left(\sum_{i=1}^{k}a_i+b_1+b_2\right)=2A$，$b_1$ 和 b_2 不可能在同一部分. 于是，存在 $I'\subseteq I$ 使得

$$\sum_{i\in I'}a_i+b_1=\sum_{i\in I-I'}a_i+b_2=2A,$$

得

$$\sum_{i\in I'}a_i=B.$$

从而 $M'=\{m_i\mid i\in I'\}\subseteq M$ 是一个匹配. 因此，这个变换是从 3DM 到划分问题的多项式时间变换. □

利用划分问题的 NP 完全性，容易证明下述两个问题是 NP 完全的.

背包问题：任给一个有穷物品集 A，每一件物品 $a\in A$ 的大小为 $s(a)\in Z^+$、价值为 $v(a)\in Z^+$，以及背包的容量 $B\in Z^+$ 和价值目标 $K\in Z^+$，问是否能在背包内装入价值不低于 K 的物品，即是否存在 $A'\subseteq A$ 使得

$$\sum_{a\in A'}s(a)\leqslant B\text{ 且 }\sum_{a\in A'}v(a)\geqslant K?$$

多处理机调度问题：任给有穷的工件集 A，每一个工件 $a\in A$ 的加工时间为 $t(a)\in Z^+$，以及机器的台数 $m\in Z^+$ 和时间限制 $D\in Z^+$，问是否能在限定的时间内加工完所有的工件，即是否能把 A 划分成 m 个不相交的集合 $A=A_1\cup A_2\cup\cdots\cup A_m$ 使得对每一个 $i(1\leqslant i\leqslant m)$ 都有 $\sum_{a\in A_i}t(a)\leqslant D$？这里假设 m 台机器是相同的，每一个工件可以在任一台机器上加工.

划分问题可以看作是这两个问题的特殊情况，从而容易把划分问题多项式时间变换到这两个问题. 限制每一件物品的大小 $s(a)$ 等于它的价值 $v(a)$，并且限制背包的容量 $B=\lfloor\frac{1}{2}\sum_{a\in A}s(a)\rfloor$，

价值目标 $K=\lceil\frac{1}{2}\sum_{a\in A}s(a)\rceil$，则背包问题成为是否能将$\{s(a)\mid a\in A\}$划分的问题. 限制机器的台数 $m=2$以及时间限制 $D=\lceil\frac{1}{2}\sum_{a\in A}t(a)\rceil$，则多处理机调度问题成为划分问题.

推论13.20 背包问题是 NP 完全的.

推论13.21 多处理机调度问题是 NP 完全的.

实际上，2台多处理机调度问题(即 $m=2$)也是 NP 完全的.

13.4.5 整数线性规划

整数线性规划是用途非常广泛的最优化模型，它的一般形式是

$$\begin{aligned}&\min \boldsymbol{c}^T\boldsymbol{x}\\&\text{s.t. } \boldsymbol{A}\boldsymbol{x}\geqslant\boldsymbol{b}\\&\boldsymbol{x}\geqslant 0,\text{整数}\end{aligned}$$

这里 $\boldsymbol{A}$ 是 $m\times n$ 的整数矩阵，$\boldsymbol{c}$ 是 n 维整数列向量，$\boldsymbol{b}$ 是 m 维整数列向量，$\boldsymbol{x}$ 是待定的 n 维向量，它的分量都是非负整数. $\boldsymbol{c}^T$ 是 $\boldsymbol{c}$ 的转置. s.t. 是 subject to 的缩写. $\boldsymbol{c}^T\boldsymbol{x}$ 是目标函数，$\boldsymbol{A}\boldsymbol{x}\geqslant\boldsymbol{b}$ 是 n 个约束条件. 它们都是线性的.

按照我们的标准做法，对应的判定问题在实例中添加一个整数 d，问是否存在非负整数向量 $\boldsymbol{x}$ 使得 $\boldsymbol{c}^T\boldsymbol{x}\leqslant d$ 且 $\boldsymbol{A}\boldsymbol{x}\geqslant\boldsymbol{b}$?

$\boldsymbol{c}^T\boldsymbol{x}\leqslant d$ 等价于 $-\boldsymbol{c}^T\boldsymbol{x}\geqslant -d$，可以把它与原有的约束条件 $\boldsymbol{A}\boldsymbol{x}\geqslant\boldsymbol{b}$ 合并在一起. 于是，对应的判定问题可写成下述形式.

整数线性规划问题(ILP)：任给 $m\times n$ 整数矩阵 $\boldsymbol{A}$ 和 m 维整数向量 $\boldsymbol{b}$，问下述问题

$$\begin{aligned}&\boldsymbol{A}\boldsymbol{x}\geqslant\boldsymbol{b}\\&\boldsymbol{x}\geqslant 0,\text{整数}\end{aligned}\qquad(1)$$

是否有解?

引理13.22 整数线性规划问题是 NP 难的.

证：把3SAT 多项式时间变换到 ILP. 设变元 $x_1, x_2, \cdots, x_n$ 和3元合取范式 $F=\bigwedge_{1\leqslant j\leqslant m} C_j$，其中 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$，每一个 z_{jk} 等于某个 x_i 或 $\neg x_i$. 如下构造对应的 ILP 实例 I：

$$
\begin{aligned}
&x_i+\bar{x}_i\geqslant 1,\\
&-x_i-\bar{x}_i\geqslant -1, \quad 1\leqslant i\leqslant n\\
&z'_{j1}+z'_{j2}+z'_{j3}\geqslant 1, \quad 1\leqslant j\leqslant m\\
&x_i\geqslant 0, \bar{x}_i\geqslant 0, \text{整数}, 1\leqslant i\leqslant n,
\end{aligned}
$$

这里当 $z_{jk}=x_i$ 时，$z'_{jk}=x_i$；当 $z_{jk}=\neg x_i$ 时，$z'_{jk}=\bar{x}_i$，$(1\leqslant j\leqslant m, k=1,2,3)$. 构造可在多项式时间内完成.

对每一个 $i(1\leqslant i\leqslant n)$，存在非负整数 x_i 和 $\bar{x}_i$ 使得 $x_i+\bar{x}_i\geqslant 1$ 且 $-x_i-\bar{x}_i\geqslant -1$ 当且仅当 x_i 和 $\bar{x}_i$ 中恰好一个为1，而另一个为0. 这恰好对应于对变元 x_i 的真值赋值.

对每一个 $j(1\leqslant j\leqslant m)$，$z'_{j1}+z'_{j2}+z'_{j3}\geqslant 1$ 当且仅当 $z'_{j1}, z'_{j2}, z'_{j3}$ 中至少有一个的值为1，从而这又当且仅当 C_j 取真值.

根据上述分析，容易看出：F 是可满足的当且仅当实例 I 有解. □

和前面的问题不同，ILP 属于 NP 的证明并不容易. 主要困难是不等式组(1)的解中可能出现很大的数，使得验证解不能在多项式时间内完成. 为了克服这个困难，下面先给出几个预备知识.

设 $\boldsymbol{A}=(a_{ij})$ 是 $m\times n$ 的整数矩阵，$\boldsymbol{b}=(b_1, b_2, \cdots, b_m)^T$ 是 m 维整数向量. 记 $\boldsymbol{A}$ 的第 i 行为 $\boldsymbol{a}_i(1\leqslant i\leqslant m)$，$a$ 为 $\boldsymbol{A}$ 的元素的最大绝对值，$q=\max(m,n)$.

引理13.23 设 $\boldsymbol{B}$ 是 $\boldsymbol{A}$ 的一个 $r\times r$ 的子方阵，则 $|\det(\boldsymbol{B})|\leqslant (qa)^q$.

证：$\det(\boldsymbol{B})$ 有 $r!$ 项，每一项是 $\boldsymbol{A}$ 中 r 个元素的乘积，其绝对值不超过 a^r. 故

$$|\det(\boldsymbol{B})|\leqslant r!a^r\leqslant (ra)^r\leqslant (qa)^q. \qquad \square$$

引理13.24 设 $\boldsymbol{A}$ 的秩为 r. 如果 $1\leqslant r<n$，则存在非零整数向

量 $z=(z_1,z_2,\cdots,z_n)^T$ 使得 $Az=0$ 且对每一个 $j(1\leqslant j\leqslant n)$，$|z_j|\leqslant (aq)^q$.

证：不妨设 A 的左上角的 $r\times r$ 子方阵 B 的秩等于 r. 把 A 的前 r 行记作 C，后 $m-r$ 行记作 D. 由于 A 的秩等于 r，C 的 r 行是 A 的 m 行的一个极大线性无关组，故存在 $(m-r)\times n$ 的矩阵 P 使得 $D=PC$. 若 $Cz=0$，则 $Dz=PCz=0$. 因此，只要 $Cz=0$，就有 $Az=0$.

考虑方程组

$$Cz=0,$$

取 $z_n=-\det(B)$，$z_{r+1}=\cdots=z_{n-1}=0$，注意到 C 的前 r 列是 B，方程组可写成

$$By=\det(B)c_n, \tag{2}$$

其中 $y=(z_1,\cdots,z_r)^T$，c_n 是 C 的第 n 列. 由 Cram 法则，(2) 有唯一解

$$z_i=\det(B_i),\quad 1\leqslant i\leqslant r,$$

其中 B_i 是用 $\det(B)c_n$ 替换 B 的第 i 列后得到的方阵. 根据引理 13.23，这样得到的 z 满足引理的要求. □

引理13.25 设 A 的秩等于 n. 如果

$$Ax\geqslant b \tag{3}$$

有整数解，则存在(3)的整数解 x 和 A 的 k 行(不妨设为 $a_1,a_2,\cdots,a_k$)使得

$$b_i\leqslant a_ix<b_i+(aq)^{q+1},\quad 1\leqslant i\leqslant k$$

且这 k 行的秩等于 n.

证：设 x_0 是(3)的整数解. 不妨设

$$b_i\leqslant a_ix_0<b_i+(qa)^q,\quad 1\leqslant i\leqslant k_0,$$

$$a_ix_0\geqslant b_i+(qa)^q,\quad k_0<i\leqslant m.$$

记 C 为 k_0 行 $a_1,a_2,\cdots,a_{k_0}$ 构成的矩阵. 如果 C 的秩等于 n，则引理已经得证. 否则由引理13.24，存在整数向量 z 使得 $Cz=0$ 且 z 的每一个分量的绝对值都不超过 $(qa)^q$. 当 $k_0=0$ 时，取 $z=((qa)^q$.

$\cdots,(qa)^q$. 于是,对任意的整数 d,

$$a_i(x_0+dz)=a_ix_0,\quad 1\leqslant i\leqslant k_0$$

$$a_i(x_0+dz)=a_ix_0+da_iz,\quad k_0<i\leqslant m.$$

由于$|a_iz|\leqslant na(qa)^q\leqslant(qa)^{q+1}$以及当$i>k_0$时$a_ix_0>b_i+(qa)^{q+1}$,可以适当地选取 d 使得

$$a_i(x_0+dz)\geqslant b_i,\quad i>k_0$$

且至少有一个$t>k_0$使得

$$b_i\leqslant a_i(x_0+dz)<b_i+(qa)^{q+1}.$$

不妨设当$k_0<t\leqslant k_1$时,

$$b_i\leqslant a_i(x_0+dz)<b_i+(qa)^{q+1}.$$

于是,令$x_1=x_0+dz$,则有

$$b_i\leqslant a_ix_1<b_i+(qa)^{q+1},\quad 1\leqslant i\leqslant k_1,$$

$$a_ix_1\geqslant b_i+(qa)^{q+1},\quad k_1<i\leqslant m,$$

这里$k_1>k_0$.

以 x_1代替 $x_0,a_{i_1},a_{i_2},\cdots,a_{i_{k_0}}$代替 $a_1,a_2,\cdots,a_{k_0}$,重复上述过程. 经过有限次重复一定可以得到引理所要求的 x 和 A 的 k 行. □

定理13.26 整数线性规划问题是 NP 完全的.

证:引理13.22已经证明该问题是 NP 难的,现在只需证明 ILP 在 NP 中.

令

$$\bar{A}=\begin{pmatrix}A\\E\end{pmatrix},\quad \bar{b}=\begin{pmatrix}b\\0\end{pmatrix}$$

其中 E 是 n 阶单位矩阵,0是 n 维零向量,$\bar{A}$ 是$(m+n)\times n$ 的整数矩阵,其秩等于 n. $\bar{A}$ 的 $m+n$ 行依次记作 $a_1,a_2,\cdots,a_{m+n}$,$\bar{b}$ 的 $m+n$ 个元素依次记作 $b_1,b_2,\cdots,b_{m+n}$. 显然,(1)可写成

$$\bar{A}x\geqslant\bar{b}\qquad(4)$$

根据引理13.25,如果(4)有整数解,则存在整数解 x 和 $\bar{A}$ 的 k 行,不妨设为 $a_1,a_2,\cdots,a_k$,使得

$$b_i\leqslant a_ix<b_i+((m+n)a)^{m+n+1},\quad 1\leqslant i\leqslant k$$

且 $\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_k$ 的秩等于 n.

令 $c_i=\boldsymbol{a}_i\boldsymbol{x}$，有 $b_i\leqslant c_i<b_i+((m+n)a)^{m+n+1}$. 由于 $\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_k$ 的秩等于 n，给定 $c_1,c_2,\cdots,c_k$，线性方程组

$$\boldsymbol{a}_i\boldsymbol{x}=c_i,\quad 1\leqslant i\leqslant k$$

有唯一解.

于是，下述过程是关于 ILP 的非确定型算法：

(1) 猜想 $\overline{A}$ 的 k 行，不妨设是 $\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_k$. 猜想 k 个整数 c_i，满足条件 $b_i\leqslant c_i<b_i+((m+n)a)^{m+n+1}$，$1\leqslant i\leqslant k$.

(2) 检查 $\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_k$ 的秩是否等于 n. 若秩小于 n，则回答“否”，结束.

(3) 解线性方程组

$$\boldsymbol{a}_i\boldsymbol{x}=c_i,\quad 1\leqslant i\leqslant k,$$

设解为 $\boldsymbol{x}$.

(4) 检查 $\boldsymbol{x}$ 是否是整数解. 若不是则回答“否”，结束.

(5) 检查 $\boldsymbol{x}$ 是否满足其余的 $m+n-k$ 个不等式

$$\boldsymbol{a}_i\boldsymbol{x}\geqslant b_i,\quad k+1\leqslant i\leqslant m+n,$$

若都满足则回答“是”；否则回答“否”，结束.

ILP 的实例 I 的规模取作 $|I|=mn+\log_2(a+1)$，其中 $a=\max\{a,b_1,b_2,\cdots,b_m\}$. 注意到猜想的 c_i 的二进制表示的长度不超过 $(m+n+1)\log_2(2(m+n)a+1)$，上述过程可以在 $|I|$ 的多项式时间内完成. □

13.5 Co-NP

设字母表 A，$L\subseteq A^*$. 把 A^*-L 记作 $\overline{L}$，称作 L 的补. 在补运算下，P 类是封闭的. 即，若 $L\in P$，则 $\overline{L}\in$P. 但是，NP 类在补运算下的封闭性至今尚未解决.

定义13.3 Co-NP$=\{L\mid\overline{L}\in$NP$\}$.

[**例13.2**] Hamilton 回路问题的补问题$\overline{\text{HC}}$是问：任给一个

图 G,G 是否没有 Hamilton 回路?

为了证明 G 没有 Hamilton 回路,需要是给出 G 中顶点所有可能的排列,并且验证这些排列都不是回路. 这和验证 G 有 Hamilton 回路不同. 前者必须检查所有的排列,而后者只需检查一个排列(当 G 有 Hamilton 回路时,总可以设想猜想对了). 检查所有的排列不可能在多项式时间内完成. 事实上,至今不知道$\overline{HC}$是否在 NP 中.

人们猜想:NP≠Co-NP. 由于 NP≠Co-NP 蕴涵 P≠NP(P=NP 蕴涵 NP=Co-NP),所以这个猜想是比 P≠NP 更强的猜想.

引理13.27 如果 $L\leqslant_m L'$,则 $\overline{L}\leqslant_m\overline{L'}$.

证:从 L 到 L'的多项式时间变换同时也是 $\overline{L}$ 到 $\overline{L'}$的多项式时间变换. □

定理13.28 如果存在 NP 完全的语言 L 在 Co-NP 中,则 NP=Co-NP.

证:由引理13.27,$\overline{L}$ 是 Co-NP 完全的. 又已知 $L\in$Co-NP,故 $\overline{L}\in$NP. 根据定理13.3(2),有 Co-NP⊆NP.

又设 $L'\in$NP,则 $\overline{L'}\in$Co-NP⊆NP,$L'\in$Co-NP. 得证 NP⊆Co-NP. □

假设 NP≠Co-NP、NP、Co-NP 以及 P 的关系如图13.8所示,其中 NPC 是所有 NP 完全的语言组成的集合,Co-NPC 是所有 Co-NP 完全的语言组成的集合. P⊆NP∩Co-NP,但这个包含关系是否是真包含也不清楚.

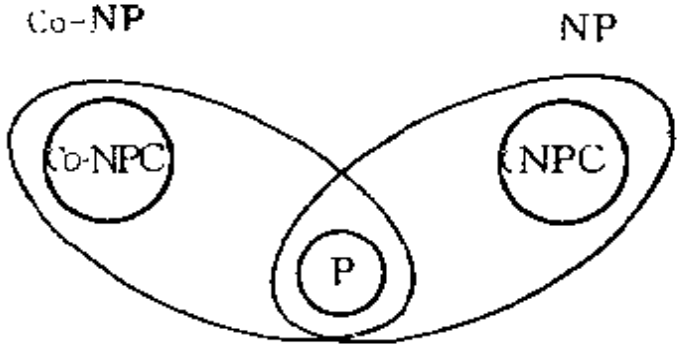


图13.8 假设 NP≠Co-NP

习　题

1. 证明定理13.4.

2. 证明定理13.6.

3. 设字母表$\{0,1\}$上的语言A,B,C. 记

$$A+B=\{x0|x\in A\}\cup\{y1|y\in B\}.$$

证明：(1) $A\leqslant_m A+B$，$B\leqslant_m A+B$.

(2) 如果$A\leqslant_m C$且$B\leqslant_m C$，则$A+B\leqslant_m C$.

证明下述问题(第4题至第11题)是NP完全的.

4. 装箱问题

任给n件物品和k只箱子，物品j的体积为$s_j\in Z^+(1\leqslant j\leqslant n)$，每只箱子的容积为$B\in Z^+$. 问：能否把这$n$件物品全部装入箱子？

5. 最小覆盖问题

任给有穷集S，$C\subseteq P(S)$以及正整数K，问：C是否包含S的大小不超过K的覆盖，即是否有$C'\subseteq C$使得$|C'|\leqslant K$且$\bigcup_{A\in C'}A=S$？

6. 子图同构问题

任给两个图G和H，问：G是否有同构于H的子图？

7. 最长通路问题

任给一个图$G=(V,E)$以及正整数$K\leqslant|V|$，问：G中是否有一条边数不少于K的简单通路？

8. 区间排序问题

任给有穷的任务集T，每一件任务$t\in T$有一个开放时间$r(t)$、一个截止时间$d(t)$和执行时间$l(t)$，这里$r(t)$、$d(t)$、$l(t)$都是非负整数. 问：是否存在关于T的可行调度表，即是否存在函数$\sigma:T\to N$使得对每一个$t\in T$满足下述条件

(1) $\sigma(t)\geqslant r(t)$，

(2) $\sigma(t)-l(t)\leqslant d(t)$，

(3) 如果$t'\in T-\{t\}$，则要么$\sigma(t')+l(t')\leqslant\sigma(t)$，要么$\sigma(t)+l(t)\leqslant\sigma(t')$？

提示：把划分问题多项式时间变换到该问题.

9. 最少拖延排序问题

任给任务集 T，每一件任务 $t\in T$ 的执行时间为1、截止时间为 $d(t)\in Z^+$，此外还有 T 上的偏序关系 $\leqslant$ 以及非负整数 $K\leqslant|T|$，问：是否存在关于 T 的调度表 $\sigma:T\to\{0,1,\cdots,|T|-1\}$ 满足下述条件：

(1) 如果 $t\neq t'$，则 $\sigma(t)\neq\sigma(t')$，

(2) 如果 $t\leqslant t'$，则 $\sigma(t)\leqslant\sigma(t)$，

(3) $|\{t\in T|\sigma(t)+1>d(t)\}|\leqslant K$？

提示：把团问题多项式时间变换到该问题.

10. 可着三色问题

任给一个图 $G=(V,E)$，问：G 是否是可着三色的，即是否存在函数 $f:V\to\{1,2,3\}$ 使得只要 $\{u,v\}\in E$ 就有 $f(u)\neq f(v)$？

提示：把3SAT 多项式时间变换到该问题.

11. 图的 Grundy 编号

任给一个有向图 $D=(V,A)$，问：是否有一个编号 $l:V\to N$，使得对于每一个 $v\in V$，$l(v)$ 是不在集合 $\{l(u)|u\in V,(v,u)\in A\}$ 内的最小数？

提示：把3SAT 多项式时间变换到该问题.

第十四章　组合优化问题的近似计算

14.1　近似算法及其近似比

从上一章已经看到，很多组合优化问题对应的判定问题是NP完全的. 像13.1节中叙述的货郎问题的最优化形式与判定形式的关系那样，通常可以借助组合优化问题的算法构造出对应的判定问题的算法，并且若组合优化问题的算法是多项式时间的，则对应的判定问题的算法也是多项式时间的. 从而，对应于NP完全的判定问题的组合优化问题不可能有多项式时间的最优化算法，除非 $P=NP$. 称这样的问题是NP难的. 货郎问题、背包问题、整数线性规划等都是NP难的.

但是，不能因为这些问题是NP难的，就不去解这些问题. 它们都是实际中经常碰到的重要问题. 既然不能在多项式时间内找到最优解，只好求其次，找一个满意的可行解(近似解). 更何况在实际中也不一定非要“最优的”不可，而往往是“足够好的”就可以了. 因此，近似算法是解决NP难的组合优化问题的一条重要途径.

组合优化问题 Π 由三部分组成：

(1) 实例集 D；

(2) 对于每一个实例 $I\in D$，有一个有穷的非空集合 $S(I)$，$S(I)$的元素称作 I 的**可行解**；

(3) 对于每一个可行解 $\sigma\in S(I)$，有一个正整数 $c(\sigma)$，称作 σ 的**值**.

当 Π 是**最小化问题**(**最大化问题**)时，如果 $\sigma^*\in S(I)$使得对于所有的 $\sigma\in S(I)$，

$$c(\sigma^*)\leqslant c(\sigma) \qquad (c(\sigma^*)\geqslant c(\sigma)),$$

则称 σ^* 是 I 的**最优解**. $c(\sigma^*)$称作 I 的**最优值**，记作 $\mathrm{OPT}(I)$.

例如，货郎问题是一个最小化问题，它的实例包括 m 个城市 $c_1, c_2, \cdots, c_m$ 以及每一对 c_i 和 c_j 的距离 $d_{ij}\in z^+$，这里 $d_{ij}=d_{ji}$. 经过这 n 个城市且每一个城市恰好经过一次的环游是一个可行解，解的值等于这个环游的长度. 长度最短的环游是最优解.

如果对于 Π 的每一个实例 I，算法 A 输出 I 的一个可行解 σ，记 $A(I)=c(\sigma)$，则称 A 是 Π 的**近似算法**. 如果对于每一个实例 I，A 总给出 I 的最优解，则称 A 是 Π 的**最优化算法**.

设 A 是问题 Π 的近似算法. 对于每一个实例 I，当 Π 是最小化问题时，记

$$r_A(I)=\frac{A(I)}{\mathrm{OPT}(I)};$$

当 Π 是最大化问题时，记

$$r_A(I)=\frac{\mathrm{OPT}(I)}{A(I)}.$$

算法 A 的近似比定义为：

$$r_A=\inf\{r\mid \text{对 } \Pi \text{ 的所有实例 } I, r_A(I)\leqslant r\}.$$

显然，$r_A\geqslant 1$ 并且 $r_A=1$ 当且仅当 A 是 Π 的最优化算法.

如果 r_A 是一个常数，则称 A 是**具有常数比的**近似算法.

下面考虑满足三角不等式的货郎问题的多项式时间的近似算法. 所谓满足三角不等式是指对于任意的 3 个不同的城市 c_i, c_j, c_k，它们之间的距离满足不等式：

$$d_{ik}\leqslant d_{ij}+d_{jk}.$$

下述算法称作**最邻近法**，记作 NN. 从某一个城市开始，在每一步取离当前所在城市最近的尚未到过的城市作为下一个城市，直至走遍所有的城市，最后回到开始出发的城市. 这是一种贪心算法. 初看起来这个算法似乎是非常合理的，很好的. 但是，实际上它不仅不能保证得到最优解，而且算法的近似性能也很不好. 下面的

定理清楚的表明了这一点.

定理 14.1 对于满足三角不等式的货郎问题的所有 m 个城市的实例 I，总有

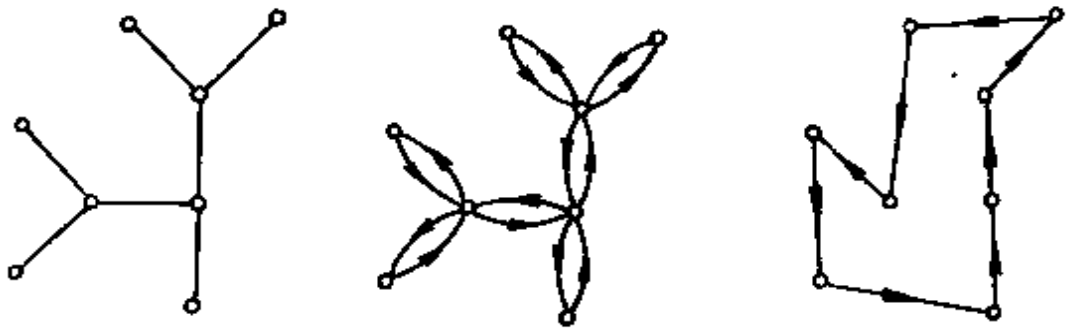
$$\mathrm{NN}(I)\leqslant\frac{1}{2}(\lceil\log_2 m\rceil+1)\mathrm{OPT}(I).$$

而且，对于每一个充分大的 m，存在 m 个城市的实例 I 使得

$$\mathrm{NN}(I)>\frac{1}{3}\left(\log_2(m+1)+\frac{4}{3}\right)\mathrm{OPT}(I).$$

定理表明 $r_{\mathrm{NN}}=\infty$. 这里略去定理的证明.

第二个近似算法叫做**最小生成树法**，记作 MST (D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, 1977.). 把货郎问题的实例看作一个带权的完全图，要找一条最短的 Hamilton 回路. 算法如下：首先求图的最小生成树，然后沿着最小生成树的每一条边走二遍，得到图的一条 Euler 回路. 最后，顺着这条 Euler 回路，跳过已走过的顶点，抄近路得到一条 Hamilton 回路. 如图 14.1 所示.



最小生成树　　走二遍得到 Euler 回路　　抄近路得到 Hamilton 回路

图 14.1 最小生成树法

由于求最小生成树和求 Euler 回路都可以在多项式时间内完成，故这个近似算法是多项式时间的. 有关图的知识可查阅参考文献[7].

定理 14.2 对于满足三角不等式的货郎问题的所有实例 I，

$$\mathrm{MST}(I)<2\mathrm{OPT}(I).$$

并且，对于任意的 $\varepsilon>0$，存在无穷多个实例 I 使得

$$\mathrm{MST}(I)>(2-\varepsilon)\mathrm{OPT}(I).$$

证：先证第一个结论. 因为从 Hamilton 回路中删去一条边就得到一棵生成树，故最小生成树的各条边的权之和必小于最短 Hamilton 回路的长度 $\mathrm{OPT}(I)$. 于是，走二遍得到的 Euler 回路的长度小于 $2\mathrm{OPT}(I)$. 由于图的各边的权满足三角不等式，抄近路不会增加长度，故算法得到的 Hamilton 回路的长度为：

$$\mathrm{MST}(I)<2\mathrm{OPT}(I).$$

图 14.2 给出第二个结论中所需要的实例 I. 图中没有标明距离的任何两点之间的距离等于这两点之间的最短路的长度. 对于这个 I，$\mathrm{OPT}(I)=2n$，$\mathrm{MST}(I)=4n-2=\left(2-\frac{1}{n}\right)\mathrm{OPT}(I)$. 当 $n>\frac{1}{\varepsilon}$ 时，

$$\mathrm{MST}(I)>(2-\varepsilon)\mathrm{OPT}(I). \qquad \square$$

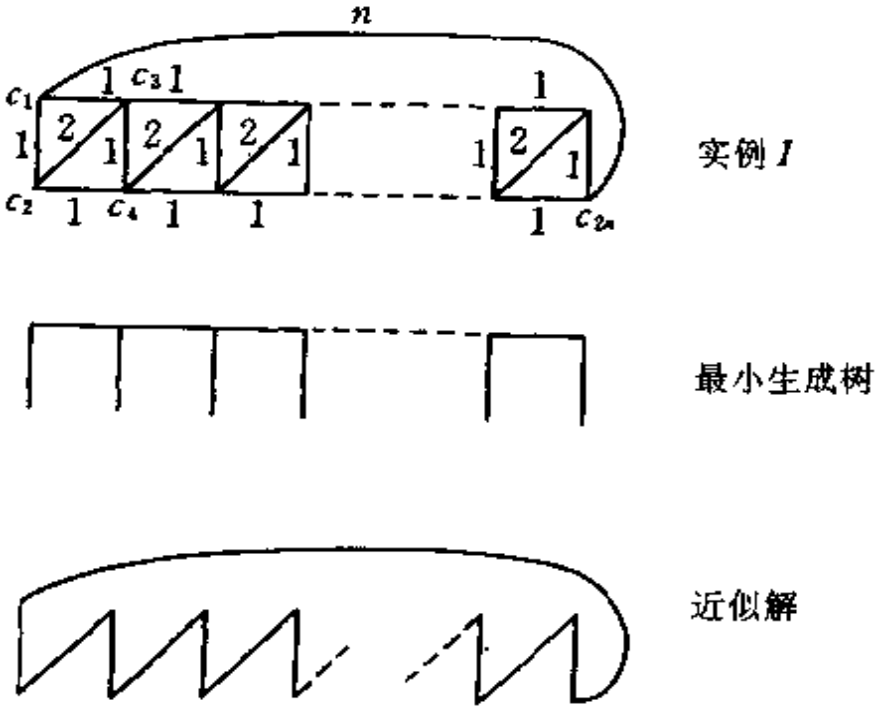


图 14.2　定理 14.2 证明中使用的实例 I

根据该定理，$r_{MST}=2$.

第三个近似算法叫做**最小权匹配法**，记作 MM（N. Christofieds，1976）. 最小生成树法通过对树的每一条边走两遍的办法得到一条 Euler 回路. 其实为了把最小生成树改造成 Euler 图没有必要复制每一条边，而只需要将最小生成树中的所有奇度顶点（一定是偶数个）两两配对，将每一对顶点之间的边加进来即可. 为了使加入的边的权之和最小，求这些奇度顶点导出的子图的最小权匹配. 把求得的匹配边加入最小生成树. 求任意图的最小权匹配有多项式时间算法（见参考文献[5]），因此这个算法仍是多项式时间的.

定理 14.3 对于满足三角不等式的货郎问题的所有实例 I，

$$MM(I)<\frac{3}{2}OPT(I).$$

并且，对任意的 $\epsilon>0$，存在无穷多个实例 I 使得

$$MM(I)>\left(\frac{3}{2}-\epsilon\right)OPT(I).$$

证：由于满足三角不等式，最小生成树中的奇度顶点导出的子图的最短 Hamilton 回路的长度不超过 $OPT(I)$. 沿着这条回路隔一条取一条边，就得到这个导出子图的一个匹配. 总可以使这个匹配的权不超过回路长度的一半. 因此，导出子图的最小权匹配的权不超过 $\frac{1}{2}OPT(I)$. 从而，求得的 Euler 回路的长度小于 $\frac{3}{2}OPT(I)$. 得证，

$$MM(I)<\frac{3}{2}OPT(I).$$

可类似于定理 14.2 的证明，给出证明第二个结论的实例.

□

这个定理表明，$r_{MM}=\frac{3}{2}$.

当不限制货郎问题的实例满足三角不等式时，这两个定理的

证明都失效. 容易给出使得这两个近似算法得到的近似解的值与最优值的比任意大的实例. 实际上,关于一般的货郎问题有下述定理.

定理 14.4 假设 P≠NP,则货郎问题(不要求满足三角不等式)不存在具有常数比的多项式时间近似算法.

证: 假设不然,设 A 是货郎问题的多项式时间近似算法,其近似比 $r_A \leqslant K$,K 为正整数.

任给一个图 $G=(V,E)$,如下构造货郎问题的实例 I_G:城市集 V,每一对城市 $u,v \in V$ 的距离为:

$$d(u,v)=\begin{cases}1 & \text{若}(u,v)\in E\\ K|V| & \text{否则}\end{cases}$$

若 G 有 Hamilton 回路,则 $\mathrm{OPT}(I_G)=|V|$, $A(I_G)\leqslant K.\mathrm{OPT}(I_G)=K|V|$. 若 G 没有 Hamilton 回路,则经过 V 中所有城市的环游中至少有两个相邻的城市在 G 中是不相邻的. 从而,$\mathrm{OPT}(I_G)>K|V|$, $A(I_G)\geqslant \mathrm{OPT}(I_G)>K|V|$. 所以,$G$ 有Hamilton回路当且仅当 $A(I)\leqslant K|V|$.

于是,下述算法可以判断任给的图 G 是否有 Hamilton 回路:首先构造 I_G,然后将算法 A 用于 I_G. 若 $A(I_G)\leqslant K|V|$,则回答"是";若 $A(I_G)>K|V|$,则回答"否".

由于 A 是多项式时间的,并且注意到 K 是固定的常数,不难看出这个算法是多项式时间的. 而 Hamilton 回路问题是 NP 完全的,与 P≠NP 的假设矛盾. □

14.2 装箱问题

装箱问题:任给 n 件物品,物品 i 的体积为 $s_i \in Z^+$, $1\leqslant i\leqslant n$. 要把它们全部装入箱子中,每只箱子的容积为 $B\in Z^+$,这里 $s_i\leqslant B$, $1\leqslant i\leqslant n$. 求使用箱子数最少的装法.

与这个问题对应的判定问题是 NP 完全的,甚至当固定箱子

数 $k=2$ 时也仍然是 NP 完全的.(见第十三章习题第 4 题)

人们都有这样的日常生活经验,先装大件物品,然后把小件物品填塞到空隙中.这种装法可以比较有效地利用空间.在这种经验的启发下,设计出下述装箱问题的近似算法.

递降首次适合法(FFD):首先将物品按体积从大到小排列,不妨设 $s_1\geqslant s_2\geqslant\cdots\geqslant s_n$.然后对 $i=1,2,\cdots n$ 依次检查每一只箱子,只要发现箱子的剩余空间的容量不小于 s_i,就把物品 i 装入这只箱子.

定理 14.5 对于装箱问题的所有实例 I,

$$\mathrm{FFD}(I)\leqslant\frac{4}{3}\mathrm{OPT}(I)+\frac{1}{3}.$$

证:记 $k^*=\mathrm{OPT}(I)$.不妨设 $s_1\geqslant s_2\geqslant\cdots\geqslant s_n$.

(1) 所有装入下标大于 k^* 的箱子中的物品的体积$\leqslant\frac{1}{3}B$.

设物品 m 是第一件装入第 k^*+1 只箱子的物品.只需证 $s_m\leqslant\frac{1}{3}B$.

假设不然,设 $s_m>\frac{1}{3}B$,则 $s_i>\frac{1}{3}B(1\leqslant i\leqslant m)$.不考虑 m 后面的物品,一定存在 $l\geqslant 0$ 使得前 l 只箱子中每只箱子只装一件,剩下的 k^*-l 只箱子每只箱子装二件(第 k^*+1 只箱子不算在内).否则,存在 $l\leqslant p<q\leqslant k^*$ 使得第 p 只箱子中装二件物品 i 和 $j(i<j)$,第 q 只箱子中装一件物品 k.根据算法,有 $s_i\geqslant s_k$,$s_j\geqslant s_m$.于是,$s_k+s_m\leqslant s_i+s_j\leqslant B$,物品 m 可装入第 q 只箱子,矛盾.

于是,对于任何装法,有 l 只箱子每只都只能装前 m 件物品中的一件.剩下的 $m-l$ 件物品的体积都大于$\frac{1}{3}B$,每只箱子至多装二件.所以,至少要 k^*+1 只箱子才能装入前 m 件物品.这与 k^* 是最优值矛盾,得证 $s_m\leqslant\frac{1}{3}B$.

(2) 至多有 k^*-1 件物品装在多用的(即下标大于 k^* 的)箱

子中.

假设有 k^* 件物品 $i_1,i_2,\cdots,i_{k^*}$ 装在多用的箱子中. 设前 k^* 只箱子中所装物品的体积分别为 $v_1,v_2,\cdots,v_{k^*}$. 对每一个 $j(1\leqslant j\leqslant k^*)$, $v_j+s_{i_j}>B$. 否则可以将物品 i_j 装入第 j 只箱子. 子是,

$$\sum_{i=1}^{n} s_i \geqslant \sum_{j=1}^{k^*} (v_j+s_{i_j})>k^*B.$$

这是不可能的.

由(1)和(2),至多有 k^*-1 件物品被装入多用的箱子中,它们的体积都不超过 $\frac{1}{3}B$,至多需要用 $\lceil\frac{1}{3}(k^*-1)\rceil$ 只箱子. 所以

$$\begin{aligned}\mathrm{FFD}(I)&\leqslant k^*+\lceil\frac{1}{3}(k^*-1)\rceil\leqslant k^*+\frac{1}{3}(k^*-1+2)\\&=\frac{4}{3}k^*+\frac{1}{3}.\end{aligned}$$ □

注意到当 $k^*=1$ 时 $\mathrm{FFD}(I)=1$,有

$$\frac{\mathrm{FFD}(I)}{k^*}\leqslant\frac{4}{3}+\frac{1}{3k^*}\leqslant\frac{4}{3}+\frac{1}{6}=\frac{3}{2}.$$

不难给出这样的实例 I:$\mathrm{OPT}(I)=2$,而 $\mathrm{FFD}(I)=3$. 从而,$r_{\mathrm{FFD}}=\frac{3}{2}$.

定理 14.6 假设 P≠NP,则装箱问题不存在近似比小于 $\frac{3}{2}$ 的多项式时间近似算法.

证:假设 A 是装箱问题的多项式时间近似算法,其近似比 $r_A<\frac{3}{2}$.

设任意的 n 个正整数 $a_1,a_2,\cdots,a_n$ 构成划分问题的实例 I,对应的装箱问题的实例 I' 中 n 件物品的体积为 $a_1,a_2,\cdots,a_n$,箱子的容积 $B=\lfloor\frac{1}{2}\sum_{i=1}^{n}a_i\rfloor$. 显然,$\mathrm{OPT}(I')=2$ 当且仅当实例 I 的答案为"是".

若 I 的答案为"是",则 $A(I')<\frac{3}{2}\mathrm{OPT}(I')=3$,即,$A(I')=$

2. 若 I 的答案为“否”，则 $A(I')\geqslant \mathrm{OPT}(I')>2$. 从而，$I$ 的答案为“是”当且仅当 $A(I')=2$.

于是，可以利用 A 构造出划分问题的多项式时间算法. 这与 P $\neq$NP 的假设矛盾. □

14.3 伪多项式时间算法与多项式时间近似方案

考虑背包问题的最优化形式：有 n 件物品和一个背包，背包的容量为 B，物品 j 的体积为 a_j，价值为 $c_j(1\leqslant j\leqslant n)$. 求使得背包内物品的价值最大的装法，即求 $J^*\subseteq\{1,2,\cdots,n\}$ 使得

$$\sum_{j\in J^*} c_j=\max\left\{\sum_{j\in J} c_j \,\middle|\, \sum_{j\in J} a_j\leqslant B, J\subseteq\{1,2,\cdots,n\}\right\}.$$

这里 a_j,c_j 以及 B 都是正整数且 $a_j\leqslant B(1\leqslant j\leqslant n)$.

这个问题可写成下述 0-1 整数线性规划：

$$\begin{aligned}&\max \sum_{j=1}^{n} c_j x_j\\ &\text{s.t.}\ \sum_{j=1}^{n} a_j x_j\leqslant B \qquad (14.1)\\ &\quad x_j=0 \text{ 或 } 1,\quad 1\leqslant j\leqslant n.\end{aligned}$$

定义背包函数

$$f_k(b)=\max\left\{\sum_{j=1}^{k} c_j x_j \,\middle|\, \sum_{j=1}^{k} a_j x_j\leqslant b,\ x_j=0 \text{ 或 } 1, 1\leqslant j\leqslant k\right\},$$

$0\leqslant k\leqslant n, 0\leqslant b\leqslant B$.

$f_k(b)$ 是装前 k 件物品且限制体积不超过 b 时可得到的最大价值. 显然，$f_n(B)$ 是问题的最优值. 下述递推公式给出背包问题的最优化算法.

递推公式(14.2)

$$f_0(b)=0$$

$$f_{k+1}(b)=\begin{cases} f_k(b) & \text{若 } b<a_{k+1} \\ \max\{f_k(b),f_k(b-a_{k+1})+c_{k+1}\}, & \text{否则,} \end{cases}$$

式中 $0\leqslant k<n, 0\leqslant b\leqslant B$.

例如,表 14-1 给出物品的体积和价值,背包的容积 $B=6$. 计算过程列于表 14-2 中. x_i 的计算如下:若 $f_{k+1}(b)=f_k(b)$ 则 $x_{k+1}=0$,否则 $x_{k+1}=1$.

表 14-1

j	1	2	3	4
a_j	4	3	2	1
c_j	3	2	1	1

表 14-2 背包函数的计算过程

b	0	1	2	3	4	5	6
f_1	0	0	0	0	3	3	3
f_2	0	0	0	2	3	3	3
f_3	0	0	1	2	3	3	4
f_4	0	1	1	2	3	4	4*
x_1	0	0	0	0	1*	1	1
x_2	0	0	0	1	0*	0	0
x_3	0	0	1	0	0	0	1*
x_4	0	1	0	0	0	1	0*

打 * 的是最优解和最优值.

最优解的值等于 $f_4(6)=4$. 按下述方法查表 14-2 得到最优解:当 $b=6$ 时 $x_4=0$;当 $b=6-a_4x_4=6$ 时 $x_3=1$;当 $b=6-a_3x_3=4$ 时 $x_2=0$;当 $b=4-a_2x_2=4$ 时 $x_1=1$.

按照这个算法要计算 n 个函数 $f_k(b)$,每一个函数要计算 $B+1$ 个值. 算法的时间复杂度为 $O(nB)$. 初看可能认为这是多项式时

间的，但这是不对的. 因为正整数 m 的二进制表示的长度为 $\lceil \log_2 m \rceil + 1$，故背包问题的实例 I 的规模应取作 $|I| = n\log_2 B$. nB 不是 $|I|$ 的多项式函数，故算法不是多项式时间的.

但是，这个算法所以不是多项式时间的，完全是由实例中的数可能很大造成的. 如果限制背包的容量 B 不超过 M，M 是一个固定的正整数，则算法成为多项式时间的. 这种算法称作伪多项式时间算法. 它的一般定义如下：

记实例 I 中的最大数为 $\max(I)$，如果算法的时间复杂度以 $|I|$ 和 $\max(I)$ 的某个二元多项式 $p(|I|, \max(I))$ 为上界，则称该算法是**伪多项式时间的**.

下面考虑背包问题的近似算法，首先考虑下述简单的贪心算法.

贪心法 G：

(1) 按单位体积的价值从大到小排列物品. 不妨设 $c_1/a_1 \geqslant c_2/a_2 \geqslant \cdots \geqslant c_n/a_n$.

(2) 顺序检查每一件物品，只要能装得下就将它装入背包，设装入背包的物品的总价值为 V.

(3) 令 $c_k = \max\{c_j \mid 1 \leqslant j \leqslant n\}$，若 $c_k > V$ 则将背包内的物品换成 k.

设物品 l 是第一件未装入背包的物品. 由于物品是按单位体积的价值从大到小排列的，故有

$$\mathrm{OPT}(I) < \mathrm{G}(I) + c_l \leqslant \mathrm{G}(I) + c_k \leqslant 2\mathrm{G}(I).$$

于是得到下述定理：

定理 14.7 对于背包问题的任何实例 I，

$$\mathrm{OPT}(I) < 2\mathrm{G}(I).$$

容易给出实例表明这个估计已经是最好的，从而 $r_G = 2$.

把贪心法嵌入一个较复杂的过程可以得到近似比任意接近 1 的近似算法.

算法 SA (S. Sahni，1975 年)

输入 $\varepsilon>0$ 和实例 I.

(1) 令 $m=\lceil 1/\varepsilon \rceil$.

(2) 按单位体积的价值从大到小排列物品. 不妨设 $c_1/a_1 \geqslant c_2/a_2 \geqslant \cdots \geqslant c_n/a_n$.

(3) 对所有 t 件物品($t=1,2,\cdots,m$),若这 t 件物品的体积之和不超过 B,则接着用贪心法尽可能地把剩余物品顺序放入背包.

(4) 比较得到的所有装法,取其价值最大的作为近似解.

设最优解为 J^*. 若 $|J^*| \leqslant m$,则算法必得到 J^*. 不妨设 $|J^*|>m$. 考虑计算中以 J^* 中 m 件价值最大的物品为初值,用贪心法得到的结果 J. 设物品 l 是 J^* 中第一件不在 J 中的物品,即

$$l=\min\{j \mid j\in J^* \text{ 且 } j\notin J\},$$

则

$$\begin{aligned}\mathrm{OPT}(I) &< \sum_{j\in J} c_j + c_l \\ &\leqslant \sum_{j\in J} c_j + \frac{1}{m}\sum_{j\in J} c_j \quad \leqslant \left(1+\frac{1}{m}\right)\mathrm{SA}(I).\end{aligned}$$

从 n 件物品中任取 t 件($t=1,2,\cdots,m$),所有可能的取法的个数为

$$c_n^1+c_n^2+\cdots+c_n^m \leqslant m\cdot\frac{n^m}{m!} \leqslant n^m.$$

算法需要的加法和比较次数为 $O(n^{m+1})$.

对每一个固定的 $\varepsilon>0$,把算法记作 SA_ε,则

$$\mathrm{OPT}(I)<(1+\varepsilon)\mathrm{SA}_\varepsilon(I),$$

SA_ε 的计算复杂度为 $O(n^{\frac{1}{\varepsilon}+2})$.

于是,得到下述结论:

定理14.8 对于每一个 $\varepsilon>0$和背包问题的实例 I,

$$\mathrm{OPT}(I)<(1+\varepsilon)\mathrm{SA}_\varepsilon(I)$$

并且 SA_ε 的计算复杂度为 $O(n^{\frac{1}{\varepsilon}+2})$.

可见对每一个固定的 $\varepsilon>0$,SA_ε 是背包问题的多项式时间近似算法且近似比小于$1+\varepsilon$.

设算法 A 以 $\epsilon>0$和问题的实例 I 作为输入. 如果对于每一个固定的 $\epsilon>0$,A 是一个多项式时间近似算法 A_ϵ 且 $r_{A_\epsilon}\leqslant 1+\epsilon$,则称 A 是一个**多项式时间近似方案**(PTAS).

定理14.8表明 SA 是背包问题的多项式时间近似方案. 多项式时间近似方案要以增加计算时间来换取更小的近似比. 在 SA 的时间复杂度中含有因子 $n^{\frac{1}{\epsilon}}$. 因此随着 ϵ 的变小,计算时间迅速地增加.

如果多项式时间近似方案 A 以某个二元函数 $p\left(|I|,\frac{1}{\epsilon}\right)$ 为时间复杂度上界,则称 A 是**完全多项式时间近似方案**(FPTAS).

背包问题不但有 PTAS、而且有 FPTAS,它是以伪多项式时间算法为基础. 不过这个伪多项式时间算法与本节前面给出的那个有点不同. 考虑模型(14.1)的对偶形式:

$$\begin{aligned}&\min \sum_{j=1}^{n} a_j x_j\\ &\text{s.t.} \sum_{j=1}^{n} c_j x_j \geqslant d\\ &\quad x_j=0\text{或}1,\ 1\leqslant j\leqslant n\end{aligned}\tag{14.3}$$

令

$$g_k(d)=\min\left\{\sum_{j=1}^{k} a_j x_j \ \Big|\ \sum_{j=1}^{k} c_j x_j\geqslant d,\ x_j=0\text{或}1,1\leqslant j\leqslant k\right\},$$

$0\leqslant k\leqslant n, 0\leqslant d\leqslant D$. 这里 $D=\sum_{j=1}^{n} c_j$ 且约定:当不存在可行解时 $g_k(d)=+\infty$.

背包问题的最优值为:

$$\mathrm{OPT}(I)=\max\{d\mid g_n(d)\leqslant B\}.$$

递推公式(14.4),

$$g_0(d)=\begin{cases}0 & \text{若 } d=0\\ +\infty & \text{若 } d>0\end{cases}$$

$$g_{k+1}(d)=\begin{cases}\min\{g_k(d),a_{k+1}\} & \text{若 } d\leqslant c_{k+1}\\ \min\{g_k(d),g_k(d-c_{k+1})+a_{k+1}\}, & \text{若 } d>c_{k+1}\end{cases}$$

$0\leqslant k<n, 0\leqslant d\leqslant D$.

记 $c_{\max}=\{c_j\mid 1\leqslant j\leqslant n\}$, $D\leqslant nc_{\max}$. 这个递推公式同样给出背包问题的最优化算法，其时间复杂度为 $O(n^2c_{\max})$. 把这个最优化算法记作 A.

算法 IK(O. H. Ibarra, C. E. Kim, 1975年)：

输入 $\varepsilon>0$和实例 I.

(1) 令 $\alpha=\max\left\{\left\lfloor c_{\max}/\left(1+\frac{1}{\varepsilon}\right)n\right\rfloor,1\right\}$.

(2) 令 $c'_j=\lceil c_j/\alpha\rceil$, $1\leqslant j\leqslant n$. 把这个新实例记作 I'.

(3) 对 I'运用算法 A，把 A 求得的解作为近似解.

由于 $\alpha(c'_j-1)<c_j\leqslant\alpha c'_j$, $1\leqslant j\leqslant n$, 对任意的 $J\subseteq\{1,2,\cdots,n\}$,

$$0\leqslant\alpha\sum_{j\in J}c'_j-\sum_{j\in J}c_j<\alpha|J|\leqslant\alpha n.$$

设 I 的最优解为 J^*，算法 IK 求得的近似解为 J(注意：J 是 I' 的最优解)，则有：

$$\begin{aligned}\mathrm{OPT}(I)-\mathrm{IK}(I)&=\sum_{j\in J^*}c_j-\sum_{j\in J}c_j\\&=\left(\sum_{j\in J^*}c_j-\alpha\sum_{j\in J^*}c'_j\right)+\left(\alpha\sum_{j\in J^*}c'_j-\alpha\sum_{j\in J}c'_j\right)\\&\quad+\left(\alpha\sum_{j\in J}c'_j-\sum_{j\in J}c_j\right)\\&\leqslant\alpha\sum_{j\in J}c'_j-\sum_{j\in J}c_j<\alpha n.\end{aligned}$$

对每一个 $\varepsilon>0$，当 $\alpha=1$时 $\mathrm{IK}(I)=\mathrm{OPT}(I)$. 不妨设 $\alpha>1$，注意到 $c_{\max}\leqslant\mathrm{OPT}(I)$，有

$$\mathrm{OPT}(I)-\mathrm{IK}(I)<\mathrm{OPT}(I)/\left(1+\frac{1}{\varepsilon}\right).$$

整理得到

$$\mathrm{OPT}(I)<(1+\varepsilon)\mathrm{IK}(I).$$

算法的时间复杂度为 $O(n^2c_{\max}/\alpha)=O\left(n^3\left(1+\frac{1}{\varepsilon}\right)\right)$.

于是，得到下述结论：

定理14.9 对每一个 $\varepsilon>0$和背包问题的每一个实例 I，

$$\mathrm{OPT}(I)<(1+\varepsilon)\mathrm{IK}_{\varepsilon}(I).$$

并且，IK 的时间复杂度为 $O\left(n^3\left(1+\frac{1}{\varepsilon}\right)\right)$.

该定理表明，IK 是背包问题的完全多项式时间近似方案.

完全多项式时间近似方案与伪多项式时间的最优化算法有相当密切的联系，下述定理清楚地表明了这一点.

定理14.10 如果存在二元多项式 q 使得对于组合优化问题 Π 的每一个实例 I，

$$\mathrm{OPT}(I)<q(|I|,\max(I)),$$

那么 Π 有完全多项式时间近似方案蕴涵它有伪多项式时间的最优化算法.

证：设 A 是 Π 的 FPTAS. 算法 A' 如下进行：对于任给的实例 I，取

$$\varepsilon=1/q(|I|,\max(I)),$$

对这个 ε 和实例 I 运用 A，A 输出的结果就是 A'的结果.

由于 A 的时间复杂度以 $|I|$和 $\frac{1}{\varepsilon}=q(|I|,\max(I))$的多项式为上界，这个多项式也是 $|I|$和 $\max(I)$的多项式，所以 A'是伪多项式时间的.

不妨设 Π 是最大化问题. 对每一个实例 I，有

$$\mathrm{OPT}(I)\leqslant(1+\varepsilon)A'(I).$$

根据 ε 的定义和定理的条件，得到

$$\mathrm{OPT}(I)-A'(I)\leqslant\varepsilon A'(I)\leqslant\varepsilon\mathrm{OPT}(I)<1.$$

由于所有的解值都是整数，由此得到 $A'(I)=\mathrm{OPT}(I)$. 因此，A'是 Π 的伪多项式时间的最优化算法. □

但是，定理14.10的逆不成立. 下一节将给出这样的例子.

至此，我们看到 NP 难的组合优化问题按其可近似性分为4类（在 P≠NP 的假设下）：

(1) 有完全多项式时间近似方案.

(2) 有多项式时间近似方案,但没有完全多项式时间近似方案.

(3) 有具有常数比的多项式时间近似算法,但没有多项式时间近似方案.

(4) 没有具有常数比的多项式时间近似算法.

货郎问题属于(4),装箱问题属于(3),背包问题属于(1). 下一节将要讨论的 k 背包问题($k\geqslant 2$)属于(2).

14.4 多背包问题

本节讨论多背包问题的可近似性.它是背包问题的自然推广.

多背包问题:有 n 件物品和 k 个背包,物品 j 的体积为 a_j、价值为 $c_j(1\leqslant j\leqslant n)$,背包 i 的容量为 $B_i(1\leqslant i\leqslant k)$,求使得装入背包的物品的总价值最大的装法. 即,求 k 个不相交的集合 $J_i\subseteq\{1,2,\cdots,n\}(1\leqslant i\leqslant k)$ 使得

$$\sum_{j\in J_i} a_j\leqslant B_i,\qquad 1\leqslant i\leqslant k$$

且

$$\sum_{i=1}^{k}\sum_{j\in J_i} c_j \qquad \text{最大.}$$

这里 a_j、$c_j(1\leqslant j\leqslant n)$ 以及 $B_i(1\leqslant i\leqslant k)$ 都是正整数.

当 k 为固定的正整数时,把这个问题称作 **k 背包问题**. 1背包问题就是背包问题.

对于 k 背包问题可以模仿上一节的做法,不难把它表示成0-1整数线性规划的形式,定义 k 背包函数 $f_t(b_1,b_2,\cdots,b_k)$,$0\leqslant b_i\leqslant B_i$,$1\leqslant i\leqslant k$,$0\leqslant t\leqslant n$,给出它的递推公式. 利用递推公式可以得到 k 背包问题的伪多项式时间算法. 但是,当 $k\geqslant 2$ 时,k 背包问题不存在 FPTAS. 在证明中要用到下述问题.

k 划分问题：任给 n 个正整数 $a_j(1\leqslant j\leqslant n)$，问是否能将它们均分成 k 组，即是否能把$\{1,2,\cdots,n\}$划分成 k 个不相交的子集 $J_i(1\leqslant i\leqslant k)$使得：

$$\sum_{j\in J_1} a_j=\sum_{j\in J_i} a_j \quad (i=2,3,\cdots,n)?$$

当 $k\geqslant 2$时，可以把划分问题多项式时间变换到 k 划分问题，从而 k 划分问题是 NP 完全的.

定理14.11 假设 P≠NP，则对于每一个正整数 $k\geqslant 2$，k 背包问题不存在完全多项式时间近似方案.

证：假设 A 是 k 背包问题的 FPTAS，其时间复杂度上界为多项式 $p\left(|I|,\frac{1}{\varepsilon}\right)$. 如下构造 k 背包问题的算法 A^*：对每一个实例 I，记 $c_{max}=\{c_j\mid 1\leqslant j\leqslant n\}$，令 $\varepsilon=1/2nc_{max}$. 将 I 和 ε 作为 A 的输入，A 的计算结果即为 A^* 的计算结果.

对于每一个实例 I，

$$\begin{aligned}0 &\leqslant \mathrm{OPT}(I)-A^*(I)=\mathrm{OPT}(I)-A_\varepsilon(I)\\ &\leqslant \varepsilon A_\varepsilon(I)\leqslant \varepsilon\mathrm{OPT}(I)\leqslant\frac{1}{2}.\end{aligned}$$

由于 $\mathrm{OPT}(I)$和 $A^*(I)$都是正整数，必有 $A^*(I)=\mathrm{OPT}(I)$. 从而，A^*是 k 背包问题的最优化算法. A^*的运行时间主要是 A 对 I 和 ε 的运行时间，因此时间复杂度上界为某个多项式 $q(|I|,2nc_{max})$.

现在利用 A^*构造 k 划分问题的算法 A'. 对于 k 划分问题的实例 I：正整数 $a_j(1\leqslant j\leqslant n)$，构造对应的 k 背包问题的实例 I'：物品 j 的体积为 a_j、价值 $c_j=1(1\leqslant j\leqslant n)$，背包 i 的容量 $B_i=\left\lfloor\sum_{j=1}^{n}a_j/k\right\rfloor(1\leqslant i\leqslant k)$. 将 A^*运用于 I'求得 $\mathrm{OPT}(I')$. 当且仅当 $\mathrm{OPT}(I')=n$ 时，A'对 I 回答“是”.

注意到 I'的 $c_{max}=1$，A^*使用的 $\varepsilon=1/2n$，运行时间不超过 $q(|I'|,2n)$. 这是关于$|I|$的多项式，所以 A'是 k 划分问题的多项式时间算法. 这与假设 P≠NP 矛盾. □

背包问题的算法 SA 可以推广到 k 背包问题上.

算法 MSA：

输入 $\varepsilon>0$和 k 背包问题的实例 I.

(1) 令 $m=\lceil k/\varepsilon\rceil$.

(2) 按单位体积的价值从大到小排列物品. 不妨设 $c_1/a_1\geqslant c_2/a_2\geqslant\cdots\geqslant c_n/a_n$.

(3) 对每一种满足条件 $t=\sum_{i=1}^{k}|J_i^0|\leqslant m$ 和 $\sum_{j\in J_i^0}a_j\leqslant B_i(1\leqslant i\leqslant k)$的 k 个不相交的子集 $J_i^0\subseteq\{1,2,\cdots,n\}(1\leqslant i\leqslant k)$，以 $J_i^0(1\leqslant i\leqslant k)$作为初值，用贪心法按照排定的顺序将剩余物品尽可能地装入背包中. 即顺序检查每一件剩余物品，只要它能装入某个背包，就把它装入这个背包.

(4) 比较得到的所有装法，取其价值最大的作为近似解.

和算法 SA 一样，可以证明对于所有的实例 I 和 $\varepsilon>0$，

$$\mathrm{OPT}(I)\leqslant\left(1+\frac{k}{m}\right)\mathrm{MSA}(I)\leqslant(1+\varepsilon)\mathrm{MSA}(I).$$

从 n 件物品中任取 t 件$(t=0,1,\cdots,m)$并且把它们任意地分成 k 组，共有

$$\sum_{t=0}^{m}c_n^tk^t=\sum_{t=0}^{m}\frac{n!k^t}{(n-t)!t!}$$

$$\leqslant\frac{n!}{(n-m)!}\sum_{t=0}^{m}\frac{k^t}{t!}\leqslant n^me^k$$

种可能. 对于每一种可能，贪心法将剩余物品装入背包所用的运算次数是 $O(kn)$. 所以，算法的时间复杂度上界为 $O(ke^kn^{m+1})=O(ke^kn^{\frac{k}{\varepsilon}+2})$.

于是，我们得到：

定理14.12 对于每一个正整数 k，MSA 是 k 背包问题的多项式时间近似方案.

综上所述，当 $k\geqslant2$时，k 背包问题有 PTAS，有伪多项式时间

最优化算法，但没有 FPTAS(除非 P=NP). 这表明定理14.10的逆不成立.

定理14.13 假设 P≠NP. 多背包问题不存在多项式时间近似算法 A 使得 $r_A \leqslant \frac{6}{5}$. 既使所有的背包容量都相同，且每一件物品的体积与价值都相等，即 $B_i = B(1 \leqslant i \leqslant k)$ 且 $a_j = c_j(1 \leqslant j \leqslant n)$，结论也仍然成立.

根据这个定理，多背包问题不存在 PTAS.

证：设 A 是多背包问题的多项式时间近似算法，其近似比 $r_A = c$. 利用 A 如下构造装箱问题的近似算法 A'. 设装箱问题的实例 I：n 件物品的体积 $s_j(1 \leqslant j \leqslant n)$ 和箱子的容积 B，这里 $s_j \leqslant B$ $(1 \leqslant j \leqslant n)$. 令 $S = \sum_{j=1}^{n} s_j$. 对 $k = 1, 2, \cdots, n$，构造 k 背包问题的实例 I_k：$a_j = c_j = s_j(1 \leqslant j \leqslant n)$ 和 $B_i = B(1 \leqslant i \leqslant k)$，把 A 运用于 I_k，直至找到 k_1 使得 $A(I_{k_1-1}) < \frac{1}{c}S \leqslant A(I_{k_1})$，这里 $A(I_0) = 0$. 然后用 FFD 法把剩余物品全部装入箱子中. 设 FFD 法使用 k_2 只箱子，A' 总共使用的箱子数为 $k = k_1 + k_2$. 显然，A' 是多项式时间的.

设 $\mathrm{OPT}(I) = k^*$，则 $\mathrm{OPT}(I_{k^*}) = S$，$A(I_{k^*}) \geqslant \frac{1}{c}S$. 因此，$k^* \geqslant k_1$. 记 $k = k^* + k'$，则 $k' \leqslant k_2$. FFD 法装入 k_2 只箱子里的物品的总体积 $\leqslant \left(1 - \frac{1}{c}\right)S$. 设装这些物品所需的最少箱子数为 l. 不妨设 $k_2 > l$，则有(见定理14.5的证明)：

(1) $k_2 \leqslant \frac{4}{3}l + \frac{1}{3}$，从而 $l \geqslant \frac{3}{4}\left(k_2 - \frac{1}{3}\right)$；

(2) 装入号码大于 l 的箱子中的物品的体积不超过 $\frac{1}{3}B$，从而前 l 只箱子中每只箱子所装物品的体积大于 $\frac{2}{3}B$.

于是

$$\left(1 - \frac{1}{c}\right)S > \frac{2}{3}Bl \geqslant \frac{1}{2}B\left(k_2 - \frac{1}{3}\right),$$

$$k_2<\frac{2}{B}\left(1-\frac{1}{c}\right)S+\frac{1}{3}\leqslant 2\left(1-\frac{1}{c}\right)k^*+\frac{1}{3},$$

$$\frac{k}{k^*}\leqslant 1+\frac{k_2}{k^*}<1+2\left(1-\frac{1}{c}\right)+\frac{1}{3k^*}.$$

注意到当 $k^*=1$ 时，$k=k^*$，所以

$$\frac{k}{k^*}<\frac{7}{6}+2\left(1-\frac{1}{c}\right).$$

如果 $c\leqslant\frac{6}{5}$，则 $\frac{k}{k^*}<\frac{3}{2}$，即 A' 是装箱问题的多项式时间近似算法，其近似比 $r_{A'}<\frac{3}{2}$. 根据定理14.6，这与 P≠NP 的假设矛盾. □

最后给出多背包问题的一个多项式时间近似算法，其近似比为2. 算法的主要思想是按“价值密度”从大到小的顺序检查每一件物品. 对于每一个背包 i，先把能装进它的物品放在一起，当这些物品的体积之和不小于 $\frac{1}{2}B_i$ 时才把它们装入背包 i.

算法 ZLH：

(1) 按单位体积的价值从大到小排列物品，不妨设 $c_1/a_1\geqslant c_2/a_2\geqslant\cdots\geqslant c_n/a_n$.

按容量从小到大排列背包，不妨设 $B_1\leqslant B_2\leqslant\cdots\leqslant B_k$.

(2) 令 $M=\{1,2,\cdots,k\}$，$J_i'=\varnothing$，$A_i'=0(1\leqslant i\leqslant k)$.

(3) 对 $j=1,2,\cdots,n$

找到 M 中使 $a_j\leqslant B_i$ 的最小的 i（如果没有这样的 i，则舍去物品 j）.

a. 如果 $a_j+A_i'<\frac{1}{2}B_i$，则令 $A_i'=A_i'+a_j$，$J_i'=J_i'\cup\{j\}$.

b. 如果 $\frac{1}{2}B_i\leqslant a_j+A_i'\leqslant B_i$，则令 $J_i=J_i'\cup\{j\}$，$M=M-\{i\}$.

c. 如果 $a_j+A_i'>B_i$ 且存在 $t\in M$ 使 $t>i$，设 $t=\min\{t'\mid t'\in M$ 且 $t'>i\}$，则令 $J_i=\{j\}$，$M=M-\{i\}$.

c.1 如果 $A_t'+A_i'<\frac{1}{2}B_t$，则令 $A_t'=A_t'+A_i'$，$J_t'=J_t'\cup J_i'$.

c. 2 否则$\left(\frac{1}{2}B_i\leqslant A_i'+A_t'\leqslant B_i\right)$令 $J_i=J_i'\cup J_t'$，$M=M-\{t\}$.

d. 否则($a_j+A_j'>B_i$ 且不存在 $t\in M$ 使 $t>i$)令 $M=M-\{i\}$，如果 $c_j\leqslant\sum_{t\in J_i'}c_t$ 则令 $J_i=J_i'$ 否则令 $J_i=\{j\}$. (记 $r_i=\min\{c_j,\sum_{t\in J_i'}c_t\}$

(4) 如果 $M\neq\varnothing$ 则对所有的 $i\in M$，令 $J_i=J_i'$.

设多背包问题的最优解为$\{J_i^*\}$，算法 ZLH 求得的近似解为$\{J_i\}$，那么不难看到：

1. 如果在步骤(4)$M\neq\varnothing$，令 $t=\max\{i\mid i\in M\}$，则所有 $a_j\leqslant B_t$ 的物品都被装进背包. 因此，存在最优解使得 $J_i^*=J_i(1\leqslant i\leqslant t)$. 此时，令 $r_i=0(1\leqslant i\leqslant t)$.

2. 如果 J_i 在步骤(3)的情况 b 或 $c.2$得到，则 $\sum_{j\in J_i}a_j\geqslant\frac{B_i}{2}$. 设 l 是下一个检查的物品，令 $r_i=\frac{c_l}{a_l}\cdot\frac{B_i}{2}$，则有 $r_i\leqslant\sum_{j\in J_i}c_j$.

3. 如果 J_i 在步骤(3)的情况 d 得到，则也有 $r_i\leqslant\sum_{j\in J_i}c_j$.

从而

$$\mathrm{OPT}(I)=\sum_{i=1}^{k}\sum_{j\in J_i^*}c_j\leqslant\sum_{i=1}^{k}\left(\sum_{j\in J_i}c_j+r_i\right)$$

$$\leqslant 2\sum_{i=1}^{k}\sum_{j\in J_i}c_j=2\mathrm{ZLH}(I).$$

于是，得到：

定理14. 14 对多背包问题的所有实例 I，有

$$\mathrm{OPT}(I)\leqslant 2\mathrm{ZLH}(I).$$

下述实例表明定理的估计已经是最好的，因此 $r_{\mathrm{ZLH}}=2$.

实例 I：$a_j=c_j=M-1(1\leqslant j\leqslant k)$，$a_j=c_j=M\ (k+1\leqslant j\leqslant 3k)$，$B_i=2M(1\leqslant i\leqslant k)$，其中 M 是正整数. 显然，$\mathrm{ZLH}(I)=k(M+1)$，$\mathrm{OPT}(I)=2kM=2\left(1-\frac{1}{M+1}\right)\mathrm{ZLH}(I)$. 任给 $\varepsilon>0$，取 $M=\lfloor\frac{1}{\varepsilon}\rfloor$，则有 $\mathrm{OPT(I)}>2(1-\varepsilon)\mathrm{ZLH(I)}$.

附　录

附录 A　记　号

记　号	首次出现的章节
$\mathrm{dom}R$	1.1
$\mathrm{ran}R$	1.1
$f(x_1,\cdots,x_n)\downarrow$	1.1
$f(x_1,\cdots,x_n)\uparrow$	1.1
A^*	1.1
$\lvert w\rvert$	1.1
$\psi_{\mathscr{P}}^{(n)}(x_1,\cdots,x_n)$	1.3,4.3
$x\dot{-}y$	1.3,2.1.4
$s(x)$	2.1.3
$n(x)$	2.1.3
$u_i^n(x_1,\cdots,x_n)$	2.1.3
$a(x)$	2.1.4
$y\mid x$	2.3.2
$\mathrm{Prime}(x)$	2.3.2
$\min_{z<y}$	2.3.3
$\min\limits_{z}$	2.3.3
$R(x,y)$	2.3.3
p_n	2.3.3
$\lfloor x/y\rfloor$	2.3.3
$(x,\vec{y})$	2.4.1
$l(z)$	2.4.1
$r(z)$	2.4.1
$[a_1,a_2,\cdots,a_n]$	2.4.2

记　号	首次出现的章节
$(x)_i$	2.4.2
$\mathrm{Lt}(x)$	2.4.2
$A(k,x)$	2.6
$\#(\mathscr{P})$	3.1
$\mathrm{HALT}(x,y)$	3.2
$\Phi^{(n)}(x_1,\cdots,x_n,y)$	3.3
$\Phi_y^{(n)}(x_1,\cdots,x_n)$	3.3
$\mathrm{STP}^{(n)}(x_1,\cdots,x_n,y,t)$	3.3
$S_m^n(u_1,\cdots,u_n,y)$	3.4
$R^*(x,y)$	4.1
$Q^*(x,y)$	4.1
$g(m,n,x)$	4.1
$h(m,n,x)$	4.1
$\mathrm{LENTH}_n(x)$	4.1
$\mathrm{CONCAT}_n^{(m)}(u_1,\cdots,u_m)$	4.1
$\mathrm{RTEND}_n(w)$	4.1
$\mathrm{LTEND}_n(w)$	4.1
$\mathrm{RTRUNC}_n(w)$	4.1
$\mathrm{LTRUNC}_n(w)$	4.1
w^-	4.1
$\mathrm{UPCHANGE}_{n,l}(x)$	4.1
$\mathrm{DOWNCHANGE}_{n,l}(x)$	4.1
x^R	第四章习题
$\chi_E(x)$	5.1,5.2
W_n	5.1
K	5.3
R_T	5.3
$\vdash$ ($\underset{\mathscr{M}}{\vdash}$)	6.1,10.3
$\vdash^*$ ($\underset{\mathscr{M}}{\vdash^*}$)	6.1,10.3
$\psi_{\mathscr{M}}^{(n)}(x_1,\cdots,x_n)$	6.1
$L(\mathscr{M})$	6.3,6.5,9.2,10.3,11.2

记　号	首次出现的章节
$\Rightarrow(\underset{\Pi}{\Rightarrow},\underset{G}{\Rightarrow})$	7.1,7.3
$\overset{*}{\Rightarrow}(\underset{\Pi}{\overset{*}{\Rightarrow}},\underset{G}{\overset{*}{\Rightarrow}})$	7.1,7.3
$\Sigma(\mathscr{M})$	7.2
$\Omega(\mathscr{M})$	7.2
DERIV(u,y)	7.3
$L(G)$	7.3
$\mathrm{K}_{\mathscr{L}}$	8.5
$\underset{\mathrm{K}_{\mathscr{L}}}{\vdash}$	8.5
$\models$	8.5
$w^{\#}$	8.5
$L_1 \cdot L_2$　(L_1L_2)	9.3
L^*	9.3
L^+	9.3
$\langle\alpha\rangle$	9.4
$r+s$	9.4
$r \cdot s(rs)$	9.4
r^*	9.4
ker(G)	10.1
$N(\mathscr{M})$	10.3
$O(f)$	12.1
$\Theta(f)$	12.1
$t_{\mathscr{M}}(n)$	12.1
$s_{\mathscr{M}}(n)$	12.1
DTIME$(t(n))$	12.3
DSPACE$(s(n))$	12.3
NTIME$(t(n))$	12.3
NSPACE$(s(n))$	12.3
L	12.4
NL	12.4
P	12.4,13.1
NP	12.4,13.1

记　号	首次出现的章节
PSPACE	12.4
NPSPACE	12.4
EXPTIME	12.4
NEXPTIME	12.4
EXPSPACE	12.4
$L[\pi,e]$	13.1
$\leqslant_m$	13.2
OPT(I)	14.1
$A(I)$	14.1
$r_A(I)$	14.1
r_A	14.1

…

记　号	含　义
inf	下确界
N	自然数集
Z^+	正整数集
e	空字符串
$\varnothing$	空集
$\lfloor x \rfloor$	不超过 x 的最大整数
$\lceil x \rceil$	不小于 x 的最小整数
$\sum_{i=0}^{y}$	求和
$\prod_{i=0}^{y}$	求积
$\cup$	并
$\cap$	交
$\wedge$	与
$\vee$	或
$\neg$	非
$\forall t$	全称量词
$\exists t$	存在量词

记　号	含　义
$(\forall t)_{\leqslant y}$	有界全称量词
$(\exists t)_{\leqslant y}$	有界存在量词
$\Leftrightarrow$	当且仅当

附录 B　中英文名词索引

三　划

四　划

五　划

六　划

七 划

八　划

九 划

十 划

十 二 划

十三划以上

其 他

单向无穷带 one-way infinite (6.4.2)

多带 multitape (6.4.3)

多维 multidimensional (6.4.3)

基本 basic (6.1)

非确定型 nondeterministic (6.5)

确定型 deterministic (6.)

在线 on-line (6.4.4)

离线 off-line (6.4.4)

k 带 k-tape (6.4.3)

$f(n)$时间界限的 $f(n)$ time-bounded (12.1)

$s(n)$空间界限的 $s(n)$ space-bounded (12.1)

Co-NP (13.5)

CFG(上下文无关文法)

CSG(上下文有关文法)

CFL(上下文无关语言)

CSL(上下文有关语言)

DCFL(确定型上下文无关语言)

DFA(确定型有穷自动机)

3DM(三维匹配问题)

DPDA(确定型下推自动机)

DTM(确定型 Turing 机)

EXPSPACE (12.4)

EXPTIME (12.4)

FA(有穷自动机)

FFD(递降首次适合算法)

FPTAS(完全多项式时间近似方案)

HC(Hamilton 回路问题)

ILP(整数线性规划问题)

L (12.4)

LBA(线性界限自动机)

MM(最小权匹配法)

MSA (14.1)

MST(最小生成树法)

NEXPTIME (12.4)

NFA(非确定型有穷自动机)

NL (12.4)

NN(最邻近法)

NP (12.4, 13.1)

NPC (13.5)

NPSPACE (12.4)

NTM(非确定型 Turing 机)

P (12.4, 13.1)

PDA(下推自动机)

PSPACE (12.4)

PTAS(多项式时间近似方案)

r.e.(递归可枚举的)

SAT(可满足性问题)

3-SAT(三元可满足性问题)

TM(Turing 机)

TSP(货郎问题)

VC(顶点覆盖问题)

ZLH (14.4)

参 考 文 献

[1] Marin D. Davis, Elaine J. Weyuker, Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science, Academic Press, Inc. 1983

中译本：张立昂、陈进元、耿素云译．可计算性、复杂性、语言，理论计算机科学基础．北京：清华大学出版社，1989

[2] John E. Hopcroft, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company. 1979

中译本：徐美瑞译，洪加威校．自动机理论、语言和计算导引．北京：科学出版社．1986

[3] Michael R. Garey, David S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman and Company. 1979

中译本：张立昂、沈泓、毕源章译．吴允曾校．计算机和难解性，NP完全性理论导引．北京：科学出版社．1987

[4] José L. Balcázar, Josep Díaz, and Joaquim Gabarró, Structural Complexity I, Springer-Verlag, 1988

[5] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization, Algorithms and Complexity. Printice-Hall Inc. 1982

中译本：刘振宏、蔡茂诚译．组合最优化，算法和复杂性．北京：清华大学出版社，1988.

[6] 王捍贫．数理逻辑，离散数学一分册．北京：北京大学出版社．1997

[7] 耿素云，集合论与图论，离散数学二分册．北京：北京大学出版社．1997