可计算性与计算复杂性

周长林　李占山 编

吉林大学计算机科学与技术学院

目　录

第一部分 预备知识

第二部分可计算性理论

第三部分 计算复杂性

第一部分 预备知识

第一章　导引

本书主要介绍可计算性与计算复杂性理论涉及到的最基本知识以及学习本书需要的入门知识，集中在计算理论的可计算性与计算复杂性。下述问题把它们联系在一起。

计算机的基本能力和限制是什么？

这个问题要回溯到 20 世纪 30 年代，那时数理逻辑学家们首先开始探究计算的含义。自那时起以来，技术进步极大地增强了我们的计算能力，并且把这个问题从理论王国带进现实世界。在可计算性与计算复杂性中的每一个领域内，对这个问题作了不同的解释，并且答案随着解释的不同而各不相同。在此，我们用相反的顺序介绍，因为从后面开始介绍能够更好地理解本书为什么这样安排顺序。

计算复杂性理论

计算的问题是各种各样的，有的容易，有的困难。例如，排序是一个容易的问题，比如说需要按升序排列一张数表，即使一台小型计算机都相当快地处理 100 万个数。与时间表问题比较一下，比如说要制定一所大学的课程表，课程表必须满足某些合理的限制，如不能有两个班在同一时间使用同一教室。时间表问题看来比排序问题困难得多。如果有 1000 个班，即使是使用一台超级计算机，制定一份最好的课程表也可能需要若干世纪。是什么使某些问题很难计算，又使另一些问题容易计算？这是复杂性理论的核心问题。值得注意的是，虽然在过去的二十多年里对它进行了深入细致的研究，但是我们仍然不知道它的答案。以后我们将探究这个迷人的问题和它的一些分支。迄今为止，复杂性理论的一个重要成果是，发现了一个按照计算难度给问题分类的完美体系。它类似按照化学性质给元素分类的周期表。运用这个体系，我们能够提出一种方法给出某些问题是难计算的证据，尽管还不能证明它们是难计算的。当你面对一个看来很难计算的问题时，有几种选择。首先，搞清楚问题困难的原因，你可能会做某些变动，使问题变的容易解决。其次，你可能会求出问题的不那么完美的的解。在某些情况下，寻找问题的近似解相对容易一些。第三，有些问题仅仅在最坏的情况下是困难的，而在绝大多数的时候是容易的。就应用而言，一个偶尔运行得很慢、而通常运行得很快的过程是能够令人满意的。最后，你可以考虑其他类型的计算，如随机计算，它能够加速某些工作。受到复杂性理论直接影响的一个应用领域是密码技术，这是一个古老的研究领域。在绝大多数的情况下，容易计算的问题比难计算的问题更可取，因为求解容易的

问题的代价要小。密码技术与众不同，它特别需要难计算的问题，而不是容易计算的问题。在不知道密钥或口令时，密码应该是很难破译的。复杂性理论给密码研究人员指出了寻找计算问题的方向，围绕这些问题已经设计出新的革命性的编码。

可计算性理论

在 21 世纪前半叶，数学家们，如歌德尔(Kurt Gödel)、图灵(Alan Turing)及丘奇(Alonzo Church)，发现一些基本问题是不能用计算机解决的。确定一个数学命题是真是假就是一个例子。这项工作是数学家的生计。用计算机来解决似乎是天经地义的，因为这是数学王国里的事。但是，没有计算机算法能够完成这项工作。关于计算机理论模型思想的发展是这一意义深远的结论的成果之一，它最终促使建造出实际的计算机。可计算理论与复杂性理论是密切相关的。在复杂性理论中，目标是把问题分成容易的和困难的；而在可计算理论中，是把问题分成可解的和不可解的。可计算性理论介绍了若干复杂性理论中使用的概念。下面简单介绍学习本书需要的一些基本知识。

1.1 集合的概念与相关运算

集合是数学中最基本的概念。既然是最基本的概念，它就不那么好定义，一般只是说明，正象数学中“点”一样。最基本的东西就是大家都知道的东西。要说明什么是集合，有多种描述方法：“所要讨论的一类对象的整体”；“具有同一性质单元的集体”等。当我们讨论某一类对象的时候，就把这一类对象的整体称为集合。而集合中的对象就称为该集合中的元素。

Cantor 是这样描述集合的：所谓集合，是指我们无意中或思想中将一些确定的，彼此完全不同的客体的总和而考虑为一个整体。这些客体叫做该集合的元素。

设 A 是一个集合，a 是集合 A 中的元素，今后将这一事实记以 $a\in A$，读做 a 属于 A；若 a 不是集合 A 中的元素，则记以 $a\notin A$,读做 a 不属于 A。

例如，这间教室里所有桌子的整体就做成一个桌子集合。每个桌子都属于这个集合,每个椅子都不属于这个集合。

又如,世界上所有哺乳动物的整体做成一个哺乳动物集合。每一条狗每一只猫都属于这个集合,而每一只鸡都不属于这个集合。

又如,平面上的所有点的整体做成平面点集；所有连续函数的整体做成连续函数集,等等。

有限个元素$a_1,\ldots,a_n$做成的集合,称为有穷集（有限集）,记以{ $a_1,\ldots,a_n$ };无限个元素做成的集合,称为无穷集。有穷集中元素的个数称为该集合的元素数，记为|A|。

特别,不含元素的集合称为空集,记以ϕ,一个元素 a 做成的集合,记以{a}。

定义 1.1 设 A,B 是两个集合。属于集合 A 而不属于集合 B 的所有元素组成的集合，称为 A 与 B 的差集，记以 A-B。

例如，令 A={a,b,c,d},B={b,c,e,f}，于是 A-B={{a,d},B-A={e,f}。

定义 1.2 设 A,B 是两个集合。所有属于 A 或者属于 B 的元素做成的集合，称为 A 和 B 的并集，记以 A∪B。

例如，令 A={a,b,c,d},B={b,d,e,f},于是 A∪B={a,b,c,d,e,f}。

我们可以把定义 1.2 推广到多个集合的并集。

定义 1.3 设 A,B 是两个集合。由属于 A 又属于 B 的元素做成的集合，称为 A 和 B 的交集(intersection set)，记以 A∩B。

例如，上面集合 A 和 B 的 A∩B={b,d}。

同样，我们可以把定义 1.3 推广到多个集合的交集。

定义 1.4 设 A,B 是两个集合，所有序偶 (x,y)做成的集合（其中 x∈A,y∈B）,称为 A,B 的直乘积(笛卡儿积)，记以 A×B。

直乘积符号化表示为 A×B={(x,y)|x∈A 且 y∈B}。

例如，令 A 是直角坐标系中 x 轴上的点集，B 是 y 轴上的点集，于是 A×B 就和平面点集一一对应。

由排列组合的基本常识不难证明,如果集合 A 的元数|A|=m,集合 B 的元数|B|=n,则

A×B 中有 mn 个元素。

1.2 关　系

1.2.1 关系的基本概念及其性质

在日常生活中,象在数学中一样,关系的概念是一个基本概念。例如,在人群中有朋友关系,父子关系,同学关系等等；在数学中有相等关系,整除关系,大小关系等等。

不难看到,每一种关系都描述了在某一集合中两个元素之间的一种特征。而这种特征也可以用一个集合描述出来。

定义 1.5 设A_1，$A_2,\ldots,A_n$是n个集合，集合$A_1\times A_2\times\ldots\times A_n$的一个子集F称为$A_1$，$A_2,\ldots,A_n$上的一个n元关系。特别地，集合A×B中的一个子集R,称为集合A与B上的一个二元关系,简称

为关系。对于x∈A,y∈B,若(x,y)∈R则称x,y有关系R,记为xRy；若(x,y)∉R,则称x,y没有关系R,记为$x\not R y$。若B=A，则R称为A上的二元关系。

由关系的定义我们能够知道关系有下列特点:

(1) A×A 上的任一子集都是 A 上的一个关系。

(2) 若|A|=n,则 A 上的关系有 2^{n^2} 个。

(3) A上的三个特殊关系,即空关系ϕ;全域关系E_A=A×A;相等关系I_A={(x,x)|x∈A}。

(4) $\bar{R}$=A×A-R。

(5) 序偶(a,b)=(c,d)的充要条件是 a=c,b=d。

例 1. 1 设 A={a, b, c, d, e, f}为学生集合，B={α, β, γ, δ}为选修课程集合，C={2,3,4,5}为学习成绩集合，学生与课程之间存在着一种关系即“选修关系”；学生、课程和成绩之间也存在着一种叫做“学习成绩关系”。设用 R 表示选修关系，S 表示学习成绩关系，那么 R 为 A 与 B 上的二元关系，S 为 A，B 和 C 上的三元关系。

R={(a, α),(a, δ),(b, β),(b, δ),(c, β),(c, γ),(e, α),(f, γ)}

表示学生 a 选修课程α，δ；学生 b 选修课程β，δ；学生 c 选修课程β，γ；学生 e 选修课程α；学生 f 选修课程γ；而学生 d 没有选修任何课程。

S={(a, α,5), (a, δ,5), (b, δ,3), (c, β,4), (f, γ,2)}

表示学生 a 所选的两门课程成绩都是 5 分；学生 b 所选δ课程的成绩是 3 分；学生 c 所选β课程的成绩是 4 分；学生 f 所选γ课程的成绩是 2 分。

既然关系是集合，因此有时用处理集合的方法处理关系是方便的。因而有子关系，关系的并、交、差、余等运算。

如，R，S 是集合 A 上的两个关系，若 R⊆S，则称 R 为 S 的子关系；对任意 x,y∈A，有

x(R∪S)y 当且仅当 xRy 或者 xSy

x(R∩S)y 当且仅当 xRy 并且 xSy

x(R-S)y 当且仅当 xRy 并且 $x\not S y$

$x\bar{R}y$ 当且仅当 $x\not R y$

当然，集合的并、交、差、余运算诸性质对关系运算也成立。需要注意的是，作为关系时，余运算是对全域关系而言的，即 A×B 作为全集 E。

定义 1.6 设 R 是集合 A 上的一个关系。令

R^{-1} ={(y,x)|x∈A,y∈A,并且有xRy}

则称关系R^{-1}为关系R的逆。

例如，小于关系的逆关系是大于关系,大于关系的逆关系是小于关系,相等关系的逆关系仍是相等关系。

定义 1.7 集合 A 上的关系 R 称为自反的（反身的）,如果对每个 x∈A,都有 xRx。

定义 1.8 集合 A 上的关系 R 称为对称的,如果 xRy,则 yRx。其中 x∈A,y∈A。

定义 1.9 集合 A 上的关系 R 称为有反对称性,如果 xRy,yRx,则必有 x=y。 其中 x∈A,y∈A。

定义 1.10 集合 A 上的关系 R 称为有传递性,如果 xRy,yRz,则 xRz。其中 x∈A,y∈A,z∈A。

例如，数之间的相等关系，具有自反性，对称性，传递性，反对称性。小于关系和大于关系没有自反性，没有对称性，但是有反对称性和传递性。父子关系既无自反性，也无对称性又无传递性，但是具有反对称性。

定义 1.11 集合 A 上的关系 R 称为反自反的,如果对任意的 x∈A，xRx 均不成立。

定义 1.12 设 R,S 是集合 A 上的两个关系，令

R•S={(x，y)|x∈A，y∈A 并且有一个 z∈A 使得 xRz，zSy}。称关系 R•S 为关系 R 和 S 的乘积或合成。

例如，兄弟关系和父子关系的乘积是叔侄关系，而姐妹关系和母子关系的乘积是姨与外甥关系。

作为练习我们可以来证明关系的乘法满足结合律以及关系的乘法不满足交换律。

由于关系的乘法运算满足结合律，因此我们可以用“幂”表示集合上同一个关系的乘积，即 $R^n = \underbrace{R \bullet R \bullet \cdots \bullet R}_{n}$，规定$R^0=I_A$。

定理 1.1 设 R 是 A 上的关系，m，n 为任意的自然数，那么

（1） $R^m \bullet R^n=R^{m+n}$;

（2） $(R^m)^n=R^{mn}$。

证明：（1）任给m，对n作归纳法。n=0，则$R^m \bullet R^0=R^m \bullet I_A= R^m = R^{m+0}$。假设$R^m \bullet R^n=R^{m+n}$，那么$R^m \bullet R^{n+1}=R^m \bullet (R^n \bullet R^1)= (R^m \bullet R^n) \bullet R^1= R^{m+n} \bullet R^1= R^{m+n+1}= R^{m+(n+1)}$ 。

使用同样方法或利用（1）的结果可证明（2），不赘述。

定理 1.2 设集合A的元数为n，R是A上二元关系，那么存在自然数i，j($0 \le i < j \le 2^{n^2}$)使得$R^i=R^j$。

证明：由关系的特点我们知道，若|A|=n，则A上的关系有2^{n^2}个，因此，在 R^0，R^1，R^2，…，$R^{2^{n^2}}$ 这2^{n^2}+1 个关系中，至少有两个是相同的（鸽巢原理），即有i，j($0 \le i < j \le 2^{n^2}$) 使得$R^i=R^j$。

关于鸽巢原理请参阅组合数学相关内容。

定理 1.3 设R是集合A上的关系，若存在自然数i，j(i<j)，使得$R^i=R^j$，则有

（1） $R^{i+k}=R^{j+k}$，k∈N;

（2） $R^{i+kp+m}=R^{i+m}$，其中k，m∈N，p=j-i。

证明：由定理 1.1，（1）显然成立。

（2）利用（1）直接证明（2），k=0，1 时显然成立，考虑 k≥2。

$R^{i+kp+m}= R^{i+j-i+(k-1)(j-i)+m}= R^{j+(k-1)(j-i)+m}= R^{i+(k-1)(j-i)+m}$ （利用（1））

$= R^{i+j-i+(k-2)(j-i)+m}= R^{j+(k-2)(j-i)+m}= R^{i+(k-2)(j-i)+m}$ （利用（1））

…

$=R^{i+j-i+m}=R^{j+m}= R^{i+m}$。 （利用（1））

请读者使用归纳法证明之。

定理 1.4 集合A上的关系R具有传递性的充要条件是$R\bullet R\subseteq R$($R\bullet R$可简记为R^2)。

证明：必要性。若R具有传递性，任取（x，y）$\in R^2$，于是存在z∈A，使得

xRz，zRy

因为 R 是传递的，所以有 xRy，即（x，y）∈R。

故$R^2\subseteq R$。

充分性。若$R^2\subseteq R$，如果xRy，yRz，则xR^2z。故xRz。所以R是传递性的。

一般来说，设 A 是一个集合，A 上的关系 R 不一定具有上面讨论过的某些性质，如自反性、对称性或传递性。例如，一个计算机网络在波士顿、芝加哥、丹佛、底特津、纽约和圣地亚哥设有数据中心。从波士顿到芝加哥，波士顿到底特津，芝加哥到底特津，底特津到丹佛和纽约到圣地亚哥，都有单向的电话线。如果存在一条从数据中心 a 到 b 的电话线，(a,b) 就属于关系 R。我们如何确定从一个中心是否有一条电话线或多条电话线（可能不直接）链接到另一个中心？由于所有的链接不一定是直接的，例如从波士顿可通过底特津到丹佛，因此不能直接使用 R 来回答这个问题。用关系的语言来说，R 不是传递的，因而它不包含可能被链接的所有的对。正如我们下面讲述的，可以通过构造包含 R 的最小的传递关系来找出每一对有着连线的数据中心。这个关系叫做 R 的传递闭包。

为此我们讨论最小的包含 R 的关系 R′，使得它具有所要求性质，这就是关系的闭包。

定义 1.13 设 A 是非空集合，R 是 A 上的二元关系。R 的自反闭包（对称闭包，传递闭包）R′满足如下条件：

（1）R′是自反的（对称的，传递的）；

（2）R⊆R′；

（3）对 A 上任意自反的（对称的，传递的）关系 R″，R⊆R″，必有 R′⊆R″。

R 的自反闭包、对称闭包和传递闭包分别记为 r(R)，s(R)，t(R)，也称 r，s，t 为闭包运算，它们作用于关系 R 后，产生包含 R 的最小的自反、对称、传递的关系。这三个闭包也可依据下述定理来计算。

定理 1.5 设 R 是集合 A 上的关系，那么

（1） $r(R)=I_A\cup R$；

（2） $s(R)=R\cup R^{-1}$；

（3） $t(R)=\bigcup_{i=1}^{\infty}R^i$ 。

证明：（1）和（2）式的证明比较简单，留给读者，我们仅证明（3）。$R\subseteq\bigcup_{i=1}^{\infty}R^i$ 是显然的。为证$\bigcup_{i=1}^{\infty}R^i$ 是传递的 ，设(x，y)∈ $\bigcup_{i=1}^{\infty}R^i$ ，(y，z)∈ $\bigcup_{i=1}^{\infty}R^i$ ，那么有正整数j，k，使得

设(x，y)∈R^j，设(y，z)∈R^k，于是有(x，z)∈$R^j \bullet R^k = R^{j+k}$，从而(x，z)∈ $\bigcup_{i=1}^{\infty} R^i$ ，即$\bigcup_{i=1}^{\infty} R^i$ 是传递的。最后，令R′传递，且R⊆R′，需要证明$\bigcup_{i=1}^{\infty} R^i \subseteq R'$。为此只要证对任意正整数n，$R^n \subseteq R'$。

对n采用归纳法，n=1 时显然有$R^1 \subseteq R'$。假设n=k时有$R^k \subseteq R'$，设(x，y)∈R^{k+1}，那么有z使(x，z)∈R^k，(z，y)∈R。根据归纳假设及题设，有(x，z)∈ R′，(z，y)∈ R′。但R′是传递的，故(x，y)∈R′，即$R^{k+1} \subseteq R'$证毕。

利用上述定理计算自反闭包和对称闭包比较容易，传递闭包的计算一般会很复杂。但是当集合 A 是有穷集合时，A 上关系的传递闭包求取便可大大简化，有兴趣的读者可参阅相关内容。

1.2.2 等价关系

在日常生活中和在数学中，我们常常碰到对一些对象进行分类的问题。例如，对一些几何图形，我们可以使用面积之间的相等关系将这些几何图形分类，即面积相等的几何图形算做一类，这种分类使得每个几何图形都必定属于某类，并且不同类之间没有公共元素。又如，在人群中，我们可以用同性关系将人群分类，即同性别的人算做一类。这种分类也使得每一个人都必定属于某类，并且不同类之间没有公共元素。

因此，任意一个分类法总是在某一观点下把一些元素看做是同样的，并且希望每一个元素在这种分类法下都必定属于而且仅仅属于某一类，具有这种功能的分类法，在数学上就叫做一个等价关系，其严格定义如下：

定义 1.14 设 A 是一个非空集合，≅是 A 上一个关系。如果≅具有自反性，对称性，传递性，则称≅是一个等价关系。

注意，当≅是 A 上的一个等价关系时，并不是说 A 中任意两个元素都有 R 关系，而是有些元素做成一个等价类组，有些元素做成另一个等价类组，即将 A 中各元素按 R 等价关系分成若干类，每一类就是 A 的一个子集，也就是等价类。

上面提到的几何图形的面积之间的相等关系，人群中的同性关系都是等价关系。

定义 1.15 设 A 是一个非空集合，≅是 A 上的等价关系。A 的一个非空子集 M 叫做一个等价类，如果

1）若 a∈M,b∈M,则 a≅b。

2）若 a∈M,b∉M,则 a≇b；或者

若 a∈M, a≅b,则 b∈M。

换句话说，如果 M 中任意两个元素等价，而 M 中任意元素与 M 外任意元素不等价，则 M 就是一个等价类。

例如，上面提到的，所有面积相等的几何图形就组成一个等价类（在面积相等关系下），所有男人就组成一个等价类（在同性关系下）。

1.2.3 部分序关系

定义 1.16 设 R 是集合 A 上的一个关系。如果 R 具有自反性，反对称性，传递性，则称 R 为一个部分序关系（或称半序关系）。集合 A 在部分序关系 R 下做成一个部分序集。

显然，一个部分序集的子集仍为部分序集。

例如，集合中的包含关系就是一个部分序关系，由一些集合做元素而做成的集合，在集合的包含关系下是一个部分序集。

通常，将部分序关系 R 写做≤，读做“小于或等于”。

定义 1.17 一个部分序集 A（其部分序关系为≤）说是一个全序集，如果对 A 中任意两个元素 a，b，必有 a≤b，或者 b≤a，全序集有时也称为链。

显然，全序集的子集仍为全序集。

例如，数的集合在数的大小关系下做成一个全序集。

定义 1.18 设 R 是集合 A 上的一个关系。如果 R 具有反自反性，传递性，则称 R 为一个拟序关系。记为＜，读做“小于”。

定义 1.19 设 A 是一个部分序集，其部分序关系为≤。

（1）如果 A 中有一个元素 a，对于所有的 x∈A，都有 x≤a(a≤x),则称 a 为集合 A 的最大（最小）元。

（2）A 中元素 a 说是一个极大（极小）元，如果除 a 之外，A 中没有元素 x，使得 a≤x(x≤a)。

(3) 对于 A 中的子集 M，A 中元素 a 称为子集 M 的一个上界（下界），如果对 M 中任意元素 m，都有 m≤a(a≤m)。M 的上界（下界）未必在 M 中，甚至 M 未必有上界（下界）。

(4) 对于 A 中的子集 M，A 中元素 a 称为 M 的一个最小上界（或称上确界），如果 a 是 M 的一个上界，并且对 M 的任意一个上界 x，都有 a≤x。

同理，可定义 M 的最大下界（或称下确界）。

注意，最大元，最小元未必存在，如果存在必唯一；极大元，极小元对有限部分序集必存在，但未必唯一。上下界未必存在，存在时又未必唯一，即使有上下界时，最小上界和最大下界也未必存在。

1.3 映　射

定义 1.20 设 A，B 是两个集合，若对 A 的每个元素 a，规定了 B 的一个确定元素 b 与之对应，则称此对应为由 A 到 B 内的一个映射。

将此映射记为 σ，于是对任意 a∈A，σ (a)表示 B 中与 a 对应之元素，称为 a 的映象，a 称为 b 的原象。

定义 1.21 设 σ 是 A 到 B 内的映射，如果 B 中每一个元素都一定是 A 中某元素的映象，就称 σ 是 A 到 B 上的映射（满射）。特别，A 到 A 上的映射，称为变换。

定义 1.22 设 σ 是 A 到 B 内的映射，如果对任意 a∈A，b∈A 且，a≠b，都有

σ （a）≠σ （b），就称 σ 是 A 到 B 的单射。

映射是数学上一个基本概念，集合A上的一个二元关系R就可看做是如下一个从集合A×A到集合{是，非}上的映射 σ；对任意a_1,a_2(都属于A)。

σ ((a_1,a_2))＝是，当且仅当（a_1,a_2）∈R。

定义 1.23 设 σ 是集合 A 到集合 B 内的映射。如果 σ 既是 A 到 B 的满射，又是 A 到 B 的单射，则称 σ 为 A 到 B 的 1－1 映射（双射）。

若 σ 是集合A到集合B的 1－1 映射，则对于B中每个元素b都对应着A中以b为映象（在 σ 下）的那个元素，这个对应显然是集合B到A上的映射，我们称这个映射为 σ 的逆映射，记为 σ^{-1}。显然，σ^{-1}也是 1－1 映射，并且对任意a∈A，都有

$$\sigma^{-1}(\sigma(a))=a$$

定义 1.24 设σ是集合A到集合B内的映射，τ是集合B到集合C内的映射，对任意a∈A，规定

$$(\tau \bullet \sigma)(a) = \tau(\sigma(a))$$

显然τ • σ是集合A到集合C内的映射，我们称此映射为映射τ与映射σ的乘积。

不难证明：映射的乘积满足结合律，但是不满足交换律。

定义 1.25 一个集合，如果它的元素为有限个，或者它与自然数集合之间存在一个1－1映射，则称此集合为可数集合。否则称该集合为不可数集合。

元素个数不是有限的可数集合称为可数无穷集合，对于可数无穷集合，可以把它的元素编号：

$$a_1, a_2, \cdots, a_n, \cdots \qquad (1)$$

前面我们讨论的无穷集合都是可数集合，并且知道了在数轴上稠密的有理数集合也可以与自然数集合建立1-1对应关系，那么是不是所有的无穷集合都是可数集合呢？

定理 1.6 全体实数做成的集合是不可数集合。

证明：由定理1.3.2知，只要证明（0，1）区间内的实数不可数就可以了。

若不然，我们可以把（0，1）区间内的数排成一个序列：

$$\left.\begin{array}{l} 0.a_{11}a_{12}a_{13}\cdots \\ 0.a_{21}a_{22}a_{23}\cdots \\ 0.a_{31}a_{32}a_{33}\cdots \end{array}\right\} \qquad (2)$$

$$\vdots$$

我们考虑下面的数：

$$0.r_1r_2\cdots r_k\cdots \qquad (3)$$

其中

$$r_k=\begin{cases} 1, & \text{当} a_{kk}\neq 1 \\ 2, & \text{当} a_{kk}=1, \ k=1, 2, \cdots \end{cases}$$

显然，（3）是（0，1）区间内的数，但它却不是序列（2）中的任一个数。事实上，对（2）中任一个数$0.a_{k1}a_{k2}\cdots a_{kk}\cdots$，因为$r_k\neq a_{kk}$,故

$$0.a_{k1}a_{k2}\cdots a_{kk}\cdots \neq 0.r_1r_2\cdots r_k\cdots$$

与假设矛盾。故（0，1）区间内的实数不可数，所以实数集不可数。

上述定理的证明方法，就是著名的“康托尔对角线法”，该方法在可计算理论中有广泛的应用。

1.4 定义、定理及其基本的证明技术

本节主要介绍什么是定义、定理及其关系，计算理论证明所使用的归纳法、、鸽巢原理、构造性证明、反证法以及三角法的简单介绍。

定理和证明是数学的精髓，而定义是它的灵魂。这三部分是包括本科目在内的每一个数学科目的核心。定义描述我们使用的对象和概念。一个定义可能是简单的，如前面给出的集合的定义；也可能是复杂的，如密码系统中安全的定义。任何数学定义必须是精确的。当定义一个对象时，必须弄清楚是什么构成这个对象、什么不构成这个对象。

在定义各种对象和概念之后，通常要给出关于它们的数学命题。典型情况下，一个命题表述某个对象具有某种性质。命题可能为真、也可能为假，但和定义一样必须是精确的。它的含义不能有任何摸棱两可的地方。

证明是一种逻辑论证，它使人们确信一个命题是真的。在数学中，一个论证必须是无懈可击的，也就是说，是令人绝对信服的。它和我们在日常生活中或在法律上使用的证据这一概念有相当大的区别。在谋杀案审判中要求“没有任何合理的疑点”的证据。重要的证据可以迫使陪审团接受嫌疑犯无罪和有罪的辩护。然而，这种证据在数学证明中不起作用。数学家需要没有任何疑点的证明。

定理是被证明为真的数学命题。通常只对特别感兴趣的命题使用这个词。有时，有兴趣证明某些命题只是因为他们有助于证明另一个更有意义的命题，这样的命题叫做引理。有时，一个定理或它的证明可以使我们容易得出另一些有关的命题为真的结论，这些命题叫做这个定理的推论。

确定一个数学命题真假的唯一办法是给出数学证明。不幸的是，找到数学证明并不总是容易的。不可能把它化简成一组简单的规则和过程。在这门课程中，会要求读者给出各种命题的证明，请你不要对此丧失信心！虽然任何人都没有制造证明的现成处方，而且证明是各不相同的，因为每一个证明是为了确认各自不同的结果。但是和下棋、打牌一样，通过大量的观察发现可以找到一些能够利用的模式、经验与技巧。下面是一些有用的一般性的证明策略可供选择，它们以各种不同的形式出现在许多证明中。

1.4.1 归纳法

归纳法是证明无穷集合的所有元素都有一种特殊性质的常用方法，例如，可以用归纳法证明一个算术表达式对于它的变量的每一组赋值都计算出想要的值，或证明一个程序在每一步或多所有的输入都工作正确。为了说明归纳法的使用过程，我们用一个例子说明。

数学归纳法的原理　如果 A 是一个自然数的集合使得

（1）　0∈A，

（2）　对于每一个自然数 n，{0，1，2，…，n}⊆蕴涵 n+1∈A，

则 A=N。

简单地说就是，任意一个包含 0 的自然数集合，如果它具有下述性质：它只要包含所有小于等于 n 的自然数就包含 n+1，则它实际上就是全体自然数的集合。

直观上这个原理是显然的：对于每一个自然数，从 0 开始，每次加 1，在有限步内能够到达它，故每一个自然数都在 A 中。在实践中，归纳法由两部分组成：归纳步骤和归纳基

础，每一部分自身是一个单独的证明，用来证明下述形式的断言：对于每一个 n≥1，如果P(n)为真，则 P(n+1)也为真，归纳基础证明 P(1)为真。只要证明了这两部分，就完成了证明。为什么呢？首先，知道 P(1)为真，因为归纳基础单独证明了它；其次，归纳步骤证明如果P(1)为真则 P(2)为真，而已经知道 P(1)为真，所以 P(2)为真论；最后，归纳步骤证明如果 P(2)为真则 P(3)为真，而已经知道 P(2)为真，所以 P(3)为真，这个过程对所有自然数可以一直下去。

1.4.2 构造法

许多定理说明存在一种特定类型的对象。证明这种定理的一个方法是说如何构造这样的对象，这种技术就是构造性证明。

让我们用构造法证明下述结论。

若一个简单图 G 的任意两点度数之和≥n-1，n=|P(G)|，则该图有 Hamilton 路。

分析：这个题如果使用一般方法证明会很复杂，那么我们是否可以利用该图的特点通过扩充构造出一个新图，从而使问题的证明简化呢？答案是肯定的。

证明：设$P(G)=\{v_1,v_2,\ldots,v_n\}$，首先在该图之外增加一点v，并把该点与G中每个点$v_1,v_2,\ldots,v_n$之间增加一条边，得到一个新图G’，显然G’有n+1 个节点且任意两点度数之和≥n+1，则G’的闭合图C(G’)的完全图，因此G’是 Hamilton图。故G’中存在一条Hamilton回路（$v, v_{i1},v_{i2},\ldots,v_{in}$），显然点v和$v_1,v_2,\ldots,v_n$都在此回路上，把v从此回路中删除得到图G，$v_1,v_2,\ldots,v_n$仍有一条Hamilton路（$v_{i1},v_{i2},\ldots,v_{in}$）相连，证毕。

1.4.3 反证法

证明定理的一种有效办法是，假设这个定理为假，然后证明这个假设导致一个矛盾的结果，也是日常生活中经常使用的推理方法。

如前面定理 1.3.2 的证明过程就是采用的反证法，为了证明定理结论成立，首先假设它不成立，然后构造一个反例子集合 B，推出矛盾，简单来说就是否定之否定等于肯定。

1.4.4 鸽巢原理

设 A 和 B 是两个有穷集合且|A|>|B|，则不存在从 A 到 B 的一一映射。换句话说，如果我们企图把 A 的元素（“鸽子”）与 B 的元素（“鸽巢”）配对，迟早不得不把一只以上鸽子放入一个巢里。

证明；基本步骤，假设|B|=0，即 B=ϕ，则没有任何从 A 到 B 的映射，更不用说一一映射。

采用归纳法。假设|A|>|B|，|B|≤n和 $f: A\rightarrow B$，这里n≥0，则f不是一一映射。设$f: A\rightarrow B$，|A|>|B|=n+1。取一个a∈A(因为|A|>|B|=n+1≥1，A非空，故总能取到)。如果有A的另一个元素a’使得f(a)=f(a’)，则显然f不是一一的，证明结束。因此，假设 a是被f映射到f(a) 的唯一元素。现在考虑集合A-{a}和B-{f(a)}，以及从A-{a}到B-{f(a)}的映射g。在A-{a}上g与f相同。因为B-{f(a)}有n个元素，|A-{a}|=|A|-1>|B|-1=|B-{f(a)}|。根据归纳假设，A-{a}有两个不同的元素被g（因而也被f）映射到B-{f(a)} 中的同一个元素，从而f不是一一的，证毕。

在很多各种各样的证明中要用到这个简单的事实。这里只是给出一个结论，在后面的各章当中会经常使用。

a) 5 对角化原理

最后，我们介绍对角线原理，虽然这个原理不象前面讨论的几个证明方法在数学中使用的那样广泛，但是它特别适合证明计算理论中的某些重要定理。

设R是集合A上的二元关系，D={a|a∈A且(a,a)∉R}称作R的对角线集合。对于每个a∈A，令R_a={b|b∈A且(a,b)∈R}，则D与每一个R_a都不同。

如果A是有穷集合，则可以把R画成一个正方形阵列。用A的元素标记行和列；正好当(a,b)∈R时，在行标记a和列标记b的方格内画一个叉。对角线集合D对应主对角线上的方格序列的补，把有叉的换成没有叉的，把没有叉的换成有叉的。集合R_a对应方阵的行。于是，可以把对角线原理叙述成：对角线的补与每一行都不同。

例子 考虑关系

R={(a,b),(a,d),(b,b),(b,c),(c,c),(d,b),(d,c),(d,e),(d,f),(e,e),(e,f),(f,a),(f,c),(f,d),(f,e)}。注意：R_a={b,d},R_b={b,c},R_c={c},R_d={b,c,e,f},R_e={e,f}以及R_f={a,c,d,e}。总之，可以把R画成下图。

	a	b	c	d	e	f
a		×		×		
b		×	×			
c			×			
d		×	×		×	×
e					×	×
f	×		×	×	×	

图 1

对角线上的方格序列是

	×	×		×	

它的补是

×			×		×

这对应对角线集合 D={a,d,f}。D 确实与方阵的每一行都不同。根据 D 的构造，它与第一行在第一个位置不同，与第二行在第二个位置不同，……。

对角化原理对于无穷集合也成立。理由是相同的：关于a是否是元素的问题，对角线集合与集合R_a总是不同的。因此对于任意的a，D不可能与R_a相同。定理 1.3.1 的证明就是对角线法的具体应用。下面再通过Cantor的一个经典的定理的证明说明对角线法的使用。

试证明集合 2^N是不可数的。

证明：假设 2^N是可数的，即能够把 2^N的所有元素枚举成

$$2^N=\{R_0, R_1, R_2,\ldots\}$$

（注意，考虑关系R={(i,j)|j∈R_i}，这些R_i是对角线原理叙述中的集合R_a。）现在考虑集合

$$D=\{n\ |n\in N且n\notin R_n\}$$

（D是对角线集合）D是一个自然数的集合，因此应该出现在枚举{R_0, R_1, R_2,…}中。但是D不可能是R_0，因为D与R_0在是否包含 0 的问题上不同（D包含 0 当且仅当R_0不包含 0）；D不可能是R_1，因为D与R_1在是否包含 1 的问题上不同，等等。矛盾。

1.5 字符串与语言

字符串是计算机科学中基本的建筑块。根据应用的要求，可以在各种各样的字母表上定义这样的串。根据我们的要求，字母表是称作符号的对象的有穷非空集合Σ(或Γ)。Σ中的符号的n元组称作Σ上的字或字符串。下面是字母表的几个例子。Σ_1={0,1}，Σ_2={a, b, c, d, e, f, g}，Γ={0, 1, x, y, z}。字母表上的字符串是该字母表中符号的有穷序列，通常写成一个符号挨着一个符号，不用逗号隔开。设Σ_1={0,1}，则 001010 是Σ_1上的一个字符串，设Σ_2={a, b, c, d, e, f, g}，则aabcdf是Σ_2上的一个字符串。设w是Σ上的一个字符串，如果$w=a_1a_2\ldots a_n$，则称w的长度为n，且记作$|w|=n$，这里每一个$a_i\in\Sigma$。长度为 0 的字符串叫做空串，记为ε。空串起着 0 在一个数系中的作用。如果字符串u反复出现在w中，则称u 为w的子串。例如，1010 是 001010 的子串。把唯一的零字写作 0，它的长度是 0。把字母表A上的所有字的集合记作A*。A*的任何子集称为A上的语言或字母表为A的语言。

第二部分 可计算性

第二章 可计算函数

2.1 程序设计语言

我们首先引进一个 FORTRAN 式的极简单的程序设计语言。语言包括赋值语句，条件转移和无条件转移语句。语句可带标号也可不带标号。

用带或不带下标的小写字母表示变元。例如：

x，y，z，x_1，…，z_1，z_2，…

等等。假定这些变元取非负整数值，在讲义中凡是提到数就指非负整数。

标号一般地用下列大写字母表示：

A，B，C，D，E，…，A_1，A_2，B_1，B_2，…

等等。

虽然我们的语言是极为简单的，但我们将看到它足以描述任何复杂的计算过程，而且指令表中的后二条也是可节省的。于是从理论上说只要三条指令就可以了。

指令(语句)结构表

指 令	说 明
x=x+1	变元 x 的值增加 1
x=x-1	变元 x 的值减 1。若 x 的值为 0，则结果仍为 0
TO A IF x≠0	若 x≠0，则转标号为 A 的指令；否则执行下一条指令
TO A	无条件转到标号为 A 的指令
y=x	把 x 的值赋给变元 y，x 值保持不变

根据定义，下列语句都是非法的

1 y=x+1

2 x=x+2

3 TO A IF x=0

4 TO A IF x≠y

5 x=2×x+1

就是说只允许指令表中所规定形式的语句。这就表明我们的语言实在太简单了。

程序中变元按其作用可分为三类：一是在执行程序前必须通过外部的方法给值的变元，这种变元称为程序的输入变元或自由变元；一是表示计算结果的变元，称这种变元为程序的输出变元，因函数值是一个整数，因此一个程序过程只有一个输出变元；一是用于存放临时值的变元，称这种变元为程序的临时变元或工作变元。

在本书中约定：

1 用 x 表示输入变元。例如：

x_0，x_1，x_2，x_3，……

2 用 z 表示临时变元。例如：

z_0，z_1，z_2，z_3，……

3 用 y 表示输出变元。

另外，对程序特做如下的规定：

1 当程序开始执行时自动认为一切变元(输入变元除外)的值为 0。

2 当程序出现下列两种情况之一时自动认为停机：

a) 转向无定义的标号。

b) 执行了程序的最后一条指令，但没有下一条指令。

下面考虑一些程序例子。

例 2.1.1 计算 y=x+3

程序

```
x=x+1
x=x+1
x=x+1
y=x
```

在这里 x 是输入变元，在执行程序前必须赋给它一个值。当执行完最后一条时自动停机。

例 2.1.2 计算零函数 n(x)=0

程序

```
x=x+1
```

在这里 x=x+1 不起任何计算作用，只是表明程序有一个输入变元 x。当程序结束时，因为 y 的初值本来就是 0，而后又没改变，因此计算结果自然 y=0。

程序亦可写成

```
x1=x1+1
x2=x2+1
```

这个程序计算结果也是y=0，但它有两个变元x_1和x_2。它所计算的函数是$n(x_1，x_2)=0$。

例 **2.1.3** 计算$y=x_1+x_2$

程序

```
y=x1
［B］TO A IF x2≠0
      TO E
［A］x2=x2-1
       y=y+1
         TO B
```

例 2.1.4 计算 y=2x

程序：

```
        TO C IF x≠0
        TO E
［C］  TO A IF x≠0
［B］  y=y+1
        z=z-1
        TO B IF z≠0
        TO E
［A］x=x-1
       y=y+1
       z=z+1
       TO C
```

每执行一条指令要改变变元的值。如果变元的集合视作内存，则程序的执行过程也就是不断改变内存状态的过程。

我们看一下上面的程序。该程序的内存可视为(x，y，z)，下面我们来考察一下当 x=3 时内存的变化情况。内存的初值是

(3，0，0)

以后的变化过程如下：

x=3	x=2	x=1	x=0	x=0	x=0	x=0
y=0	y=1	y=2	y=3	y=4	y=5	y=6
z=0	z=1	z=2	z=3	z=2	z=1	z=0

例 2.1.5 计算$y=x_1 \bullet x_2$

现在我们可以把上述程序做为宏指令来使用了。比如，可直接用$y=x_1+x_2$，y=2x等宏指令。$y=x_1 \bullet x_2$的程序如下。

```
［B］  TO A IF x2≠0
        TO E
［A］x2=x2-1
       y=y+x1
       TO B
```

2.2 无条件转移和赋值的消去

我们将说明下面两种指令是可消去的：

```
TO  A
```

```
y=x
```

就是说有它们也并不增加功能，因此可视为宏指令。事实上，它们分别表示下面的指令组：

TO A：

```
z=z+1
TO A IF z≠0
```

y=x的展开稍微复杂些，算法的思想是把x复写到y_1，当复写完时x=0。再把y_1复写到y中，复写完时x值也被恢复。

下面是把x复写到y_1的程序。

```
［B1］  TO A1 IF x≠0
        TO E
［A1］  x=x-1
        y1=y1+1
        TO B1
```

把y_1复写到y，并恢复x值的程序部分如下：

```
［B2］  TO A2 IF y1≠0
        TO E
［A2］  y1=y1-1
        y=y+1
        x=x+1
        TO B2
```

在它执行时x=0，y=0，y_1=x。

总的程序是：

```
[B1] TO A1    IF x≠0
TO B2
[A1] x=x-1
     y1=y1+1
TO B1
[B2] TO A2 IF y1≠0
TO E
[A2] y1=y1-1
y=y+1
x=x+1
TO B2
```

定义 2.2.1 若程序 P 恰有 n 个输入变元

x_1，x_2，…，x_n

而没有其它 x 变元，则称 P 为 n 元程序。

定义 2.2.2 设 P 是 n 元程序，则变元的下列状态称为程序 P 的初始状态：

$$\begin{cases} x_1=a_1 \\ x_2=a_2 \\ \cdots\cdots \\ x_n=a_n \end{cases} \qquad \begin{cases} y=0 \\ z_1=0 \\ \cdots\cdots\cdots \\ z_n=0 \end{cases}$$

定义 2.2.3 程序的第一个语句称为初始语句。

任何程序都从初始语句开始执行，这时变元处于某初始状态。

2.3 可计算函数

在这一节将引进整个讲义的最基本概念－－可计算函数。因为以后还会提到别的语言，所以把前面所介绍的语言称为原语言。

设 P 为一 n 元程序，则对每组输入值

$x_1=a_1, x_2=a_2, \ldots x_n=a_n$

都计算出一个值 y，因此可使每一个程序 P 对应一个函数。不同的程序可对应于同一函数。

设P是以$x_1,x_2,\ldots x_n$为输入变量的n元程序，则将用

$\varphi_p(x_1, x_2, \ldots, x_n)$

表示程序 P 所对应的函数。具体如下：

$$\varphi_p(a_1, a_2, \ldots, a_n)=\begin{cases} b，若程序P对输入值a_1, a_2, \ldots, a_n停机且y=b。 \\ 无定义，若程序P对于输入值a_1, a_2, \ldots, a_n不停机 \end{cases}$$

例 2.3.1 程序 P1：

```
［B］  x=x+1
       TO B
```

从程序可以看出，对于任意输入值a，P_1不停机，因此$\varphi_{p1}(x)$是无定义函数。

程序 P2：

```
[B] x1=x1+1
    x2=x2+1
    TO   B
```

因此$\varphi_{p2}(x_1，x_2)$是无定义函数。

上面两个程序是处处无定义的一元和二元函数。

程序 P3：

```
[B] TO A IF x2≠0
    TO E
[A] x2=x2-1
    y=y+x1
    TO B
```

因此$\varphi_{p3}(x_1，x_2)= x_1 \bullet x_2$是一个处处有定义的函数。

程序 P4：

```
[C]   To   A IF x2≠0
      y=x1
      TO E
[A]   TO B IF x1≠0
      TO A
[B]   x1=x1-1
      x2=x2-1
      TO C
```

因此$\varphi_{p4}(x_1，x_2)= x_1 \ominus x_2$是一个函数其中 $x_1 \ominus x_2=\begin{cases} x_1-x_2，x_1\geq x_2 \\ 无定义，\ 若x_1< x_2 \end{cases}$

定义 2.3.1 函数$f(x_1，x_2，…，x_n)$被称为部分可计算的，若有一程序P使得

$\varphi_p(x_1，x_2，…，x_n)= f(x_1，x_2，…，x_n)$

这里等号“=”表示：或者两边都无定义，或者两边都有定义并且其值相同。

定义 2.3.2 函数$f(x_1，x_2，…，x_n)$被称为全函数，若它对一切$x_1，x_2，…，x_n$的值都有定义。

定义 2.3.3 函数$f(x_1，x_2，…，x_n)$被称为可计算函数，若它是部分可计算的而且是全函数。

可计算函数不只要求有计算它的程序，且要求永远有定义。

可计算函数是部分可计算函数的特殊情形。

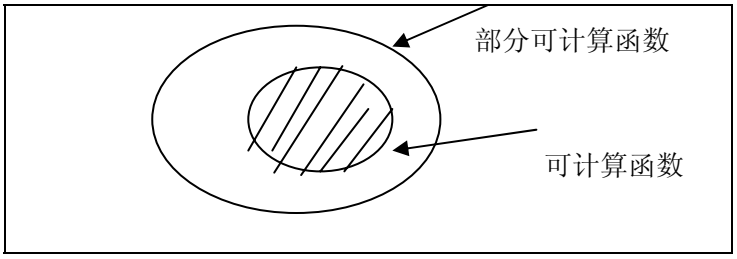


图 2.1

在整个讲义中可计算函数和部分可计算函数是基本概念。

程序是定义的一种方式，现在我们暂时离开定义函数的程序方式，而是考虑另一种定义方式，即递归方式。为此，我们请看下一章递归函数。

习 题

用原语言程序(只准用基本指令)证明下列函数是可计算的

1 x^y。

2 $[\sqrt{X}]$。（向下取整）

第三章 递归函数

3.1 复合递归与取极小算子

我们要研究一些函数类，这些函数是从一初始函数出发，通过几种方式构造出来的。这些方式包括：复合，递归，取极小值。

假设有两个函数

$y=f(x)$

$z=g(x)$

则由它们可构造出一新函数

$y=f(g(x))$

我们说函数 $y=f(g(x))$是复合算子作用于函数 $y=f(x)$和 $z=g(x)$的结果。在这里并不要求 f 和 g 为全函数，只要 g 对 x 有定义，f 对 g(x)有定义即可。

一般情形与此相仿，只是多了变元。

定义 3.1.1 假设有函数

$y=h(z_1,z_2,\ldots,z_m)$

和函数

$z_1=g_1(x_1，x_2，\ldots，x_n)$

$z_2=g_2(x_1，x_2，\ldots，x_n)$

：

：

$z_m=g_m(x_1，x_2，\ldots，x_n)$

则(m 元)函数

$y=h(z_1,z_2,\ldots,z_m)$

$=h(g_1(x_1，x_2，\ldots，x_n)，\ldots，g_m(x_1，x_2，\ldots，x_n))$

被称为函数f和$g_1，\ldots，g_m$的复合函数。

再看递归算子，先看一元情形：

$$\begin{cases} h(0)=m \\ h(t+1)=\varphi(h(t)，t) \end{cases}$$

这时我们说函数 h 是由φ和 m 递归定义的，例如，假定有

$$\begin{cases} h(0)=0 \\ h(t+1)=\varphi(h(t)，t) \end{cases}$$

其中φ(h(t)，t)表示 h(t)+t，则函数 h 的计算过程如下：

h(3)=h(2)+2

=h(1)+1+2

=h(0)+0+1+2

=0+0+1+2=3

对一般情形我们有下面定义。

定义 3.1.2 设$m(x_1，x_2，…，x_n)$和$\varphi(x_1，x_2，…，x_{n+2})$是全函数。我们定义

$$\begin{cases} h(x_1，x_2，…，x_n，0)= m(x_1，x_2，…，x_n) \\ h(x_1，x_2，…，x_n，t+1)= \varphi(x_1，x_2，…，x_n，h(x_1，x_2，…，x_n，t)，t) \end{cases}$$

这时我们说 h 是递归算子作用于函数 m 和φ的结果。

注意：我们只对全函数 m 和φ施行递归算子。显然，结果函数 h 亦为全函数。

下面给出极小算子的概念。

定义 3.1.3 设$f(x_1，x_2，…，x_n，z)$为全函数，我们定义$h(x_1，x_2，…，x_n)=\min\{z|f(x_1，x_2，…，x_n，z)=0\}$，这时我们说函数h是取极小算子作用函数f的结果。

n+1 元的全函数通过min算子可定义出一个n元函数。上述定义表示：$h(x_1，x_2，…，x_n)$的值是使$f(x_1，x_2，…，x_n，z)=0$ 的最小z值，如果不存在使得$f(x_1，x_2，…，x_n，z)=0$ 的z，则$h(x_1，x_2，…，x_n)$无定义。

和递归算子一样，在这里也要求 f 是全函数。但取极小后的函数 h 不一定是全函数。

定义 3.1.4 函数$f(x_1，x_2，…，x_{n+1})$被称为正则的，若对任何一组$x_1，x_2，…，x_n$，都有z使

$f(x_1，x_2，…，x_n，z)=0$

由定义可知，对于正则函数取极小算子结果得全函数。

3.2 原始递归函数

定义 3.2.1 下列函数定义为初始函数：

S(x)=x+1	后继函数
n(x)=0	零函数
$\bigcup_i^n$ $(x_1, x_2, \ldots, x_n)=x_i$	投影函数(1≤i≤n)

定义 3.2.2 由初始函数

S(x)，n(x)，$\bigcup_i^n$ $(x_1, x_2, \ldots, x_n)$ (1≤i≤n)

出发，只用复合和递归算子得到的函数称为原始递归函数。它们都是全函数。

下面给出原始递归函数的例子

1 加法函数 x+y

要说明它是原始递归的，只要表明它是由初始函数经复合和递归算子而得。

令 add(x,y)=x+y 显然有

$$\begin{cases} add(x, 0)=x \\ add(x, y+1)=add(x, y)+1 \end{cases}$$

具体可写成如下

$add(x,0)= \bigcup_1^1(x)$

$add(x,y+1)=S(\bigcup_2^3(x, add(x,y), y))$

其中$\bigcup_1^1(x)$，$\bigcup_2^3(x_1, x_2, x_3)$和 S(x)是初始函数，故为原始递归函数。$S(\bigcup_2^3(x_1, x_2, x_3))$为原始递归函数的复合，故亦为原始递归函数。再用递归算子得到 add，故 add 为原始递归函数。

2 乘法函数 x•y

令 mul(x，y)=x•y，则其递归式可写成如下：

$$\begin{cases} mul(x, 0)=0 \\ mul(x, y+1)=mul(x, y)+x=add(\bigcup_2^3(x, mul(x,y), y), \bigcup_1^3(x, mul(x,y), y)) \end{cases}$$

进一步可写成

$$\begin{cases} mul(x, 0)=n(x) \\ mul(x, y+1)= add(\bigcup_2^3(x, mul(x,y), y), \bigcup_1^3(x, mul(x,y), y)) \end{cases}$$

其中 n(x)为零函数。

n(x)，$\bigcup_2^3(x_1,x_2,x_3)$，$\bigcup_1^3(x_1,x_2,x_3)$，add(x_1，x_2)

是原始递归函数，故

add($\bigcup_2^3(x_1,x_2,x_3)$，$\bigcup_1^3(x_1,x_2,x_3)$)

是原始递归函数。最后，由于上面式子是合法的递归式子，故 mul(x，y)是原始递归函数。

以后用更简单的形式来显示这一函数是原始递归的。

3 阶乘函数 x!

令 fac(x)=x!递归式如下：

$$\begin{cases} fac(0)=1 \\ fac(x+1)=mul(fac(x),s(x)) \end{cases}$$

4 指数函数 x^y

令exp(x，y)=x^y。递归定义如下：

$$\begin{cases} exp(x，0)=1 \\ exp(x，y+1)=mul(exp(x，y)，x) \end{cases}$$

5 前驱函数 P(x)

这是前驱函数。对应于指令 x=x-1。若 x≠0,则 p(x)=x-1，否则 p(0)=0，递归式如下：

$$\begin{cases} p(0)=0 \\ p(x+1)=x \end{cases}$$

6 点减函数 x÷y

令 sub(x,y)=x÷y。x>=y 时,sub(x,y)取 x-y，否则取 0。递归式如下：

$$\begin{cases} sub(x，0)=x \\ sub(x，y+1)=p(sub(x，y)) \end{cases}$$

7 |x-y|

由下式得证

|x-y|=(x÷y)+(y÷x)

8 α函数 α(x)

函数α(x)的定义如下：

$$\alpha(x)=\begin{cases} 1，若\ x=0 \\ 0，若\ x\neq 0 \end{cases}$$

因为α(x)=1∸x，故它是原始递归函数。

9 n+1 元求和函数 $\sum_{t=0}^{y} f(x_1,...,x_n,t)$

令h(x_1，x_2，…，x_n，y)= $\sum_{t=0}^{y} f(x_1,...,x_n,t)$，则其递归式可写成如下：

$$\begin{cases} h(x_1，x_2，…，x_n，0)= f(x_1，x_2，…，x_n，0) \\ h(x_1，x_2，…，x_n，y+1)=add(h(x_1，x_2，…，x_n，y)，f(x_1，x_2，…，x_n，y+1)) \end{cases}$$

因此如果f(x_1，x_2，…，x_n，t)是原始递归函数，则h(x_1，x_2，…，x_n，y)亦为原始递归函数。

10 n+1 元求积函数 $\prod_{t=0}^{y} f(x_1,...,x_n,t)$

若令

g(x_1，x_2，…，x_n，y)= $\prod_{t=0}^{y} f(x_1,...,x_n,t)$ 则可写成如下：.

g(x_1，x_2，…，x_n，0)= f(x_1，x_2，…，x_n，0)，

g(x_1，x_2，…，x_n，y+1)=mul(g(x_1，x_2，…，x_n，y)，f(x_1，x_2，…，x_n，y+1)

因此如果f(x_1，x_2，…，x_n，t)是原始递归函数，则g(x_1，x_2，…，x_n，y)是原始递归函数。

不难证明下面函数也是原始递归函数：

g_1(x_1，x_2，…，x_n，y)= $\sum_{t=1}^{y} f(x_1,...,x_n,t)$

g_2(x_1，x_2，…，x_n，y)= $\prod_{t=1}^{y} f(x_1,...,x_n,t)$ 事实上，我们有

g_1(x_1，x_2，…，x_n，y)= $\sum_{t=0}^{y\dot{-}1} f(x_1,...,x_n,t+1)$

g_2(x_1，x_2，…，x_n，y)= $\prod_{t=0}^{y\dot{-}1} f(x_1,...,x_n,t+1)$

11 判断函数 d(x，y)

d(x，y)这样一个函数，如果 x=y，则其值为 0，否则其值为 1。即有

$$d(x，y)=\begin{cases} 0，若\ x=y \\ \end{cases}$$

1，若 $x \neq y$

从下式可知 d(x，y)是原始递归函数

$d(x，y)=\alpha(\alpha(|x-y|))$

进一步的更有用的原始递归函数将在后面介绍。其中用到原始递归谓词和受囿取极小算子。

3.3 原始递归谓词和原始递归集合

为了简化问题，这里引入数理逻辑知识以简化模型。

定义 3.3.1 设有谓词$P(x_1，x_2，…，x_n)$，则可定义函数

$$\delta(x_1，x_2，…，x_n)=\begin{cases}0，当P(x_1，x_2，…，x_n)时\\ 1，当\sim P(x_1，x_2，…，x_n)时\end{cases}$$

称$\delta(x_1，x_2，…，x_n)$为谓词$P(x_1，x_2，…，x_n)$的特征函数。

对于集合 S 亦可定义其特征函数：

$$\delta(x_1，x_2，…，x_n)=\begin{cases}0，当\ (x_1，x_2，…，x_n)\in S时\\ 1，当\ (x_1，x_2，…，x_n)\notin S时\end{cases}$$

这样，对集合和谓词同样可以研究其可计算性和递归性。注意我们只考虑全谓词。

定义 3.3.2 说谓词 P(集合 S) 是原始递归的，如果其特征函数是原始递归的。

定义 3.3.3 说谓词 P(集合 S) 是可计算的，如果其特征函数是可计算的。

因为谓词是全的，所以不用部分可计算一词了。

定理 3.3.1 若$P(x_1，x_2，…，x_n)$是原始递归谓词，则$\sim P(x_1，x_2，…，x_n)$也是原始递归谓词。

证明：设δ和δ′分别为谓词 p 和～p 的特征函数。我们要证明δ是原始递归的，则δ′也是原始递归的。事实上，由下式立即得出结论：

$\delta'(x_1，x_2，…，x_n)=\alpha(\delta(x_1，x_2，…，x_n))$

其中α是上一节定义的函数。注意，我们今后一直使用δ_P表示P的特征函数。

定理 3.3.2 若$P(x_1，x_2，…，x_n)$和$Q(x_1，x_2，…，x_n)$是原始递归谓词。则$P(x_1，x_2，…，x_n)\vee Q(x_1，x_2，…，x_n)$是原始递归谓词。

证明： 设δ_P和δ_Q以及$\delta_{P\vee Q}$分别为上述三个谓词的特征函数。已知δ_P和δ_Q是原始递归函数，要

证明$\delta_{P\vee Q}$是原始递归函数。显然有

$\delta_{P\vee Q}(x_1，x_2，\ldots，x_n)=\delta_P(x_1，x_2，\ldots，x_n)\cdot\delta_Q(x_1，x_2，\ldots，x_n)$

因此$\delta_{P\vee Q}$是原始递归函数。

定理 3.3.3 设$P(x_1，x_2，\ldots，x_n)$和$Q(x_1，x_2，\ldots，x_n)$是原始递归谓词，则$P(x_1，x_2，\ldots，x_n)\wedge Q(x_1，x_2，\ldots，x_n)$也是原始递归谓词。

证明：从下式得证

$$\delta_{P\wedge Q}(x_1，x_2，\ldots，x_n)=\alpha(\alpha(\delta_P(x_1，x_2，\ldots，x_n)+\alpha(\delta_Q(x_1，x_2，\ldots，x_n)))$$

下面三个具体谓词是原始递归的。

12 等值函数 $\boxed{x=y}$

其特征函数是前面所定义的原始递归函数：

$$d(x，y)=\begin{cases}0，若\ x=y\\ 1，若\ x\neq y\end{cases}$$

13 $\boxed{x>y}$

其特征函数是$\alpha(x\dot{-}y)=$

$$\begin{cases}0，若\ x>y\\ 1，若\ x\leq y\end{cases}$$

它是原始递归函数。

14 $\boxed{x\leq y}$

$x\leq y\Leftrightarrow x<y\vee x=y$，故 $x\leq y$ 是原始递归谓词。

定理 3.3.4 设 P 为原始递归谓词，g 和 h 为原始递归函数，则下面函数 f 是原始递归的：

$$f(x_1，x_2，\ldots，x_n)=\begin{cases}g(x_1，x_2，\ldots，x_n)，当P(x_1，x_2，\ldots，x_n)时\\ h(x_1，x_2，\ldots，x_n)，当\sim P(x_1，x_2，\ldots，x_n)时\end{cases}$$

证明：显然 f 可写成如下

$$f=g\cdot\alpha(\delta_P)+h\cdot\delta_P$$

其中δ_P表示P的特征函数。从上式可知下面定理成立。

定理 3.3.5 若$P(t，x_1，x_2，\ldots，x_n)$是原始递归谓词，则

$(\exists t)_{\le y}$ P(t，x_1，x_2，…，x_n)

也是原始递归谓词。

证明：定理中的谓词将定义为以y，x_1，x_2，…，x_n为变元的n+1 元谓词。记它为Q(y，x_1，x_2，…，x_n)。要证明Q是原始递归的。用δ_P表示P的特征函数并令

$$\delta_Q(y，x_1，x_2，…，x_n)=\prod_{t=0}^{y}\delta_P(t,x_1,x_2,...,x_n)$$

下面证明δ_Q(y，x_1，x_2，…，x_n)是谓词Q (y，x_1，x_2，…，x_n)的特征函数。事实上，如果存在一个 $0\le t_0\le y$，使得P (t_0，x_1，x_2，…，x_n)取真值，则δ_P(t_0，x_1，x_2，…，x_n)=0，从而δ_Q(y，x_1，x_2，…，x_n)=0 。即当Q (y，x_1，x_2，…，x_n)取真值时有δ_Q(y，x_1，x_2，…，x_n)=0 。如果对一切 $0\le t\le y$，都使P(t，x_1，x_2，…，x_n) 取假值，则对一切 $0\le t\le y$，都有δ_P(t，x_1，x_2，…，x_n)=1。从而有δ_Q(y，x_1，x_2，…，x_n)=1。即当Q (y，x_1，x_2，…，x_n)取假值时有δ_Q(y，x_1，x_2，…，x_n)=1。综上所述，δ_Q(y，x_1，x_2，…，x_n)是Q (y，x_1，x_2，…，x_n)的特征函数，由δ_Q的式子不难知道δ_Q是一个原始递归函数。故 $(\exists t)_{\le y}$ P(t，x_1，x_2，…，x_n)是原始递归谓词。

定理 3.3.6 若P(t，x_1，x_2，…，x_n)是原始递归谓词，则

$(\forall t)_{\le y}$ P(t，x_1，x_2，…，x_n)

也是原始递归谓词。

证明：从下式得证

δ_Q (y，x_1，x_2，…，x_n)

$$=\alpha(\alpha(\sum_{t=0}^{y}\delta_P(t,x_1,...,x_n)))$$

其中δ_P和δ_Q分别为P和$(\forall t)_{\le y}$ P(t，x_1，x_2，…，x_n)的特征函数。

注意：在上述定理中我们用到的是受囿存在量词和受囿全称量词。

对于集合可以给出下面定理。

定理 3.3.7 设 S 和 R 是原始递归集合，则

$\bar{R}$，R，S，R∩S，R∪S

也是原始递归集合。

由前面定理可知，在原始递归谓词前面加一些个受囿量词后仍然是原始递归谓词，例如假定P(x_1,…,x_n)是原始递归谓词，则下面谓词也是原始递归谓词。

$(\exists x_1)_{\le y1}(\forall x_2)_{\le y2}$P($x_1$，$x_2$，…，$x_n$)

$(\forall x_1)_{\le y1}(\exists x_2)_{\le y2}P(x_1，x_2，\ldots，x_n)$

$(\forall x_1)_{\le y1}(\forall x_2)_{\le y2}P(x_1，x_2，\ldots，x_n)$，

$(\exists x_1)_{\le y1}(\exists x_2)_{\le y2}P(x_1，x_2，\ldots，x_n)$

定理 3.3.8 设$P(x_1，x_2，\ldots，x_n)$是原始递归谓词，$h_1，h_2，\ldots，h_n$是n个k元原始递归函数，则下面谓词也是原始递归的：

$P(h_1(x_1，x_2，\ldots，x_k)，\ldots，h_n(x_1，x_2，\ldots，x_k))$

证明：令δ'为所要证谓词的特征函数，δ_P为给定谓词P的特征函数，则有

$\delta'(x_1，x_2，\ldots，x_n)=\delta(h_1(x_1，x_2，\ldots，x_k)，\ldots，h_n(x_1，x_2，\ldots，x_k))$

其中$\delta，h_1，h_2，\ldots，h_n$均为原始递归函数，因此$\delta'$也是原始递归函数，故定理成立。

定理 3.3.9 设$f(x_1，x_2，\ldots，x_n)$和$g(x_1，x_2，\ldots，x_n)$是原始递归函数，则

$f(x_1，x_2，\ldots，x_n)=g(x_1，x_2，\ldots，x_n)$

也是原始递归谓词。

证明 我们已知 x=y 是原始递归谓词，因此由前一定理

$f(x_1，x_2，\ldots，x_n)=g(x_1，x_2，\ldots，x_n)$

也是原始递归谓词。

3.4 受囿取极小值

现看由谓词到函数的算子。它是称之为受囿取极小值的一种取极小值算子。

设$P(t，x_1，x_2，\ldots，x_n)$为谓词。我们定义一函数

$$\min_{t\le y}\{P(t，x_1，x_2，\ldots，x_n)\}$$

这是由谓词定义出来的函数。这个函数的具体定义如下：

$$\min_{t\le y}P(t,x_1,x_2，\ldots,x_n)=\begin{cases}\min\limits_{t}\{t\le y\wedge P(t，x_1，x_2，\ldots，x_n)\},若(\exists t)_{\le y}P(t，x_1，x_2，\ldots，x_n)\\ 0，否则\end{cases}$$

由定义可知上述函数是全函数。

由可计算性理论的角度看，当不存在小于等于 y 的 t，使得 P 为真时，对函数赋值为零是合理的。对不受囿取极小值来说是不合理的。关键之处在于对受囿来说只要看有穷多个情形即可，而对一般情形(不受囿)来说，要对无穷多个 t 进行试验。显然，不可能通过一个计

算过程来实现这一点。

定理 3.4.1 若P(t，x_1，x_2，…，x_n)是原始递归的谓词，则对它取受囿取极小后的函数

f(y，x_1，x_2，…，x_n)

$=\min\limits_{t\le y}$ {P(t，x_1，x_2，…，x_n)}

是原始递归函数。

证明 为简单起见用 $\vec{X}$ 表示x_1，x_2，…，x_n。假设有

$\delta_P(t_0, \vec{X})=\delta_P(1, \vec{X})=\ldots=\delta_P(t_0\text{-}1, \vec{X})=1$

$\delta_P(t_0, \vec{X})=0$

即t_0是第一个使P(t，$\vec{X}$)取真值的t值。假设δ是P(t，$\vec{X}$)的特征函数，并且考虑函数

$\prod\limits_{t=0}^{u}\delta_P(t,\overline{X})$。我们不难验证

$$\prod_{t=0}^{u}\delta_P(t,\overline{X})=\begin{cases}1, & \text{当}u<t_0\text{时}\\ 0, & \text{当}u\ge t_0\text{时}\end{cases}$$

因为当$u<t_0$时乘积中的每个因子取 1，而当$u\ge t_0$时在乘积中包含$\delta(t_0, \vec{X})$因子，其中$\delta_P(t_0, \vec{X})=0$。

不难验证

$$\sum_{u=0}^{y}(\prod_{t=0}^{u}\delta_P(t,\overline{X}))=t_0$$

而t_0为使P(t，$\vec{X}$)真的最小t值，因此有

$$f(y，x_1，\ldots，x_n)=\begin{cases}\sum\limits_{u=0}^{y}(\prod\limits_{t=0}^{u}\delta_P(t,\overline{X}))，\text{当}(\exists t)_{\le y}P(t,\overline{X})\text{时}\\ \\ 0，\text{否则}\end{cases}$$

其中$\delta_P(t, \vec{X})$，所以有P(t，x)，因此$(\exists t)_{\le y}P(t,\overline{X})$ ，$\sum\limits_{u=0}^{y}(\prod\limits_{t=0}^{u}\delta_P(t\ ,\overline{X}))$ 是原始递归的，

根据定理 3.3.4，因此f(y，x_1，…，x_n)是原始递归的。

下面给出出使用受囿极小算子和受囿量词的原始递归函数和原始递归谓词例子。

15 y|x

y|x 表示“y 整除 x”。例如

3|24 为真

3|22 为假

当 x=0 时，对于任何 y， y|x 都为真。

当 y=0 时，若 x≠0，则 y|x 为假。

⇔说明，若右式的特征函数为δ ，则左式也为δ。因为当左式为真时，右式也为真即δ为 0，反之δ为 1。

它是原始递归谓词。因为

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

16 [y/x]

[y/x]表示这样一个函数。它的值是用 x 除 y 所得的商，我们有：

$]y/x] = \min_{t \le y} \{x(t+1) > y)\}$

17 Prim(x)

这是一个谓词“x 是素数”我们有

$prim(x) \Leftrightarrow (x>1) \wedge (\forall t)_{\le x}(t=1 \vee t=x \vee \sim(t|x))$

18 P_i

P_i是函数。是P(i)的简写(这里P不是指前驱函数)。P(i) 的值就是第i个素数的值。具体如下：

P(0)=0 (约定)－－这和“0 不是质数”不是一回事。

$P_1=2$

$P_2=3$

$P_3=5$

……

可给出下面递归式

$$\begin{cases} P_0=0 \\ P_{i+1}= \min\limits_{t \le (P_i!+1)} \{prim(t) \wedge t > P_i\} \end{cases}$$

这就是说第 0 个素数是 0 且第 i+1 个素数是大于第 i 个素数 Pi 的最小素数。

我们必须证明在$P_i<t\le P_i!+1$ 中有素数。即$P_{i+1}\le P_i!+1$。如果$P_i!+1$ 是素数。那么自然就有$P_{i+1}<t\le P_i!+1$。现假设$P_i!+1$ 不是素数。这时它一定有素数因子。这个因子不会是P_0。因为

0 不是任何数的因子。P_1，P_2，…，P_i也不可能是因子，因为用它们去除$P_i!+1$ 后都是余数 1。所以$P_i!+1$ 的素数因子大于P_i，显然它小于$P_i!+1$。

19 R(x，y)

R(x，y)表示用 y 除 x 时的余数，有下面等式

$$x/y=[x/y]+\frac{R(x,y)}{y}$$

于是有

R(x，y)=x÷(y．[x/y])

当y=0 时R(x,y)=x 。

20 t(x)

t(x)表示这样一个函数：在x的素因子分解中非零指数的个数。t (0)定义为 0。例如，$20=2^2\cdot3^0\cdot5^1$

因此有t(20)=2

令 *l* h(i，x)为前 i-1 个素数中能整除 x 的素数个数。

l h(0，x)=0

$$l\,h(i+1,x)=\begin{cases} l\,h(i,x)+1, & \text{当}P_i|x\text{时} \\ l\,h(i,x), & \text{当}P_i\nmid x\text{时} \end{cases}$$

t(x)= *l* h(x，x)

21 $(x)_i$ =f(i，x)——x素因子分解中P_i的指数。

$(x)_i$表示x的素因子分解中的第i个指数，即P_i的指数。有下面式子

$$(x)_i=\begin{cases} \min\limits_{t\le x}\{t\,|\,P_i^t\,|\,x\wedge\sim P_i^{t+1}\,|\,x\} & ,\ (i\neq0)\wedge(x\neq0)\text{时} \\ 0 & ,\ (i=0)\text{或 } x=0\text{ 时} \end{cases}$$

22 Lt(x)

Lt(x)表示：在x的素因子分解中最后一个非零指数是第几素数的指数，即第Lt(x)个以后的指数均为 0。例如，我们有 $20=2^2\cdot3^0\cdot5^1\cdot7^0\cdot11^0\ldots$，其中最后一个非零指数是第三个，因此有Lt(20)=3。Lt(x)的形式定义如下：

Lt(x)=$\min_{i\le x}\{(x)_i \neq 0 \wedge (\forall j)_{\le x}(j \le i \vee (x)_j = 0)\}$

23 | GN(x)

GN(x)是一个谓词。GN(x)真表示在x的素因子分解中没有零指数，即指数均为正。例如，$60=2^2 \cdot 3^1 \cdot 5^1$，因此有GN(60)真。

GN(x) ⇔x>1∧$(\forall i)_{\le Lt(x)}$ (i=0∨$(x)_i$ ≠0)

24 | $[a_1, a_2, \ldots, a_n]$

这是歌德尔数，用 $[a_1, a_2, \ldots, a_n]$表示 $P_1^{a_1} \bullet P_2^{a_2} \bullet \cdots \bullet P_n^{a_n}$，即表示

$$\prod_{i=1}^{n} P_i^{a_i}$$

25 | x*y

设$x=[a_1, a_2, \ldots, a_n]$

$y=[b_1, b_2, \ldots, b_m]$

则我们定义

$x*y=[a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m]=\prod_{i=1}^{n} P_i^{a_i} \bullet \prod_{j=n+1}^{n+m} P_j^{b_{j-n}}$

其形式定义如下：

$x*y=\prod_{i=1}^{Lt(y)} P_{Lt(x)+i}^{(y)_i} \bullet \prod_{j=1}^{Lt(x)} P_j^{(x)_j}$

26 | #(a，x)

#(a，x)表示在 x 的素因子分解中指数为 a 者有几个。例如，假定 x=14700，则其分解为

$x=2^2 \cdot 3^1 \cdot 5^2 \cdot 7^2$

其中指数为 2 者有 3 个，因此#(2，14700)=3

#(a，x)的形式定义如下

$$\#(a，x)=\sum_{i=1}^{Lt(x)}F(a,i,x)$$

其中 F(a，i，x)= $\begin{cases} 1，若a=(x)_i因子的分解中P_i的指数为a时为\ 1 \\ 0，否则 \end{cases}$

下面引进在后面常用到的原始递归函数-----配对函数。具体引进一个二元函数<x，y>是两个一元函数 l (z)和 r(z)。它们具有下面性质：

(1) 它们都是原始递归函数。

(2) l (<x，y>)=x

(3) < l (z)，r(z)>=z

(4) l (z)，r(z)≤z

有许多方式实现这一点。我们将用称为康托配对函数的这一标准而十分简单的方式。

直观表示为：

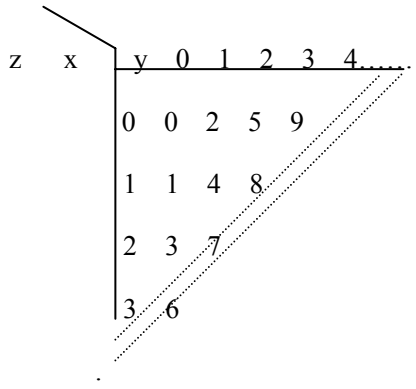


图 3.1

我们定义

h(n)=1+2+3+…+n

对分给的 z，总可以找到 n(唯一)使得

h(n) ≤z≤h(n+1)

于是可令

z=h(n)+y

这种 y 显然是唯一的，再

n=y+x

这种 x 显然也是唯一的。故对任给的 z 有唯一的对偶(x,y)与之对应。

下面考虑由(x，y)找对应 z 的方法。由

z=h(n)+y

n=y+x

我们有

z=h(x+y)+y

$$=\frac{(x+y)(x+y+1)}{2}+y$$

于是有下面函数。

27 <x，y>

$$<x，y>=\frac{1}{2}(x+y)(x+y+1)+y$$

28 l (z)

$$l(z)=\min_{x\le z}\{\exists y)_{\le z}\, z=<x,y>\}$$

29 r(z)

$$r(z)=\min_{y\le z}\{\exists x)_{\le z}\, z=<x,y>\}$$

例如，<2，3>=18，l (18)=2，r(18)=3

<3，2>=17，l (17)=3，r(17)=2

假设有谓词(∃x)(∃y)P(x，y，z)，则因为“存在两个数 x 和 y”与“存在一个数 w”（该 w 为<x，y>）一对一。因此可把上述谓词写成如下形式：

(∃w)P(l (w)，r(w)，z)

即有

(∃x) (∃y)P(x，y，z)⇔ (∃w)P(l (w)，r(w)，z)

对函数也有类似情形。

3.5 部分递归函数，递归函数及其可计算性

前面我们研究了特殊一类的函数－原始递归函数。我们将介绍更为一般的递归函数，并证明每个递归（部分）函数都是可（部分）计算的。

到目前为止，对于函数我们只用了复合和递归算子，并且有取极小算子：

$h(x_1，x_2，\ldots，x_n)$

$=\min\{z|f(x_1，x_2，\ldots，x_n, z)=0\}$

如果对于任意一组$x_1，x_2，\ldots，x_n$都有一个z，使得$f(x_1，x_2，\ldots，x_n, z)=0$，则称f为正则函数。

定义 3.5.1 由初始函数

$S(x)，n(x)，\cup_i^n (x_1, x_2, ..., x_n)$ $1\le i\le n$

出发，用复合、递归和正则函数取极小算子所得的函数称为递归函数。因此取极小结果可能导出部分函数，所以加上“部分”二字。

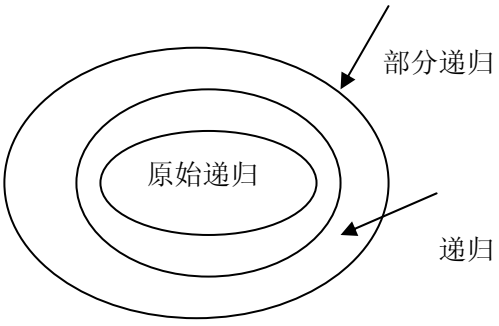


图 3.2 原始⊂递归⊂部分递归

定义 3.5.2 由初始函数 $S(x)，n(x)，\cup_i^n (x_1, x_2, ..., x_n)$ $1\le i\le n$

出发，用复合、递归和对正则函数取极小算子所得的函数称为递归函数。递归函数是全函数。原始递归函数、递归函数和部分递归函数之间的关系如图 3.2 所示。

定理 3.5.1 每个递归函数是可计算的。

定理 3.5.2 每个部分递归函数是部分可计算的。

我们在本书中不去证明上述定理的逆命题，即每个可计算函数是递归函数，每个部分可计算函数是部分递归函数。

证明第一个定理要证两件事：

A 每个初始函数是可计算的。

B 复合、递归和正则函数的取极小算子保持可计算性。定理的具体证明由以下一些引理组成。

引理 1　初始函数 S（x），n(x)，$\cup_i^n (x_1, x_2, ..., x_n)$ 是可计算的。

证明：

1　程序

```
x=x+1
y=x
```

它计算全函数 y=S(x)，故它是可计算的。

2　程序

```
x=x+1
```

计算全函数 y=n(x)。因为 y 开始时为零。故 n(x)是可计算的。

3　对给定的 n，i (1≤ i≤ n)，程序

```
x1=x1+1
x2=x2+1
……
xi-1=xi-1+1
y=xi
xi+1=xi+1+1
…….
xn=xn+1
```

计算全函数 y=$\cup_i^n (x_1, x_2, ..., x_n)$。故投影函数是可计算的。

引理 2　设下面函数是(部分)可计算的

$f(z_1, z_2, \ldots, z_k)$

$g_1(x_1, x_2, \ldots, x_n)$

…… …… ……

$g_k(x_1, x_2, \ldots, x_n)$

则由它们经复合而得的函数

$h(x_1, x_2, \ldots, x_n)=f(g_1(x_1, x_2, \ldots, x_n), \ldots, g_k(x_1, x_2, \ldots, x_n))$

是(部分)可计算的，即复合法保持可计算性，如果上面函数都是可计算的，则 h 也是可计算的。

证明：因为f，g_1，g_2，…，g_k皆部分可计算的，故有计算它们的程序。于是可用下面程序去计算h：

```
z1= g1(x1，x2，…，xn)
z2= g2(x1，x2，…，xn)
…… …… ……
zk= gk(x1，x2，…，xn)
y=f(z1，z2，…，zk)
```

其中每个语句表示相应一段程序。故函数 h 是部分可计算的。

如果f和g_1，g_2，…，g_k是可计算的，即全函数，则显然h也是全函数，故是可计算的。

引理 3 设f (x_1，x_2，…，x_n，y)是可计算的，则函数

h(x_1，x_2，…，x_n)=min{ y | f (x_1，x_2，…，x_n，y)=0}是部分可计算的。

证明：程序

```
[B]   z= f (x1，x2，…，xn，y)
      TO   A   IF z≠0
      TO   E
[A]   y=y+1
      TO   B
```

计算h(x_1，x_2，…，x_n)。程序开始y＝0，首先计算z＝f(x_1，x_2，…，x_n，0)，然后检查是否z＝0，若是，则y＝0 作为h (x_1，x_2，…，x n)的值，否则y＝y＋1，然后计算z＝f(x_1，x_2，…，x_n，1)，并检查是否z＝0？若是，则y＝1 做为h(x_1，x_2，…，x_n)的值，否则y＝y＋1，以此类推。如果没有y使f(x_1，x_2，…，x_n，y)＝0，则程序无穷的进行下去，这时h(x_1，x_2，…，x_n)将无定义。

引理 4 设f (x_1，x_2，…，x_n，y)是正则的可计算函数，则函数h(x_1，x_2，…，x_n)=min{ y |f(x_1，x_2,…，x_n，y)=0}是可计算的。

证明　证明仿引理 3.3。区别只在于此情况下不论对那组x_1，x_2，…，x_n终有y使f(x_1，x_2，…，x_n，y)＝0 且使程序停止。故h(x_1，x_2，…，x_n)是可计算的。

引理 5 设g(x_1，x_2)是可计算的，则如下定义的函数h是可计算的。

$$\begin{cases} h(0)=k \\ h(n+1)=g(n，h(n)) \end{cases}$$

证明：下面程序计算 h(x)，开始令 y=k。如果 x=0 则 y=k 为 h(x)=h(0)的值。若 x≠0，则计算 y=g(0，h(0))，并做 x=x-1 和 z=z+1。如果 x=0，则 y 的值为 h(x)的值，否则重复类似上述的过程。其中

```
y=k
```

是下面程序的简写(y 的初值为 0)

```
y=y+1
y=y+1
……
y=y+1
```

(k 次)

引理 6 若f (x_1，x_2，…，x_n)

g(x_1，x_2，…，x_n，x_{n+1}，x_{n+2})

是可计算，则如下定义的函数是可计算的。

$$\begin{cases} h(x_1, x_2, \ldots, x_n, 0)=f(x_1, x_2, \ldots, x_n) \\ h(x_1, x_2, \ldots, x_n, n)=g(x_1, x_2, \ldots, x_n, h(x_1, x_2, \ldots, x_n, n-1),n-1) \end{cases}$$

证明略

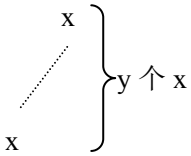
从以上引理可以证明我们的定理。

```
    y=k
[B] TO A IF x ≠0
    TO E
[A] y=g(z，y)
    x=x-1
    z=z+1
    TO B
```

习 题

证明下列函数是原始递归的

1



2 $[\sqrt[y]{x}]$

3 称三边均为整数的直角三角形的勾股数，将它们从小到大排列，第 n 个勾股数记作 R(n)

证明 R(n)是原始递归。

4 设 f(x)表示 x 的各位数字之和，证明 f(x)是原始递归函数。

5 证明$[\log_2 x]$是原始递归函数。

6 设 $a_n = f(n)$，$b_n = g(n)$ 都是递增的原始递归函数，将 a_n，b_n $(n = 0,1,2,\cdots)$ 混合在一起，证明在按从小到大排列得到函数 $c_n = \varphi(n)$ 是原始递归函数。

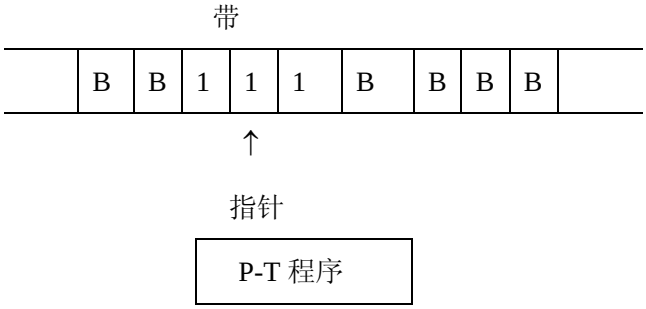
7 用元语言程序证明函数α(x÷3)是可计算函数。

8 用元语言程序证明每个原始递归函数都是可计算函数。

第四章　Post-Turing 程序和 Turing 机

4.1Post-Turing 程序

前面用的程序有一变元素。其中每个变元能存一个数。程序对这些变元进行操作。现在要引进的 Post-Turing 机却不同，它没有变元，而是有一个所谓的“带”，并对带进行操作。带分成很多（无穷）格，每个格可记录一个符号。



带有一个指针“↑”，它指向正在被扫描的格。P-T 程序可以把指针 左移或右移一格，亦可在所注视的格上输出符号。开始时把初始符号串给在带上，并且指针指向开始符，我们认为带的两边是无穷的。

我们约定在带的两端永远有额外的空白，当指针左移或右移时不会由带上“掉”下来。在这里我们首先考虑两个符号的机器。这两个符号是：

B 和 1

其中 B 是空白符。

机器有下面一些指令：

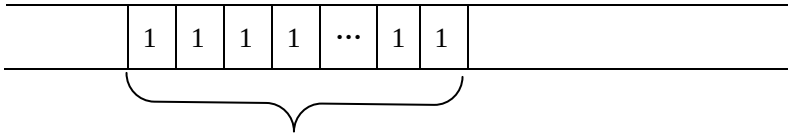
RIGHT	指针右移一格
LEFT	指针左移一格
WRITE　1	抹去当前所注视格内的符号并且在该格内打印 1
WRITE B	抹去当前所注视格内的符号并且在该格内打印 B

TO A IF B	若所注视符号是空白符 B，则转向标号为 A 的指令， 否则执行下一条指令，在二情况下都不改变指针位置。

TO A IF 1	若所注视符号是 1，则转向标号为 A 的指令， 否则执行下一条指令，在二情况下都不改变指针位置。

P-T 程序和原来的程序设计语言的程序类似， 它是带或不带标号的指令的序列，其中没有二指令具有相同标号，也约定程序的结束是转去程序中没有的标号。

带上数的表示法如下。设 N 为一个数，则在带上把它表示成 N+1 个 l：



N+1

为方便计，将用 $\overline{x}$ 表示数 x 的带上表示。例如，

$\overline{3}$=1111

$\overline{1}$=11

$\overline{0}$=1

带上输入值的表示法如下。设一个P-T程序计算函数f (x_1, x_2, …, x_n)，则把变元x_1, x_2, …, x_n的值按下法给在带上：

1 1⋯1 1 B 1 1⋯1 1 B⋯⋯⋯⋯⋯B 1 1⋯1 1

x_1+1　　x_2+1　　x_n+1

可简写成如下：$\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}$ 。例如，(2，0，3)的带表示如下：

111B1B1111

计算结果是计算终止带上 1 的个数减 1。例如，计算后带上形状为

B11BB1B1111BB

则它表示结果 6。

例 4.1.1 计算 P(x)=x-1 的 P-T 程序。

若 x=0，则函数值为 0，因此不必做任何工作；若 x≠0，则要抹去一个 1，具体 P-T 程序如下：

```
RIGHT
TO E IF B
WRITE B
```

若要抹去最后一个 1，则程序如下：

```
        RIGHT
        TO E IFB
［A］   RIGHT
        TO A IF 1
        LEFT
        WRITE B
```

例 4.2.2 计算函数

$$f(x)=\begin{cases} x \div 2 & (x>1) \\ x+2 & (x\le 1) \end{cases}$$

首先检查初始带的第三个符号是否 B，若是 B，则表示初始值 x≤1，否则 x>1。

若 x≤1，则多写两个 1，若 x>1，则 x≥2，这时带上有三个以上的 1。我们将抹去最左边的两个 1。

程序如下：

```
      RIGHT
      RIGHT
      TO A IF 1
      WRITE 1
      RIGHT
      WRITE 1
      TO E IF 1
［A］LEFT
```

WRITE B LEET WRITE B

仿过去的(部分)可计算函数，可定义 Post-Turing 可计算函数。

定义 4.1.1 称函数 f 是 P-T 部分可计算的，若有一个 P-T 程序计算它。

定义 4.1.2 称函数 f 为 P-T 可计算的，如果它是 P-T 部分可计算的并且是全函数。

4.2 可计算性与 P-T 可计算性

现在有两种面向可计算函数定义的机器。可提出问题：它们之间有什么关系？我们将看到它们是等价的。即(部分)可计算函数是 P-T(部分)可计算函数，反之亦然。

定理 4.2.1 每个(部分)可计算函数是 P-T(部分)可计算函数。

证明：基本思想是论证任意一个可计算函数的程序 P 均可模拟为等价的 P-T 程序。它们的初始信息和结果相同。当然表示法不相同的。

一般程序的操作对象是变元，而 P-T 程序的操作对象是带，因此首先必须把变元状态反映到带上来，一般程序包含如下变元

x_1，x_2，…，x_n，z_1，z_2，…，z_m，y

其中x_1，x_2，…，x_n是自变元z_1，z_2，…，z_m是中间变元，y是结果变元。一旦给定程序，上述m和n是确定的。这些变元在带上的表示法如下：

$$\mathrm{B}\,\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}\,\mathrm{B}\,\overline{z_1}B\overline{z_2}B\cdots\cdots B\overline{z_m}B\overline{y}\,\mathrm{B}$$

初始值x_1，x_2，…，x_n是给定的，因此可假定P-T程序的初始带如下：

$$\mathrm{B}\,\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}\,\mathrm{B} \qquad (*)$$

而程序的初始状态是要求z_1，z_2，…，z_m和y均为 0。因此P-T程序的第一部分应把带(*)改变为带(**)：

$$\underbrace{\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}}_{\text{x 变元值 (n 个)}}\ \mathrm{B}\underbrace{1\mathrm{B}1\mathrm{B}\ldots\mathrm{B}1}_{\text{z 变元值 (m 个)}}\mathrm{B}\underbrace{1}_{\text{y 初值 (1 个)}}\ \mathrm{B} \qquad (**)$$

从带(*)到带(**)部分的 P-T 程序部分将表示为

BEGIN

在写出 BEGIN 部分后，可把原程序翻译成等价的 P-T 程序。这一部分表示为

MIDDLE

在这里原程序的一条指令对应 P-T 程序的一组指令。

当 P-T 程序工作完时带的状态为

$$B\,\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}\,B\,\overline{z_1}B\overline{z_2}B\cdots\cdots B\overline{z_m}B\overline{y}B$$

其中 $\overline{y}$ 表示计算结果，带上还有多余的x_i和z_j，因此还必须抹掉它们的信息，完成这部分的P-T程序部分表示为

END

这样模拟一般程序的 P-T 程序结果如下：

BEGIN MIDDLE END

为方便计，首先引进几条宏指令

宏指令	意义
TO A (无条件转移)	TO A IF B TO A IF 1
RIGHT TO NEXT B	［A］RIGHT TO A IF 1
LEFT TO NEXT B	［A］LEFT TO A IF 1
WRITE B1	WRITE B RIGHT WRITE 1 RIGHT
指令β[k]	指令β 指令β (k 次) 指令β

以上宏指令的含义是清楚的。下面将用这些宏指令给出程序的模拟方法。

BEGIN 程序：

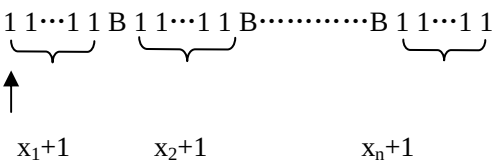
RIGHT TO NEXT B[n]

WRITE B1[m+1]

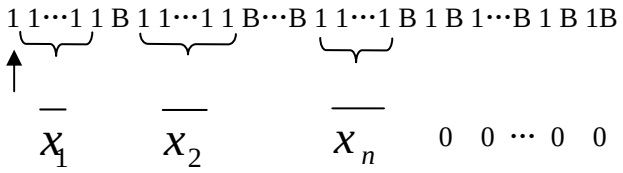
LEFT TO NEXT B [n+m+1]

RIGHT

该程序的操作对象是 $\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}$ 即



工作结果是



(B)MIDDLE

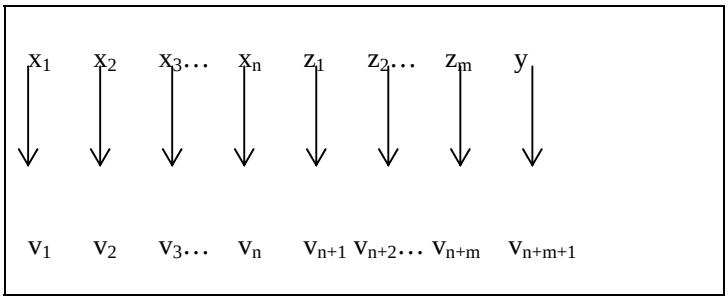
现在介绍用 P-T 指令模拟程序 P 的方法。我们可假设程序 P 只包含下列三种指令：

x=x+1

x=x-1

TO A IF x≠0

为方便计，将变元统一改名为v_i，具体如下：



下面介绍三种指令的模拟方法。

a

$V_i=V_i+1$

$V_i=V_i+1$ 的模拟程序如下：

```
      RIGHT TO NEXT B［i］
［A］WRITE   1
      RIGHT
      TO C IF B
      WRITE B
      RIGHT TO NEXT B
      TO A
［C］LEFT TO NEXT B［n+m+1］
      RIGHT
```

工作过程是：

1 把指针移到 $\overline{x_i}$ 后面的 B 上。

$$\overline{x_1}B\overline{x_2}B\cdots\overline{x_i}B\overline{x_{i+1}}B\cdots\cdots$$

2 把 $\overline{x_i}$ 后面的 B 改为 1，再把 $\overline{x_{i+1}}$ 的第一个 1 改为 B。对于后面的 $\overline{x_{i+1}}$，…，$\overline{x_n}$ $\overline{z_1}$ …，$\overline{z_m}$，$\overline{y}$ 重复上述过程。

3 指针从最右移到最左边。

工作结果是：

$$\overline{x_1}B\overline{x_2}B\cdots\overline{x_i}B\overline{x_{i+1}}B\cdots\cdots$$

b

TO A IF $V_i\neq0$

模拟程序的工作过程是：

1 把指针移至 $\overline{v_i}$ 块的紧后边。

2 检查$\overline{v_i}$块的最后第二位是否B。若是，则表示V_i=0，否则表示V_i≠0。

3 若V_i≠0，则把指针移至最左边并转标号A。

4 若V_i=0，则把指针移至最左边并执行下一条。

程序：

```
        RIGHT TO NEXT B［i］
        LEFT［2］
        TO B IF 1
        TO C
［B］LEFT TO NEXT B［i］
        RIGHT
        TO A
［C］LEFT TO NEXT B［i-1］
        RIGHT
```

下面还用到一条宏指令：

```
                WRITE B
MOVE BLOCK      RIGHT TO NEXT B
                WRITE 1
                RIGHT
```

MOVE 宏指令的主要功能是把某当前块$\overline{v_i}$整个地右移一格，指针放在$\overline{v_i}$块(右移后)的紧后边。其算法要点如下：

1 把$\overline{v_i}$块的第一个 1 改为 B。注意 MOVE 程序要求开始指针位于$\overline{v_i}$块的第一格。

2 跳到$\overline{v_i}$块后面的第一个 B 上，并将它改为 1。

3 再指针右移一格，使指针指向$\overline{v_i}$块的后继格，即指向$\overline{v_{i+1}}$块的第一位。若不继续执行此指令，$\overline{v_i}$块和$\overline{v_{i+1}}$块将连成一块。

```
C       Vi=Vi-1
```

已知 P-T 指令可模拟指令

TO A IF $V_i \neq 0$

因此可在宏指令中应用它。

当$v_i=0$ 时，执行结果$v_i=v_i-1$ 的值应为 0。因此，在模拟时不做任何操作而直接出口即可。若$v_i \neq 0$ ，则在$\overline{v_i}$ 块内消去一个 1。做法是把$\overline{v_1}$，$\overline{v_2}$，…，$\overline{v_{i-1}}$ 块统统右移一格，再把$\overline{v_i}$ 块的最左 1 改为B。最后把指针移回最左。

程序：

```
        TO A IF Vi≠0
        TO E
[A]     MOVE BLOCK RIGHT[i-1]
        WRITE B
        LEFT TO NEXT B[i-1]
        RIGHT
```

这就完成了 MIDDLE 部分。

(C) END 部分

END 部分的作用是消去带上的所有$\overline{x_i}$ 块和$\overline{z_i}$ 块，只保留$\overline{y}$ 块。这里用到一条宏指令：

ERASE A BLOCK	[A] TO C IF B WRITE B RIGHT TO A [C] RIGHT

ERASE 指令的主要用途是把一个块内的 1 全部改为 B，并把指针放在该块的后继格上。该指令工作时要求指针位于所要抹的块的第一位上。

实现算法是：

1 检查注视符是否 B？若是，则把指针右移一格并停止，否则转 2。

2 把注视符 1 改为 B，然后转 1。

END 部分的模拟指令如下：

ERASE A BLOCK [n+m]

这里 P-T 带的最后样子是：

$$B\ B\ \underbrace{1\ 1\cdots1\ 1}_{y}\ B\ B$$

以上我们就证明了每个(部分)可计算函数是 P-T(部分)可计算函数。

4.3 广义 Post-Turing 机

前面考虑了两个字母的 Post-Turing 机。这一节考虑多个字母的 Post-Turing 机。称这种机器为广义 Post-Turing 机。我们将会证明这两种机器类是等价的。

我们假设广义 Post-Turing 机有下面字母表

S_0，S_1，S_2，…，S_k

为了和过去相联系约定：S_0=B，S_1=1。

机器指令如下：

RIGHT

指针右移一格

LEFT

指针左移一格

WRITE S_i

在注视的格内写下符号S_i

(i=0，1，2，…，k)

TO A IF S_i

若所注视的格是符号S_1 ，则转标号A，否则执行下一条。这两种情况下指针不移动。

对输入输出做同样的约定。

定理 4.3.1 P-T(部分)可计算函数都是广义 P-T(部分)可计算函数。

定理 4.3.2 广义 P-T 部分程序计算函数是部分可计算函数。

为了证明这定理，要表明给定一个 P-T 程序后，可用原语言构造一个与之等价的程序。P-T 程序的工作对象是带上的符号串，而一般程序的工作对象是变元中的数，因此，必须给出一种编码方法，使得带上的每个符号串唯一对应一个数。P-T 程序对于带上符号串的操作，也就对应于带上符号串所对应数的等操作。

a 字母表的编码如下：

$S_0 \longrightarrow 1$

$S_1 \longrightarrow 2$

：

：

$S_k \longrightarrow k+1$

b 带上符号串的编码方法如下。例如

$S_2S_3S_0S_1S_2 \rightarrow [3，4，1，2，3]$

$=2^3 \cdot 3^4 \cdot 5^1 \cdot 7^2 \cdot 11^3$

$S_3S_0S_0S_1S_1 \rightarrow [4，1，1，2，2]=2^4 \cdot 3^1 \cdot 5^1 \cdot 7^2 \cdot 11^2$。

我们将用 U 变元表示带上符号串所对应(哥德尔数)，用变元 V 记录指针的位置。

注意在这里有两个 1。一个是字母表中的 1(即S_1)，它是一个符号，一个是S_0符号的编码 1，它是一个数，带上的 1 是符号 1。为了区别起见，将用黑体字 1 来表示符号 1(S_1)。

c 我们将用 TAPE(x)表示 x 的带上表示 $\bar{x}$ 的哥德尔数编码。

$$\bar{x} = \underbrace{11\ldots\ldots11}_{x+1\text{ 个}} = \underbrace{S_1S_1\ldots\ldots S_1S_1}_{x+1\text{ 个}} \rightarrow \underbrace{[2, 2,\ldots 2,2]}_{x+1\text{ 个}}$$

于是有

$$\text{TAPE}(x) = \underbrace{[2,2，\cdots，2]}_{x+1} = 2^2 \cdot 3^2 \cdots P^2_{x+1} = \left(\prod_{i=1}^{x+1} p_i\right)^2$$

例如

$\text{TAPE}(3)=[2,2,2,2]=(2 \cdot 3 \cdot 5 \cdot 7)^2$

TAPE(0)=[2]=2^2

另外，再定义

TAPE ^{n}(x_1，…，x_n)=TAPE(x_1)*[1]*TAPE(x_2)*2*…*2*TAPE(x_n)

这就是 $\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}$ 的哥德尔编码。其中 2 是空白符B(即S_0)的哥德尔编码,而$S_0\rightarrow 1$。

不难看出TAPE^n(x_1，…，x_n)是原始递归的。

仿前，模拟前三部分：

BEGIN

MIDDLE

END

我们已假定 U 表示带上符号串的编码，V 指出指针位置。P-T 程序的指令不断改变 U 和 V。

BEGIN 部分如下：

U= TAPE^n(x_1，…，x_n) V=1

END 部分是

Y=#(2，U)-1

MIDDLE 部分是 P-T 指令的具体模拟部分，在这里须要知道的有以下几点：

a. $(U)_v$表示当前所注视符的编码。

b.［U/ $P_V^{(U)_V}$ ］• $(P_V)^{i+1}$表示把当前指针所注视符改为S_i符。

模拟方法如下：

WRITE S_i	U=[U/ $P_V^{(U)_V}$] • $(P_V)^{i+1}$
TO A IF S_i	TO A IF $(U)_V$=i+1 (令z=α(δ $(U)_v$=i+1)) TO A IF z≠0) 特征函数
	TO A IF V≠Lt(U) U=U*2→因为，执行此语句后，指针（P-T）指向

RIGHT	S_0 [A] V=V+1
LEFT	TO A IF V≠1 U=2*U TO E } V 值不变，此时指针已指向第一个数 [A] V=V-1

在这里我们用了形如

TO A IF $P(x_1, x_2, \ldots, x_n)$

的指令，其中$P(x_1, x_2, \ldots, x_n)$是原始递归谓词。在原指令中，转移指令只有以下两种：

TO A IF X≠0

TO A

事实上，我们有

TO A IF $P(x_1, x_2, \ldots, x_n)$

等价于

$z=\alpha(\delta_P(x_1, x_2, \ldots, x_n))$

TO A IF z≠0

其中$\delta_P(x_1, x_2, \ldots, x_n)$是$P(x_1, x_2, \ldots, x_n)$的特征函数。

上述指令都是合法的。因此，综合上述。我们有：多符号 P-T 程序所计算的函数都是部分可计算的。我们有下面图 4.1：这个图给出下面推论：

推论 1 多符号 P-T(部分)可计算函数一定是二符号 P-T(部分)可计算函数。

推论 2 二符号 P-T(部分)可计算函数是(部分)可计算函数。

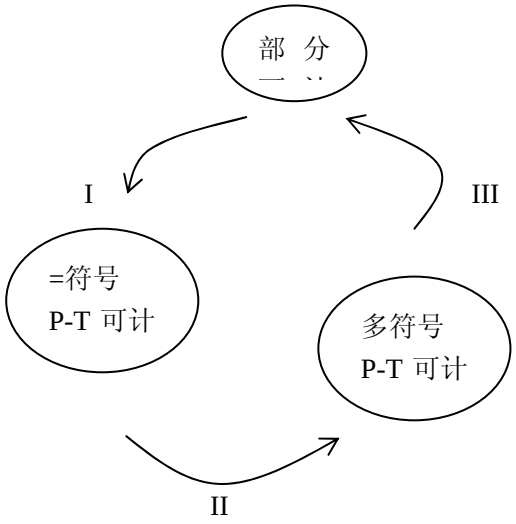
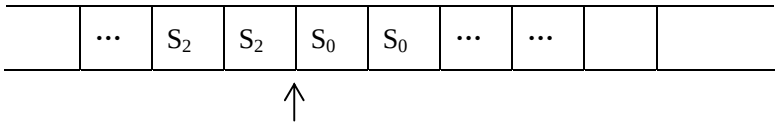


图 4.1

4.4 Turing 机

现在开始研究一种新的计算装置—Turing(图灵机)。Turing 机和 Post-Turing 机类似，它有一个带和指针



Turing 和 Post-Turing 机一样有三个基本动作：

1 往带上输出符号

2 把指针右移一格；

3 把指针左移一格。

每个 Turing 机都有自己的字母表和状态集：

字母表$\Sigma=\{S_0，S_1，S_2，\cdots，S_K\}$

状态集$Q=\{q_1，q_2，q_3，\cdots，q_N\}$

在状态集中有一个特殊的状态叫做初始状态。

我们假定q_1为初始状态。

Turing 机的每步动作由所谓的四元组来确定。每个四元组完成一个基本动作并转入下一个状

态。

四元组有三类，具体如下：

四 元 组	意　　义
$q_i\ S_j\ S_k\ q_l$	当前状态为q_i，当前注视符为S_j，把符号S_k印在所注视的格内，并转入状态q_l。
$q_i\ S_j\ L q_l$	当前状态为q_i，当前注视符为S_j时，把指针左移一格并转入状态q_l。
$q_i\ S_j\ R\ q_l$	当前状态为q_i，当前注视符为S_j时，把指针右移一格，并转入状态q_l。

四元组中L和R是特殊符，它们区别于字母表中符号S_i。四元组中前两个符号表示条件，第三个符号表示动作内容，第四个符号表示要转入的下一个状态。

定义 4.4.1 Turing机是四元组的有穷集，其中没有两个四元组的头两个符号$q_i\ S_j$是相同的。

由于四元组的头两个符号彼此不相同，每次只能选中一个四元组，通常的图灵机都有终止状态，但我们这里不用这种状态，而是规定：如果当前状态q_i和当前符S_j的对偶$q_i\ S_j$不出现于Turing机的任何四元组的前两个字符，则Turing机停机。

带上的初始信息　还是表示成如下：

$$\overline{x_1}B\overline{x_2}B\cdots\cdots B\overline{x_n}$$

即

1 1…1 1 B 1 1…1 1 B……B 1 1…1 1

x_1+1　　x_2+1　　x_n+1

这些都和 Post-Turing 机一样，计算结果仍然规定为带上 1 的个数减 1。

例 4.4.1 构造计算函数S(x)=x+1 的Turing机。字母表取Σ={1，B}，Turing机把指针移至$\overline{x}$后面的B上，然后把该B改为 1 并停机，初始状态为q_1。四元组集：

$q_1\ 1\ R\ q_1$
$q_1\ B\ 1 q$

注意初始指针和结束指针可以在任何位置，并不要求恢复到指定位置。

例 4.4.2 构造计算函数S(x)=2x的Turing机。字母表取Σ={1，B，a，b}，状态取Q={q_1，q_2，q_3，q_4，q_5，q_6，q_7，q_8，q_9

q_1→复制

$q_1\ 1\ a\ q_2$
$q_2\ a\ R q_3$
$q_3\ 1\ R\ q_3$
$q_3\ b\ R\ q_3$
$q_3\ B\ b\ q_4$
$q_4\ b\ L\ q_4$
$q_4\ 1\ L\ q_5$
$q_4\ a\ L\ q_6$

$q_5\ 1\ L\ q_5$
$q_5 a\ R\ q_1$
$q_6\ a\ L\ q_5$
$q_6\ B\ R\ q_7$
$q_7\ a\ B\ q_8'$
$q_8'\ B\ R\ q_8$
$q_8\ a\ 1\ q_9$
$q_9\ 1\ R\ q_8$

B a a…a 1 1…1 b b…b B

q_2

四元组集：

q_1复写a状态　　q_8写 1 状态

q_2复写a后状态

q_3右移到尾状态　　q_9右移

q_4复写b后状态“左移”　　q_{10}结束

q_5左移到 1 段状态

q_6复写完状态“左移”

q_7左移到头

q_8' 消去一个 1 状态（因x=x+1，所以 $\overline{zx} = \overline{zx} - 1$）

x=0 和 x=1 情形计算φ(x)的过程如下：

(x=0)	(x=1)	
B q_1 1 B	B q_1 1 1 B	
→B q_2 a B	→ B q_2 a 1 B	→ B a a q_4 b b
→B a q_3 b B	→B a q_3 1 B	→ B a q_4 a b b
→Ba q_4 b B	→B a 1 q_3 B	→ B q_6 a a b b
→B q_4 a b B	→B a 1 q_4 b B	→ q_6 B a a b b
→q_6 B a b B	→B a q_4 1 b B	→B q_7 a a b b
→B q_7 a b B	→B q_5 a 1 b B	→ B q'_8 1 B a b b
→B q'_8 B b B	→B a q_1 1 b B	→B B q_8 a b b
→B B q_8 b B	→B a q_2 a b B	→B B q_9 1b b
→B q_9 1 B	→B a a q_3b B	→B1 q_9 b b
→B 1 q_8 B	→B a a b q_3 B	→B 1q_9 1 b
→B 1 q_{10} B	→B a a b q_4 b	→B 1 1 q_8 b
		→B 1 1 q_9 1
		→B 1 1 1 q_8 B
		→B 1 1 1 q_{10} B

结果带为　　结果带

B 1 B	B 1 1 1B

(表示 0)　　（表示 2）

定义 4.4.2 函数f(x_1，x_2，…，x_n)是Turing部分可计算的。若有一Turing机计算它。(而不是程序)

定义 4.4.3 函数f(x_1，x_2，…，x_n)是Turing可计算的，若函数f(x_1，x_2，…，x_n)是Turing部分可计算并且是全函数。

定理 4.4.1 每个 P-T(部分)可计算函数都被具有相同字母表的 Turing 机所计算。

证明 设想 P-T 程序 P 为有编号 1-N 的语句列。

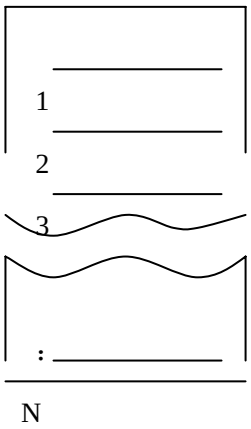


图 4.2

想法是让编号为i的语句对应于Turing机的状态g_i，然后用四元组来模拟各语句。

因为 Turing 机的初始数据和结果数据在带上的表示形式与 Post-Turing 机完全相同，因此我们只要把 P-T 程序的每个语句模拟为 Turing 机的四元组形式即可。

下面是语句(设它为第 i 个语句)与四元组的对应表：

我们假定了下述表中的每个语句编号为i，因此它们对应状态q_i。

WRITE S_j语句表示把S_j印到当前所注视的格内(不管原来有什么符号)，然后执行下一个语句，下面的四元组集(K+1 个)。

WRITE S_j	$q_iS_kS_jq_{i+1}$ (k=0,1,…,k)
LEFT	$q_iS_kLq_{i+1}$

	(k=0,1,…,K)
RIGHT	$q_iS_kRq_{i+1}$ (k=0,1,…,K)
TO A IF S_j A 为标号的语句 编号假定为 l	$q_i\ S_j\ S_j\ q_l$ $q_iS_kS_kq_{i+1}$ (k=0,1,…,K 且 k≠j)

$q_i\ S_0\ S_j\ q_{i+1}$

$q_i\ S_1\ S_j\ q_{i+1}$

$q_i\ S_2\ S_j\ q_{i+1}$

……

$q_i\ S_k\ S_j\ q_{i+1}$

表示不管状态q_i当前所注视的格内有什么符号将把S_j印到当前格内并转状态q_{i+1}模拟了编号为i的语句WRITE S_j。

如果标号A是出口标号，则其编号规定为N+1，它对应于状态q_{n+1}。

定义 4.4.4 下面两种形式称为五元组：

$q_i\ S_j\ S_k\ Lq_l$

$q_i\ S_j\ S_k\ Rq_l$

我们已知四元组只完成一个基本动作，而五元组则完成两个基本动作：输出一个符号并移动指针，Turing 原来的陈述是使用了五元组。

定理 4.4.2 四元组 Turing 机和五元组 Turing 机可彼此模拟。

证明：设四元组 Turing 机的状态为

q_1，q_2，…，q_n

则五元组 Turing 机的状态多二倍，记为

q_1，q_2，…，q_n，q_{n+1}，…，q_{2n}，q_{2n+1}，…，q_{3n}。

四元组到五元组的过程如下表：

$q_iS_jS_kq_l$	$q_iS_jS_kRq_{n+l}$ $q_{n+l}S_mS_mLq_l$ (m=0,1,…,K)
$q_iS_jRq_l$	$q_iS_jS_jRq_l$

$q_iS_jLq_l$	$q_iS_jS_jLq_l$

五元组到四元组过程如下表：

$q_iS_jS_kq_l$	$q_iS_jS_kq_{n+l}$ $q_{n+l}S_mRq_l$ (m=0,1,…,K)
$q_iS_jS_kLq_l$	$q_iS_jS_kq_{2n+l}$ $q_{2n+l}S_mLq_l$ (m=0,1,…,K)

定理 4.4.3 五元组 Turing 机可被具有相同字母表的广义 P-T 程序模拟。

证明：设 Turing 机的状态为

q_1，q_2，q_3，…，q_n

使每个状态对应一个广义 P-T 程序的标号，记为

A_1，A_2，A_3，…，A_n

再对每个对偶q_iS_j对应标号B_{ij}。

在标号A_i处有

$[A_i]$ TO B_{i0} IF S_0 TO B_{i1} IF S_1 …… TO B_{ik} IF S_k

在标号B_{ij}处有以q_iS_j为头的五元组的动作。具体如下：

$q_iS_jS_kRq_l$	$[B_{ij}]$ WRITE S_k RIGHT TO A_l
$q_iS_jS_kLq_l$	$[B_{ij}]$ WRITE S_k LEFT TO A_l
无q_iS_j为头的五元组	［B_{ij}］ TO E

以A_i为标号的程序段的作用是判断属于哪个q_iS_j并转相应B_{ij}程序段。B_{ij}程序段在完成以q_iS_j为头的五元组的动作，转状态q_l被模拟为转A_l。

模拟后的 P－T 程序结构如下：

$[A_1]$ TO B_{10} IF S_0

…

TO B_{1k} IF S_k

$[A_2]$ TO B_{20} IF S_0

…

TO B_{2k} IF S_k

…

$[A_n]$ TO B_{n0} IF S_0

TO B_{nk} IF S_k

$[B_{10}]$ ……

……

……

$[B_{nk}]$ ……

至此我们证明了下面图所示的循环过程。

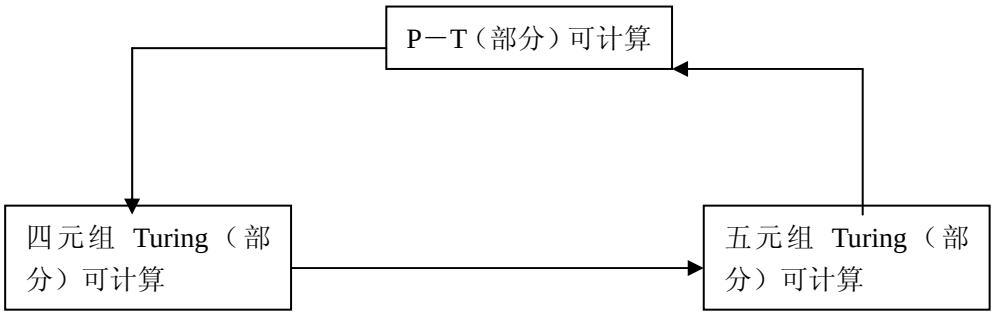


图 4.3

4.5 Post-Turing 程序的数字编码

这一节要建立二符号字母表∑={1，B}上的 P－T 程序的数字编码。这样每个 P－T 程序将对应整数。

设 P－T 程序 P 的指令序列为

R_1，R_2，R_3，…，R_n

且指令R_i的编码为r_i，则程序P的编码定义为序列(r_1，r_2，…，r_n)的哥德尔数［r_1，r_2，…，r_n］，即

P的编码＝［r_1，r_2，…，r_n］

$$2^{r_1}\cdot 3^{r_2}\cdot 5^{r_3}\cdots\cdot P_n^{r_n}$$

指令可带标号，因此指令是对偶（A_i，β_i），其中A_i是标号，β_i是无标号语句。假定在P－T程序中标号都具有上述形式（即A_i），并且i从 1 开始。对无标号情形，我们把β_i视为（A_0，β_i）。这样指令的编码可用配对函数＜x，y＞给出：

（A_i，β_i）编码＝（A_i编码，β_i编码）

A_i的编码定义为整数，无标号指令β_i的编码如下表（其中i≥1）。

指　　令	编　　码
RIGHT	1
LEFT	2
WRITE　1	3
WRITE　B	4
TO　A_i　IF　1	2i+4
TO　A_i　IF　B	2i+3

考虑 P－T 程序

```
[A2]    RIGHT
        TO A1 IF B
        TO A2 IF 1
[A1]    WRITE 1
        RIGHT
        WRITE 1
```

对上面的程序我们有：

指　　令	标号代码	无标号指令代码	指令代码

［A_2］RIGHT	2	1	＜2，1＞＝7
TO A_1 IF B	0	5	＜0，5＞＝20
TO A_2 IF 1	0	8	＜0，8＞＝44
［A_1］WRITE 1	1	3	＜1，3＞＝13
RIGHT	0	1	＜0，1＞＝2
WRITE 1	0	3	＜0，3＞＝9

于是整个程序 P 的编码为：

［7，20，44，13，2，9］

$=2^7 3^{20} 5^{44} 7^{13} 11^2 13^9$

定义 4.5.1 由 P－T 程序得的数称为 P－T 程序的哥德尔数。

由上可知，一旦给定一个 P－T 程序即可确定相应哥德尔数，而且是正的。我们要问，任给一个整数是否都有相应 P－T 程序，即是否任意整数都是某 P－T 程序的哥德尔数？对这个问题的回答是否定的。

设 n 为一个 P－T 程序的哥德尔数，则 n 的素数分解中的指数具有形式：

＜i，j＞

其中i≥0，j>0，因为在一个程序中不能有相同标号，故在上述指数中不能有两个相同的非零左分量，即对于任意两个指数＜i_1，j_1＞、＜i_2，j_2＞来说都有$i_1 \neq i_2$，其中i_1，i_2不为 0，如果一个数的素数分解满足上述条件，那末它定是某P－T程序的哥德尔数。

下面引进一个新的谓词 PROG（x）：若数 x 是某 P－T 程序的哥德尔数，则取真值；否则取假值。

$PROG(x) \Leftrightarrow (\forall i)_{\leq Lt(x)} (i=0 \vee r((x)_i)\neq 0) \wedge \sim(\exists i)_{\leq Lt(x)} (\exists j)_{\leq i} (l((x)_i)= l((x)_j) \wedge l((x_i))\neq 0)$

i相当于程序序号。$(x)_i$相当于第i条程序的编码。

由定义不难看出 PROG（x）是原始递归谓词。

设给定数 17150，则其素数分解为

$17150=2^1 3^0 5^2 7^3$

且

l(1)=1　　r(1)=0

l(2)=0　　r(2)=1

l(3)=2　　r(3)=0

Lt(17150)=4

因此 PROG（17150）＝false，即 17150 不是 P－T 程序的哥德尔数。

4.6 通用程序

现在我们来介绍用原来语言写的一个特殊程序u，称之为通用程序。

通用程序 u 有二个输入变元－－Z，X。它的主要功能是模拟编码为 Z 的 P－T 程序对编码为 X 的输入工作。

特别地，假定 P 是编码为 Z 的 P－T 程序，原 P 程序对带上的 $\bar{x}$ 停机当且仅当通用程序 u 对于 z 和 x 停机，同时计算结果一致。

下面开始构造通用程序 u。

首先建过“带计数器”V 和“指令计数器”I。它们的初值分别为 1：

表示 P－T 程序的指令

V＝1
I＝1

这表示指针放在带的初始位置上，并从每条指令开始执行。

P－T 程序的停机可模拟为

I=0∨I>Lt(Z)

于是通用程序模拟 P－T 程序停机的语句如下：

[A]　TO　G　IF　I=0∨I>Lt(Z)

其中标号 G 表示做结尾工作。

P－T 程序的当前指令的编码为（Z）I。其中标号部分为 l((Z)I)，无标号指令部分为 r((Z)I)。

若 R 是模拟 RIGHT 指令的程序标号，L 是模拟指令 LEFT 的程序标号，F 是模拟指令 WRITE 1 的程序标号，B 是模拟指令 WRITE　孤程序标号，我们有

TO R IF $r((Z)_I)=1$
TO L IF $r((Z)_I)=2$
TO F IF $r((Z)_I)=3$
TO B IF $r((Z)_I)=4$

我们已知

TO A_i IF B 的编码＝2i+3　　(i≥1)

TO A_i IF 1 的编码＝2i+4　　(i≥1)

从而可知，当$r((Z)_l)>4$ 且$r((Z)_l)$为奇数时表示

TO Ai IF B

指令，当$r((Z)_i)>4$ 且$r((Z)_i)$为偶数时表示

TO A_i IF 1

指令。因此，若假定T是模拟指令TO　A_i　IF HB的程序标号，W是模拟指令TO　A_i　IF　1的程序标号，则我们有

```
TO T IF R(r((Z)i)，2)=1
TO W
```

带上的注视符是否B被模拟为是否（X）$_V$＝1。因此T和W的标号的程序部分分别如下：

```
［T］　TO　C　IF　（X）V＝1
　　TO　D
```

```
［W］　TO　C　IF　（X）V＝2
　　TO　D
```

其中C是模拟“黑心A_i”功能的程序部分。主要是使I取相应值。

```
［C］　W＝［（r((Z)I)-S）/Z］
　　I=min{t{l((z)t)=W}
　　t≤Lt(z)
　　TO A
```

因为指令TO　A_i　IF　B的编码的右分量形为 2i+3，因此计算结果正好有W＝1。I取左分量为W的指数序号。该指数对应于以A_i为标号的P－T程序的指令。

D 是执行下一条指令的程序标号。

```
［D］　I＝I＋1
　　TO　A
```

I=0 表示转到没有的标号。下面给出 R，L，FB 的程序部分。

```
［R］TO S IF V≠Lt（X）
      X=X·2
［S］V=V+1
      TO D
```

其中 X=X·2 表示带右边加一个空白符 B，V=V+1 表示(模拟)指针右移一位。

```
［L］ TO M IF V≠1
        X=2·X
        TO D
［M］V=V-1
       TO D
```

其中 X=2·X 模拟在带左边加一空白符 B，V=V-1 则模拟指针左移一位。

```
［F］ TO D IF (X)V=2
       X=X·PV
       TO D
```

```
［B］ TO D IF (X)V=1
       X=[X/PV]
       TO D
```

当$(X)_V \neq 2$，即$(X)_V=1$ 时，做$X=X\cdot P_V$的结果，使得X的素数分解中P_V的指数为 2。这就模拟了往带的注视格上输出 1 的功能。关于［B］与［F］类似。

```
［G］y=#(2，X)-1
```

当 P-T 程序要停机时通用程序 u 执行语句［G］。它将模拟带上有多少个 1 并减 1 后做为计算结果。

综合上述，通用程序 u 如下：

V=1

T=1

［A］ TO G IF I=0 V_I>Lt(Z)

TO R IF r($(z)_I$)=1

TO L IF r($(z)_I$)=2

TO F IF r($(z)_I$)=3

TO B IF r($(z)_I$)=4

TO T IF R($(z)_I$)，2)=1

TO C IF $(X)_V$)=2

［D］ I=I+1

TO A

［C］ W-［$(r(Z)_I$÷3)/2］

I=$\min_{t\leq Lt(z)}${l(z)t}=W}

TO A

［］ TO C IF $(X)_V$=1

TO D

［B］TO D IF $(X)_V$=1

X=［X/P_V］

TO D

［F］TO D IF $(X)_V$=2

X=X·P_V

TO D

［L］TO M IF V

X=2·X

TO D

［M］V=V-1

TO D

```
［R］ TO S IF V≠Lt(X)
      X=X·2
［S］ V=V+1
     TO D
［G］Y=#(2，X) ÷1
```

我们将引进在后面不止一次用到的函数

$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$

这是 n+1 元函数。对于每一个 n，这些函数的定义如下。

定义 4.6.1 若Z是某P-T程序P的哥德尔数，并且程序P计算部分函数$g(X_1，X_2，…，X_n)$，则定义$\Phi^{(n)}(Z，X_1，X_2，…，X_n)=g(X_1，X_2，…，X_n)$

若Z不是任何P-T程序的哥德尔数。或者当g对给定$X_1，X_2，…，X_n$无定义时，$\Phi^{(n)}$对$Z，X_1，X_2，…，X_n$)也是无定义的。

由定义可知$\Phi^{(n}$是部分函数。下面给出关于函数$\Phi^{(n}$的一条重要定理，它被称为枚举定理。

定理 4.6.1(枚举定理) 对每个n部分函数$\Phi^{(n}$是(部分)可计算函数。

证明：为了表明$\Phi^{(n}$是部分可计算函数。我们必须构造计算它的原语言程序。其实所要构造的程序之核心部分是上面给出的通用程序u。

在转入通用程序 u 之前要做两件事。

①首先检查 z 是不是 P-T 哥德尔数？如果不是，则应无穷循环。这一部分的程序可如下表示：

```
［H］TO K IF PROG(Z)
      TO H
```

②其次模拟初始带：

```
［K］ X=TAPE^n(X1，X2，…，Xn)
```

在做上述两步工作后转入通用程序 u，即可计算出哥德尔数为 z 的 P-T 程序的计算结果。

综上所述，计算$y=\Phi^{(n)}(Z，X_1，X_2，…，X_n)$的程序如下：

```
[H]  TO  K  IF  PROG(Z)
     TO  H
     [K]  X= TAPE^n(X1，X2，…，Xn)
                  通用程序
                     u
```

我们再定义在后面反复用到的一个谓词。

定义 4.6.2 计步谓词$STP^{(n)}$定义如下：

$STP^{(n)}(Z，X_1，X_2，…，X_n)$⇔PROG(Z)并且编码为Z的P-T程序对于输入$X_1，X_2，\cdots，X_n$在≤M步内停机。

这里的一步是 P-T 程序的一条指令的一次执行。

定理 4.6.2 (计步定理)

对任意n，谓词$STP^{(n)}(Z，X_1，X_2，…，X_n，M)$是可计算的。

证明：主要用程序u。要给出计算$STP^{(n)}$的程序。该程序的主要思想是

①检查 z 是不是某 P-T 哥德尔数？如果不是，则取假值，即 y=1。

②模拟初始带。

③建立一个步计数器 j，每当执行一条 P-T 指令时 j 加 1，如果 j>M，则取假值，即 y=1，若在 M 步内停机，则 y 自动取②

$STP^{(n)}(Z，X_1，X_2，…，X_n，M)$的计算程序如下

TO G IF ~PROG(Z)

$X=TAPE^{(n)}(X_1，X_2，…，X_n)$

J=0

V=1

I=1

[A] J=J+1

TO G IF J>M

TO E IF I=0 ∨I>Lt(Z)

TO R IF $r((Z)_I)=1$

：

：

[S] V=V+1

TO D

[G] y=1

这一节给出了计算部分可计算函数$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$的原语言程序。根据过去的

定理，我们也可以写出计算$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$的P-T程序，也可构造出计算$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$的Turing机。这就可引出通用Turing机的概念。

这一节给出了枚举定理。之所以称为“枚举”是因为它能枚举所有部分可计算函数。具体说，n+1 元函数$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$将枚举所有n元部分呆计算函数。事实上，任给一个n元部分可计算函数$g(X_1，X_2，…，X_n)$，则必有一个P-T程序P计算它。令Z_0为程序P的哥德尔数，则由$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$函数的定义。我们有

$\Phi^{(n)}(Z_0，X_1，X_2，…，X_n)$

$=g(X_1，X_2，…，X_n)$

4.7 迭代定理

迭代定理给出n取不同值时$\Phi^{(n)}(Z，X_1，X_2，…，X_n)$之间的关系。

定理 4.7.1（迭代定理） 对一切n，有原始递归函数$S^{(n)}(Z，X_1，X_2，…，X_n)$使得对一切m

$\Phi^{(n+m)}(Z，X_1，X_2，…，X_n，v_1，…，v_m)= \Phi^{(m)}(S^{(1)}(Z，X_1，X_2，…，X_n)，v_1，…，v_m)$

证明：施归纳于 n。

首先对n=1 要证明存在原始递归函数$S^{(1)}(Z，X)$使得

$\Phi^{(m+1)}(Z，X，v_1，…，v_m)= \Phi^{(n)}(S^{(1)}(Z，X)，v_1，…，v_m)$

因为$\Phi^{(m+1)}(Z，X，u_1，…，u_m)$是部分可计算函数，必有P-T程序P计算它。程序P的哥德尔数为Z，程序P的初始带是：

$$\overline{x}B\overline{u_1}B\overline{x_2}B\cdots\cdots B\overline{u_m}$$

现在我们需要构造一个 P-T 程序 D，它以

$$\overline{u_1}B\overline{x_2}B\cdots\cdots B\overline{u_m}$$

为初始带，并且计算

$\Phi^{(m+1)}(Z，X，u_1，…，u_m)$

其中 Z 和 X 是确定的。

程序 D 的工作可如下：第一步把初始带

$$\underbrace{1\,1\cdots1\,1}\quad B\quad \underbrace{1\,1\cdots1\,1}\,B\cdots B\,\underbrace{1\,1\cdots1\,1}$$

↑ $\overline{u_1}$ $\overline{u_2}$ $\overline{u_m}$

加工为下面带

1 1…1 1 B 1 1…1 1 B…B 1 1…1 1

X $\overline{u_1}$ $\overline{u_m}$

第二步就执行前面给出的 P-T 程序 P。

D { 带上加 X 的指令部分 | 程序 P 的指令部分 }

显然，若令Z_0为P-T程序Φ的哥德尔数，则

$\Phi^{(m)}(Z_0, u_1, \ldots, u_m) = \Phi^{(m+1)}(Z, X, u_1, \ldots, u_m)$

亦即程序D关于U_1，U_2，…，U_m计算结果与程序P关于X，U_1，U_2，…，U_m的计算结果一致。

现在我们只需要证明存在一个原始递归函数$S^{(1)}$使得

$S^{(1)}(Z, X)=Z_0$

在带左边加 X+1 个 1 的 P-T 指令是

LEFT
LEFT
WRITE 1
LEFT
WRITE 1
:
:
LEFT
WRITE 1

} 2X+2 条指令

这些指令都不带标号。因此指令的编码分别为

LEFT：<0，2>=5

WRITE 1：<0，3>=9

于是这 2X+2 条指令部分的哥德尔数是

$$2^5\cdot3^9\cdot5^5\cdot7^9\cdot11^5\cdots P^5_{2x+1}\cdot P^9_{2x+2}=(\prod_{i=0}^{x}P_{2i+1})^5(\prod_{i=0}^{x}P_{2i+2})^9$$

已知程序 P 的哥德尔数是 Z，故程序 D 的哥德尔数是

$$S^{(1)}(Z，X)=((\prod_{i=0}^{x}P_{zx+1})^5(\prod_{i=0}^{x}P_{zx+2})^9)^*=Z$$

显然$S^{(1)}(Z，X)$是原始递归函数。

设n=k成立，要证n=k+1 时也成立。归纳假设表示原始递归函数$S^{(k)}$使得对一切m

$$\Phi^{(m+k)}(Z，X_1，\cdots，X_k，v_1，\ldots，v_m)=\Phi^{(m)}(S^{(k)}(Z，X_1，\cdots，X_k)，v_1，\ldots，v_m)$$

用 n=1 的结果有

$$\Phi^{(m+k+1)}(Z，X_1，\cdots，X_{k+1}，v_1，\ldots，v_m)$$

$$=\Phi^{(m+k)}(S^{(1)}(Z，X_1)，X_2，\cdots，X_{k+1}，v_1，\ldots，v_m)$$

$$=\Phi^{(m)}(S^{(k)}(S^{(1)}(Z，X_1)，X_2，\cdots，X_{k+1})，v_1，\ldots，v_m)$$

其中S^K是原始递归的。若令

$$S^{k+1}(Z，X_1，\cdots，X_{k+1})=S^{(k)}(S^{(1)}(Z，X_1)，X_2，\cdots，X_{k+1})$$

则显然S^{k+1}是原始递归函数，因此定理成立。

习　题

1 给出计算谓词$x|y$的特征函数的广义 P-T 程序。

2 给出计算函数$\left[x/y\right]$的四元组 TM。

3 给出计算函数$f(x)=x+[x/2]$的四元组 TM。

4 利用四元组 TM 计算函数$\left[\log_3(1+x)\right]$

5 用双向无穷带四元组 TM 计算[$\log_2 n$]（n 为输入）。

6 给出计算谓词$P(x,y)\Leftrightarrow(3x=2y)$的特征函数的五元组图灵机。

第五章　半可计算性

5.1 半可计算谓词和半可计算集合

前面我们介绍了可计算性。可计算性一词可用于函数、集合以及谓词。说一个谓词(集合)是可计算的，如果它的特征函数是可计算的。

半可计算性一词主要用于谓词和集合。在集合论中常常用到递归和递归可枚举概念。在谓词理论中则用到可判定和半可判定概念。

说一个集合 S 是递归的，如果存在一个算法 A，使得对任给的 $\vec{X}$，能够在有限步内回答是否 $\vec{X}\in$S? 即有

$$A(S,\ \vec{X})=\begin{cases}\text{“是”，}\ \text{当}\ \vec{X}\in S\\ \text{发散，当}\ \vec{X}\notin S\end{cases}$$

显然，任何有穷集合都是递归集合。

说一个集合 S 是递归可枚举的，如果存在算法 A，使得对任何 $\vec{X}$ 若 $\vec{X}\in$S，则在有限步内给出回答“是”，若 $\vec{X}\notin$S，则算法不能给出回答，即算法不能终止。

$$A(S,\ \vec{X})=\begin{cases}\text{“是”，}\ \text{当}\ \vec{X}\in S\\ \text{发散，当}\ \vec{X}\notin S\end{cases}$$

我们不难看出集合的可计算性和递归性是一回事，而半可计算性将要与递归可枚举性对应。说一个谓词 P($\vec{X}$)是可判定的，如果存在一个算法 A，使得对任给的 $\vec{X}$，能够在有限步内回答 P($\vec{X}$)是“真”还是“否”即有

$$A(P,\ \vec{X})=\begin{cases}\text{“是”，}\ \text{当 P}(\vec{X})\text{真}\\ \text{否，当 P}(\vec{X})\text{假}\end{cases}$$

说一个谓词 P($\vec{X}$)是半可判定的，如果存在一个算法 A（尽管存在别的算法 B 使 P($\vec{X}$)可判定，但这时也认为 P($\vec{X}$)不可判定，所以可判定包含于不可判定之中），使得对任给 $\vec{X}$，如果 P($\vec{X}$)为真，则在有限步内回答“是”，否则算法不终止(即不能给出回答)，即有

$$A(P,\ \vec{X})=\begin{cases}\text{“是”，}\ \text{当 } P(\vec{X}) \text{真}\\ \\ \text{发散，当 } P(\vec{X}) \text{假}\end{cases}$$

不难看出谓词的可性与可判定性是一回事，谓词的半可计算性则与半可判定对应。

在定义半可计算性之前，首先引进一种记号

$f(x_1, x_2, \ldots, x_n)\downarrow$

这表示函数f对于$(x_1, x_2, \ldots, x_n)$有定义，如果函数f对于$(x_1, x_2, \ldots, x_n)$无定义，则表示为

$f(x_1, x_2, \ldots, x_n)$？？

这样可把$f(x_1, x_2, \ldots, x_n)\downarrow$视为谓词，$f(x_1, x_2, \ldots, x_n)\downarrow$取真值当且仅当函数f对于变元组 $(x_1, x_2, \ldots, x_n)$有定义。显然

$D_f=\{(x_1, x_2, \ldots, x_n)|\ f(x_1, x_2, \ldots, x_n)\downarrow\}$

表示函数 f 的定义域。

半可计算性的定义方式有几种，下面是其中的一种方式。

定义 5.1.1 谓词$P(x_1, x_2, \ldots, x_n)$称为半可计算的，如果存在一部分可计算函数$g(x_1, x_2, \ldots, x_n)$使得下式成立

$P(x_1, x_2, \ldots, x_n)\Leftrightarrow g(x_1, x_2, \ldots, x_n)\downarrow$

这实际上是用部分可计算函数的定义域来定义了谓词的半可计算性。就是说，一个谓词是半可计算的，当且仅当它的真值域等于某部分可计算函数的定义域。

上面的定义也等价于下面的说法：说谓词是$P(x_1, x_2, \ldots, x_n)$是半可计算的，当且仅当存在一个程序P_1，使得

$P(x_1, x_2, \ldots, x_n)\Leftrightarrow P_1(x_1, x_2, \ldots, x_n)$停机

定义 5.1.2 一个集合S是半可计算的，如果存在一个部分可计算函数$g(x_1, x_2, \ldots, x_n)$，使得

$(x_1, x_2, \ldots, x_n)\in S\Leftrightarrow g(x_1, x_2, \ldots, x_n)\downarrow$

亦可定义为：S 是半可计算的当且仅当它的特征谓词是半可计算的。

只要回顾一下关于谓词的半可判定定义和集合的递归可枚举定义，即可认识到用函数的定义域来定义半可计算性是很自然的事情。

在我们的可计算性理论中，多元和一元没有本质的差别，在此我们主要考虑的是一维的集合 S⊆N

例如，假定给了n元谓词$P(x_1，x_2，…，x_n)$可引进新的一元谓词P(X)，使得P的判定问题变为$\widetilde{P}$的判定问题，$\widetilde{P}(X)$可定义如下：

$\widetilde{P}(X)=P((X)_1，(X)_2，…，(X)_n)$

又假设 $S⊆N^n(n>1)$，则这 n 维集合的判定问题亦可变一维集合$\widetilde{S}$的判定问题。$\widetilde{S}$可定义如下：

$\widetilde{S}=\{X|((X)_1，(X)_2，…，(X)_n)∈S\}$

事实上我们有

$(x_1，x_2，…，x_n)∈S⇔(([x_1，x_2，…，x_n])_1，([x_1，x_2，…，x_n])_2，⋯，([x_1，x_2，…，x_n])_n)∈S⇔[x_1，x_2，…，x_n]∈\widetilde{S}$

$P[x_1，x_2，…，x_n]$

$⇔P(([x_1，x_2，…，x_n])_1，⋯，([x_1，x_2，…，x_n])_n)$

$⇔\widetilde{P}([x_1，x_2，…，x_n])$

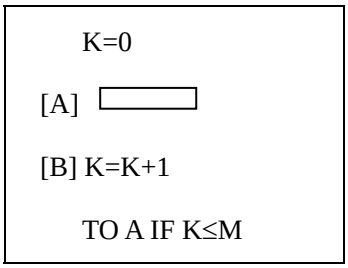
下面给出一定理，该定理表示半可计算的整数域亦可用部分可计算函数的值域来定义。就是说 ，任一半可计算整数集都是某部分可计算函数的值域；反之，任一部分可计算函数的值域是半可计算整数集。显然，对多维集合来说，我们无法用值域来定义，因为值域是一维的整数集。

定理 5.1.1 整数集H是半可计算的当且仅当存在某部分函数$f(x_1，x_2，…，x_n)$使得

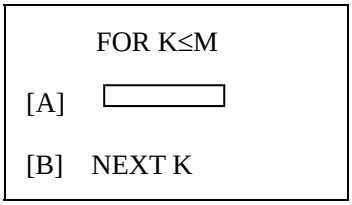
$H=\{f(x_1，x_2，…，x_n)|f(x_1，x_2，…，x_n)↓\}$

换言之半可计算集是某部分可计算函数的值域，反之亦然。

证明　为了方便起见，把下面循环程序



简写成如下：



下面开始证明定理。设f $(x_1，x_2，…，x_n)$是部分可计算函数，且令R为f $(x_1，x_2，…，x_n)$的值域。

$f(x_1，x_2，…，x_n)$

$=\Phi^{(n)}(z_0，x_1，x_2，…，x_n)$

我们要证明 R 是半可计算集，为此我们只需证明存在一个部分可计算函数 g(X)使得

$R=\{X|g(X)\downarrow\}$

即存在一个程序使得

$x\in R\Leftrightarrow$程序对 X 停机。

程序如下：

```
[A]   FOR W1≤K
      FOR W2≤K
        :
        :
      FOR Wn≤K
      FOR M≤K
      U=STP(n)(Z0，W1，…，Wn，M)
      TO B IF U≠0
      V=f(W1，…，Wn)
      TO E IF X=V
[B]   NEXT M
      NEXT Wn
        :
        :
      NEXT W1
      K=K+1
      TO A
```

$$g(x)=\begin{cases}0 & \text{若}(\exists y)\{x=f(y)\}\\ \text{无定义}\end{cases}$$

注意：如何判定$(\exists y)\{x=f(y)\}$成立是一个难处理的问题。

如果依次增加y，并判断x=f(y)就出现某个y_0使得$f(y_0)$无定义，从而即使有x=f(y')也不能得到此结果，因为在计算$f(y_0)$时将无限循环。

这个程序的定义域等于函数$f(x_1，x_2，…，x_n)$的值域。

($\Rightarrow$)假设 H 是半可计算集。若 H=Φ，则只要令 f(x)为对任意 x 无定义的部分可计算函数，则显然有 $H=\{f(x)|\ f(x)\downarrow\}$

要证明 有一个部分可计算函数 f 值域为 H。考虑下面程序：

U=STP(Z_0，X，M)

TO A IF U≠0

y=x

TO E

或[A] TO A

[A] y=x_0(x_0∈g的值域

$$g(x)=\begin{cases} x & \text{当}(\exists y)(h(x)=y) \\ \text{无定义} & \end{cases}$$

$$=\begin{cases} x & \text{当}(\exists M)STP(z_0，x，M) \\ \text{无定义} & \end{cases}$$

$$g(x，M)=\begin{cases} x & \text{当}(\exists M)STP(z_0，x，M) \\ \text{无定义} & \end{cases}$$

这里h(x)与H相对应，z_0与h(x)相对应。该程序的输入变量为X和M。设它所计算的部分可计算函数为f(X_0，M)。下面证明它就是所要找的函数，即有

H={f(x，M)| f(x，M)↓}

显然，f 的值域小于等于 H，即 R⊆H (R 值域)。

下面证明R⊇H。假充X∈R，则有M使STP(z_0，x，M)=0。这时由上述程序不知道f(x，M)=x ，因此有R⊇H，于是得所要结论。

H={f(x，M)| f(x，M)↓}

5.2 半可计算的一些封闭性

这一节主要研究半可计算性对于哪些运算是封闭的。

定理 5.2.1 若 P($\vec{X}$)和 Q($\vec{X}$)是半可计算的，则

$P(\vec{X})\wedge Q(\vec{X})$

亦为半可计算谓词。

证明： 因为 $P(\vec{X})$和 $Q(\vec{X})$是半可计算的，故有部分可计算函数 $f(\vec{X})$和 $g(\vec{X})$使得

$P(\vec{X})\Leftrightarrow f(\vec{X})\downarrow$

$Q(\vec{X})\Leftrightarrow g(\vec{X})$

为了证明定理必须指出确有一个部分可计算函数 $h(\vec{X})$使得

$P(\vec{X})\wedge Q(\vec{X})\Leftrightarrow h(\vec{X})\downarrow$

亦即存在一个程序 R 使得

$P(\vec{X})\wedge Q(\vec{X})\Leftrightarrow$R 对于 $\vec{X}$ 停机

计算 $y= h(\vec{X})$的程序如下：

```
Z=f(X)
Y=g(X)
```

显然，假设 $P(\vec{X})\wedge Q(\vec{X})$真，则 $P(\vec{X})$和 $Q(\vec{X})$为真，于是计算 $f(\vec{X})$和 $g(\vec{X})$的程序集，故计算 $h(\vec{X})$的程序停机。反之，$P(\vec{X})\wedge Q(\vec{X})$为假，则 $P(\vec{X})$或 $Q(\vec{X})$为假。于是 $f(\vec{X})$或 $g(\vec{X})$程序不停机，故 $h(\vec{X})$的程序亦不停机，即 $h(\vec{X})\downarrow$不成立。

定理 5.2.2 若 $P(\vec{X})$和 $Q(\vec{X})$是半可计算谓词，则

$P(\vec{X})\vee Q(\vec{X})$

亦为半可计算谓词。

证明： 因为很像前一定理，所以可能认为类似。事实并非如此。证明要比前定理难些。若用相仿方法证明。则由于 P(X)和 Q（X）半可计算。就有部分 $f(\vec{X})$和 $g(\vec{X})$使得

$P(\vec{X})\Leftrightarrow f(\vec{X})\downarrow$

$Q(\vec{X})\Leftrightarrow g(\vec{X})\downarrow$

我们要有函数 $h(\vec{X})$，使得当且仅当 $f(\vec{X})\downarrow$或 $g(\vec{X})\downarrow$时 $h(\vec{X})\downarrow$。我们不能用先计算 f()然后

再计算 g($\vec{X}$)的办法，因为 f($\vec{X}$)是半可计处的，如果 f($\vec{X}$)不成立，那么计算过程总也不能终，从而也无法去试 g($\vec{X}$)。由于以上原因，我们将采用交错计算法，即先执行 f($\vec{X}$)程序 M 步，然后执行 g($\vec{X}$)程序 M 步，若两个程序都没有停止，则 M＝M＋1，然后财重复上述过程。

因为f($\vec{X}$)和g($\vec{X}$)是部分可计算函数，因此由枚举定理有整数Z_1和Z_2使得

Φ（Z1，$\vec{X}$）＝f($\vec{X}$)

Φ（Z1，$\vec{X}$）＝g($\vec{X}$)

(注意Φ没有上标，因为不知是$\vec{X}$多长)。

下面开始写出交错计算 f($\vec{X}$)和 g($\vec{X}$)的程序。在这里用到一个计数器 M，它从 0 开始依次增加 1（计步数）。对每个 M 先检查 f($\vec{X}$)程序是否在 M 步内停机，然后再检查 g($\vec{X}$)程序是否在 M 步内停机。

如果 P（$\vec{X}$）∨Q（$\vec{X}$）为真，则必有 f($\vec{X}$)或 g($\vec{X}$)的程序使其在某 M 步内停机。

计算 y=h(X)的程序如下：

```
      M＝0
［A］U1＝STP（Z1，X，M）
      U2＝STP（Z2，X，M）
      TO  E  IF  U1＝0
      TO  E  IF  U2＝0
      M＝M＋1
      TO  A
```

若函数 f($\vec{X}$)的程序在 M 步内停止，则 U1 值为 0，否则为 1。类似地，如果函数 g($\vec{X}$)的程序在 M 步内停止，则 U2 值为 0，否则为 1。

显然，

P（$\vec{X}$）Q（$\vec{X}$）⇔

f($\vec{X}$)↓或 g($\vec{X}$)↓⇔h($\vec{X}$)

故谓词 P（$\vec{X}$）Q（$\vec{X}$）是半可计算的。

类似的，对集合有以下定理。

定理 5.2.3 如果S_1和S_2是半可算集，则

亦是半可计算的。

定理 5.2.4 谓词 H（$\vec{X}$）是半可计算的当且仅当存在一个呆计算谓词 C（$\vec{X}$，y）使得

H（$\vec{X}$）⇔（∃y）c($\vec{X}$,y)

证明（⇔）（∃y）c($\vec{X}$,y)

其中 C（$\vec{X}$,y）是可计算的。要证明 H（$\vec{X}$）是半可计算的，即要找一个程序 h($\vec{X}$)使得

H（$\vec{X}$）⇔程序 h（$\vec{X}$）对于 $\vec{X}$ 停机。

程序 h（$\vec{X}$）可如下：

```
[A]  U＝C（X,y）
       为可计算谓词的特征函数
       TO  E  IF  U＝0
       y=y+1
       TO A
```

这程序逐次对 y=0，1，2，…，验证 C（$\vec{X}$，y）是否为真，因为 C（$\vec{X}$,y）是可计算的，程序的第一语句是合法的。因此可计算出 U 值。若一个 y 使 C（$\vec{X}$，y）为真，则可找到这 y 且停机，否则程序永远继续下去（不停机）。显然有

H（$\vec{X}$）⇔程序 h（$\vec{X}$）对 X 停机

故 H（$\vec{X}$）是半可计算的。

（⇒）设 H($\vec{X}$)是半可计算谓词。证明存在一个可计算谓词 C($\vec{X}$,y)使得

H($\vec{X}$)⇔（∃y）C($\vec{X}$，y)

因为 H（X）是半可计算的，必有部分函数 h($\vec{X}$)使得

H($\vec{X}$)⇔h($\vec{X}$)↓

由枚举定理有Z_0使得

Φ(Z0，$\vec{X}$)＝h($\vec{X}$)

因此有

H($\vec{X}$)⇔Φ（Z0，$\vec{X}$）↓

Φ（Z0，$\vec{X}$）↓的意思是说程序 Z0（哥德尔数）对于$\vec{X}$在步内停机，即有

H($\vec{X}$)⇔(∃M)STP(Z0，$\vec{X}$,M)

正确写法为：

因为 STP 是可计算的，因此结论成立。

定理 5.2.5　设 H1(V，$\vec{X}$)是半可计算谓词则

(∃V)H(V，$\vec{X}$)

亦是半可计算的。即半可计算性封闭于运算。

证明（1′）因为 Hr 是半可计算谓词，由前定理有一可计算谓词 C（W，V，$\vec{X}$）使得

H1（V，$\vec{X}$）⇔（∃W）C（W，V，$\vec{X}$）

即有

（∃V）H（V，$\vec{X}$）⇔（∃V）（∃W）C（W，V，$\vec{X}$）

用配对函数我们

（∃V）H1（V，$\vec{X}$）⇔（∃Z）C（l（Z），r（Z），$\vec{X}$）

其中 l,r 和 C 都是可计算的。因此 C（l(Z),r($\vec{X}$)）是可计算的。由前定理可知（∃V）H1（V，$\vec{X}$）是半可计算的。

证明（2）令

H1（V，$\vec{X}$）⇔Φ（Z0，V，$\vec{X}$）↓

要找一个程序 h($\vec{X}$)使得

（∃V）H1（V，$\vec{X}$）⇔程序 h($\vec{X}$)对于 X 停机。

H($\vec{X}$)程序如下：

```
［A］  FOR  V≤K
       FOR  M≤K
       U＝STP（Z0，V，$\vec{X}$，M）
       TO  E  IF  U＝0
       NEXT  M
       NEXT  V
       K＝K＋1
       TO  A
```

本定理表示半可计算谓词类在存在量词下封闭。即么对全作称量词（∀y）也是封闭呢？将可知是不封闭的，但可以证明半可计算性封闭囿全称量词，（∀y）≤z。

定理 5.2.6　设 H（V，$\vec{X}$）是半可计算谓词，则

（∀V）≤H（V，$\vec{X}$）

是半可计算的。

证明（1）因为 H（V，$\vec{X}$）是半可计算谓词。有部分可计算函数 h（V，$\vec{X}$）使得

H(V，$\vec{X}$)⇔h(V，$\vec{X}$)↓

我们要证明有部分可计算函数 g（Z，$\vec{X}$）使得

（∀V）≤ZH(V，$\vec{X}$)⇔g(Z $\vec{X}$)

计算 g（Z，$\vec{X}$）的程序是

```
[A]  U=h(V，$\vec{X}$)
     TO E IF V=Z
     V=V+1
     TO A
```

如果没 Z 的限制，将无法判断（VV）H（V，Z）为真。

程序时，Z，$\vec{X}$ 停机⇔h（V，$\vec{X}$）程序对 X，V＝0，1，…，Z 停机

⇔H（V，$\vec{X}$）对 X，V＝0，1，…，X 级真值

⇔（∀V）H（V，$\vec{X}$）真。

故（∀V）H（V，$\vec{X}$）是半可计算的。

证明（2）设 H（V，X）是半可计算的，于是有可计算谓词 C（W，V，$\vec{X}$）使得

H（V，$\vec{X}$）⇔（∃W）C（W，V，$\vec{X}$）

故有

（∀V）≤ZH（V，$\vec{X}$）⇔（∀V）≤Z（∃W）C（W，V，$\vec{X}$ ）

我们想证明

（∀V）≤Z（W）C（W，V，$\vec{X}$）⇔（∃W）（∀V）≤Z（W，V，$\vec{X}$）

如果能证明这一点，那末由可计算谓词对受囿量诩的封闭性以及半可计算性对于存在量诩的封闭性，即可得出我们的结论。

下面证明上述结论。

（∀V）≤Z（∃W）C（W，V，X）真表示对任何 0≤V≤Z_0都存在相应W，使得C（W，V，$\vec{X}$）为真。假设对应关系如下：

V	0　1　2 ……Z
W	W_0 W_1 W_2…… W_Z

这时我们有

（∀V）≤Z（∃W）C（V，V，$\vec{X}$）⇔存在$W_0W_1…W_Z$使C（Wi，i，$\vec{z}$）真，（0≤i≤x）

⇔存在一个 W 使 W＝［W0，W1，…，Wz］C（W）i+1,i,lz），（0≤i≤z）

⇔(∃W)(∀V)≤ZC((W)v+1V, $\vec{z}$)

这就证明了我们的断言。

5.3 半可计算性与可计算性

这一节主要给出一谓 P（$\vec{X}$）是可计算的充要条件，同时指出一特殊谓词 K（$\vec{X}$）是半可计算但不是可计算的。也缎带出连半可计算都不是的谓词。

定理 5.3.1 若 P（$\vec{X}$）是可计算的，则它是半可计算的。

证明 要证明 P（）是半可计算的，就要找一部分可计算函数 h()使得

P（$\vec{X}$）⇔h($\vec{X}$)↓

也就是要构造一个程序 Q 使得

P（$\vec{X}$）⇔Q 对 $\vec{X}$ 停机

这个程序 Q 就是：

```
［A］ y=P'( X⃗ )
      TO A IF y=1
```

其中 P（$\vec{X}$）是可计算的，因此 y=p`($\vec{X}$)是合法的。这里 P（$\vec{X}$）是宾特征函数。为方便起见，以后直接把谓词本身写在程序中。

定理 5.3.2 若 P（$\vec{X}$）是可计算的，则～P（$\vec{X}$）半可计算。

证明 下面程序 Q 具有性质

～P（$\vec{X}$）⇔Q 对 $\vec{X}$ 停机

```
[A]  y=p(X)
     TO A IF y=0
```

定理 5.3.3 （post）

P（$\vec{X}$）是可计算的当且仅当 P（$\vec{X}$）和～P（$\vec{X}$）是半可计算的。

证明（⇔）

由前定理已知：若 P（$\vec{X}$）是可计算的，则 P（$\vec{X}$）是半可计算的，另外，～P（$\vec{X}$）是可计算的。由此可推出～P（$\vec{X}$）也是半可计算的。

（⇔）设 P（$\vec{X}$）和～P（$\vec{X}$）是半可计算的。即有部分可计算的函数 f（$\vec{X}$）和 g($\vec{X}$)使得

P（$\vec{X}$）⇔f($\vec{X}$)↓

$\sim P(\vec{X}) \Leftrightarrow g(\vec{X})\downarrow$

下面证明存在一个程序 Q 使得它计算函数 h：

$$h(\vec{X})=\begin{cases}0 & \text{当 P（}\vec{X}\text{）时}\\ 1 & \text{当}\sim\text{P（}\vec{X}\text{）时}\end{cases}$$

显 $h(\vec{X})$是 P（$\vec{X}$）的特征函数。因此如果确有上述程序则 P（$\vec{X}$）是可计算的。事实上，这个程序可如下：

```
      M=1
[A]   U1=STP(Z1, X⃗ ,M)
      U2=STP(Z2, X⃗ ,M)
      TO E IF U1=0
      TO C IF U2=0
      M=M+1
      TO A
[C]   y=1
```

其中Z_1是计算$f(\vec{X})$程序的哥德尔数，Z_2是计算$g(\vec{X})$ 哥德尔数，对任意输入$\vec{X}$，在计算$f(\vec{X})$和$g(\vec{X})$的程序中必有一个程序是要停机的，因此上述程序对任意$\vec{X}$都是停机的，并且计算函数$h(\vec{X})$。

定义 5.3.1 一变元谓词 K（X）定义为

K（X）⇔Φ（X，X）↓

这表示 K（X）真当且仅当哥德尔数为 X 的 P－T 程序对输入 X 停机。

定理 5.3.4 谓词 K（X）是半可计算的，但不是可计算的。

证明 已知$\Phi^{(1)}$（Z，X）是部分可计算的。故Φ（X，X）是部分可计算的定义。

K（X）⇔Φ（X，X）↓

K（X）是半可计算的。

下面证明 K（X）不是可计算的。假设是可计算的，则～K（X）是半可计算的，因此必有部分可计算函数 g(X)使得

$$\sim K(X) \Leftrightarrow g(X)\downarrow$$

由枚举定理有某Z_0使得

$$g(X)= \Phi(Z_0, X)\downarrow$$

从而

$$\sim K(X) \Leftrightarrow \Phi(Z_0, X)\downarrow$$

令$X=Z_0$，则有

$$\sim K(Z_0) \Leftrightarrow \Phi(Z_0, Z_0)\downarrow$$

但由 K（X）的定义有

$$K(Z_0) \Leftrightarrow \Phi(Z_0, Z_0)\downarrow$$

这就产生矛盾

$$\sim K(Z_0) \Leftrightarrow K(Z_0)$$

因此我们假设“K（X）是可计算的”不成立。

推论 ～K（X）不是半可计算的。

因为如果～K（X）是半可计算的，则 K（X）就是成为可计算的，而这是不可能的。

本定理表示确有半可计算但不是可计算的谓词，即半可计算谓词的集合真包含可计算谓词集。

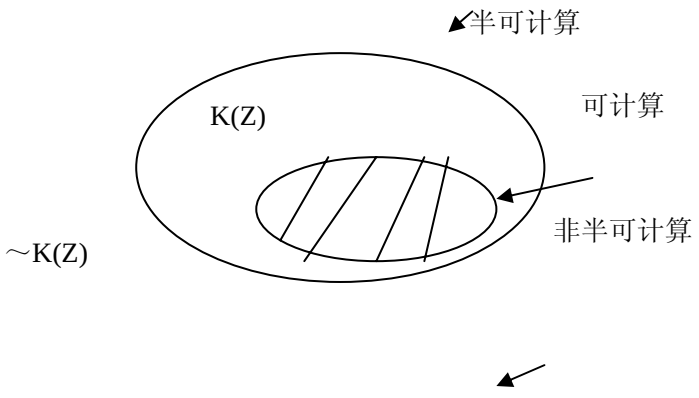


图 5.1

定理 5.3.5（图形定理）

函数$f(X_1,X_2,\ldots,X_n)$部分可计算⇔

谓词$V=f(X_1,X_2,\ldots,X_n)$是半可计算的。

证明（⇒）令

$f(X_1,X_2,\ldots,X_n)=\Phi n(Z_0, X_1,X_2,\ldots,X_n)$

$$\varphi(X_1,X_2,\ldots,X_n,V)=\begin{cases}0 & \text{若}V=f(X_1,X_2,\ldots,X_n)\\ 1 & \text{否则}\end{cases}$$

只要证明φ是部分可计算的。

计算φ的程序如下：

```
［A］  U＝STP（n）（Z0，X1，…,Xn,M）
       TO B IF U=0
       W=f(X1,…,Xn)
       TO K IF W=V
[B]    M=M+1
       TO A
```

（⇐）设$V=f(X_1,X_2,\ldots,X_n)$是半可计算谓词，则有一可计算谓词C（V，X，Z）使得

$V=f(\vec{X})\Leftrightarrow(\exists z)C(V,\vec{X},Z)$

用 C 程序可给出计算 f 的程序。

```
[A]  FOR y≤K
       FOR Z≤K
       U=C(y, X⃗ ,Z)
       TO E IF U≠0
       NEXT Z
       NEXT y
       K=K+1
       TO A
```

定义 5.3.2　定义集合W_Z为

集W_Z表示程序Z（哥德尔数）的定义域，即使得哥德尔数为Z的P－T程序机停的所有那些X的集合。

定理 5.3.6　半可计算集之集是可数（可枚举）的，即可把半可计算集排成如下序列

H_1，H_2，H_3，……

其中每个H_i是半可计算集。

证明　事实上，下面序列就是我们所要的排列

W_0，W_1，W_2，W_3,……

其中每个 Wi 是前面所定义的集合。

我们已知每个W_i是半可计算的。设H是一半可计算集，则必有Z_0使得

$$\tilde{H} = \Phi（Z', X）\downarrow$$

即　　$\tilde{H} = WZ'$

故任一半可计算集都在上述排列之中，因此定理成立。

那么一切整数集之集是否也可枚举呢？对这个问题的回答是否定的。

定理 5.3.7　整数集之集是不可枚举的，即不能把整数集之集排成如下序列

S_0，S_1，S_2，S_3……

其中 Si 是整数集。

证明：（康托方法）假设是可排列的：

S_0，S_1，S_2，S_3……

则将推出矛盾。具体方法是构造一个 S'，使得它不含于上述排列中，用到康托的对角线方法，即定义集合

$S'=\{n|n\notin S_n\}$

下面证明对一切 i 使得

$S'\neq Si$

假若不是，则必有某i0 使$S'=S_{i0}$，于是有

$n\in S_{i0}\Leftrightarrow n\notin S'$（n任意）

若令 n＝1，则有

$i_0\in S_{i0}\Leftrightarrow i_0\in S_{i0}$

显然这是矛盾的。故定理成立。

我们亦可用这种对角线方法来证明～K（X）不是半可计算的。事实上，我们有

$K(X)\Leftrightarrow X\notin W_X$随∑而变

$\sim K(X)\Leftrightarrow X\notin W_X$

～K(X) 半可计算$\Leftrightarrow\{X|X\notin W_X\}$半可计算

但由对角线方法已知$\{X|X\notin W_X\}$不属于序列

W_0，W_1，W_2，W_3……

之中，而这序列是半可计算集的枚举，因此上面集合$\{X|X\notin W_X\}$不昱可计算集集，故～K（X）不是半可计算集。

最后指出半可计算集不封闭于全称量词。

已知 K（X）是半可计算的，因此有一可计算谓词 C，使得

K（X）⇔（∃y）C(X,y)

进而有

～K（X）⇔～（∃y）C(X,y)

～K（X）⇔（∀y）～C(X,y)

因 C（X,y）是可计算的，～C（X,y）亦为可计算的，从而也是半可计算的，若半可计算谓词封闭于全称量词，则出～K（X）也是半可计算的。但这是不可能的，因此半可计算谓词不封闭于全称量词运算。

习 题

1 给出一个不是半可计算的谓词。

2 给出一个不是半可计算的谓词。

第六章　半图厄系统

6.1 半图厄系统

现在考虑处理符号串一种系统，这部分的内容和形式语言理论有密切联系。每个半图厄（Semi-Thue）系统都定义在某字母表上

D＝｛$a_1,a_2,\ldots,a_n$｝

D 上的字是指字母表中符号的有穷序列。

定义字母表 D 上的半图厄处理是开始下面的产生式集合：

$$\beta\rightarrow\overline{\beta}_i \quad (i=1,2,\ldots,m)$$

其中β_i和$\overline{\beta}_i$是D上的字。

例如：产生式集

1 ab→aa

2 ba→bb

3 bb→ab

组成一个半图厄处理。

半图厄处理给出了符号串的推导方法。例 1，对符号串 aba 我们有

$$aba \overset{1}{\Longrightarrow} aaa$$

$$aba \overset{2}{\Longrightarrow} abb \overset{1}{\Longrightarrow} aab \Rightarrow aaa$$

$$aba \overset{2}{\Longrightarrow} abb \overset{3}{\Longrightarrow} aab \Rightarrow aaa$$

若有串$\omega_1,\omega_2,\ldots,\omega_n(n\leq 1)$，使得

$\omega_1\Rightarrow\omega_2\Rightarrow\ldots\Rightarrow\omega_n$

则记为

$\omega_1 \overset{0}{\Longrightarrow} \omega_n$（现在只假定此式只对初始字成立）

对任意字ω，我们定义$\omega \overset{0}{\Longrightarrow} \omega_0$

（我们认为在每个半图厄处理中有一个产生式∅→∅）

定义 6.1.1，设σ是半图厄处理，若产生式β→$\overline{\beta}$在σ中，则产生式$\overline{\beta}$→β也定在σ中，那么称半厄处理σ为图厄处理。

例如，下面是一个图厄处理

1 ab→aa

2 ba→bb

3 aa→ab

4 bb→ba

定义 6.1.2 （一半）图厄系统是（一半）图厄处理加上一个字，A 称为这个系统的公理或初始字。（半）图厄系统可记做

π＝（P•A）

其中 P 是（半）图厄处理，A 是公理。由公理 A 可推出的符号 W 称之为定理。因为有 A⇒A，因此公理是一个定理。

定义　说半图厄系统是单演的，若对系统产生的每个字W最多有一字U使得W⇒U_0（当然可以没有一个字U）

一个半图厄系统π＝（P，A）也可能无穷地进行替换，例如

π= ((a→aa),a)

aa→ca

bb→ab

abb⇒aab

abb⇒aab

这也是单演的，尽管对一个字可以用一个以上的产生式，但推导结果都相同。

将如下无穷做下去

这个定义指出推导是确定的。如果加一些限制，比如推导时只能替换最左出现，（注把产生式编号）那么非单演的，也是确定的。

a⇒aa⇒aaa⇒aaaa⇒…

这个系统不但有无军多个定理且是单演的。

6.2 用半图厄系统模拟图灵机

我们已知四元组形式的图灵机有三种不同类型的四元组指令：

1 $q_iS_jS_{Kq}l$

2 $q_iS_jR_ql$

3 $q_iS_jL_ql$

下面我们来用半图厄系统模拟给图灵机 M。即对给定图灵机 M 构造和 M 密切相关的半图厄系统*。

*的字母表是:

1 图灵机M的字母表$\{S_0,S_1,\ldots,S_k\}$

2 图灵机M的状态符$\{q_1,q_2,\ldots,q_N\}$

3 再加三个新符{h,q,q'}

设有图灵机 M 有一状况

$$S_1S_2S_0S_1S_2S_3$$

$$q_4$$

则可把它表示成如下的一个字

$$hS_2S_3S_0q_4S_1S_2S_3S_1h$$

这种字称之为波斯特字。

图灵机的工作过程也就是状况的变化过程，也就是波斯特字的变化过程。于是可以想像用半图厄系统来模拟这个过程。我们将用 (M)表示所要造的半图厄系统的处理(即产生式表)。

具体方法如下：

(1)对于形如$q_iS_jS_{Kq}l$的图灵机指令，在∑(M)中放入产生式

$q_iS_j\rightarrow q_lS_k$

(2)对于形如$q_iS_jR_ql$的图灵机指令，在∑(M)中放入产生式

$q_iS_jS_m\rightarrow S_jq_lS_m$ (m=0,1,…,k)

$q_iS_jh\rightarrow S_jq_lS_0h$

其中S_0是空白符。

(3)对于形如$q_iS_jL_ql$的图灵机指令，在∑(M)中放入产生式

$S_mq_iS_j\rightarrow q_lS_mS_j$ (m=0,1,…,k)

$hq_iS_j\rightarrow hq_lS_0S_j$

(4)对每个“非法”的q_iS_j，在∑(M)中加上产生式，在四元组中不出现，这表示“停机”

$qiSj \rightarrow qRj$

(5)在∑(M)中加上以下 2k+3 个产生式

$qS_1 \rightarrow q$ (I=0,1,…,k)

$qh \rightarrow q'h$

$S_1q' \rightarrow qi$ (I=0,1,…,k)

这里第一个产生式的作用是擦去 q 右边的一切符号(h 除外)，第三个产生式的作用是擦去 q'左边的一切符号(h 除外)。

∑(M)最后半导出 hqh 字，这时再也不能用产生式业推导了。例如，当前的字为

$hS_1S_2S_5q_2S_4h$

并且在M中没有以q_2S_2为头的四元维，则下面的推导过程如下：

$\Rightarrow hS_1S_2S_5q_3S_2S_4h$

$\Rightarrow hS_1S_3S_3qS_2S_4h$

$\Rightarrow hS_1S_2S_5qS_4h$

$\Rightarrow hS_1S_3S_5qh$

$\Rightarrow hS_1S_2S_5q'h$

……

$\Rightarrow hq'h$

由上述构造不难证明：对任意 postω,∑(M)中最多有一产生式可用ω，于是，∑(M)是演的。

Turing 机的初始带是

111…11

q1↑x+1 个

它对应于 post

hq_1 111…11h

x+1 个

由∑(M)产生式的逆组成的半图厄处理称为Ω(M)。M 和∑(M)及Ω(M)之间有下面关系。

引理　1　M对X停机$\Leftrightarrow hq_1Xh \overset{*}{\underset{\Sigma(M)}{\Leftrightarrow}} hq'h$

2　M对X停机$\Leftrightarrow hq'h \overset{*}{\underset{\Omega(M)}{\Leftrightarrow}} hq_1Xh$

6.3 半图厄系统和半可计算集合

这研究半图厄系统和半可计算集之间的关系。设 A 是一个字母表，则通过编码可使 A 上的每个字对应一整数。因此，字母的集合可视为整数集。这样半可计算概念亦可用于字的集合。

给定 Turnig 机 M，则可以构造半图厄处理∑(M)和Ω(M)。下面再定义半图厄处理了Γ(M)：

$$\Gamma(M)=\Omega(M)\cup\begin{cases} hq1\rightarrow r \\ r11\rightarrow 1r1 \\ r1h\rightarrow 1 \end{cases}$$

新加的三个产生式用于擦去post字hq中的和hq11111…1h中的hq_1和h。

$hq111\ldots11h\Rightarrow r11\ldots11h$

$\Rightarrow 1r1\ldots11h$

……

$\Rightarrow 11\ldots r1h$

$\Rightarrow 11\ldots1$

定义 6.3.1 设 M 为任一图灵机，则用Φ(M)表示使 M 停机的所有 X 的集合。

定义 6.3.2 设π=(P，A)为一半图厄系统，则用 T(π)表示π的所有定理的集合。

定理 6.3.1 对任一 Turning 机 M，都有一半图厄系统π，使得

$$X\in\Phi(M)\Leftrightarrow \overline{X}\in T(\pi)$$

注意，定理的结论不是Φ(M)=T(π)。这里 X 是整数，而 X 是形如 11……11 的字，但 T(π)中不只包含上述类型的定理。

证明　由 M 构造Ω(M)和Γ(M)，并且π=(Γ(M),hq'h)，即可，事实上，假设 X∈Φ(M)，则有(由引理)

$$hq'h \overset{*}{\underset{\Omega(M)}{\Leftrightarrow}} hq_1\,\overline{X}\,h$$

再由Γ(M)的最后三个产生式得

$$hq' \overline{X} h \overset{*}{\underset{\Gamma(M)}{\Leftrightarrow}} \overline{X}$$

从而有

$$hq' h \overset{*}{\underset{\Gamma(M)}{\Leftrightarrow}} \overline{X} \text{ 即 } \overline{X}$$

反之，若有

$$hq'h \overset{*}{\underset{\Gamma(M)}{\Leftrightarrow}} \overline{X}$$

则最后一步只能用产生式 ，因为只有它才能水运非符号，进而可推出：

$$hq'h \overset{*}{\underset{-\Omega(M)}{\Leftrightarrow}} hq_1 \overline{X} h$$

由前的引理有 X∈Φ(M)。

定义 6.3.3 用 И(π)表示半图厄系统不产生 X 的整数 X 的集合，即

$И(\pi)=\{x|\overline{x}\in T(\pi)\}$

$И(\pi)=\{X|\overline{X}\in T(\pi)\}$

π=(Γ(M),hq'h)不一定

其中 X 是整数。

定理 6.3.2 对任一半可计算集 S 都有一个半图厄系统π，使得

S=И(π)

证明 已知 S 为半可计算集，因此应有一部分可计算函数 g(X)使得

S={X|g(X)}

设 Turing 机 M 计算 g(X)。我们有

S=Φ(M)

于是由前的定理有

$X\in\Phi(M)\Leftrightarrow \overline{X}\in T(\pi)$

$\Leftrightarrow X\in И(\pi)$

其中π=(Γ(M),hq'h)，从而可推出

Φ(M)=S=И(π)

本定理表示：任给一个半可计算集 S，我们都可以构造出一个半图厄系统 ，使得在 的定理中所有形如 111…11 的定理所表示的整数之集恰好构成 S。反之，任给一半图厄系统π，И(π)是半可计算集。其证明因太繁，故略去。

6.4 判定问题

判定问题可归结成谓词的可计算问题，因此下面用谓词来描述判定问题。

定义 6.4.1 设给定一个“是/否”判定问题，则定义一谓词谓词μ，使得μ取真值当且仅当问题的回答是“是”，我们称μ为特征谓词。

定义 6.4.2 说一个判定问题是可判定的，若其特征谓词μ是可计算的。

定义 6.4.3 说一个判定问题是半可判定的，若其特殊谓词μ是半可计算的。

前面几节介绍了一些半可计算理论，这一节将给出一些典型的不可判定问题。

具体研究 Turing 机(程序，P-T 程序)的停机问题、半图厄系统的判定问题和字问题以及正则系统的判定问题和字问题。

(1) Turing 机的停机问题 I：

对任给的 Turing 机 M 和任一输入 X 判定 turing 机 M 是否停机，(其算法的结果要与 M，X 无关，即两变元的判定谓词)

(2)Turing 机的集机问题 II：

对已给的 Turing 机 M，判定对输入 X ，Turing 机 M 是否停机。(其算法与 M 无关，但其判定谓词只有一个变元 X)

(3)半图厄系统的字问题：

对已给的半图厄系统σ，判定对任给的两个字ω_1和ω_2，是否有

$$\omega_1 \overset{*}{\underset{\sigma}{\Rightarrow}} \omega_2$$

(4)关图厄系统的判定问题：

对已给的关图厄系统σ，判定任给的一字ω是不是σ的定理，即有

$$A \overset{*}{\underset{\sigma}{\Rightarrow}} \omega$$ （A－公理）

(5)正则系统的判定问题：

对给定的正则系统γ，判定任给的一字ω是不是γ的定理，即有

$$A \overset{*}{\underset{\sigma}{\Rightarrow}} \omega$$ (A－公理)

(6)post 对应问题：

对给定对偶序列

$(\omega_1,\varpi_1),(\omega_2,\varpi_2),\ldots,(\omega_n,\varpi_n)$

判定是否存在序列$i_1,i_2,\ldots,i_p$使

$$\omega_{i1}\omega_{i2}\ldots\omega_{ip}= \quad \varpi_{i1}\varpi_{i2\ldots}\varpi_{ip}$$

其中ω_{ij}和ϖ_{ij}是字，$p\geq 1, 1\leq j\leq n, (j=1,\ldots,p)$。

定理 6.4.1 Turing 机的停机问题(I 是不可判定的)

证明 这里有两个变元，一是任给的 Turing 机 M，一是任给的输入 X，利用配对函数可视为只有一个变元 y。

$$M=l(y), X=r(y)$$

设问题的特殊谓词为$\mu(y)$。若问题是可判的，则$\mu(y)$是可计算的，于是有P-T程序Z_0(哥德尔数)计算它的特征函数。即有

$$\mu'(y)=\Phi(Z_0,y)$$

令$y=Z_0$，则有

$$\mu'(Z_0)=\Phi(Z_0,Z_0)$$

其中μ’表示μ的特征函数，进而有

$$\mu(Z_0)\Leftrightarrow\Phi(Z_0,Z_0)\downarrow$$

$$\Leftrightarrow K(Z_0)$$

如果μ是可计算的，则 K 就成为可计算的。这与 K 的不可计算性相矛盾，故μ不能是可计算的，因此该判定问题是不可判定的。

定理 6.4.2 存在-Turing 机 M，使得它对任意 X 的停机问题是不可判定的。

证明 令 M 为计算Φ(X X)的 Turing 机，则

M 对 X 停机$\Leftrightarrow\Phi$(X，X)↓

$\Leftrightarrow$K(X)

由 K(X)的非可计算性得出 M 对 X 的停机问题是不可判定的。

因为我们的兴起是在于整数集，因此把半图厄系统和则系统的判定问题中的字均理解为形如 11111…11 的字。结论是一致的。

定理 6.4.3 存在一半图厄系统σ，使得对任给字(111…11)的判定(是否为定理)问题是不可判定的。

证明 设 S 为半可计算但不是可计算的整数集，则根据过去的定理它必为其半图厄系统σ的形如 11…11 的定理的集合，即有

$$S=И(\sigma)$$

这时 И(σ)是半可计算而不可计算。因此对这个σ来说任给的整数 X 是是否∈И(σ)的问题是不可判定的，也就是 X∈T(σ)的问题是不可判定的。

定理 6.4.4 存在一半图厄系统σ，使得它的字问题是不可判定的。

证明 若不存在这种半图厄系统，则任何半图厄系统的字问题都是可判定的，而系统的判定问题是字问题的特例，于是得出结论任何半图厄系统的判定问题是可判定的。这与上述定理相矛盾，故定理成立。

在字问题“$\omega_1\Rightarrow\omega_2$”中把$\omega_2$视为形如 11…11 的字，而$\omega_1$是任意字。

我们再强调一次：上述判定问题对一般字而言也都是不可判定的。

定理 6.4.5 存在一正则系统 V，使得它们的字问题是不可判定的。

证明 可仿定理 6.4.4 证明。

Post 的对应问题，我们就不给出其证明了。

以上给出的判定问题都是典型的判定问题。下面给出 Post 判定问题的一些应用。

定理 6.4.6 由两个任意的 cfg 所产生语言是否相交是不可判定的。

证明 我们把这个问题归结为 Post 的对应问题。设给定对偶序列

$(\omega_1,\varpi_1'),(\omega_2\varpi_2'),\ldots,(\omega_k\varpi_k')$

对此我们构造两个 cfg

$$\text{cfg}\begin{cases} S_A\to\omega_iS_Aa_i \quad (1\le i\le k) \quad a_i\text{的引入目的是为了证明}a_ia_j\text{存在} \\ S_A\to\omega_ia_i \end{cases}$$

$$\text{cfg}\begin{cases} S_B\to\omega_i'S_Ba_i \quad (1\le i\le k) \quad a_i\text{的引入目的是为了证明}a_ia_j\text{存在} \\ S_B\to\omega'_ia_i \end{cases}$$

其中$a_i(1\le i\le k)$是新符，令

$L(cfg_1)=L_A$

$L(cfg_2)=L_B$

则容易证明

$L_A=\{\omega_{j1}\omega_{j2}\ldots\omega_{jm}a_{jm}a_{jm-1}\ldots a_{j1}|m\ge1\}$

$L_B=\{\omega'_{j1}\omega'_{j2}\ldots\omega\square_{jm}a_{jm}a_{jm-1}\ldots a_{j1}|m\ge1\}$

这时我们有

$L_A \cap L_B \neq \varnothing \Leftrightarrow$对于$(\omega_1,\omega'),\ldots(\omega'_k,\omega'_k)$的post的对应问题是可判定的。

但我们已知post的对应问题是不可判定的，因此两个cfg语言是否相交的问题是不可判定的。

存在着$j_1j_2\ldots j_m$以及$j'_1j'_2\ldots j'_n$使得：

$\omega_{j1}\omega_{j2}\ldots\omega_{jm}a_{jm}a_{jm-1}\ldots a_{j1}=\omega'_{j1}\omega'_{j2}\ldots\omega\square_{jm}a_{jm}a_{jm-1}\ldots a_{j1}$从这式我们可推得

(1) $a_{ji}=a'_{ji}(2)n=m(3)j_2=j_1j_2=j_2\ldots j_m=j_m$因此存在$j_1j_m$有$\omega_{j1}\omega_{j2}\ldots\omega_{jm}=\omega'_{j1}\omega'_{j2}\ldots\omega\square_{jm}$

习 题

1 给出一半图厄系统π，使得$N(\pi)=S$。其中，

$$S=\left\{x\middle|(\exists n)(\exists m)\left(x=n^m\right)\right\}，（m>1）$$

2 设集合 S={x|x 能被 3 整除，但不能被 4 整除的二进制数}

给出半图厄系统σ使得 $S=T(\sigma)\cap\{0,1\}^*$。

3 设 S 是所有能被 3 整除并且是技术的二进数组成的集合。给出半图厄系统σ，使得

$S= T(\sigma)\cap\{0,1\}^*$.（要求：用尽量少的产生式）。

4 给出半图厄系统σ，使得，$N(\sigma)=\left\{x\middle|(\exists n)(x=2^{3^n})\right\}$。（限制：产生式不超过 10 个）

5 构造半图厄系统σ，使得$N(\sigma)=\left\{x\middle|(\exists n)\left(x=n\left[\log_2 n\right]\right)\right\}$。

第七章 图灵机

本章，我们将介绍图灵机－计算机的一种简单数学模型。尽管图灵机简单，但它具有模拟通用计算机计算的能力。人们研究图灵机不仅是为了研究它所定义的语言类（称为递归可枚举集合）。也是为了研究它所计算的整数函数类（称为部分递归函数）。还将引用其它各种计算模型，它们在计算能力上都等价于图灵机。

7.1 引言

算法或有效过程的直觉概念已经出现过多次。在第三章中，我们给出一个有效过程来判定有穷自动接受的集合是否是空的，有穷的或无穷的。人们也许会朴素地设想，对于具有有军描述的任何语言类，总存在一个有效过程来回答这类问题，然而，情况并非如下，例如，不存在一个算法来辨别一个 CFL 的补是否为空（虽然我们能够辨别这个 CFL 本身是否为空）。注意，我们不是要求一个过程对一个具体的上下文无关语言回答这个问题，而是要求单独一个过程，对所有的 CFL 正确的回答这个问题。显然，如果我们只需要确定一个具体的 CLFL 是否有空补集，那么回答这个问题的算法是存在的。亦即，有一个算法，不管其输入是什么，总说“是”，还有一个算法，它总说“否”，两个算法中必有一个是正确的。自然，两个算法个正确地回答了问题可能不清楚。

本世纪初，有若·Rilbert 着手一项计算，打算寻求一个算法，用以确定任何数学命题的真和假，特别是，他要寻找一个过程，用来确定整数上的一价谓词演算中，一个任意公式是否为真，因为一阶谓词演算是以表达下面的命题：一个上下文无关文法产生的语言是Σ^*。故若 Hilbert 成功了，则我们判定一个 CFL 的补是否为空的问题将被解决，但在 1931 年，K·Godel 发表了他的著名不完全性定理，该定理证明，这样的有效过程不可能存在，在应用于整数的谓词演算中，他构造了一个公式，正是该公式的定义表明：该公式本身在这个逻辑系统中即不能被证明，也不能被否定。这个讨论的形式化以及后来关于有效过程真觉概念的澄清和形式化是本世纪中伟大的智力成就之一。

一旦有效过程概念被形式化，就可证明，对于许多具体函数的计算没有有效过程。实际上，这种函数的存在性，容易从一种计数论中看出，考虑将非负整数映射到（0，1）上的函

数类。这个函数类可以与实验一一对应。然而，如果我们假定有效过程有穷描述，那么全部有效过程的类就能与整数一一对应。因为在整数与实数之间没有一一对应，于是必然存在一些函数，对于这些函数没有对应的有效过程来计算它们，这样的函数简直太多了，即有不可数个函数，但只有可数过程，因此，不可计算函数的存在就不会令人吃惊了，使人吃惊的倒是某些在数学、计算机科学和其它科学中有真实意义的问题和函数是不可计算的。

今天，图灵机已成为被人接受的有效过程的形式定义。显然，人们不能证明图灵机模型等价于我们关于计算的直觉概念，但却有关于等价性的令人信服的证据。它就是为人熟知的 cnurcn 假设，特别地，就象我们今天知道的，图灵机的计算能力上等价于数字计算机，它也等价于计算的所有最一般的数学概念。

7.2 图灵机模型

有效过程的形式模型应该具有某些性质，首先，每个过程都应该是有穷可描述的。其次，过程应该由离散的步组成，每一步能够机械地被执行。A.Turing 在 1936 年介绍了这样一个模型。

这里我们介绍它的一种变形。

图 6.1 说明的基本模型有一个有限控制器，一条输入带和一个带头，带被分成许多单元，带头在每一时刻扫视带上的一个单元。该带有一个最左单元，向右则是无限的，带的每个单元正好可容纳有军个带符号中的一个，开始时，最左边 n 个单元对于某个有穷数 n≥0 装着输入，它是一个字符串，符号都自带符号的一个子集，即所谓输入符号集合，余下的无军多个元都存放空白符，它是一个特殊的带符号，但不是输入符号。

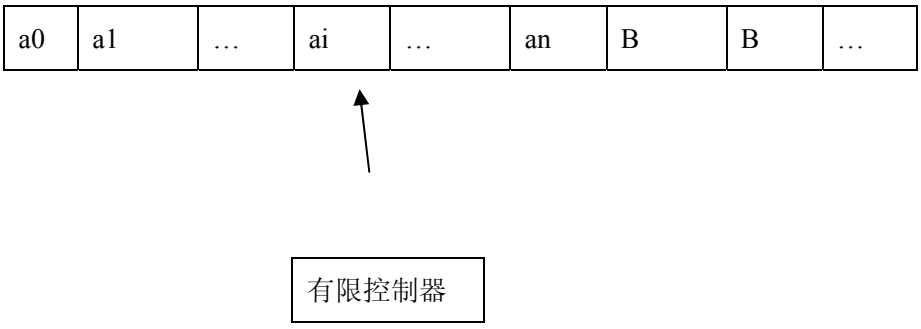


图 7.1 基本图灵机

在一个动作中，图灵机根据带头扫视的符号和有限控制器的状态。

（1）改变状态；

（2）在补充扫视的带单元上打印一个符号，以代替原来写在这里的符号；

（3）将带头向左或右移一个单元。

注意，图灵机和双向有穷自动机的区别在于：前者能够改变它带上的符号。

形式上，一个图灵机（TM）被记作：

M＝（Q，∑，Γ，δ，q_0，B，F）

这里

Q 是状态的有军集合；

Γ是所允许的带符号的有军集合；

B 是Γ的一个符号，即空白符；

∑是Γ的一个不包含 B 的子集，即输入符号集合；

δ是次动作函数，是一个从 Q×Γ到 Q×Γ×（L，R）的映射（然而，δ可以对某些自变量没有意义）

q_0在Q中，是初始状态；

F⊆Q 是终结状态集合。

我们用a_1qa_2来表记图灵机M的一个瞬时描述（ID），其中，q在Q中，是M的当前状态：a_1a_2是P中的字符串，它或是一直到最右非空白符号为止的带上的内容，或是一直到带头左边符号为止的带上的内容，a_1a_2等于其中较长的一个（可以看到，空白符B可以出现在a_1a_2中），为避免混淆。我们假设Q和P是不相交的，最后，假定带头正在扫视a_2的最左符号，如果a_2＝ε，那么带头正在扫视一个空白符。

我们定义M的一个动作如下。设$X_1X_2…X_{i-1}qX_iX_n$是一个ID。假定δ（q,X_i）＝（P,Y,L），这里如果i-1=n，那么X_i取为B。如果i=1，那么就没有下一个ID，因为带头不允许落在带左端的外面，如果i>1，那么我们写出

$X_1X_2…X_{i-1}qX_iX_n$ ⊦ $X_1X_2…X_{i-1}pX_iX_n$然而，如果$X_{i-1}qYX_{i+1}…X_n$的任何后缀全是空白符，那么在上式中，这个后缀可以删掉。

另一方面，假定δ（q,X_i）＝（P,Y,R），那么我们写出

$X_1X_2…X_{i-1}qX_i…X_n$　$X_1X_2…X_{i-1}YPX_{i+1}…X_n$

注意，在i-1=n情况下，字符串$X_i…X_n$是空的，上式的右边要比左边长。

如果把两个 ID 用 M 连接起来，我们就说，第二个是通过一个动作从第一个产生的，如果一个 ID 是通过有限个动作（包括零个动作）从另一个产生的，那么它们可用符号连结起来，

在不起混淆时，我们可以从$|\overline{M}|$和中省掉下标 M。

被M接受的语言记作L（M），它是Σ^*中那样一些字的集合，当这些字对齐子左端被放在M的带上，M处于状态q，且M的带头处在最左单元上时，这些字将使M进入一个终结状态，形式上，被M＝（Q，∑，Γ，δ，q_0，B，F）接受的语言是：

{ω|ω∑Γω}

缎带定一个识别语言 L 的 TM，不失一般性，我们假定，当输入被接受时，TM 将停止，也就是说，没有下一个动作，然而，对于不被接受的字，TM 可能永不停止。

例　接受语言 L＝{≥} 的 TM　M 的设计如下，起初，M 的带包含　，后面跟着无穷多个空白符，反复用 X 替换 M 最左边的 0，右移至最左边的 1，用 Y 替换它，左移去寻找最右边的 X，然后右移一个空白符，那么 M 停止而不接受，若在将一个 1 改变成 Y 后，M 再也找不到 0 了，那么 M 检查一下是否还剩 1，如果没有，M 接受。

设Q＝{q,q,q,Q}，∑＝{0，1}，Γ＝{0，1，X，Y，B}，而F＝{q}，非形式地，每个状态都表示程序中的一个语句或一组语句。状态　在开始时被进入，又恰在每次用X替换一个最左的 0 之前被进入，用状态　向右搜索，跳过所有的 0 和Y，直到发现最左的 1，如果M找到一个 1，M就将它变成Y，同时进入状态q_2。状态q_2向左搜索X，刚一找到X，就进入状态q_0，当它改变状态时，右移到最左的 0。当M在状态q_1中向右搜索时，若在一个 1 之前，遇到一个B或X，则输入被拒绝—或者是有太多的 0，或者输入不是在0^*1^*中。

状态	0	1	符号 X	Y	B
q_0	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	(q_1,0,R)	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	(q_2,0,L)	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
q_4	-	-	-	-	-

图 7.2

$q_0$0011	X$q_1$011	X0$q_1$11	X$q_2$0Y1
q_2X0Y1	X$q_0$0Y1	XXq_1Y1	XXY$q_1$1
XXq_2YY	Xq_2XYY	XXq_0YY	XXYq_3Y
XXYYq_3	XXYYBq_4		

图 7.3

那么 0 已经被耗尽了，扫视Y的同时，就从q_0 转入状态q_3，以便扫过Y并检查是否再没有 1 了。如果Y后面着B，则进入状态q_4，并接受输入；否则字符串被拒绝。函数δ示于图 7.2。图 7.3 表示M在输入 0011 上的计算。例如，第一个动作可以用δ（q,0）＝（q,X,R）这个事实画解释；最后一个动作可以用δ（q,,B）＝（q,,B,R）这个事实来解释；读者应该在某些被拒绝的输入上模拟 M，诸如 001101，001 和 011。

7.3 图灵机的变形

把图灵机作为计算的一种通用模型的原因之一就是：我们一直在讨论的这一模型等价于许多修改过的、初看去似乎有更强的计算能力的变形。本节，我们将缎带出某些等价定理的非形式证明。

7.3.1 双向无限带

一个具有双向无限带的图灵机，象在原来模型中一样，仍记作M＝（Q，∑，Γ，δ，q_0，B，F）。就象其名字所暗示的，带向左和向右都是无限的。我们象对单向无限TM那样来表记这个装置的ID。不过，我们想象，在带上现时非部分的左面和右面都有无穷多个空白单元。.

关系 $\vdash_M$ 关联两个 ID，右边的 ID 可以从左边的 ID 通过一个动作得到。可以象原来模型那样来定义 $\vdash_M$ 但下面的情况除外：如果δ=（q，X）＝（p,Y,L），那么 qXa $\vdash_M$ pBYa（在原来模型中，此时不能作出动作），如果δ(q,X)=(p,B,R)，那么 qXa├Pa（在原来模型中，B 将出现在 p 的左边）。

初始ID是$q_0\omega$，在原来中，图灵机的带有一个左端，而现在图灵机的带却没有左端，也不会“脱出”左端，故 0 它可以继续向左到承受意远。象通常一样，如果右边这个能通过若干个动作由左边那个得出，则关系 $\vdash_M^*$ 关系着这两个ID。

定理 7.3.1 L 被一个具有双向无限带的图灵机识别，当且仅当它被一个具有单向无限带的 TM 识别。

证 关于具有双向无限带的 TM 模拟一个具有单向无限带的 TM 的证明是容易的。前者在其带头初始位置的左侧单元上做上记，然后开始模拟后者，若在模拟期间到达作为记号的

单元，模拟结束而不接受。

反过来，设 M2（）是一个具有双向无限带的 TM，我们来构造一个图灵机 M，它模拟 M2，且有一条只在右方无限的带，M1 有两道，一道表示 M2 开始扫视的带单元及其右边所有的单元，另一道在相反的顺序下表示开始单元左方的所有单元。M2 和 M1 的带间关系示于图 7.3，M2 的初始单元的编号为 0，右边单元的编号为 1，2，……，左边单元为－1，－2，…。

…	A-5	A-4	A-3	A-2	A-1	A0	A1	A2	A3	A4	A5	…

A0	A1	A2	A3	A4	A5
∅	A-1	A-2	A-3	A-4	A-5

图 7.4 (a)M 的带　(b)M 的带

M_1带的第一单元，在下道中包含符号∅，它指明这是最左单元，M_1的有限控制器能够分辨M_2是正在扫视M1 上道中的符号，还是其下道中的符号。

十分明显，M_1能够在下面意义下被构造出来，以模拟M_2：当M_2在其输入带头初始位置右方时，M_1在其下道工作，但移动方向恰好和山的移动方向相反。M_1的输入符号是这样一些符号，下道上为空白符，上道上为M_2的一个输入符号，这样的符号与M_2的相应的输入符号可视为同一。B等同于[B，B]。

我们现在给出M_1＝（Q1，∑1，Γ_1，δ_1，q_1，B，F_1）的形式结构。M_1的状态集Q_1是所有形如[q,U]或[q,D]的对象的集合，这里q在Q_2∪｛q_1｝中。注意，第二分量指出M_1工作在上道（U表示上），还是工作在下道（D表示下）。Γ_1中的带符号是所有形如［X，Y］的对象，这里X和Y都在I中。此外，Y可以是∅，一个不在I_2中的符号。$\sum_1$由所有的符号［a,B］组成，这里a在$\sum_2$中。F_1是｛[q,U]，［q,D］丨q在F_2中｝，我们定义δ_1如下：

（1）对于$\sum_2$∪{B}中每个a，如果δ2(q2,a)=(q,X,R)，那么

$\delta_1(q_1,[a,B])=([q,U],[X,\varnothing],R)$

如果M_2在第一个动作中向右移，那么M_1在下道中打印，用以标记带的端点，将其状态的第二分量设置为U，并向右移，M_1状态的每一分量保存M_2的状态。在上道上，M_1打印符号X，它是M_2要打印的符号。

（2）对于$\sum_2 \cup \{B\}$中的每个，如果$\delta_2(q_2,a)=(q,X,L)$,那么

$\delta_1(q_1,[a,B])=([q,D],[X,\varnothing],R)$

如果M_2在其每个动作中左移，象在（1）中一样，M_1记录M_2的次状态和M_2打印的符号，但将其状态第二分量设置为D，并向右移。还是打印在下道，以标记带的左端点。

（3）对于Γ_1中的每个［X，Y］，$Y\neq\varnothing$，且A＝L或R，如果$\delta_2(q,X)=(p,,Z,A)$，那么

$\delta_1([q,U],[X,Y])=([p,U],[Z,Y],A)$

M1 在上道上模拟M_2。

（4）对于Γ_1中每个［X，Y］，$Y\neq\varnothing$，如果$\delta_2(q,Y)=(q,Z,\overline{A})$,那么

$\delta_1([q,D],[X,Y])=([p,D],[X,Z],A)$

这里，如果$\overline{A}$是R，A就是L，如果$\overline{A}$是L，A就是R。M_1在其下道上模拟M_2。M_1带头的移动方向与M_2的相反。

（5）如果$\delta_2(q,X)=(p,,Z,A)$，那么

$\delta_1([q,U],[X,\varnothing])=\delta_1([p,D],[X,\varnothing])=([p,C],[Y,\varnothing],R)$

这里，如果A＝R，则C＝U，如果A＝L，则C＝D。在M_2初始扫视单元上，M_1模拟M_2的一个动作。然后M_1工作在上道或下道，这取决于M_2的移动的方向。在这种情况下，M_1永远右移。

7.3.2 多带图灵机

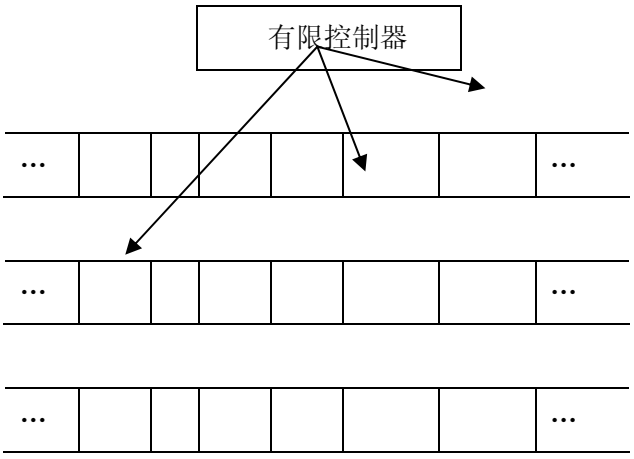


图 7.5 多带图灵机

图 7.5 表示了一个多带图灵机，它由有限控制器，k 个带头和 k 条带组成，每条带都是

双向无限的，在一个动作中，根据有限控制器的状态和每个带头扫视的符号，机器能够

（1）改变状态

（2）在其带头扫视的单元上，打印一个新符号；

（3）独立地将每个带头向左或向右移动一个单元，或保持不动。开始时，输入出现在第一条带上，其它带是空的。我们不去形式地定义这个装置，因为这个形式系统非常烦琐，而且是单带 TM 的一种直接推广。

定理 7.3.2　如果一个语言 L 被一个多带图灵机接受，它就能被一个单带图灵机接受。

证　设 L 是一个有 k 条带的 TM　M1 接受。我们可以构造一个具有 2k 道的单带 TM M2，M1 的每条带对应两条道。一条道记录 M1 对应带的内容，另一道为空白，不过，在 M1 的对应带头扫视的符号所在的单元中有一记号。如图，M2 的有限控制器存贮 M1 的状态以及 M2 带头右方的带头记号的数目。

带头 1		X				
带 1	A_1	A_2	…		…	A_m
带头 2				X		
带 2	B_1	B_2	…		…	B_m
带头 3	X					
带 3	C_1	C_2	…		…	C_m

图 7.6　一条带对三条带的模拟

为了模拟M_1的一个动作，M_2的带头要从左至右，然而从右至左的作一次巡视。开始时，M_2的带头在最左面含有带头记号的单元上。然而M_2向左巡视，访问每一个具有带头记号的单元，并记录下M_1的每个带头所扫视的符号。当M_2超过一个带头记号时，它必须修改右方带头标记的数目。当再没有带头记号在右方时，M_2就看过M_1的每个带头所扫视的符号了，故M_2已拥有足够的信息来确定M_1的动作。现在，M_2再作一次向左的运动，直到它到达左带

头记号，右方记号总数使得M_2能知道何时它已直到头了。当M2 在向左运动中通过每一个带头记号时，它修改M_1的被该带头记号“扫视”的符号，并将该带头记号向左或向右移动一个单元，以此来模拟M_1的动作。最后，M_2改变记录在有限控制器中的M_1的状态，完成对M_1的一个动作的模拟。如果M1 的新状态是接受状态，则M_2接受。

注意，在本节的第一个模拟－－用一个单向无限带TM对一个双向无限带TM进行的模拟中，模拟是一个动作对一个动作的。然而，在现在的模拟中，需要用M_2的多个动作来模拟M_1的一个动作。事实上，因为在k个动作后，M_1的带头可能相距 2k个单元，所以需要M_2的大约约$\sum_{i-1}^{k} 4i \approx 2k^2$个动作来模拟M1 的k个动作（直，模拟带头向右移动就可能需要 2k个动作）。从一个多带TM转到一个单带TM时出现的这种二次方减速，对于某些语言来说，是真正必需的。我们把证明放到以后章节，这里，我们给出多带TM效力的一个例子。

例 7.3.1 语言｛$\omega\omega^R|\omega$在（0＋1）*中｝能够在一个单带TM上被识别。方法是将带头在输入上移向移后，从两端检查符号，并比较它们，该过程类似于上例的过程。

为了用双带 TM 识别 L，输入将被复制到第二条带上。用两个带头按相反方向移动的办法，一条带上的输入和另一条带上的逆输入可以进行比较，输入的长度亦被检查，以查明它是否为偶数。

注意，用单带机识别 L 时用到的动作数大约是输入长度的平方，而用双带机器时，同输入长度成正比时间就足够了。

7.3.3 非确定图灵机

非确定图灵机是一个具有一个有限控制吕和单独一条单向无限带的装置。对于一个给定的状态和被带头扫视的带符号，机器对次动作可以有有限个选择。每个选择包括一个新状态，一个要打印的带符号和一个带头移动方向。注意，非确定 TM 不允许作这样的动作，其下一状态来自一个选择，而打印符号和（或）带头移动方向却来自中车个选择，如果有任何一个动作选择序列导致一个接受状态，那么非确定 TM 就接受它的输入。

就象有穷自动机那样，对图灵机增加非确定性并未使这个装置接受新的语言。实际上，非确定性与任何介绍过的或将要介绍的推广（例如双向无穷 TM 或多带 TM）组合起来，也不会增加额外的能力。我们把这些结果留作练习，只去证明用确定的 TM 模拟一个非确定的 TM 的一个基本结果。

定理 7.3.3 如果 L 被一个非确定的图灵机 M1 接受，那么 L 将被某个确定的图灵机 M2 接受。

证 对于 M1 的任何状态和带符号，有有限多个关于次动作的选择，它们可以用 1，2，…加以编号，设 r 是对于任何状态－－带符号偶来说的最大选择数。于是任何有限的选择序列都可以表成一个由数字 1 到 r 组成的序列，并不是所有这样的序列都表示一个动作选择序列，因为在某些情况下，可能只有少于 r 个的选择。

M2 将有三条带，第一条保存输入，在第二条上，M2 将以一种系统的方式产生数字 1 到 r 的序列。具体地讲，先产生短的序列，等长序列则按数值大小顺序产生。

对于在带 2 上产生的每个序列，M2 都把输入复制到带 3 上，然后在带上 3 上模拟 M1，同时使用带 2 上的序列来指明 M1 的各动作。如果 M1 进入一个接受状态，M2 也将接受，如果存在一个导致接受的选择序列，那么它终穷会在带 2 上产生出来。在模拟时，M2 将会接受。但若 M1 没有选择序列导致接受，M2 不会接受。

7.3.4 多维图灵机

让我们考虑图灵机的另一种修改－－多维图灵机，它也不增加额外的能力。这种装置具有通常的有限控制器，但带却由 k 维 （对于某个固定的 k）单元阵列组成，这里有所有 2k 个方向都是无限的。根据状态和所扫视的符号，该装置改变状态，打印一个新符号，在 2k 个方向之一上移动它的带头，即沿着 k 轴中的一轴，作正向或负向移动。开始时，输入沿着一个轴排列，带头在输入的左端。

任何时刻，在任何一维中，只有有穷多行包含非空白符号，其中每一行只有有穷多个非空白符号。例如，考虑图中的二维 TM 的带格局。仍如图所示，画一个关于非空符号的矩形。该矩形能够如图（b）那样，一行接一行地表示在一条带上，＊用来分隔各行，第二道可以用来指出二维 TM 的带头位置。

我们将证明，一个一维 TM 能够模拟一个二维 TM，并把二维以上的一般情形留作练习。

定理 7.3.4 如果 L 被一个二维 TMM2 接受，那么 L 将被一个一维 TMM1 接受。

证 M_1如图（b）那样表示M_2的带。M_1还有第二条带，下面我们将描述它的用途，在该动作中，带头不会离开已被M_1的带表示好的矩形。如果动作是水平方向的，那么M_1打印一个新符号，改变记录在M_1的控制器中的M_2状态，然后，简单地将带头记号向左或向右移一个单元。如果动作是铅直方向的M_1就用它的第二带来统计它的带头位置与其左边＊之间单元

的个数。如果动作是向下的，M_1就向右边＊移动，如果动作是向上的，M_1就向左边＊移动，最后，使用第二带上的计数，将带头记号置于新段（两个＊之间的区域）的相应位置上。

B	B	B	a1	B	B	B
B	B	a2	a3	a4	a5	B
a6	a7	a8	a9	B	A10	B
B	a11	a12	a13	B	a14	a15
B	B	a16	A17	B	B	B

(a)

**BBBa_1BBB*BB$a_2a_3a_4a_5$B*$a_6a_7a_8a_9$Ba_{10}B*B$a_{11}a_{12}a_{13}a_{14}a_{15}$

*BB$a_{16}a_{17}$BBB**

(b)

图 7.7　用一维对二维的模拟

（a）两维带（b）一维模拟

现在考虑M_1的带头移出被M_1表示好的矩形的情况。如果动作是铅直方向的，那么增加一个新的空白符号行到左边或右边，并用第二带来统计原来段的当前长度。如果动作是水平方向，M_1就运用“移动”技术，适当地在每个段的左端或右端加一个空白符。注意，双＊标志着包含各段的整个区域的两端，故M_1能够知道它何时已经加长了所有的段，腾出空间来作动作后，M_1就可以如上述那样来模拟M_1的动作。

7.3.5 多头图灵机

一个 k 头图灵机有 k（某个固定数目）个带头。带头从 1 到 k 编号，TM 的一个动作依赖于状态和被每个带头所扫视的符号。在一个动作中，每个带头都可独立地左移、右移或保持不动。

定理 7.3..5　如果L被某个k头TM　M_1接受，那么它能被一个单头TM　M_1接受。

证　这个证明类似于多带 TM 的定理 6.2 的证明。M2 在其带上有 k+1 道，最后一道保存 M1 的带，第 i 道保存一个记号，用以指出第 i 个带头（1≤i≤k）的位置。我们把此证明的细节留作练习。

7.3.6 离线图灵机

一个离线图灵机是一个多带 TM，其输入带是只读的。通常我们用端点记号来界定输入，∅在左边，$在右边。该图灵机不允许输入带头移出∅和$之间的区域。显然，离线 TM 只是多带 TM 的一种特殊情况，因而不会比我们已经考虑过的模型更为力。反之，离线 TM 能够模拟任何一个 TM　M，方法是比 M 多用一条带。离线 TM 所做的第一件事。就是将自己的输入复制到这条额外的带上，然后模拟 M，就就这条额外的带本来就是 M 的输入一样，在以后的章节中，当我们考虑把存贮空间的数量限制到小于输入长度的时候，对离线 TM 的需要将变得明显了。

CHURCH 假设

“可计算函数”的直觉概念能够等同于部分递归函数类这一假定是大家熟知的 Church 假设或 Church-Turing 论题。只要“可计算的”这个非形式概念仍然是一个非形式概念，我们就不能期望证明 Church 假设。然而，我们却能给出其合理性的证据。只要对我们直觉的“可计算”概念在步数和存贮量上不施加限制，看来部分递归函数直觉上就是可计算的，尽管有人会争论说，除非我们能事先对计算加以限制，或者至少可确定是否计算最终能结束，否则一个函数就不算是“可计算的”。

不太清楚的是，部分递归函数是否包括所有“可计算的”函数。逻辑学家提出了许多其它的形式系统，诸如λ演算，Post 系统和一般递归函数。已经证明，所有这些系统都定义了同样的函数类，也就是部分递归函数类。此餐，抽象计算机模型，例如随机存取机器（RAM），也产生部分递归函数。

RAM 包含：无穷个存贮字，编号为 0，1，…，其中每个都能保存任意整数；有穷个能保存任意整数的算术寄存器，整数可以译码成通常的各类计算机指令。我们不去更形式地定义 RAM 模型，但应该清楚，如果我们选择一个适当的指令集合，RAM 就可以模拟任何现存的计算机。我们将在下面证明图灵机形式系统与 RAM 形式系统具有同样的能力。某些鞭它的形式系统将在练习中讨论。

习 题

1 以作 $\bar{x}$ 为输入，在一条带上（要指明哪条带）给出结果 $\bar{y}$ ，用多带 TM 计算下面函数

$$y = x - 2^{[\log_2 x]}$$

2 给出计算 x_1， x_2 的最小公倍数的双向无穷多带图灵机。

3 利用多带图灵机计算级数 1，2，4，7，11，…，$\frac{1}{2}$(n(n-1))+1,…的前 n 项和，并计算其时间复杂度。

4 给出计算谓词 7|X 特征函数的离线 TM M。这里输入带上不是 X 而是 X 的二进制表示。并且要求 M 的空间复杂度的阶最小。（说明：若 $\lim_{n\to\infty}\frac{s_1(n)}{s_2(n)}=0$，则称 $s_1(n)$ 的阶小于 $s_2(n)$ 的阶。如上述极限为常数则阶相等）

第三部分　计算复杂性

第八章　计算复杂性理论

语言理论按集合结构的复杂程序，对集合进行分类。这样，就把正规集合看成是比 CFL 更简单的类，因为有穷自动机的结构比 DPA 更简单。另外一种分类称为计算复杂性分类。在某个通用计算装置（例如图灵机）上识别一个语言，需要一定数量的时间、空间和其它资源。这种分类法的根据就是识别集合时所需资源的数量。

虽然计算复杂性主要时间和究竟，然而还有其它许多可能的量度，例如单带 TM 上带头运动方向的逆转次数。实际上，人们可以在更一般的装置上抽象地定义一种复杂性量度，并证明许多结果，我们有选择地介绍了一些关于时间和空间这种特殊情况的结果，因为这种方法提供了比较直观的证明。

8.1 复杂性定义

8.1.1 空间复杂度

考虑图 8.1 中的离线图灵机 M。M 有一条具有端记号的只读输入带，和 k 条半无穷存贮带。如果对于每个长度为 n 的输入字，M 在任一条存贮带上都不能超过多扫视 S（n）个单元，那么称 M 是 **S（n）空间有界图灵机**，或**具有空间复杂度 S（n）**。被 M 识别的语言也称作具有空间复杂度 S（n）。

注意，在输入带上，图灵机不能重写，故计算带界限时，只统计所用的存贮带的长度。这个限制，使我们可以考虑比线性增长更小的带界限。如果 TM 能在输入带上重写，那么在计算空间界限时，必须把输入带长度包含进去。这样，空间界限就不能小于线性函数。

8.1.2 时间复杂度

考虑图 8.2 的多带 TMM。TM 有 k 条双向无限带，其中一条包含输入，如果对于每个长度为 n 的输入字，M 在停机前最多做 T（n）个动作，那么就称 M 是一个 **T（n）时间有**

界的图灵机，或 **M 具有时间复杂度 T（n）**。被 M 识别的语言称为具有时间复杂度 T(n)。

对时间和空间复杂度，选用两种不同的模型，目的就在于简化某些证明。在模型上的某些变化是行得通的。例如，若 S（n）≥n，那么我们可以使用单带 TM 作为我们的模型。而不会改变该语言类在空间 S（n）内接受的事实。然而，当讨论时间复杂度时，我们就不能使用单带 TM，或具有任何固定多条带的 TM，而不致从时间复杂度为 T（n）的语言类中推动某些语言。

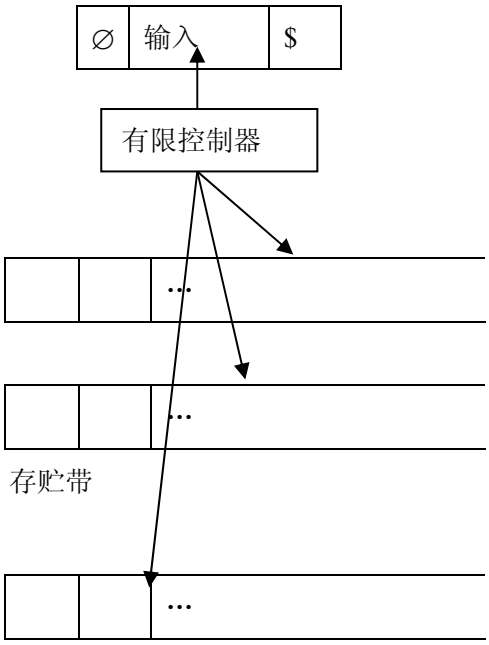


图 8.1 具有只读输入带的多图灵机

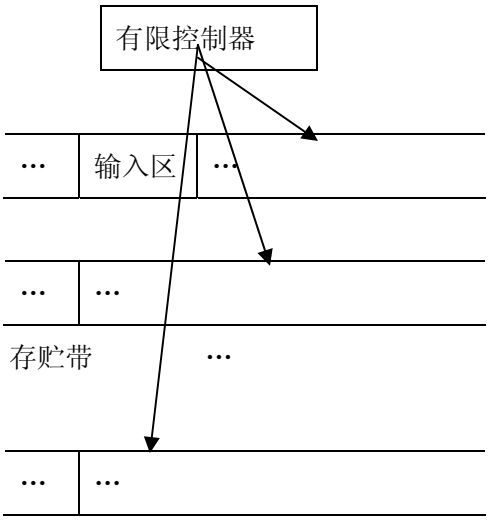


图 8.2 多带图灵机

例 8.1.1 考虑语言：L＝｛wcw^R|w在(0+1)*中｝

语言L具有时间复杂度n＋1，因为存在一个图灵机M_1，它具有两条带，并把c左边的输入复制到第二带上。然而，当发现c时，M_1使第二带带头向左，通过它刚刚复制出的字符串，同时使输入带头继续向右。在带头移动过程中，比较两个带头下的符号，如果每对符号都相同，且c左边和右边的符号数相等，那么M_1接受。容易看出，如果输入长度是n，则M最多做n＋1 个动作。

存在另一个图灵机M_2，它接受L，具有空间复杂度$\log_2 n$。M_2用两条存贮带来作二进制计数器。首先，检查输入，看看是否只出现一个c，以及c左边和右的符号数是否相等。然后，逐个符号地比较c左边的字和右边的字，同时用上述计算器来找出对应的符号。如果它们不一致，M_2停机，且不接受。如果所有的符号都一致，M_2就接受。

8.1.3 关于时间和空间复杂度函数的特殊假定

显然，对于一切输入，每个TM都至少要用一个单元，故若S（n）是一个空间复杂性量度，我们就可以假定对于一切n，S（n）≥1。我们来作一个有用的假设，即当我们谈到“空间复杂度S（n）”时，我们实际是指max(1,⌈S(n)⌉)。例如，在例 7.1 中，我们TM M_2具有“空间复杂度$\log_2 n$”。这对于n=0 或n=1 是无意义的，除非人们接受这样的假定：“$\log_2 n$”是max(1，⌈S(n)⌉)的简写形式。

类似地，下面的假定也是适当的：任何时间复杂度函数 T（n）至少是 n＋1，因为这恰好是读输入并根据所读到的第一个空白符来检验是否到达末端所需要的时间，因此，我们约定，“时间复杂度 T（n）”的意思是 max(n+1, ⌈T(n)⌉)。例如，在 n=1 时，时间复杂度 nlog2n 是 2，不是 0，而在 n＝2 时，它的值是 3。

8.1.4 非确定时间和空间复杂度

时间和空间有界图灵机的概念，同样也可以应用于非确定机器。一个非确 TM 具有时间复杂度 T（n），如果任何动作选择序列都不会使机器做多于 T（n）个的动作，它具有空间复杂度 S（n），如果任何动作选择序列都不会使机器在任何存贮带上扫视多于 S（n）个单元。

8.1.5 复杂性类

具有复杂度S（n）的语言族记作DSPACE（S(n)）具有非确定空间复杂度S（n）的语言族记作NSPACE（S(n)）。具有时间复杂度T（n）的语言族记作DTIME（T（n））, 而具有非确定时间复杂度T（n）的语言族记作NTIME（T（n））。所有这些语言族都被称为**复杂性类**。例如，例 7.1 中的语言L在DTIME（n）中，又在DSPACE（$\log_2 n$）中，因而L也在NTIME（n）和NSPACE（$\log_2 n$）中，同样也在更大的语言类中，例如DTIME（n^2）或NSPACE（$\sqrt{n}$）之中。

1）然而，应当注意，有一些 TM，它无须读完其全部输入就可以接受或拒绝，我们决定不考虑它们。

2）我们记得，对于时间复杂度来说，n 的真实意思是 max(n+1,n)=n。

8.2 线性加速、带压缩和带数目的减少

因为一个图灵机的状态数和带字母的大小可以任意大，故识别一个集合所需的空间量总可以压缩一个常数因子，将几个带符号编码成一个符号，就可做到这一点，类似地，人们可以使一个计算加速一个常数因子，因此，在复杂性的结果中，重要的是函数增长率（例如，线性的、二次方的和指数的增长率），常数因子则可忽略。例如，我们将谈到复杂度logn，而不规定对数的基数。因为$\log_b n$和$\log_c n$只差一个常数因子，即$\log_b c$。在本节中，我们将给出一些基本事实，它们是关于线性加速和带的，以及带数目对复杂性的影响的。

8.2.1 带压缩

定理 8.2..1 如果 L 被一个具有 k 条存贮带的 S（n）空间有界图灵机接受，那么对于任意 c＞0，L 被一个 cS(n)空间有界 TM 接受。

证 设M_1是一个S（n）空间有界离线图灵机，它接受L。证明的关键在于构造一个新的图灵机M_2，它模拟M_1，在这里，对于某个常数r，M_2的每个存贮带单元保存一个符号，这个符号代表了M_1相应带上r个相邻单元的内容。M_2的有限控制器能记住在所代表的那些符号

中，M_1的哪一个单元真正被M_1扫视。

由M_1的规则详细构造出M_2的规则，这一任务留给读者。设r满足rc≥2。在任何带上，最多用⌈S(n)/r⌉个单元，M_2就能模拟M_1。如果S（n）≥r，上面这个数就不大于cS(n)。如果S（n）<r，那么M_2可以将任何一条带的内容存在一个单元中。因此，在后一种情况下，M2只使用一个单元。 □

系 如果 L 在 NSPACEL(S(n))中，那么 L 在 NSPACE（cS(n)）中，这里 c 是任意一个大于零的常数。

证 如果上面 M1 是非确定的，就在上述构造过程中，让 M2 也是非确定的。 □

8.2.2 对于空间复杂性类的带数目的减少

定理 **8.2..2** 如果语言 L 被一个具有 k 条存贮带的空间有界 TM 接受，那么它可以被一个具有一条存贮带的空间有界 TM 接受。

证 设M1 是一个具有k条存贮带的空间有界TM，它接受L，我们可以构造一个新的TM M_2，它具有一条存贮带。M_2在k条道上模拟M_1的存贮带。这种技术在前面已经用过。M2 使用的单元数不超过s(n)。

从现在起，我们假设：任何 S(n)空间有界 TM 只有一条存贮带，且如果 S(n) ≥ n，那么它是一个单带 TM，而不是一个具有一条存贮带和一条输入带的离线 TM。

8.2.3 线性加速

在考虑时间界限以前，我们引进下述记号。f（n）是n的函数。表示式$\sup_{n\to\infty}f(n)$是当$n\to\infty$时，f(n),f(n+1),f(n+2),…的最小上界的极限。类似地，$\inf_{n\to\infty}f(n)$是当$n\to\infty$时，f(n),f(n+1),f(n+2),…的最大下界的极限。如果当$n\to\infty$时，f(n)收剑于一个极限值，那么这个极限既是$\inf_{n\to\infty}f(n)$，又是$\sup_{n\to\infty}f(n)$。

例 8.2..1 设当n为偶数时，f(n)=1/n，当n为奇数时，f(n)=n。对于任意n,f(n),f(n+1),…的最小上界显然是∞，原因在于n为奇数的项。因此，$\sup_{n\to\infty}f(n)=\infty$，然而，由于n为偶数的项，$\inf_{n\to\infty}f(n)=0$。

另一个例子，假定 f(n)=n/(n+1),那么对于任意 n,n/(n+1),(n+1)/(n+2),…的最小上界是 1。

因此，

$$\sup_{n\to\infty}\frac{n}{n+1}=1$$

n/(n+1),(n+1)/(n+2),…的最大下界是n/(n+1)，且$\inf_{n\to\infty}n/(n+1)=1$，故$\inf_{n\to\infty}n/(n+1)=1$。

定理 8.2..3 如果L被一个k带T(n)时间有界图灵机M_1接受，那么只要k>1，且$\inf_{n\to\infty}T(n)=\infty$，L就可被一个k带cT(n)时间有界TM M_2接受，这里c为大于零的任意数。

证 可以构造一个TM M_2，它以下述方式模拟M_1，首先，M_2将输入复制到一条存贮带上，与此同时将m个符号编码成一个符号（m的值将在后面确定）。由现在起，M_2用这条存贮带作为输入带，并用原来的输入带作为存贮带。通过将m个符号合并成一个符号的办法，M_2对M_1存贮带的内容进行编码。在模拟过程中，M_2在一个基本步中模拟M_1的一大批动作。这里，一个基本步包含M_2的八个动作。被M2 带头扫视的单元被为**基地单元**。对于每一条带，M_2的有限控制器都要记下带上基地单元所代表的M_1的m个符号中哪一个正在被M_2相应的带头扫视着。

开始一个基本步时，M_2将每个带头向左移一次，向右移两次，再向左移一次，同时把基地单元左、右邻单元的符号，记录在自己的有限控制器中。这需要M_2的四个动作，在这些动作以后，M_2又回到它的基地单元。

然后，在M_1的某个带头第一次离开由基地单元及其左、右邻单元所代表的区域时，M_2要确定出各基地单元及其左、右邻单元所代表的M_1的带单元的内容（注意，M_2的这个计算不占时间，因为它已安排在M_2的转移规则中了）。如果在某个带头离开所代表的区域之前，M_1接受了，那么M_2就接受，如果M_1停机，M_2也停机。否则M_2在每条带上再次访问基地单元的两个相邻单元，必要的话，它要改变这些符号以及基地单元的符号。M_2把它的带头放到一个单元上，这个单元代表了在该次模拟动作结束时M_1相应的带头正扫视的符号。这至多需要M_2的四个动作。

M_1要将一个带头移出由基地单元及其相邻单元所代表的区域至少m个动作。因此，在八个动作中，M_2至少模拟了M_1的m个动作。选择m满足cm≥16。

如果M_1做了T(n)个动作，那么M_2最多用⌈T(n)/m⌉个动作就可模拟它们。此外，M_2必须对输入进行复制和编码（m个单元变成一个），然后将模拟输入带的带头移回左端。这要用n+⌈n/m⌉个动作，总共要

$$n+\lceil n/m\rceil+8\lceil T(n)/m\rceil \qquad 8.1$$

个动作。因为对于任何 x,⌈x⌉<x+1,故式 8.1 的上界为：

n+n/m+8T(n)/m+9　　8.2

现在我们假定$\inf_{n\to\infty}T(n)/n=\infty$，故对任意常数d，都有一个$n_d$，使得对于所有的$n\geq n_d$。换句话说，$n\leq T(n)/d$。因此当$n\geq 9$ (故$n+9\leq 2n$)且$n\geq n_d$时，式 7.2 的上界是

$$T(n)[\frac{8}{m}+\frac{9}{d}+\frac{1}{md}]　　8.3$$

我们还没有规定d，我们已选m满足$cm\geq 16$，现在选$d=m/4+1/8$，并用 16/c去代替式 8.3 中的m。那么对于一切$n\geq \max(9, n_d)$，M_2所做的动作数不会超过cT（n）。

为了识别有限个其长度小于 9 和n_d中较大者的字，M_2只要利用它的有限控制器，用n+1个动作来读其输入，和到达作为输入末端标志的空白符即可。因此，M_2的时间复杂度是cT(n)。我们知道，对于时间复杂性来说，cT(n)表示$\max(n+1, \lceil cT(n)\rceil)$。 □

系　如果$\inf_{n\to\infty}T(n)/n=\infty$　且c>0,那么

$$DTIME(T(n))=DTIME(cT(n))$$

证　对于 2 带或多带 DTM 在时间 T（n）内接受的任何语言 L，定理 7.3 是一个直接的证明。显然，若 L 被一个 1 带 TM 接受，那么它被一个具有相同时间复杂度的 2 带 TM 接受。□

当T（n）是n的常数倍时，定理 7.3 不适用，因为$\inf_{n\to\infty}f(n)/n$是常数。不是无穷。然而，只要对M_2的时间界限作更细致的分析，定理 7.3 的证明过程即可以证明下述定理。

定理 8.2..4　如果对于 k>1 和某个常数 c，L 被一个 k-带 cn 时间有界 TM 接受，那么对于每个$\epsilon>0$，L 被一个 k 带（$1+\epsilon$）n 时间有界 TM 接受。

证　在定理 7.3 中取 $m=1/16\epsilon$。 □

系　如果对于某个 c>1，T（n）＝cn，那么对于任何$\epsilon>0$，DTIMET（n）=DTIME(($1+\epsilon$)n)

系（定理 7.3 和 7.4 的系）

（a）如果$\inf_{n\to\infty}T(n)/n=\infty$，那么对于任何c>0，NTIME（T(n)）＝NTIME(cT(n))。

（b）如果对于某个常数 c，T（n）＝cn，那么对于任何$\epsilon>0$，NTIME（T(n)）＝NTIME（($1+\epsilon$)n）。

证　此证与定理 8.2..3 和 8.2..4 的类似。

8.2.4 对于时间复杂性类的带数目的减少

我们来看看，当限制在单带时，时间复杂度会发生什么变化。一个象$L=\{wcw^R|w在(a+b)^*中\}$这样的语言，能够在线性时间内被一个双带机器识别，这正象我们在例 8.1..1 中看到的

那样，然而，在一个单带机器上，L需要时间cn^2（对于某个c>0）因此，只允许一条带，就可能使识别一个语言的时间按平方增加。下面定理表明，这也是可能发生的最坏情况。

定理 8.2..5 如果L在DTIME（T(n)）中，那么L可被一个单带TM在时间T^2（n）内接受。

证 在定理 8.2.1 的证明过程中，由一个多带TM到一个单带TM，当时M_2至多用 $2T^2$（n）步来模拟M_1的T（n）步。根据定理 8.2..2，我们可加速M_1，使之在T（n）/$\sqrt{2}$ 时间内运行。那么M_2是一个在T^2（n）步内接受L的单带TM。 □

系 如果L在NTIME（T(n)）中，那么L可被一个非确定时间复杂度T^2（n）来单带NTM接受。

证 类似于于本定理的证明。 □

如果我们限制在双带情形，那么，正如下面定理表明的，与限制在单带情形比较，时间的损失将大大减少。

定理 **8.2.6** 如果L被一个k带T（n）时间有界图灵机M_1接受，那么L可被一个双存贮带TM M_2 在T（n）logT(n)时间内接受。

证 在M_2的第一条存贮带上，对于M_1的每条存贮带，都将有两条道。为方便计，我们将集中精力，处理对应于M_1的一条特定带的两条道。对于M_1其它的带，可完全照此进行模拟。M_2的第二条带只用于打草稿和传送带 1 上的数据块。

带 1 有一个特殊单元，叫做B_0，它保存M_1每个带头所扫视的存贮符号。M_2不移动带头记号，而是移动数据，使之穿过B_0，移动方向恰好与被模拟的M_1的带头移动方向相反。因此，M_2只要注视B_0，就能模拟M_1的每个动作。单元B_0的右边是数据块B_1，B_2，…，它们的长度按指数增长，即B_i的长度为 2^{i-1}。同样地，B_0的左边是数据块 $B_1,B_2,\ldots B_{-i}$，它的长度是2^{i-1}。数据块间的标记假定是存在的，尽管在用到数据块之前，它们并不真的出现。

令a_0为开始时M_1的这条带带头扫视单元的内容。该单元右面单元的内容是$a_1,a_2,\ldots$，左面是$a_{-1},a_{-2},\ldots$。当这些a_i进入B_0时，它们可以改变。重要的不是它们的值，而是在M_2的带 1 的道上它们所处的位置。开始时，假定M_2对应于M_1所述带的那条上道是空的，而下道假定存有$\cdots a_{-2},a_{-1},a_0,a_1,a_2,\ldots$，如图 7.3 所示，它们被放在数据块$\cdots B_{-2},B_{-1},B_0,B_1,B_2,\ldots$中。

如上所述，数据将被移过B_0，也许在越过时，数据会发生变化。在模拟M_1的每个动作后，将会有

（1）对于任意i>0，或者B_i是满的（指在两条道上）而B_{-1}是空的，或者B_i是空的而B_{-i}是满的，或者B_i和B_{-i}的下道都是满的而上道都是空的。

（2）任何B_i或B_{-i}的内容都表示了被表示的M_1的带上连续的单元，对于i>0，上道表示在下道单元左边的单元；对于i<0，上道表示在下道单元右边的单元。

（3）对于i<j。B_i表示B_j那些单元左边的单元。

（4）B_0永远只有满的下道，其上道作了特殊的标记。

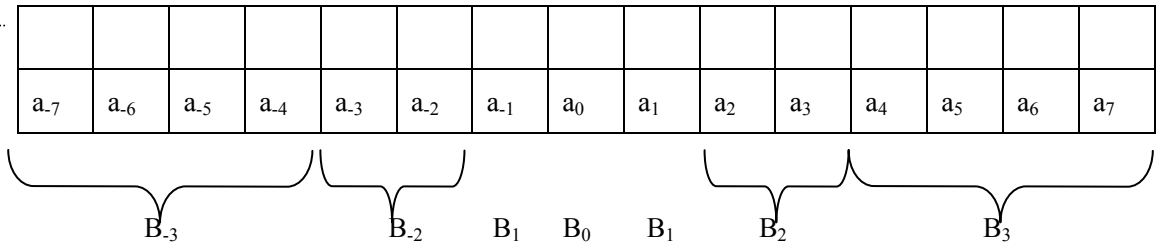


图 8.3 带 1 上的数据块

为了看清数据是怎样传送的，可设想M_1的所述带头向左移，那么M_2必须把相应的数据向右移。为此，让带 1 的带头从B_0（带头停留的地方）出发向右移动，直到它找到它的第一个不是上下全满的数据块，比如说B_i，然后，M_2把$B_0,B_1,\ldots,B_{i-1}$，的全部数据复制到带 2 上，并将它们再存贮到$B_1,B_2,\ldots,B_{i-1}$的下道，以及B_i的下道上去，这时，假定B_i的下道是空的。如果B_i的下道已经填满，则使用B_i的上道。无论哪种情况，都刚好有足够的空间分配给这些数据。注意，取出数据并将它们存到新位置所花的时间，与B_i的长度成正比。

其次，在与B_i长度成正比的时间内，M_2能够找到B_{-i}（利用带 2 来测量从B_i到B_0的距离，将使这件事变得容易）。如果B_{-i}是全满的，那么M_2取出B_{-i}的上道，并将它们存在带 2 上。如果B_{-i}是半满的，那么下道放到带 2 上。无论哪种情况，复制到带 2 上的数据随后都被复制到$B_{-(i-1)},B_{-(i-2)},\ldots,B_0$的下道上（根据规则 1，这些下道必是空的，因为$B_1,B_2,\ldots,B_{-1}$是满的）。还请注意，刚好有足够的空间来存贮这些数据，上述的所有操作都能在正比于B_i长度的时间内实现。还要注意，数据可以被分配得满足上述规则（1）（2）和（3）。

我们称上面所描述的一切为一个B_i-操作。M_1的带头向右移的情形是类似的。当M_1将所讨论的带头向左移五个单元时，各数据块的相继内容如图 8.4 所示。

我们注意，对于M_1的第条带来说，对M_1的每 2^{i-1}个动作，M_2至多执行一次B_i操作，因为在一个B_i-操作之后，$B_1,B_2,\ldots,B_{i-1}$都是半空的，要填满它们，必须用这么长的时间。此外，在M_1的第 2^{i-1}步之前，是不会开始执行第一个B_i操作的。因此，如果M_1在T（n）时间内操作，那么M_2将只执行这样的一些B_i-操作，其中的i满足$i\le\log_2 T(n)+1$。

我们已经看到，存在一个常数m，使得M_2至多使用$m2^i$个动作来实现一个B_i-操作。如果M_1作T（n）个动作，在模拟M_1的一条带时，M_2至多作

			B_{-3}			B_{-2}		B_{-1}	B_0	B_1		B_2			B_3		
	a_{-7}	a_{-6}	a_{-5}	a_{-4}	a_{-3}	a_{-2}	a_{-1}	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7		
								a_0									
	a_{-7}	a_{-6}	a_{-5}	a_{-4}	a_{-3}	a_{-2}		a_{-1}	a_1	a_2	a_3	a_4	a_5	a_6	a_7		
										a_0	a_1						
	a_{-7}	a_{-6}	a_{-5}	a_{-4}			a_{-3}	a_{-2}	a_{-1}	a_2	a_3	a_4	a_5	a_6	a_7		
									a_{-2}	a_0	a_1						
	a_{-7}	a_{-6}	a_{-5}	a_{-4}				a_{-3}	a_{-1}	a_2	a_3	a_4	a_5	a_6	a_7		
												a_0	a_1	a_2	a_3		
					a_{-7}	a_{-6}	a_{-5}	a_{-4}	a_{-3}	a_{-2}	a_{-1}	a_4	a_5	a_6	a_7		
									a_{-4}			a_0	a_1	a_2	a_3		
					a_{-7}	a_{-6}		a_{-5}	a_{-3}	a_{-2}	a_{-1}	a_4	a_5	a_6	a_7		

图 8.4　M_2各数据块的内容

$$T_1(n)=\sum_{i=1}^{\log_2 T(n)+1} m2^i \frac{T(n)}{2^{i-1}} \qquad 8.4$$

个动作。

由 8.4 我们得到

$T_1(n)=2mT(n)\ \lceil \log_2 T(n)+1 \rceil$　　8.5

由 8.5 式

$T_1(n)<4mT(n)\log_2 T(n)$

读者应该看出，M_1在操作时使用了几条不同的存贮带，而不仅是我们关注过的一条存贮带，M_2还是在比于T_1（n）的时间内操作，根据定理 8.2.2，我们可修改M_2，使其运行不超过$T(n)\log_2 T(n)$步。

系　如果 L 被一个 k-带、时间复杂度为 T（n）的 NTM 接受，那么 L 也可被一个双带、

时间复杂度为 T(n)logT(n)的 NTM 接受。

证 与本定理的证明类似。 □

8.3 谱系定理

直观上，给了更多的时间或空间，我们应该能接受更多的语言，或计算更多的函数，然而，线性加速定理和压缩定理告诉我们，必须增加越过一个常数因子的可用空间或时间。但若用一个缓慢增长的函数，诸如 loglogn 来乘空间或时间，情况将如何呢？是否还是不能识别任何新的语言？是否有一个时间或空间界限 f(n)，使得任何递归语言都在 DTIME(f(n))中？或者，也许在 DSPACE(f(n))中？

正如我们将在下个定理中要证明的那样，最后一个问题和答案是“否”。然而，对第一个问题的回答却取决于我们是否从一个“性质良好”的函数开始。本节我们将给“性质良好”的适当定义，并证明：对于性质良好函数，少量额外的时间和空间，将会增加我们的计算能力。

在 8.6 中，我们将考虑任意全递归函数和它们所定义的复杂性类。那时，我们将看到，一些奇怪的现象的出现。在任何复杂性谱系中，都有一些“间隙”，即存在一个函数T(n)，使得DTIME(T^2(n))=DTIME(T(n))，而且，一般地，对于任何全递归函数f，都有一个时间复杂度T_f(n)，使得DTIME(T_f(n))=DTIME(T_f(n))。类似的表述对空间也成立，且实际上，对任何适当的计算复杂性量度也都成立。我们还将看到，有这样一些语言，对于它们，不存在“最好的”识别器，更确切些说，存在一个无穷序列的TM，它们都识别L，而其中每一个运行起来都比前一个快得多。

定理 8.3.1 给定任何全递归的时间界限(空间界限)T(n)，存在一个递归语言 L，它不在DTIME(T(n))(DSPACE(T(n)))中。

证 我们将证明关于时间的结果，对空间的证明类似。证明基本上使用对角线方法。因为T(n)是全递归的，故有一个能停机的TM M来计算它。我们来构造一个 $\hat{M}$ ，它接受一个语言L⊆(0+1)*，L是递归的，但不在DTIME(T(n))中。设x_i是在(0+1)*的正则顺序中第i个字符串。在前面，我们对具有带字母表{0，1，B}的单带TM确定了一个顺序。对具有任意带字母表的TM，我们可以类似地定序，方法是用二进制字符串来替换它的转移函数。唯一重要的一点在于：象状态名一样，带符号的名字是不重要的，故我们可以假定，输入字母表为{0，1}

的一切TM都具有带字母表 0，1，B，x_4,x_5,…直到某个有穷的x_m，然后可用 0，00，000 对 0，1 和B编码，用 0_i对x_i编码，这里i≥4 。我们还允许在M的代码前面有任意个数的 1，这些代码都表示M，故M具有任意长的代码。

因此，我们可以随意谈论M_i，即第 个多带TM。现在定义L={x_i|M_i在T(|x_i|)个动作内不接受x_i}。我们断言L是递归的。为识别L，执行下面的算法。这个算法肯定能在一个图灵机上实现。给定一个长度为n的输入w，在n个模拟M，用以计算T(n)。然后确定满足w=x_i的i。写成二进制形式的整数i是某个多带TM M_i的转移函数(如果二进制下的i不是一个转移函数应有的形式)那么Mi没有动作)。在w上对Mi模拟T(n)个动作，如果M_i停机但不接受，或者越过T(n)个动作还没有接受，则接受w。

为看出L不在DTIME(T(n))中，假定L=L(M_i)，且M_i是T(n)时间有界的。 x_i在L中吗？如果在，那么M_i在T(n)步内接受x_i，这里n=|x_i|。根据L的定义，x_i不在L中，这产生矛盾。如果x_i不在L中，那么M_i不接受x_i，故根据L的定义，x_i在L中，又是一个矛盾。两种假设都导致矛盾，故M_i是T(n)时间有界的假定必是错误的。 □

如果对于一切 n，T’(n)≥T(n)那么立刻可以从时间复杂性类的定义推出 DTIME(T(n))⊆DTIME(T'(n))。如果 T(n)是一个全递归函数，定理 8.3.1 就推出，存在一个递归集合 L，它不在 DTIME(T(n))中，设$\hat{T}(n)$是接受 L 的的某个图灵机的运行时间，并令 T'(n)=max{T(n), $\hat{T}(n)$}，那么 DTIME(T(n)⊆DTIME(T'(n))，因为 L 在后者中而不在前者中，于是我们得知，有一个确定的时间复杂性类的无穷谱系，类似的结果对确定的空间复杂性类也成立，且对非确定的时间和复杂性也成立。

定理 8.3.1 表明，对于任何递归的时间或空间复杂度函数 f(n)，总有一个 f'(n)，使得某个语言是在由 f'(n)定义的复杂性类中，而不在 f(n)定义的类中。我们现在证明，对于一个性质良好的函数 f(n)，只需对 f(n)的增长率稍作增加，就能产生新的复杂性类。定理 7.8 和 7.9 关系到为得到一个新的确定的复杂性类所需的增长量，后来用这些定理建立种种问题的复杂度的下界。对于非确定类的类似结果要困难得多，我们将在 8.5 节提到关于非确定空间的一个稠密谱系。

8.3.1 空间谱系

现在我们来引进“性质良好”的空间复杂度函数的概念，称一人函数S(n)为空间可构造

的，如果有某个图灵机M，M是S(n) 空间有界的，且对每个n，都存在某个长度为 n的输入，对于这个输入，M实际用了S(n)个带单元，空间可构造函数集包括logn，n，n^k，2^n和n!。如果 $S_1(n)$ 和$S_2(n)$是空间可构造的，那么 $S_1(n)$ $S_2(n)$，$2^{S1(n)}$和$S_1(n)^{S2(n)}$也是空间可构造的，因此，空间可构造函数的集合是相当丰富的。

注意，上述 M 无需对一切长度为 n 的输入都使用 S(n)，只需对某个这种长度的输入使用 S(n)空间就行了。如果对于一切 n，M 对长度为 n 的任何一个输入，都实际使用恰好 S(n)个单元，那么我们说 S(n)是**完全空间可构造的**。任何空间可构造的 S(n)≥n 都是完全空间可构造的（可作为练习由读者完成）。

为了简化后面的结果，我们证明下述引理。

引理 8.1 如果L被一个S(n)≥$\log_2$n空间有界的TM接受，那么L被一个S(n)空间有界且对所有输入都能停机的TM接受。

证 设M是一个S(n)空间有界的离线图灵机，它具有s个状态和t个带符号，并接受L。如果M接受，那么它可以至多用(n+2)sS(n)$t^{S(n)}$个动作来做到这一点。因为，不然的话，某个ID就要重复了。即有n+2 个输入带头位置，s个状态， S(n)个带头位置，$t^{S(n)}$个存贮内容。如果增加一条额外的道，作为动作数计数器，M就可以在$(4st)^{S(n)}$ ≥(n+2)sS(n)$t^{S(n)}$个动作之后，使自己停下来。实际上，M将建立一个长度为logn的计数器，计数的基数为 4st。每当M扫视一个新单元，即保存计数器的那些单元之外的一个单元时，M就增加计数器的长度。因此，如果M只使用i个单元而进入循环，那么当计数达到$(4st)^{\max(i,\ \log_2 n)}$时(这个数至少为(n+2)sS(n)$t^{S(n)}$)，计数器就能探测出这一点来。 □

定理 8.3.2 如果 S2(n) 是一个完全空间可构造函数，

$$\inf_{n\to\infty}\frac{S_1(n)}{S_2(n)}=0$$

且$S_1(n)$和 $S_2(n)$ 都至少是$\log_2$n，那么有一个语言，它在DSPACE($S_2(n)$)中，但不在DSPACE($S_1(n)$)中。

证 这个定理可用对角线方法证明。考虑具有输入字母表{0，1}且有一条存贮带的离线图灵机的一个枚举。这些机器是根据前面的二进制编码法编码的，且有一个由 1 组成的前缀，故每个TM具有任意长的编码。我们构造一个TM M，它使用$S_2(n)$ 空间，而与任何一个$S_1(n)$空间有界TM比较，至少在一个输入上两者不同。

对于输入w，开始时，M在一条带上，对$S_2(n)$个单元作上记号。这里n是w的长度。因为$S_2(n)$是完全空间可构造的，定有某个TM，它对每个长度为n的输入，都愉好使用 $S_2(n)$个单

元。模拟这个TM，就能完成作记号的任务。此后，如果M企图离开作记号的单元，M就停机，并拒绝w。这样就能保证M是S(n)空间有界的。

然后，在输入w上，M对TM M_w开始进行模拟，这个TM是二进制字符串w所编码的机器。如果M_w是 $S_1(n)$ 空间有界的，且它有t个带符号，那么模拟所需的空间为$\lceil \log_2 t \rceil S_1(n)$ M接受w，仅当M能够在$S_2(n)$ 空间内完成模拟，且M_w停机而不接受w。

因为M是$S_2(n)$ 空间有界的，L(M)在DSPACE($S_2(n)$)中。L(M)不在DSPACE($S_1(n)$)中。假定有一个 $S_1(n)$空间有界的TM $\hat{M}$ ，它具有t个带符号，且接受L(M)。根据引理 7.1，我们可以假定，$\hat{M}$ 对所有的输入都能停机。因为 $\hat{M}$ 无穷经常地出现在枚举中，且

$$\inf_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$$

故存在一个足够长的w，|w|=n ，使得$\lceil \log_2 t \rceil S_1(n) < S_2(n)$，并且$M_w$是 $\hat{M}$ 。对于输入w，M有足够的空间模拟M_w，M接受当且仅当M_w拒绝。于是$L(M_w) \neq L(M)$，产生矛盾。因此，L(M)在DSPACE($S_2(n)$)中，但不在DSPACE($S_1(n)$)中。 □

尽管大多常见函数都是完全空间可构造的，然而我们只需空间可构造，就能使定理 7.8 成立。因此我们有下面的结果。

系 即使$S_2(n)$ 是空间可构造而不是完全空间可构造的，定理 7.8 仍然成立。

证 设M_1是一个TM，它在某个输入上构造$S_2(n)$ 。设∑是M_1的输入字母表。我们设计M来接受字母表∑×{0,1}上的一个语言。即对M的输入被看成两道：第一道用于对M_1的输入，第二道用来放具有输入字母表∑×{0,1}的一个TM的代码。对于M的设计，唯一要修改的是：通过在M的第一道上模拟M_1，M必须在带 1 和 2 上标出区间段来。我们可以证明，M与任何 $S_1(n)$ 空间有界的TM $\hat{M}$ 都会在一个输入上不同，这个输入的长度n充分大。输入的第一道是$\sum^n$中的一个字符串，该字符串导致M_1使用$S_2(n)$ 个单元，而输入的第二道是 $\hat{M}$ 的一个编码。 □

在定理 8.3.3 和它的系中，$S_2(n) \geq \log_2 n$这个条工不真正需要，我们将此证明留作练习。这个证明不用对角线方法，其关键在于证明下述事实：如果

$$\inf_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$$

且$S_2(n) < \log_2 n$，那么语言

{$wc^i w$|w在(a+b)*中,|w|=$S_2(n)$且i=n-2$S_2(n)$}

可在$S_2(n)$空间内接受，但不能在 $S_1(n)$ 空间内接受。

注意，如果$\inf_{n\to\infty}[S_1(n)/S_2(n)]=0$，且对一切n，$S_1(n)\le S_2(n)$，那么

$DSPACE(S_1(n))\subsetneq DSPACE(S_2(n))$

但是，若无$S_1(n)\le s_2(n)$，那么$DSPACE(S_1(n))$和$DSPACE(S_2(n))$可能有一些语言不在另一个中。

8.3.2 时间谱系

确定的时间谱系不象空间谱系那样致密。其原因在于：一个对所有多带 TM 进行对角化的 TM 的带数目是一个固定的数。为了模拟一个带数目很大的 TM，我们要利用多带 TM 的双带模拟，因而出现了对数减速。在给出构造过程之前，我们介绍一下时间可构造性概念。称函数 T(n)为**时间可构造的**，如果存在一个 T(n) 时间有界多带图灵机 M，使得对于每个 n，都存在某个输入，对于这个输入，M 实际作了 T(n) 个动作。正象空间样，有一个时间可构造函数的丰富谱系。我们说 T(n) 是完全时间可构造的，如果有一个 TM，它对一切长度为 n 的输入，都使用 T(n) 时间。

定理 8.3.3 如果$T_2(n)$ 是一个完全时间可构造函数，且

$$\inf_{n\to\infty}\frac{T_1(n)\log T_1(n)}{T_2(n)}=0$$

那么，有一个语言，它在$DTIME(T_2(n))$中，但不在$DTIME(T_1(n))$中。

证 这个证明与定理 8.3.2 的类似，我们只给出必需的构造过程的概述。构造一个$T_2(n)$时间有界的TM M。其操作如下。M把输入w看作一个图灵机 $\hat{M}$ 的编码，并在w上模拟 $\hat{M}$ 。产生困难的原因在于M的带数目是某个固定的数，故对于某些w，$\hat{M}$ 将比M有更多的带。幸而根据定理 8.2.6，要模拟任何 $\hat{M}$ ，只需要两条带，虽然这时模拟将损失一个因子$\log T_1(n)$。此外，因为 $\hat{M}$ 可能有许多带符号，它们必需用某个固定数目的符号来进行编码。故M对 $\hat{M}$ 的$T_1(n)$个动作的模拟需要$cT_1(n)\log T_1(n)$时间，这里c是一个与 $\hat{M}$ 有关的常数。

为确保对 $\hat{M}$ 的模拟是$T_2(n)$时间有界的，M必须同时执行一个TM的各个步骤(用附加的带)，这个TM对一切长度为n的输入都恰好用$T_2(n)$ 时间，这就是为什么$T_2(n)$ 必须是完全时

间可构造的原因。在$T_2(n)$ 步以后，M停机。M接受w，仅当对 $\hat{M}$ 的模拟完成了且 $\hat{M}$ 拒绝w。$\hat{M}$ 的编码被设计得象在前面定理中一样，每个 $\hat{M}$ 有任意长的编码。因此，若 $\hat{M}$ 是一个$T_1(n)$ 时间有界的图灵机，那么将有一个充分长的w，它编码 $\hat{M}$ ，使得

$$cT_1(|w|)\log T_1(|w|)\leq T_2(|w|)$$

并使模拟得以完成。在这种情况下，w在L(M)中，当且仅当w不在L($\hat{M}$)中。于是，对于任何$T_1(n)$ 时间有界的 $\hat{M}$ ，L(M)≠L($\hat{M}$)。因而L(M)在DTIME（$T_2(n)$）-DTIME（T_1）(n) ）中。 □

例 8.3.1 设$T_1(n)=2^n$，$T_2(n)=n^2 2^n$，那么

$$\inf_{n\to\infty}\frac{T_1(n)\log T_1(n)}{T_2(n)}=\inf_{n\to\infty}\frac{1}{n}=0$$

于是定理 8.3.3 适用，且有DTIME(2^n)≠DTIME($n^2 2^n$)。因为对于一切n，T_1（n）≤T_2（n），我们可以推出：DTIME（2^n）⊊DTIME($n^2 2^n$)

8.4 复杂性量度间的关系

对于语言，我们已定义了四种复杂性量度，一个给定的语言 L 的复杂性量度间有若干简单的关系，还有一种不那么明显的关系。这些简单关系表述在下面的定理中。

定理 8.4.1

(a)如果 L 在 DTIME（f(n)）中，那么 L 在 DSPACE（f(n)）中。

(b)如果L在DSPACE（f(n)）中，且f(n)≥$\log_2 n$，那么有某个常数c（它依赖于L），使得L是在DTIME（$c^{f(n)}$）中。

(c)如果L在NTIME（f(n)）中，那么有某个常数c（它依赖于L），使得L是在DTIME（$c^{f(n)}$）中。

证

(a)如果 TM　M 所做动作数不超过 f(n)，那么它在任何一条带上所扫视的单元数不可能超过 f(n)＋1。修改 M，使每个单元能保存两个符号，我们就能把存贮要求降到「(f(n)＋1）

/2⌉，它最多为 f(n)。

(b)可以看到，如果TM M_1有s个状态，t个带符号，最多使用f(n)空间，那么当输入长度为n时，M_1所有不同ID的数目最大为$s(n+2)f(n)t^{f(n)}$。因为$f(n)\geq \log_2 n$，故有某个常数c，使得对于$n\geq 1$，$c\ f(n)\geq s(n+2)f(n)t^{f(n)}$。

由M_1构造一个多带TM M_2，它用一条带进行直到$c^{f(n)}$的计数，而用另外两条带来模拟M_1。如果在计数到$c^{f(n)}$时，M_1还没有接受，那么M_2停机，且不接受。在这个数目的动作之后，M_1必然要重复一个ID，故决不会再接受了。显然，M_2是$c^{f(n)}$时间有界的。

(c)设M_1是一个f(n)时间有界的非确定TM，具有s个状态，t个带符号和k条带。给定长度为n 的输入，M_1所有可能的ID的数目不超过$s(f(n)+1)^k t^{kf(n)}$，即状态数、带头位置和带内容的乘积。令$d=s(t+1)^{sk}$，于是d满足

$d^{f(n)}\geq s(f(n)+1)^k t^{k f(n)}$　　对一切$n\geq 1$

通过构造一个表，即M_1的从初始ID可以达到的所有ID的表，一个确定的多带TM就能断定M_1是否接受长度为n的输入w。这个过程在表长度的平方时间界限内就能实现。因为可达ID表的长度不大于$d^{f(n)}$乘一个ID的长度，而一个ID可编码成 1+k(f(n)+1)个符号，故时间界限是$c^{f(n)}$，这里c是某个常数。 □

定理 8.4.2（Savitch定理）如果L在NSPACE（S(n)）中，那么L在DSPACE（S(n)）中，这里假定 S(n)是完全空间可构造的，且$S(n)\geq \log_2 n$。

证　设L=L（M_1），这里M_1是一个非确定的S(n)空间有界的TM。对于长度为n的一个输入，最多有$c^{S(n)}$个ID，这里c是某个常数。于是，如果M_1接受它的输入，那么它最多通过$c^{S(n)}$个动作就能做到这点。因为在M_1的导致接受的最短计算中，没有ID会重复。

让$I_1|^{(i)}-I_2$表示；ID I_2可以通过最多 2^i个动作由I_1达到。对于$i\geq 1$，通过测试每个I'，看看是否有$I_1|^{(i-1)}-I'$和$I'|^{(i-1)}-I_2$，我们可以确定是否有$I_1|^{(i)}-I_2$。因此，为确定能否在 2^i个动作内有一个ID得到另一个ID所需的空间，就等于为记录现在正被测试的ID　I'所需的空间加上确定能否在 2^{i-1}个动作内由一个ID得到另一个ID所需的空间。可以看出，测试一个ID是否可在 2^{i-1}个动作内由另一个达到时所使用的空间，可以在每个这样的测试中重复使用。

测试w是否在L（M_1）中的细节已在图 7.5 中给出，图 7.5 的算法可以在一个图灵机M_2上实现，M_2使用一条带来作为调用TEST时运行记录的栈。每次调用都有一个运行记录。参数I_1，I_2，和i的值以及局部变元I'的值都置于其中。因为I_1，I_2和I'都是单元数不超过S（n）的ID，故都能在S（n）空间中表示。在二进制下，输入带头的位置使用$\log n\leq S(n)$个单元。注意，

在所有ID中，输入带是固定不变的，且同对M_2的输入是一样的，故我们不需要在每个ID中复制输入。参数i可以在二进制下编码，它最多用mS(n)个单元。因此，每个运行记录点用 0（S(n) ）空间。

begin

设|w|=n且m=⌈$\log_2 c$⌉；

设I_0是输入为w时M_1的初始ID；

for每个长度最多为S(n)的终结ID I_f do

if TEST(I_0，I_f，mS(n)) **then accept；**

end；

Procedure TEST(I_1，I_2，i)；

if i=0 且(I_1=I_2或I_1|－I_2) **then return true；**

if i≥1 **then**

for 每个长度最多为 S(n)的 ID I' **do**

if TEST(I_1，I'，i-1)且TEST(I'，I_2，i-1) **then；**

return true；

return false

end TEST

图 8.5 模拟M_1的算法

因为每调用一次TEST，第三个参数就要减 1，开始调用时，i=mS(n)，当i到达 0 时，不产生调用，故在栈上，运行记录数目最大是O（S(n)）。因此，所使用的总空间是O（S^2(n)），根据定理 8.2.1，我们可以重新设计M_2，使得空间正好是S^2(n)。 □

例 **8.4.1**

NSPACE（logn）⊆DSPACE($\log^2 n$)

NSPACE(n^2)⊆DSPACE(n^4)

且NSPACE(2^n)⊆ DSPACE(4^n)

注意，对于S(n)≥n，即使S(n) 是空间可构造而不是完全空间可构造的，Savitch定理仍然成立。开始时，M_2在每个长度为n的输入上，对构造S(n) 的一个TM M进行模拟，取所用的最大空间量作为S(n)，并用这个长度来标出用于运行记录的空间。然而，可以看到，如果我们即使在S^2(n) 空间也无法计算S(n) 的话，我们就不可能轮流通过I_f或I'的一切可能值，而

不至于达到某些占用过量空间的ID。

8.5 转换引理和非确定谱系

在定理 8.3.2 和 8.3.3 中，我们看到，确定的空间和时间谱系是很稠密的。看来，在空间情况下，非确定机器相应的谱系似乎需要一个平方的增长量，在时间的情况下，需要一个指数的增长量，以便模拟用于对角线方法的一个非确定机器，然而，一种转换证法可以用来给出一个稠密得多的非确定机器谱系。我们来表述空间情况下的这种技术。

8.5.1 转换引理

第一步，证明包含关系可以向上转换。例如，假定碰巧NSPACE(n^3)⊆NSPACE(n^2)成立（实际并不成立）。这个关系可以向上转换。方法是用n^2代替n，于是产生

NSPACE(n^6)⊆NSPACE(n^4)

引理 8.2 设S_1.(n)， S_2(n) 和f(n)是完全空间可构造的，且S_2(n)≥ n，f(n)≥n，那么，由

NSPACE(S_1(n))⊆NSPACE(S_2(n))

可以推出

NSPACE(S_1(f(n)))⊆NSPACE(S_2(f(n)))

证 设L_1被M_1接受，而M_1是一个非确定S_1（f(n)）空间有界的TM。设

L_2＝｛x\$^{i}|M_1在S_1（|x|+i）空间内接受x｝

这里\$是一个新符号，它不在L_1的字母表中，那么，L_1被一个如下的TM M_2接受。对于输入x\$^{i}，M_2划出S_1（|x|+i）个单元来，这可以办到，因为S_1是完全可构造的。然后，M_2在x上模拟M_1，M_2接受当且仅当M_1接受且使用的单元数不超过S_1（|x|+i）。显然，M_2是S_1（n）空间有界的。

我们做过的事情是在NSPACE（S_1（f(n)））中取一个集合L_1，并用符号\$来增补字符串，使得增补后的语言L_2是在NSPACE（S_1(n)）中，现在，根据假设，NSPACE(S_1（n））⊆NSPACE((（S_2（n））），因此有一个非确定S_2(n) 空间有界的TM M_3，它接受L_2。

最后，我们构造M4，它在空间空间S_2(f(n)) 内接受原来的集合L_1，M_1划出f(n)个单元，然后划出S_2(f(n))个单元，这都能办到，因为f和S_2是完全可构造的。因为S_2(n)≥n，(n) ≤S_2(f(n))，故M_4使用的单元数不超过S_2(f(n))。

然后M_4在输入x上模拟M_3在x\$^{i}上的操作（对于i=0，1，2，…）。为此，M必须记住M_3在x\$^{i}上时带头的位置。如果M_3的带头在x中，M4 的带头就就在其输入的对应位置上。当M_3的

带头移到$中时，$M_4$就将这个位置记在一个计数器中。计数器的长度最长为logi。

如果在模拟过程中，M_3接受，那么M_4接受。如果M_3不接受，那么M_4就增加i，直到计数器不再限于$S_2(f|x|)$）个带单元为止。这里M_4停机。现在，如果x是在L_1中，那么，$x\$^i$是在$L_2$中，这里i满足$S_1$（|x|+i）＝$S_1(f(|x|))$。因为$f(n)\geq n, i=f(|x|)-|x|$将满足这个等式，因此，计数器需要$log(f|x|-|x|)$空间。因为$S_2(f(|x|))\geq f(|x|)$，故推出计数器有足够的空间。于是，$x$在L（$M_4$）中，当豕仅当对于某个$i$,$x\$^i$在L（$M_3$）中，因而，L（$M_4$）＝$L_1$，且$L_1$在NSPACE（$S_2$（f(n))）中。

注意，我们可以条件S_2（n）≥n，只要求$S_2(n) \geq \log_2 n$，但要假定$S_2(f(n))$是完全空间可构造的。这样，M_4可以划出$S_2(f(n))$个单元，而不必去划出 f(n)个单元。因为$S_2(f(n))\log f(n)$，故仍然有空间提供给M_4的计数器。

例 8.5.1 利用确定时间情况下类追认的转换结果，我们可以证明DTIME（2^n）⊆DTIME($n2^n$)。注意，这个结果不能由定理 7.9 推出，因为

$$\inf_{n\to\infty} \frac{2^n \log 2^n}{n2^n} = 1$$

假定

DTIME（$n2^n$）⊆DTIME(2^n)

那么令S_1（n）＝$n2^n$，$S_2(n)$＝2^n和f(n)＝2^n,我们得到

DTIME（$2^n 2^{2n}$）⊆DTIME(2^{2n})

类似地，通过令 f(n)＝n+2^n，我们得到

DTIME（$(n+2^n)2^n 2^{2n}$）⊆DTIME($2^n 2^{2n}$)

将上面两式合起来，我们得到

DTIME（$(n+2^n)2^n 2^{2n}$）⊆DTIME(2^{2n})

但

$$\inf_{n\to\infty} \frac{2^{2n} \log 2^{2n}}{(n+2^2)2^n 2^{2n}} = \inf_{n\to\infty} \frac{1}{n+2^n} = 0$$

于是，由定理 8.3.3 推出 8.8 式不成立，故我们关于DTIME（$n2^n$）⊆DTIME(2^n)的假定必是错的。因为DTIME（2^n）⊆DTIME($n2^n$)，我们得到DTIME（2^n）⊆DTIME($n2^n$)。

例 8.5.2 转换引理可以用来证明NSPACE（n^3）真包含在NSPACE（n^4）中。假定相反，NSPACE(n^4)⊆NSPACE（n^3），而 f(n) ＝n^3,我们得到NSPACE(n^{12})⊆NSPACE（n^9），类似地，令f(n)=n^4，我们得到NSPACE(n^{16})⊆NSPACE（n^{12}）而f(n)＝n^5则给出NSPACE(n^{20})⊆NSPACE

（n^{15}）。把这些放在一起，得到NSPACE(n^{20})⊆NSPACE（n^9）.但根据定理 7.11，我们知道NSPACE(n^9)⊆NSPACE（n^{18}），而根据定理 7.8，DSPACE(n^{18})⊆DSPACE（n^{20}）。因此，把这些结果合在一起，我们得到

NSPACE(n^{20})⊆NSPACE（n^9）⊆DSPACE（n^{18}）

DSPACE(n^{20})⊆NSPACE（n^{20}）

这是一个矛盾。因而，我们关于NSPACE(n^4)⊆NSPACE（n^3）的假定是错的，我们推出NSPACE(n^3)⊊NSPACE（n^4）。

8.5.2 一个非确定空间谱系

可以推广例 8.5.2，以展示多项式范围内非确定空间的一个稠密谱。

定理 8.5.1 如果∈＞0，且 r≥0，那么

NSPACE(n^r)⊊NSPACE（$n^{r+\in}$）

证 如果 r 是任意非负实数，我们就能找到正整数 s 和 t，使得 r≤s/t，r+∈(s+1)/t，因而我们只需证明：对于一切正整数 s 和 t，有

NSPACE($n^{s/t}$)⊆NSPACE（$n^{(s+t)/t}$）

假定相反，即

NSPACE($n^{(s+1)/t}$)⊆NSPACE（$n^{s/t}$）

那么，根据引理 8.2，在$f(n)=n^{(s+i)t}$情况下，我们得到，对于i=0，1，…，s，

NSPACE($n^{(s+1)(s+i)}$)⊆NSPACE（$n^{s(s+i)}$ ）

因为对于 i≥1,s(s+i)≤(s+1)(s+i-1)，故我们知道

NSPACE($n^{s(s+i)}$)⊆NSPACE（$n^{(s+1)(s+i-1)}$ ）

交替使用上面两式，我们有

NSPACE($n^{(s+1)2s}$)⊆NSPACE（$n^{s(2s)}$）

⊆ NSPACE($n^{(s+1)(2s-1)}$)⊆NSPACE（$n^{s(2s-1)}$）

⊆…⊆NSPACE（$n^{(s+1)t}$）⊆NSPACE（n^{s^2} ）

亦即

NSPACE(n^{2s^2+2s})⊆NSPACE（ n^{s^2} ）

但根据 Savitch 定理，

$$\text{NSPACE}(ns^2)\subseteq\text{NSPACE}(n^{s^2})$$

又根据定理 8.3.2，

$$\text{NSPACE}(n^2s^2)\subsetneq\text{NSPACE}(n^{2s^2+2s})$$

显然，

$$\text{DSPACE}(n^{2s^2+2s})\subseteq\text{NSPACE}(n^{2s^2+2s})$$

将这些结果合在一起，我们得到

$$\text{NSPACE}(n^{2s^2+2s})\subsetneq\text{NSPACE}(n^{2s^2+2s})$$

这是矛盾，我们推出下面的假设：

$$\text{NSPACE}(n^{(s+1)/t})\subseteq\text{NSPACE}(n^{s/t})$$

是错误的。因为反向的包含关系是明显的，我们推出

$$\text{NSPACE}(n^{s/t})\subseteq\text{NSPACE}(n^{(s+1)/t})$$

这里 s 和 t 是任意正整数。 □

在高于多项式的范围中，可以证明有类似的非确定空间稠密谱系，我们把某些这样的结果留作练习。定理 8.5.1 不能直接推广到非确定时间上去。因为 Savitch 定理起了关键作用，但对于时间，还不知道有类似的结果，然而，一个类似于定理 8.5.1 的时间方面的结果已由 Cook 证得。

7.6 一般复杂性量度的性质：间隙定理、加速定理和并定理

本节，我们将讨论复杂性量度的某些非直观性质，尽管我们仅对确定的空间复杂度证明了这性质，然而在下节会看到，它们将被应用到复杂性量度上。

定理 8.3.2 和 8.3.3 表明，空间和时间谱系是非常稠密的。但在这两个定理中，都要求函数是可构造的。这个条件可以去掉吗？回答是否定的，即在确定的空间和时间谱系中存在任意大的间隙。

我们说一个带参数 n 的命题是**几乎处处.(a.e.)真的**，如果对于除有穷个以外的一切 n，命题都是真的。我们说一个命题是**无穷经常（i.o.）真的**，如果该合理对无穷多个 n 都是真的。注意，一命题及其否定可以都是 i.o.真的。

引理 8.3 如果 L 被一个 TM M 接受，而 M 是 a.e.S(n)空间有界的，那么 L 可被一个 S（n）空间有界的 TM 接受。

证 对于使 M 不是 S（n）有界的有穷个 n，我们用有限控制器来接受或拒绝长度为这些 n 的字符串。注意，构造过程不是有效的，因为在没有一个时间界限的情况下，我们无法确定这些字中哪些被 M 接受。 □

引理 8.4 存在一个算法，对于给定的 TM M，输入长度 n 和整数 m，该算法能确定对于某个长度为 n 的输入，M 使用带单元的最大数目是否是 m。

证 对于每个 m 和 n，在长度为 n 的输入上，若要求任何存贮带上所使用的单元数不超过 m，又不重复任何 ID，那么 M 所能做的动作数定有一个界限 t。从每个长度为 n 的输入出发，模拟直到 t 个动作的一切动作序列。

定理 **8.6.1**（Borodin 间隙定理）给定任何一个全递归函数 g(n)≥n，存在一个全递归函数 S（n），使得 DSPACE（S（n））＝DSPACE(g(S(n)))。换言之，在空间界限 S（n）和 g(S(n))之间有个“间隙”，没有任何语言的最小空间复杂度会在这个间隙内。

证 设M_1，M_2，……是TM的一个枚举，设S_i（n）是M_i在长度为n的任何输入上使用的带单元数的最大值。如果M_i总能停机，那么S_i（n）是一个全函数，并为M_i的空间复杂度函数；但若M_i在某个长度为n的输入上不停机，那么S_i（n）就没有定义。我们这样来构造S（n），使得对于每个k，或者

$$S_k(n) \le S(n) \quad \text{a.e.}$$

或者

$$S_k(n) \ge g(S(n)) \quad \text{i.o.}$$

即，没有任何S_k（n）会对几乎所有的n，处于S（n）和g（S（n））之间。

对于一个给定的n 值，在构造S（n）的过程中，我们仅限于注意TM的有穷集合M_1，M_2，…，M_n。这样选择S（n）的值，使得对于 1 和n之间的i，S_i（n）不处于S（n）和g（S（n））之间。如果我们能计算出S_i（n）（1≤i≤n）的最大有穷值，那么我们可以让S（n）等于这个值。但因某些$S_i(n)$没有定义，故我们无法计算出这个最大值。但因某些Si(n)没有定义，故我们无法计算出这个最大值。替代的办法是：开始时，令j=1，看看在我们的有穷集合中，是否有某个M_i，它的$S_i(n)$是在j+1 和g(j)之间。如果有某个这样的$S_i(n)$，就让j为$S_i(n)$，并重复这个过程。如果没有，就让S（n）为j，并结束。因为只考虑有穷个TM，又根据引理 8.4，对于任意固定的m，我们能确定是否有$S_i(n)=m$，故这个过程最终将计算出一个j值，使得对于 1≤i≤n，要么$S_i(n) \le j$，要么$S_i(n)>g(j)$。规定S(n)等于这个j值。

假定有某个语言L，它在DSPACE（g(S(n))中，但不在DSPACE（S(n)）中，那么有某个k，使得L＝L（Mk），且对于切n，$S_k(n)\leq g(S(n))$。根据S(n)的构造，对一切$n\geq k$，$S_k(n)\leq S(n)$，也就是说，$S_k(n)\leq S(n)$ a.e.，因此，根据引理 7.3，L将在DSPACE（S(n)）中，这就产生了矛盾。我们推出，DSPACE（S(n)）＝DSPACE(g(S(n)))。

定理 8.6.1 以及对其它三个复杂性量度的类似结果，都有若干高度非直观的推论，诸如下述推论。

例 8.6.1 有一个全递归函数 f(n)，使得

DTIME（f(n)）＝NTIME（f(n)）＝DSPACE（f(n)）＝NSPACE（f(n) ）

显然，DTIME（f(n) ）包含在NTIME（f(n)）和DSPACE（f(n)）中。类似地，NTIME（f(n)）和DSPACE（f(n)）又都包含在NSPACE（f(n)）中。根据定理 7.10，对于所有的 $f(n)\geq \log_2 n$，如果L是在NSPACE（f(n)）中，那么存在一个常数c，它仅依赖于L，使得L是在DTIME($c^{f(n)}$)中。因而，有某个TM M，使得L＝L（M），且M的时间复杂度从上面界于$f(n)^{f(n)}$ a.e.。根据对于DTIME 的类似于定理 7.13 的定理，当$g(x)=x^x$时，可以得到这样一个f(n)，使得DTIME（f(n)）＝DTIME（$f(n)^{f(n)}$），这就证明了上述结果。

类似地，如果人们有两种通用的计算模型，然而，其中一个很简单，又非常慢，比如说一个图灵机，它一百年做一个动作，而另一个非常快，比如说，一个随机存取机器，有很强的乘法、方幂等内部指令，每秒执行一百万次运算。容易证明，存在一个全递归函数，使得在一个模型上 T（n）时间内可计算的任何函数，在另一个模型上也是 T（n）时间内可计算的。

8.6.1 加速定理

关于复杂性量度的另一个奇怪现象是：有一些函数，它们没有最好的程序（图灵机）。我们已经看到，对每个 TM，都允许时间上的线性加速，以及空间上的压缩。我们现在证明，有一些语言，它们没有“最好的”程序。也就是说，这些语言的识别器可以无限加速。我们将只处理空间情况，并证明在一个语言 L，使得对于任何接受 L 的图灵机来说，永远存在另一个图灵机接受 L，并且只使用，比如说，前一个机器使用的空间的平方根空间。这个新识别器自然又可被一个更快的识别器代替等等，无穷加速下去。

证明的基本思想十分简单。用对角线法，我们构造一个 L，使得 L 不能很快地被任何一个“小的”机器识别，所谓小的机器就是一个具有较小整数下标的机器，这里整数下标即机

器的编码。随着机器下标的增长，对角线过程就使接受 L 的机器越来越快。给定任何识别 L 的机器后，这个机器就有某个固定的下标，因此，它只能这样快的识别 L。然而，具有更大下标的那些机器，就能以任意更快的速度识别 L。

定理 8.6.2（Blum加速定理）设r(n)是任意全递归函数。存在一个递归语言L，使得对于任意一个接受L的图灵机M_i,都存在一个图灵机M_j,M_j接受L,且对几乎所有的n,$r(S_j(n))\leq S_i(n)$。

证　不失一般性，假定 r(n)是一个单调不减的完全空间可构造函数,且 r(n)≥n2。定义 h(n)如下：

h(1)=2,h(n)=r(h(n-1))

那么，正如读者容易证明的那样，h(n)是一个完全空间可构造函数。

设M_1，M_2，…是所有离线TM的一个枚举，它类似于前面单带TM的枚举。特别地，我们假定，M_i代码的长度是$\log_2 i$。构造L，使之满足

（1）如果L（M_i）＝L，那么S_i（n）≥h(n-i) a,e.;

（2）对于每个k，在一个图灵机M_j,使得$L(M_j)=L$,且$S_j(n)\leq h(n-k)$。

上面关于L的条件，能确保对于每个接受L的M_i，都存在一个接受L的M_j，满足

S_i（n）≥$r(S_j(n))$　a.e

为看清这一点，选择M_j，使$S_j(n)\leq h(n-i-1)$。根据（2）M_j存在。然后，根据（1），

$S_i(n)\geq h(n-1)=r(h(n-i-1))\geq r(S_j(n))$ a.e.

现在我们来构造$L\subseteq 0^*$，使之满足（1）和（2）。对于，1，2，…，我们依次规定0^n是否在L中。在这个过程中，某些M_i被规定为“被删掉的”。一个被的删掉的TM肯定不接受L。设σ(n)是一个最小的整数j≤n，它使得S_j（n）＜h(n-j)，而M_j又不曾被i=0,1,…,n-1 删掉的。当我们考虑n时，如果σ(n)存在，就规定$M_{\sigma(n)}$为被删掉的。然后，0^n被放入L中，当且仅当σ(n)存在且0^n不被$M_{\sigma(n)}$接受。

其次，我们证明L满足条件（1），即如果L（M_i）＝L，那么$S_i(n)\geq h(n-1)$ a.e.。设L（M_i）＝L，在构造L的过程中，所有被删掉的TM　$M_j(j<i)$，都是在考虑了有穷个n以后（比如说直到n_0）被删掉的。注意，n_0不能有效地计算出来，但它是存在的。假定对于某个$n>\max(n_0, i)$，$S_i(n)<h(n-i)$。当我们考虑n时，没有满足j<i的M_j会被删掉。于是σ(n)＝i，且M_i将被删掉，倘若它在先前被删掉的话。但一个被删掉的TM肯定不会接受L。因此，当$n>\max(n_0, i)$时，$S_i(n)\geq h(n-i)$，即$S_i(n)\geq h(n-i)$ a.e.。

为了证明条件（2），我们要表明，对于给定的k ，存在一个TM M=M_j，使得$L(M_j)=L$，

且对一切n，$S_j(n)\le h((n-k)$。为了确定 0^n是否在L中，M必须在 0^n上模拟$M_{\sigma(n)}$ 。为了确定σ(n)，M必须确定那些M_i已经被 0^l删除掉了，这里l<n。然而，要构造被删除掉的TM表，马上就要看看M_i是否使用了超过h(l-i)的空间，这里 $0\le l\le n$，$1\le i\le n$。对于i<k+l-n，这就需要超过h(n-k)的空间。

解决的办法是要注意，任何一个被删掉的TM $M_i(i\le k)$，都在我们考虑某个小于一个特定值n_1的l时被删掉。对于每个$l\le n_1$，我们把 0^l是否在L中的信息放到M的有限控制器中，同时也把任何$l<n_1$删掉的所有TM M_i的表放到有限控制器中，因此，若$n\le n_1$，M就完全不需要任何空间了。如果$n>n_1$，为了计算σ(n)和在 0^n上模拟$M_{\sigma(n)}$，只需在输入 0^l上模拟TM M_i，这里$n_1\le l\le n$，k<i≤n，并看看M_i是否被l删掉。

为了检查M_i是否被l删掉，我们只需要使用h(l-i)个M_i的单元来模拟M_i，这个数目小于h(n-k)，因为$l\le n$，i>k。因$n>n_1$，故若σ(n)存在，σ(n)就必大于k。因此，在输入 0^n上模拟$M_{\sigma(n)}$将占用h(n-σ(n))个$M_{\sigma(n)}$的单元，这个数小于h(n-k)。

最后，我们必须证明，M可以被构造在空间h(n-k)内操作。我们只需要在输入 0^l上模拟TM M_i，这里k<i≤n，$n_1\le l\le n$，并看看这些M_i是否被删掉，故对于任何一个模拟，我们只需要表示不超过h(n-k-1)个的M_i带单元。因为$i\le n$，M_i的整数代码的长度不大于$\log_2 n$。因此，M_i的任何带符号都能用$\log_2 n$个M的单元进行编码。因为$r(x)\ge x^2$，故知道$h(n-k-1)\ge 2^{2^x}$ 。根据h的定义，$h(n-k)\ge\log_2 n$ a.e.，故对几乎所有的n，用h(n-k)空间来模拟都足够。

除了模拟TM所需的空间，还需要用来保存被删掉的TM表的空间，这个表最多包含n个TM，每个TM的代码长度至多为$\log_2 n$。保存被删掉的TM表所需要的空间nlogn，它也小于等于h(n-k)a.e.。根据引理 7.3，可以对M进行修改，使之在有限控制器中识别这样一些字 0^n，这里$n\log_2 n>h(n-k)$或 $2^{2n-k-1}<\log_2 n$。最后所得的TM具有空间复杂度h(n-k)（对一切n）,这正是我们所要的M。 □

8.6.2 并定理

本节的最后一个定理称为并定理，它研究的是复杂性类的命名问题。作为引言，我们知道，每个诸如n^2或n^3的多项式，都定义了一个空间复杂性类，（以及其它三种形式的复杂性类）。然而，多项式空间是否构成一个复杂性类呢？也就是说，是否存在一个S(n) ，使得DSPACE(S(n))包含了一切在一个多项式空间界限内能够识别的集合，而不包含其它集合

呢？显然，S(n)必须几乎处处大于任何一个多项式，但它又必须足够的小，以致不能将另外的函数插在它和多项式之间，而这个函数又是某个TM使用的空间。这里的“插在”必须当作一个术语，下面的定理中精确地定义了其含义。

定理 8.6.3 设$\{f_i(n)|i=1,2,\ldots\}$是递归函数的一个递归可枚举集合。也就是说，有一个TM，它枚举一系列的TM，这些TM中的第一个计算f_1，第二个计算f_2等等。又假设对于每个i和n，$f_i(n)<f_{i+1}(n)$，那么存在一个递归的S(n)，使得

$$DSPACE(S(n))=\bigcup_{i\geq 1} DSPACE(f_i(n))$$

证 我们来构造一个函数 S(n)，它满足下面两个条件：

(1)对于每个i，$S(n)\geq f_i(n)$ a.e. .

(2)如果$S_j(n)$恰好是某个TM M_j的空间复杂度，且对于每个i，$S_j(n)> f_i(n)$ i.o.，那么对于某个n(事实上，对于无穷多个n)，$S_j(n) >S(n)$。

第一个条件能保证：

$$\bigcup_i DSPACE(f_i(n))\subseteq DSPACE(S(n))$$

第二个条件能保证，DSPACE(S(n)) 仅仅包含那样一些集合，即有某个i，该集合在$DSPACE(f_i(n))$中，将两个条件合起来，就推出

$$DSPACE(S(n))\subseteq \bigcup_i DSPACE(f_i(n))$$

如果设定$S(n) =f_n(n)$，就能保证条件(1)。但可能不满足条件(2)。可能有一个TM M_j，其空间复杂度$S_j (n)$大于每个$f_i(n)$ i.o.，但对一切n，$S_j(n)<f_n (n)$。因此，可能有一些集合在$DSPACE(f_n(n))$中，但不在 $\bigcup_i DSPACE(f_i(n))$中。为解决这个问题，我们构造一个S(n)，使之降到每个$S_j(n)$之下，这里的 $S_j(n)$ 是i.o.大于每个$f_i(n)$的，事实上，S(n) 将对无穷多个n降到$S_j(n)$之下。这可以办到，只要对每个TM M_j，猜一个i_j，使$f^{i}{}_j(n)\geq S_j(n)$ a.e.这个“猜测”不是非确定的，而是属于如下确定形式的。如果在某个时刻，我们发现，猜测不正确，我们就猜一个更大的值来作为i_j，对某个特定的n，定义S(n) ，使之小于$S_j(n)$。如果碰巧S_j 增长速度越过任何一个f_i，S将无穷经常地小于S_j。另一方面，如果某个f_i几乎处处大于S_j ，那么我们最终将猜出一个这样的f_i，并停止对S赋小于S_j的值。

在图 8.6 中，我们给出一个产生S(n) 的算法。这里保存有一个称作LIST的表，表中有形如“$i_j=k$”的一些猜测，j和k是各种整数。对于每个j，在任何时刻，LIST中将最多有一个猜测k。和前一定理一样，M_1，M_2，…是一切离线TM的一个枚举，而$S_j(n)$ 是M_j 在任何长度为n

的输入上所用空间的最大值。我们知道，对某些n值，$S_j(n)$ 可能没有定义(即为无穷)。

为了证明

$$DSPACE(S(n))=\bigcup_i DSPACE(f_i(n))$$

我们首先证明 S(n)满足条件（1）和（2）。考虑条件（1）。

Begin

1) LIST:=空表

2) **for** n=1，2，3，…**do**

3) **if** 对于LIST 中一切“$i_j=k$”,$f_k(n)\geq S_j(n)$ **then**

4) 将“$i_n=n$”加到LIST中，定义$S(n)=f_n(n)$

else

begin

5) 在LIST 的使$f_k(n)<S_j(n)$”的一切猜测中，令“$i_j=k$”是具有最小k的，且给定这个k后，又具有最小j的猜测；

6) 定义$S(n)=f_k(n)$

7) 用“$i_j=n$”替换LIST中的“$i_j=k$”;

8) 将“$i_n=n$”加到LIST中

end

end

图 8.6 S(n)的定义

为了看出对于每个m，$S(n)\geq f_m(n)$ a.e. 我们注意，仅在图 7.6 的行(4)和(6)中，S(n)才被赋给一个值。当S(n) 在第(4)行对n≥m定义时，S(n) 的值至少是$f_m(n)$。因此，对于在行(4)被定义的S(n)值，除了小于m的有穷个n以外， $S(n)\geq f_m(n)$。现在考虑行(6)定义的S(n) 值。当n达到m时，LIST将会有有穷个猜测。每个这样的猜测，后来在某个n>m时，都可能使S(n) 有一个值，这个值小于$f_m(n)$。然而，当发生这种情况时，在行(7)，这个猜测将被一个的猜测”$i_j=p$”(p≥m)代替，而这个猜测，如果在行(5)被选出，它就不再使S(n)小于$f_m(n)$了，因为当p≥m 时，$f_p(n)\geq f_m(n)$。因此，由行(6)只会产生有穷多个大于m的n(其个数至多为n=m时表LIST的长度) ，对于这些n,$S(n)<f_m(n)$，.因为只有有穷多个n小于m，故 $S(n)\geq f_m(n)$ a.e.。

其次，我们必证明条件(2)，即如果存在TM M_j，使得对于每个i，$S_j(n)>f_i(n)$ i.o.,那么有无穷多个n'，使$S_j(n')>S(n')$。在n=j以后，无论何时，LIST将含有一个对i_j的猜测，且LIST总

是有穷的。对于n=j，我们把“$i_j=j$”放到LIST中。因为$S_j(n)>f_j(n)$ i.o.，故将有任意多个后面的n值，使第(3)行的条件不成立。每当这样的时刻，或者我们的“$i_j=j$”在行(5)被选中，或者n=j时在LIST中的有穷多个猜测中的某个另外的猜测被选中。对于后面这种情况，该猜测将被一个形如“$i_p=q$”的猜测所代替，这里q>j。加到LIST中的一切猜测都具有$i_p=q(q>j)$的形式，故我们的“$i_j=j$”最终将会在第(5)步被选中，并且对n的这个值，我们有$S_j(n)<fj(n)=S(n)$。因此条件(2)成立。

最后，我们必须证明，由条件(1)和(2)能推出：

$$DSPACE(S(n))\subseteq \bigcup_i DSPACE(f_i(n))$$

假定L在$\bigcup_i DSPACE(f_i(n))$中，那么有某个m，L在$DSPACE(f_m(n))$中。根据条件(1)，$S(n)\geq f_m(n)$ a.e.。于是根据引理 7.3，L在DSPACE(S(n))中。现在假定L在DSPACE(S(n))中。设$L=L(M_j)$，这里对一切n，$S_j(n)\leq S(n)$。如果没有i使L在$DSPACE(f_i(n))$中，那么根据引理 7.3，对于每个i, 每个接受L的TM M_k，都有$S_k(n)>f_i(n)$ i.o. 。因此根据条件(2)，存在某个n，使$S_k(n)>S(n)$。令k=j，产生矛盾。 □

例 8.6.2 令$f_i(n)=n^i$。我们肯定可以枚举出一个TM 序列M_1，M_2，…，使得在以 0^n为输入时，M_i能在其带上写出0^{n^i}，并停机。于是，定理 8.6.2 表明，有某个S(n)满足：

$$DSPACE(S(n))\subseteq \bigcup_i DSPACE(n^i)$$

因为每个多项式都小于等于某个n^i a.e.，故DSPACE(S(n))也是在一切多项式p(n)上DSPACE(p(n))的并。可以将这个并看成是一个确定的空间复杂性类，在难解问题理论中，它将起重要作用。

8.7 公理化复杂性理论

读者也许注意到，本章的许多定理并不依赖于下述事实：我们正在测量的是所使用的时间和空间的数量。而仅仅依赖于这样一个事实，我们正在测量的是某种资源，它们是计算过程上正在被消耗着的。事实上，我们可假定各种资源应满足的一公理，并给出复杂性理论的完全公理化的研究。本节，我们将概述这种方法。

8.7.1Blum 公理

设M_1，M_2，…是图灵机的一个枚举，这些图灵机定义了每个部分递归函数。由于技术上的原因，我们把这些M_i看成是计算部分递归函数ϕ_i的机器，而不是识别集合的机器。原因在于：在测量时，把复杂度当作输入的函数比当作输入的长度的函数，在记号上要简单些。设$\phi_i(n)$是被M_i计算的一元函数，又设$\Phi_1(n),\Phi_2(n),\ldots$是满足下面两个公理（Blum公理）的部分递归函数的集合。

公理 1　$\Phi_i(n)$有定义，当且仅当 $\phi_i(n)$有定义。

公理 2　如下定义的函数R量个全递归函数：R（i，n，m）被定义为 1，如果$\Phi_i(n)$ ＝m，否则被定义为 0。

函数$\Phi_i(n)$ 给出了第i个图灵机对输入n的计算的复杂度，公理 1 要求：$\Phi_i(n)$ 有定义，当且仅当第i个图灵机对输入n能停机。因此，一种可能的$\Phi_i(n)$ 将是第i个图灵机的步数。所使用的空间量将是另一种可能的Φ_i，倘若当TM进入循环时，我们定义所使用的空间为无穷。

公理 2 要求，我们能够确定第i个图灵机对输入n的复杂度是否为m。例如，如果我们的复杂性量度是计算过程中的步数，那么给定i，m和n以后，我们可以在 0^n上对M_i模拟m步，并看看是否停机。引理 7.4 及类似的结果表明，对于我们曾考虑过的四种量度，公理 2 都成立。

例 8.7.1　确定的空间复杂度满足Blum公理，倘若M_i对输入 0^n不停机时，我们假定$\Phi_i(n)$无定义，即使Mi在 0^n上所用的空间量是有界限的。确定的时间复杂度同样也满足这些公理，倘若当M_i永远运行时，或者虽停机但在带上没有任何 0^j时，我们都规定$\Phi_i(n)$无定义。为了计算R（i，n，m），只要在输入 0^n上，对M_i模拟m 步。

可以证明，非确定的时间和空间满足这些公理，如果我们对一个NTM计算一个函数的含义作一个恰当定义的话。例如，我们可规定：$\phi_i(n)$＝j当且仅当 0^n为输入时，M_i有某个选择序列，它导致M_i停机，同时带上是 0^j；而不会有任何其它选择序列，也使M_i停机但带上为 0^k，这里k≠j。

如果我们定义$\Phi_i(n)$＝ϕ_i (n)，就不会满足公理 2。假定R（i，n，m）是递归的，那么存在一个算法，来辨别以 0^n为输入时M_i是否停机，并在带上留下 0^m。给定任何TM M，我们可以构造 $\hat{M}$ 来模拟M。如果M停机，有任意的带内容，那么 $\hat{M}$ 就擦去它自己的带。如果i是 $\hat{M}$ 的下标，那么R（i，n，m）为 1，当且仅当在输入 0^n上M能停机。因此，如果R（i，

n，m）是递归的，我们就能辨别一个给定的TM　M，在给定的输入上是否停机，而这个问题是不可判定的。

8.7.2 复杂性量度间的递归关系

关于复杂性的许多定理，可以仅由上面两个公理证明出来。特别地，下述结果可以这样来证明：存在任意的复杂函数、加速定理、间隙定理和并定理。这里我们只证明一个定理，以说明这种技术。我们所选的定理是：所有量度都是递归相关的。也就是说，给定任何两个复杂性量度Φ和$\hat{\Phi}$，有一个全递归函数r，使得TM M_i在一种量度下的复杂度$\hat{\Phi}_i(n)$最大为r(n，$\Phi_i(n)$)。例如定理 7.10 和 7.11 曾证明，对于我们已经研究过的四种复杂性量度，最多用一个指数函数就可以将任意两个这样的复杂性量度联系起来。从某种意义上说，在一种量度下是容易的函数，在任何其它量度下也是“容易”的，虽然“容易”这个术语必须作弱一点的理解，因为r可能是一个增长迅速的函数，诸如Ackermann函数。

定理 8.7.1　设Φ和$\hat{\Phi}$是两个复杂性量度，那么存在一个递归函数 r，使得对于一切 i，

$$r(n,\Phi_i(n)) \geq \hat{\Phi}_i(n) \text{ a.e.}$$

证　令

$$r(n，m)=\max\{\hat{\Phi}_i(n) \mid i\leq n且\Phi_i(n)=m\}$$

函数r是递归的，因为根据公理 2，可以检验是否有$\Phi_i(n)=m$。如果它真的等于m，那么根据公理 1，$\phi_i(n)$和$\hat{\Phi}_i(n)$必有定义，从而能够计算出最大值来。显然，对于一切n≥i，r(n，$\Phi_i(n)$)≥$\hat{\Phi}_i(n)$，因为对于n≥i，r(n，　$\Phi_i(n)$)至少是$\hat{\Phi}_i(n)$。　□

虽然公理化方法是精巧的，并使我们一个更为广泛的范围内证明一些结果，但是它至少还不能抓住我们直观的复杂度概念的一个重要方面。如果我们构造一个图灵机M_k，M_k首先在n上执行M_i，然后在其结果上执行M_j，我们能期望，M_k在n上的复杂度至少要象M_i在n上的一样大。然而，对于有些复杂性量度，情况却并非如此。换句话说，进行更多的计算，我们反倒可以减少我们已经做过的计算的复杂度。我们把构造这样一个复杂性量度的任务留作练习。

习 题

1 写出完全空间可构造函数的定义。

2 证明[$\sqrt{x}$]+1 是完全空间可构造的。

3 证明函数$s(n)=n^2$是完全时间可构造的。

4 证明$DTIME(2^{kn})\subset DTIME(n2^{kn})$，其中 k 为一常数。（$\subset$表示真包含）

5 令函数 $$f(n,m)=c_n^m=\frac{n(n-1)\cdots(n-m-1)}{m!}$$，（$m\le n$）

用双存贮带离线 TM 计算函数$f\left(n,\left[\frac{n}{2}\right]+1\right)$，并使其空间和时间复杂度均为$O(2^n)$。

6 用离线图灵机证明：$\left[\sqrt{n}\right]+3$是空间可构造函数。

（限制： a） 只有一条存贮带

b)输入带与存贮带都是单道的

c)带上符号除空格 B 外只有一个符号“1”。）

7 证明函数$(n+1)^2$是完全时间可构造的。

8 证明 DTIME(2^{3n})$\subset$DTIME($n2^{3n}$)。

第九章 NP 完全问题简介

在前面几章中，我们发现，人们能够提出一些在计算机上不可解的问题。本章，我们将看到，在可判定问题中，某些问题十分困难，以致从各种实用目的来看，总的说来它们都是在计算机上不能解决的。这类问题虽是可判定的，但已证明，欲求其解，需要指数时间。另外有些问题，对它们求解，可能非常需要指数时间。如果有一种比指数时间更快的方法来解决它们，那么在数学、计算机科学和其它领域中的一大批重要问题（对于这些问题，为了寻求好的解法，人们花费了许多年的时间，仍无结果）就能用比现在更好的方法加以解决。

9.1 *P*类和 *NP*类

本节介绍 NP 完全问题的基本概念：问题的 *P* 类和 *NP* 类（即确定型 TM 和非确定型 TM 分别在多项式时间里能解答的问题），以及多项式时间归约的技术。本节还定义:“NP 完全性“的概念，即 *NP* 中某些问题所具有的性质：这些问题至少是和 NP 中任意问题一样难的（在时间个至多相差多项式）。

9.1.1 可在多项式时间内解答的问题

如果当给定图灵机TM M长度为n的输入w时，TM M无论接受与否，都至多移动T(n)步之后停机，则说TM M具有时间度量复杂性T(n)（或具有“运行时间T(n)”）。这个定义适用于任意函数T(n)，比如$T(n)=40n^2$或$T(n)=2^n+7n^5$；我们主要是对T(n)是n的多项式的情形感兴趣。如果存在某个多项式T(n)和某个具有时间复杂性T(n)的确定型TM M，使得L=L(M)，则说语言L属于*P*类。Kruskal算法就是可在多项式时间内求解最小生成树的一个例子。

9.1.2 非确定型多项式时间

难解性研究中基本的问题类是在多项式时间里运行的非确定型 TM 能解答的问题。形式化地说，如果存在非确定型 TM M 和多项式时间复杂性 T(n)使得语言 L=L(M)，并且当给定 M 长度为 n 的输入时，M 没有移动序列超过 T(n)步，则说 L 属于 *NP* 类（非确定型多项式）。

第一个事实是：因为每台确定型 TM 都是从来也不选择移动的非确定型 TM，所以 $P\subseteq NP$。但 *NP* 似乎包含了许多不属于 *P* 的问题。直觉的理由是在多项式时间里运行的 NTM 有能力猜测问题的指数个可能解，并在多项式时间里“并行地”验证每个解。无论如何：

是否 *P*=*NP*，即是否 NTM 在多项式时间里能够做到的每一件事情事实上 DTM 在多项式时

间（也许更高次的多项式）里也能做到，这是数学中最深奥的未解决问题之一。旅行商问题(TSP)就是一个典型的 *NP* 问题的例子。

9.1.3 NP 完全问题

下面将遇到最著名的属于 *NP* 而不属于 *P* 的候选问题族。设 L 是 *NP* 中的一个语言（问题）。如果下列关于 L 的命题为真则说 L 是 NP 完全的：

1L 属于 *NP*。

2 对于 *NP* 中每个语言 L'，都存在从 L' 到 L 的多项式时间归约。

即将看到，一个 NP 完全问题的例子是旅行商问题。由于似乎 *P≠NP*，具体地说，似乎所有 NP 完全问题都属于 *NP-P*，所以通常认为，一个问题的 NP 完全性证明就是这个问题不属于 *P* 的证明。

通过证明每个多项式时间 NTM 的语言都有到所谓 SAT 问题的多项式时间归约，将证明第一个 No 完全问题 SAT（表示布尔可满足性）。但是，一旦有了一些 NP 完全问题，则通过使用多项式时间归约，把某个已知 NP 完全问题归约到新的问题上，就能证明新的问题是 NP 完全的。下面定理说明为什么这样的归约证明目标是 NP 完全的。

定理 9.1.1 若P_1是NP完全的，并且存在从P_1到P_2的多项式时间归约，并且P_2属于*NP*，则P_2是NP完全的。

证明　需要证明*NP*中每个语言L都多项式时间归约到P_2。已知存在从L到P_1的多项式时间归约；这个归约花费某个多项式时间p(n)。因此，L中长度为n的串w转换成P_1中长度至多为p(n)的串x。

还已知存在从P_1到P_2的多项式时间归约；设这个归约花费多项式时间q(m)。于是这个归约至多花费q(p(n))时间把x变换成P_2中某个串y。因此，从w到y 的变换至多花费p(n)+q((p(n))时间，这个时间是多项式的。结论是：L可多项式时间归约到P_2。L可能是*NP*中任意语言，所以已经证明*NP*的所有语言都多项式时间归约到P_2；即P_2是NP完全的。

还要证明一个更重要的关于 NP 完全问题的定理：如果任何一个 NP 完全问题属于 *P*，则所有 *NP* 问题都属于 *P*。由于人们深信，*NP* 中有许多问题不属于 *P*，因此就把一个问题是 NP 完全的这样的证明当作这个问题没有多项式时间算法，因此没有好的计算机解决的同等证明。

定理 9.1.2 若某个 NP 完全问题 P 属于 *P*，则 *P=NP*。

证明 假设 P 既是 NP 完全的又属于 *P*。则 *NP* 中所有语言 L 都在多项式时间里归约到

P。在前面曾讨论过，若 P 属于 P，则 L 属于 P。

9.2 多项式时间和空间

确定的多项式时间内能识别的语言，构成一个自然的重要的类，即$\cup_{i\geq D}$TIME（n^i），我们将它记作$\mathcal{J}$。下述想法在直觉上是很有吸引力的：$\mathcal{J}$是一个能有效解决的问题组成的类。虽然人们会争辩说，一个n^{57}步的算法不是很有效的，但在实践中我们发现，$\mathcal{J}$中的问题通常都具有低阶多项同式时间解法。

有一些重要的问题，看来似乎不在$\mathcal{J}$中，但却有有效的非确定算法。这些问题归入语言类$\cup_{i\geq D}$TIME（n^i），我们用N$\mathcal{J}$来表示它。汉密尔顿回路问题就是一个例子，该问题是：在一个图中，是否存在一个回路，在这个回路上，该图的每个顶点都恰好出现一次？看来不象是有一个确定的多项式时间算法，它能识别具有汉密尔顿回路的图。然而，却有一个简单的非确定算法：猜测在回路中的边，然后验证它们是否真的构成一 个汉密尔顿回路。

$\mathcal{J}$ 和 N$\mathcal{J}$ 的区别类似于下述区别，即有效地寻找一个命题（诸如“这个图有一个汉密尔顿回路”等）的证明和有效地验证一个证明（亦即检查一个具体回路是否是汉密尔顿回路）之间的区别。直观上，我们感到检查一个给定的证明比寻找一个证明更容易。但我们不知道这是不是事实。

另外两个自然的类是：

$$\text{PSPACE}=\bigcup_{i\geq 1} DSPACE(n^i)$$

和

$$\text{NSPACE}=\bigcup_{i\geq 1} NSPACE(n^i)$$

注意，根据Savitch定理（定理 8.4.2），PSPACE＝NSPACE，这是因为NSPACE（n^i）⊆DSPACE(n^{2i})。显然，$\mathcal{J}\subseteq$N$\mathcal{J}\subseteq$PSPACE，现在还不知道，在这些包含关系中，是否有真包含，不仅如此，正象我们将要看到的，解决这些问题所需要的这样或那样的数学工具，看样子还没有建立起来。

在 PSPACE 中，我们有复杂性类的两个谱系：

DSPACE(logn)⊊DSPACE($\log^2 n$) ⊊NSPACE($\log^3 n$) ⊊....

和

NSPACE(logn)⊊DSPACE(log²n) ⊊NSPACE(log³n) ⊊....

显然，DSPACE(logᵏn) ⊆ NSPACE(logᵏn)。因此，根据 Savitch 定理，

$\bigcup_{k\geq 1} NSPACE(\log^k n) = \bigcup_{k\geq 1} DSPACE(\log^{ki} n)$

虽然人们能证明 $\mathcal{P} \neq \bigcup_{k\geq 1} DSPACE(\log^{ki} n)$

但还不知道是否其中任何一个类包含在另一个类中。然而 DSPACE(logn) $\subseteq \mathcal{P} \subseteq$ N$\mathcal{P} \subseteq$ PSPACE

且至少有一个包含关系是真包含，因为根据空间谱系定理，DSPACE(log) ⊊PSPACE。

还记得，在上章中，我们证明一个语言 L 是不可判定的，方法是取出一个已知不可判定的语言 L'，并将它归约到 L。也就是说，我们给出了一个映射 g，它被一个总是能停的 TM 计算，且对一切字符串 x，x 是在 L'中当且仅当 g(x)是在 L 中。那么若 L 是递归的，L'就可通过计算 g(x)和判定 g(x)是否在 L 中的办法加以识别。

通过把 g 限制为一个容易的可计算函数的办法，我们就能确定，L 是不是在某个类中，例如是不是在 $\mathcal{P}$，N$\mathcal{P}$ 或 PSPACE 类中。我们将对下述两类可归约性特别感兴趣：多项式时间可归约性和 log 空间可归约性。我们说 L'是多项式时间可归约到 L 的，如果有一个多项式时间有界的 TM，对于每个输入 x，它都产生一个输出 y，使得 y 在 L 中，当且仅当 x 在 L'中。

引理 1 设 L'是多项式时间可归约到 L 的，那么

(a)若 L 在 N$\mathcal{P}$ 中，L'就在 N$\mathcal{P}$ 中；

(b)若 L 在 $\mathcal{P}$ 中，L'就在 $\mathcal{P}$ 中。

证 (a)和(b)的证明是类似的。我们只证(b)。假定归约是$p_1(n)$时间有界的，又假定L在$p_2(n)$时间内可识别，这里p_1和p_2都是多项式，那么，可用下述方法，在多项式时间内识别L'。给定长度为n的输入x，用多项式时间归约产生y。因为归约是$p_1(n)$时间有界的，每个动作最多打印一个符号，故推出$|y|\leq p_1(n)$。然后，我们可在$p_2(p_1(n))$时间内y是否在L中。因此，确定x是否在L'中的总时间是$p_1(n)+p_2(p_1(n))$，它是n的多项式。因而L'是在$\mathcal{P}$中。

Log 空间转换器是一个总能停机的离线 TM，它具有 logn 起草用的存贮空间，和一条只

写输出带，在这条带上，带头决不向左移，我们说 L’是 log 空间可归约到 L，如果且个 log 空间转换器，对于给定的输入 x，它产生一个输出字符中 y，使得 y 在 L 中，当且仅当 x 在 L’中。

引理 2 如果 L’是 log 空间可归约到 L，那么

(a) 若 L 在 $\mathcal{J}$ 中，则 L’在 $\mathcal{J}$ 中；

(b)若L在NSPACE($\log^k n$)中，则L’是在NSPACE($\log^k n$)中；

(c)若L在DSPACE($\log^k n$)中，则L’是在DSPACE($\log^k n$)中。

证明：

(a)只需证明：log 空间归约所需的时间不会超过多项式时间。这样一来，结论就可从引理 1(b)推出。证明时要注意：输出带不会影响计算，故状态数、存贮带内容所占单元数、输入带头和存贮带头的位置数之乘积，将是 log 空间转换器在必定进入循环之前所能作的动作数的上界，而进入循环将与总能停机的假设相矛盾。如果存贮带的长度为 logn，那么容易看出，这个上界是 n 的多项式。

(b)和(c)的证明只有微小的差别。我们只证(c)，(b)的证明实质上和(c)的证明相同。

(c)设M_1是log空间转换器，它将L’归约到L，又设M_2是一个$\log^k n$空间有界的TM，它接受L。在长度为n的输入x上，M_1产生一个长度不超过n^c的输出，这里c是某个常数。因为这个输出不能写在$\log^k n$空间以内，故不能通过将M_1的输出存贮在一条带上的办法，来模拟M_1和M_2。替代的办法是将M_1的输出存贮在一条带M_2，一次送一个符号，只要M_2在其输入上向右移，这种办法就行得通。如果M_2向左移，M_1就必须重新启动，以便确定送给M_2的输入符号，因为M1 的输出并没有保存起来。

我们如下构造一个M_3，用以接受L’。用M_3的一条存贮带来保存M_2的输入位置(以 2^c为基)。因为输入位置越过n^c，故这个数能存贮在logn空间之内。用M_3的其它存贮带来模拟M_1和M_2的存贮带。假定在某个时刻，M_2的输入带头在位置i，而M_2又要作一个左移或右移。M_3适当地调整一下M_2的状态和存贮带。然后，M_3重新进行对M_1从头开始模拟，直到M_1产生出i-1 或i+1 个输出符号，这取决于M_2的输入带头是向左移速是向右移，最后产生的这个输出符号就是M_2的输入带头要扫视的符号，故M_3可以模拟M_2的下一个动作了。作为特殊情况，如果 i=1 且M_2向左移，那么我们假定M_2的下个扫视的符号是左端记号M_2向右移时，在产生i+1 个输入符号之前，M_1就停机了，那么我们假定M_2扫视的下一个符号是右端记号，M_3接受它的输入，如果M_2接受送给它的那个模拟输入。因此，M_3是一个接受L’的$\log^k n$空间有界的TM。

引理 3 两个 log 空间(或多项式时间)归约的结合，仍是一个 log 空间(多项式时间)归约。

证 简单推广引理 1 和 2 的构造法，即可证明本引理。

9.3 某些 NP 完全问题

NP 完全问题类的重要意义在于：它包含了许多这样的问题，这些问题都是非常自然的，并且人们认真地研究过它们的有效解。如果这些问题中任何一个在 $\mathcal{P}$ 中，那么它们全部都在 $\mathcal{P}$ 中，这个事实加强了这样的想法：它们不象是有多项式时间解。此外，一个新问题被证明是 NP 完全的，那么就象对现有问题一样，我们有同样信心认为，这个新问题也是困难的。

我们要证明其 NP 完全性的第一个问题是布尔表达式的可满足性问题。碰巧，这也是历史上第一个 NP 完全问题。我们从精确地定义这个问题开始。

9.3.1 可满足性问题

一个布尔表达式是一个由变元、括号、运算符∧(逻辑与)、∨(逻辑或)和¬(非)组成的表达式。这些运算的优先性以¬为为最高，然后是∧，最后是∨。变量取值 0(假)和 1(真)。如果E_1和E_2是布尔表达式，那么$E_1 \wedge E_2$的值是 1 倘若E_1和E_2的值都是 1，否则$E_1 \wedge E_2$的值为 0。$E_1 \vee E_2$的值为 1，倘若E_1或E_2的值为 1，否则为 0。$\neg E_1$的值为 1，倘若E_1是 0；$\neg E_1$的值为 0，若E_1是 1。一个表达式是可满足的，如果存在用 0 和 1 对诸变元进行的某个赋值，使得表达式的值为 1。可满足性问题是：对于给定的布尔表达式，确定它是否是可满足的。

我们可以按下述方式将可满足性问题表成一个语言L_{sat}。设某个表达式的变元是$x_1, x_2, \cdots, x_m$(对于某个m)。把x_i编码成符号x，后面再跟着写成二进制形式的i,于是，L_{sat}的字母表就是：

{∧，∨，¬，(，)，x，0，1}

容易看出，具有n个符号的表达式，其编码形式的长度不会大于$\lceil n\log_2 n \rceil$.因为每个非变元符号都编码成一个符号，而在一个长度为n的表达式中，不同变元的个数至多为$\lceil n/2 \rceil$，一个了代码需要的符号不超过 $1+\lceil \log 2n \rceil$，以后，我们将把L_{sat}中的表示一个长度为n的表达式的字，看成仿佛字本身的长度就是n，我们的结果将不依赖于是用n还是nlogn来表示字的长度，因为log(nlogn)≤2logn，而我们的又是log空间归约。

一个布尔表达式称为是合取范式（CNF），如果其形式为$E_1 \wedge E_2 \wedge \cdots \wedge E_k$,这里每个$E_i$(称子式或合取项)的形式又为$a_{i1} \vee a_{i2} \vee \cdots \vee a_{ir}$，I其中每个$a_{ij}$是一个文字，亦即是x或¬x，x是某个变

元。我们通常写$\bar{x}$而不写¬x。例如，$(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge \overline{x_3}$是 CNF.一个表达式称为是 3CNF，如果每个子式恰好有三个不同的文字。上面的例子不是 3CNF，因为第一个子式和第三子式所含文字都少于三个。

9.3.2 可满足性问题是 NP 完全的

首先我们给出NP中每个语言到L_{sat}的一个log空间归约的。

定理 9.3.1 可满足性问题是 NP 完全的。

证 证明L_{sat}在NP中是比较容易的。为了确定一个长度为n的表达式是否是可满足的，非确定地猜测所有变元的值，然后计算表达式的值。因此，L_{sat}在NP中。

为了证明NP中每个语言都可归约到L_{sat}，对于每个多项式p(n)时间有界的NTM M，我们给出一个log空间算法。它以一个字符串x作为输入，产生一个布尔公式E_x，并使得E_x是可满足的，当且仅当M接受x，我们现在来描述E_x。

设$\#\beta_0\#\beta_1\#\ldots\#\beta_{p(n)}$是M的一个计算,这里每个$\beta_i$都是一个恰好由p(n)个符号组成的ID。 如果在第p(n)个动作以前就接受了，我们就让接受的ID重复，这样就使得每个计算都正好有 p(n)+1 个ID。在每个ID中，我们把状态和被扫视的符号结合起来，构成一个单独的组合符号。此外，在第i个ID中的组合符号还包含一个整数m，它指示出一个动作，按这个动作，第 i＋1 个ID可由第个ID推出来。给定状态和带符号后，M可以有若干个可选择的动作，对这些选择的有穷集合任意安排一个顺序以后，就可以给每个可选择的动作分配一个数字。

对于每个能出现在一个计算中的符号和每个i，$0 \le i \le (p(n)+1)^2$，我们都产生一个布尔变元C_{iX}，它指明在该计算中，第i个符号是不是X（在该计算中的第 0 个符号是初始的＃）。对于一个给定的对各个C_{ix}的赋值，我们所要构造的表达式，E为真，当且仅当是真值的那些C_{ix}恰好对应于一个有效计算。表达式E_x表述了下面几件事：

(1)所有其值为真的C_{iX}对应于一个符号串，这里对于每个i，恰好有一个C_{iX}为真。

（2）IDβ_0是M的一个具有输入x的初始ID。

（3）最后一个 ID 包含一个终结状态。

（4）每个 ID 都可按照被指出的 M 的动作由前一个 ID 推出。

公式Ex是四个公式的逻辑与，其中每个公式都体现上述一个条件。第一个公式是说，对于每个在 0 和$(p(n)+1)^2-1$ 之间的i，恰好有一个C_{iC}是真的，这个公式是：

$$\bigwedge_i [\bigvee_X C_{iX} \wedge \neg(\bigvee_{X \neq Y} (C_{iX} \wedge C_{iY}))]$$

对于一个给定的i值，$\bigvee_X C_{iX}$ 这一项体现了至少有一个C_{iX}是真的，而$\neg(\vee_{X \neq Y}(C_{iX} \wedge C_{iY}))$这一项体现了最多有一个是真的。

设$x=\alpha_1\alpha_2...\alpha_n$。第二个公式表示：$\beta_0$是一个初始ID，这个公式是下述各项集资进行的逻辑与：

(a)$C_{0,\#} \wedge_{p(n)+1,,\#}$在 0 位置和p(n)+1 位置上符号是＃。

(b)$C_{1Y1} \vee C_{2Y2} \vee ... \vee C_{1Yk}$，这里$Y_1$，$Y_2$，，$Y_k$是所有如下的组合符号，它们代表了带符号$a_1$、初始状态$q_0$以及在状态$q_0$下且读着符号$a_1$时M的一个合法动作的相应数字。这个子式说明$\beta_0$的第一个符号是正确的。

(c)$\wedge_{2 \le i \le n} C_{iai}$。$\beta_0$的第二个到第n个符号都是正确的。

(d) $\wedge_{n \le i \le p(n)} C_{iB}$的其余符号是空白。

第三个公式是说，最后一个 ID 有一个接受状态。它可以写成 $\bigvee_{p(n)(p(n)+1) \prec i \prec (p(n)+1)^2} \left(\bigvee_{X在F中} C_{iX} \right)$

这里，F 是包含一个终结状态的组合符号的集合。

第四个公式是说，每个IDβ_i(i≥1)都是根据出现在Dβ_{i-1}的组合符号中的动作、由β_{i-1}推导出来的，为了弄清怎样写出这个公式，我们注意β_i的每个符号实际上都可由β_{i-1}的对应符号及其两边符号（其一边可以是＃）推演出来。亦即，β_i中的符号与β_{i-1}中对应的符号相同，只是具有状态和动作的那个符号，以及相邻符号中那个曾具有状态和动作的符号应当除外，后面个动作正好使带头位置移到β_i中我们正在考虑的符号的位置上。注意，如果β_i中这个符号是一个表示状态的符号，那么它也表示M的一个任意的合法动作，故也许有一上的合法符号。还要注意，如果前一个ID有接受状态，那么现在的ID和前一个ID相同。

这样，我们就能比较容易地规定一个谓词 f(W,XY,Z),它为真当且仅当给定 W，X，Y 为某个 ID 的位置 j-1，j 和 j+1 上的符号时（若 j=1，则 W 为＃，若 j=p(n)，则 Y 为＃），符号 Z 能出现在后一个 ID 的位置 j 上。规定 f(W.#,X，#)为真也是比较方便的，这样我们就可以象处理 ID 中的符号一样来处理 ID 之间的记号了。现在我们可以把第四个公式表示成：

$$\bigwedge_{p(n) \prec j \prec (p(n)+1)^2} \left(\bigvee_{使f(W,X,Y,Z)成立的一切W,X,Y,Z} (C_{j-p(n)-1,W} \wedge C_{2j-p(n)-1,X} \wedge C_{j-p(n),Y} C_{jZ} \right)$$

给定M关于x的一个可接受计算，容易找出各个C_{iX}的真假值，以使E_x为真。这只要使 C_{iX}为真，当且仅当该计算中第i个符号是X。反之，给定一个使E_x为的赋值，上面四个公式就能

保证有一个M关于x的可接受计算。注意，即使M是非确定的，动作选择已经包含在每个ID中这一事实，就能保证从一个ID到下一个ID时，下一状态，被打印的符号和带头移动的方向全都会和M的某个动作选择完全一致。

进一步，组成E_x的诸公式的长度为$0(p^2(n))$，并且这些公式非常简单，以致当在其输入上给定x时，一个log空间TM就能产生它们。这个TM只需要有足够的存贮空间来进行直到$(p(n)+1)^2$的计数。因为一个n的多项式的对数是某个常数乘上log n，故只要用 0（log n）的存贮空间就行了。因此，我们证明了NP中每个语言都log空间可归约到L_{sat}这就证明了L_{sat}是NP完全的。

我们刚才证明了布尔表达式的可满足性问题是NP完全的。这意味着，一个接受L_{sat}的多项式时间算法可以用来接受□𝓕中任何语言。设L是被某个p(n)时间有界非确定图灵机M接受的语言，设A是将x转换成Ex的log空间（因而多项式时间）转换器。这里Ex是可满足的，当且仅当M接受x。那么，如图 8.1 所示，将A与接受L_{sat}的算法结合起来，就构成一个接受L的确定的多项式时间算法。因此，刚才这个问题，即布尔表达式可满足性问题有一个多项式时间算法，可推出：𝓕 ＝N𝓕。

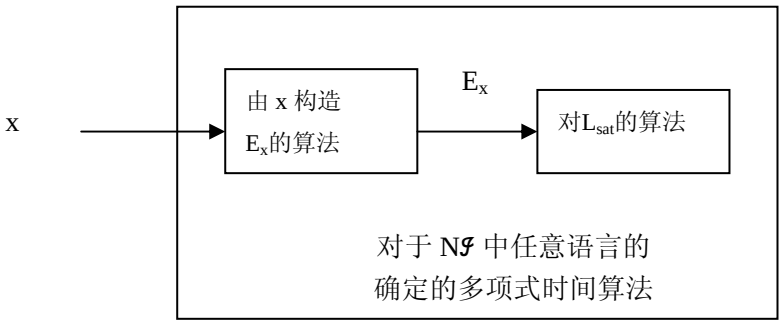


图 9.1 给定对L_{sat}的算法时，对N𝓕中任意集合L的算法

9.3.3 NP－完全的受限可满足性问题

我们记得，一个布尔公式是合取范式（CNF），如果它是一些子句的逻辑与，而这些子句又是一些文字的逻辑或。我们说布尔公式是 kCNF，如果每个子句正好有 k 个文字。例如，

我们现在考虑两个语言：L_{sat}（即可满足的CNF布尔公式的集合）和L_{sat}（即可满足的 3CNF布尔公式的集合）。我们给出从L_{sat}到L_{csat}和从L_{csat}到L_{3sat}的log空间归约，因而根据引理 3，证明了后两个问题是NP完全的。在每种情况下。我们都将一个表达映射到另一个表达式，它可能

不与原来的等价，然而它是可满足的，当且仅当原来的表达式是可满足的。

定理 9.3.2　L_{sat}，即CNF表达式的可满足性问题是NP完全的。

证　显然，在L_{sat}在N　中，因为L_{sat}在其中，我们将L_{sat}如下归约到L_{sat}。设E是一个长度为n的任意布尔表达式。自然，出现在E中的变元个数不会超过n，运算符∧和∨的数目也不会超过n，使用恒等式

$$\neg(E_1 \wedge E_2)=\neg(E_1)\vee\neg(E_2)$$

$$\neg(E_1 \vee E_2)=\neg(E_1) \wedge\neg(E_2)$$

$$\neg\neg E_1=E_1$$

我们能将 E 转换成一个等价的表达式 E’，在 E’中，运算仅应用于变元，决不会用于更复杂的表达式。考虑 0 和 1 对 E1 和 E2 的四种赋值，可以验证等式组式的正确性。顺利提一句，大家知道，前两个等式是狄摩根法则。

这个置换可以看成两个 log 空间转换的合成，作为第一个置换的结果，每个紧靠在变元前面的“非”符号都用该变元顶上加一道横杠代替，对于每个闭括号，若与之匹配的开括号前面是一个“非”符号，那么该闭括号将用）代替，符号表明一个“非”符号辖域的结束。用一个计数器来确定相匹配的括号的位置，第一个置换就能容易地在 log 空间内完成。

第二个转换是由一个有穷自动机来完成的。该机自左至右扫描输入，记住有效“非”符号个数的奇偶性（模 2 和），所谓有效“非”符号就是其后紧跟的开括号已被看过，但其闭括号尚未看到的非符号。当这些非符号的奇偶性是奇时，x 用 $\bar{x}$ 代替，$\bar{x}$ 用 x，∨用∧而∧用∨代替，符号¬和¬被删掉。这个置换是正确的，用关于一个表达式长度的简单归纳法，并使用等式组式就可证明这一点。我们现在有了一个表达式 E’，其中所有的非符号都直接应用到变元上。

其次，我们来构造E’’，一个CNF表达式，它是可满足的，当且仅当E是可满足的，设V_1和V_2是变元集合，且$V_1 \subseteq V_2$，我们说对V_2的一个赋值是对V_1 的一个赋值的扩展，如果两个赋值在V_1的变元上是相同的。我们将用关于r（表达式中和的数目，这里是一个其所有非符号都用到变元上的表达式）的归纳法，证明若|E’|=n　　，那么存在一个最多有n个子句的表：F_1，F_2，...，F_k，它们是一个包含E’的变元且最多包含n个其它变元的赋值而有 1 值，当且仅当存在着这个赋值的一个扩展，使$F_1 \wedge F_2 \wedge ... \wedge F_k$满足。

基始 r=0。那么 E’是一个文字，我们可以选取这个文字本身作为一个子句，以满足所述条件。

归纳 如果E'= $E_1 \wedge E_2$,设F_1，F_2，...，F_k和G_1，G_2，...，G_l是对应于E_1和E_2的子句，根据归纳假设，它们是存在的。不失一般性可以假定，没有不出现在E'中的变元会既出现在这些F中又出现在这些G中，那么F_1，F_2，...，F_k，G_1，G_2，...，G_l将满足对于E'的条件。

如果E'= $E_1 \vee E_2$，设F_1，F_2，...，F_k和G_1，G_2，...，G_l同上，并设y是一个新变元，那么$y \vee F_1$，$y \vee F_2$，... ，$y \vee F_k$，$\overline{y} \vee G_1$，$\vee \vee G_2$，...，$\overline{y} \vee G_1$，将满足所要求的条件。在证明中假设有一个赋值满足E'，那么它一定满足E_1或者E_2。如果该赋值满足E_1，那么这个赋值的某个扩展将满足F_1，F_2，...，F_k。于是，使得y=0 的这个赋值的任何进一步的扩展，都将满足E'的全部子句，如果该赋值满足E_2，一个类似的论证也成立。反之，假定对应于E'的全部子句都能被某个赋值满足，若在这个赋值中y=1，那么G_1，G_2，...，G_l必全能满足，故E_2被满足。若y=0，可以应用一个类似的证明。所希望的表达式E''就是由对应于E'的全部子句通过∧符号连结而成的。

为了看出上面的转换能在log空间内完成，考虑E的语法分析树。设y_i是由第i个∨符号引出的变元。最后的表达式是各子句的逻辑与，这里每个子句都包含原来表达式的一个文字y_i。若文字是在第i个∨的右子树中，那么该子句包含$\overline{y}_i$ 。输入被从左到右地扫描，每遇到一个文字，就输出一个子句。为了确定是那些y_i和$\overline{y}_i$包含在这个子句中，我们使用一个长度为logn的记数器来记住我们在输入上的位置。然后，我们来扫描整个输入，对于每个∨符号，比如说从左数第i 个，我们来确定它的左、右操作数，这时使用另一个长度为log n的计数器来统计括号。如果现在的文字是在左操作数中，则产生y_i；若它是在右操作数中，则产生$\overline{y}_i$，如果不在任何一个操作数中，那么既不产生y_i，也不产生$\overline{y}_i$。

于是，我们已把每个布尔表达式E归约到一个CNF表达式E''，E''在L_{csat}中，当且仅当E中L_{sat}中。因为归约是在log空间内完成的，由L_{sat}的NP完全性就可推出L_{csat}的NP完全性。

□

例 9.3.1 设$\neg(\neg(x_1 \vee x_2)) \wedge (\neg x_1 \vee x_2)$

应用德摩根法则得到：

$E' = (x_1 \vee x_2) \vee (x_1 \wedge \overline{x}_2)$

到CNF的转换引出变元y_1和y_2，并给出

$E'' = (x_1 \vee y_1 \vee y_2) \wedge (x_2 \vee \overline{y}_1 \vee y_2) \wedge (x_1 \vee \overline{y}_2) \wedge (\overline{x_2} \vee \overline{y}_2)$

定理 9.3.3 L_{3sat}，3-CNF表达式的可满足性问题是NP完全的。

证明　显然，L_{3sat}在N$\mathcal{P}$中，因为L_{sat}在N$\mathcal{P}$中。设$E=F_1 \wedge F_2 \wedge \ldots \wedge F_k$是一个CNF表达式。假设某个子句$F_i$有多于三个的文字，比如说$F_i=\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_l$，$l>3$

引入新变元y_1，y_2，...，y_{l-3}，用下式替换F_i：

$$(\alpha_1 \vee \alpha_2 \vee y_1) \wedge (\alpha_3 \vee \overline{y}_1 \vee y_2) \wedge (\alpha_4 \vee \overline{y}_2 \vee y_3) \wedge \ldots \wedge (\alpha_{l-2} \vee \overline{y}_{l-4} \vee y_{l-3}) \wedge (\alpha_{l-1} \vee \overline{y}_{l-3} \vee \alpha_l)$$

那么F_i被一个赋值满足，当且仅当这个赋值的一个扩展满足上式。对于一个满足F_i的赋值，必然有某个j，使$\alpha_j=1$。于是假定该赋值给文字α_1，α_2，...，α_{j-1}赋 0 值， 给α_j赋 1 值，那么，对于$m \leq j-2$，令 $y_m=1$；对于$m \geq j-1$，令$y_m=0$，这就是该赋值的一个满足式（上式）的扩展。反过来，我们必须证明，对任何满足上式的赋值，必然有某个j，使$\alpha_j=1$，因而该赋值满足F_i。假设不然，该赋值使所有的α_m为 0，那么因为第一个子句为 1，推出$y_1=1$。又因为第二个子句为 1，故y_2必然为 1，通过归纳，这与上面的可满足式的假设相矛盾。因此任何满足上式的赋值也满足F_i。

当F_i包含一个或两个文字时，还必须另作考虑。在后一种情况下，用$(\alpha_1 \vee \alpha_2 \vee y) \wedge (\alpha_1 \vee \alpha_2 \vee \overline{y})$替换$\alpha_1 \vee \alpha_2$，这里y 是一个新变元。在前一种情况下，引入两个新变元就够了。于是E可以转换成一个 3CNF表达式。这个转换容易在log空间内完成。因此，我们有了一个从L_{csat}到L_{3sat}的log空间归约，从而推出L_{3sat}是NP完全的。

9.3.4 其他 NP 完全问题

除了上面描述的 NP 可满足问题以外，顶点覆盖问题、Hamilton 回路问题、整数线性规划问题、图着色问题以及旅行商问题等都是 NP 完全问题，我们这里就不一一介绍了，有兴趣的读者可参阅相关文献。