第7章　NP 完全问题

前面各章针对许多不同类型的问题，介绍了各种算法设计方法和许多具体算法，并对这些算法的时空复杂性进行了分析讨论。我们从中可以发现，有些问题人们已经找到多项式时间复杂性的算法，例如分类（又称排序）问题等等。但也有一些问题，人们已经设计出实现它的时间复杂性为指数阶的算法，并且已证明该问题不存在多项式时间复杂性的算法（例如 Hanoi 问题）等等。

不可能设计出多项式时间复杂性算法的一个更加简单的例子是，输出图 G 中代价不超过某个数 B 的所有周游路线。对于这个问题，我们显然可以找到一个实例，它有关于图的阶的指数多个长度不超过 B 的周游路线，而任何关于图的阶的多项式时间算法不可能将这些路线全部列举出来。

上面列出的问题虽然难解，但是它的难解性很容易判断，因为解的总长度不可能以问题规模的多项式函数为界。但是还有许多问题，问题及其解的表示都以问题规模的多项式函数为界，虽然人们通过多方面的努力与探讨，却至今尚未找到求解它们的多项式时间算法。例如，将前面在动态规划、贪心法、回溯法、分枝限界法等章节中讨论过的那些困难问题改成只求一个解，或者是判定有没有解，得到的问题都属于这一类。正因为有大量的问题至今没有找到多项式时间算法，所以人们有理由怀疑，很可能不存在求解这些问题的多项式时间算法。这类问题被认为是难解的问题。然而，求解这些问题的难度有多大，哪些是真正难解的呢？这一章就要初步探讨与回答这些提问。

§7.1　确定型图灵机

从三十年代开始，就不断地有人设计各种计算模型，其中有代表性的模型约有三十多种。图灵机（Turing Machine）是英国数理逻辑学家图灵（A. M. Turing）于一九三六年提出的一种计算模型。它具有结构简单和计算能力强等许多优点，在理论研究中占有重要地位。

一台 K 带图灵机是由一个有限状态控制器和 k 条带（$k \geq 1$）组成。这些带的右端是无限的。每一条带都从左到右划分成方格（或称单元），每个方格都可以存放一个带符号。每条带都有一个与有限状态控制器相连的带头，它可以对这条带进行读写操作，既可以读出带头下当前扫描着的那个方格中的符号，也可以将某个带符号写入带头当前扫描着的方格中。图 7.1 为一台图灵机

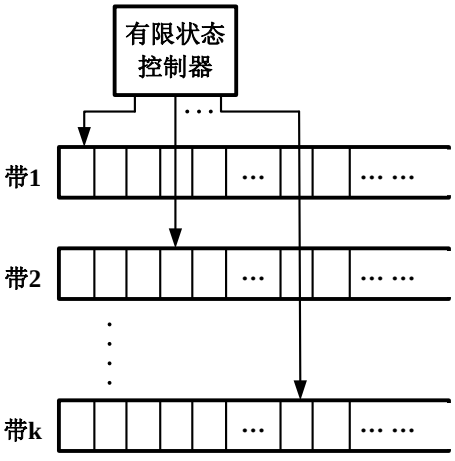

图 7.1　K 带图灵机

的示意图。

根据有限状态控制器的当前状态，以及各个带头扫描的当前符号，图灵机的一个计算步可以完成以下三个操作之一，或者全部：

- 转换状态：根据定义的映射关系，把当前状态改变为新的状态。
- 印刷符号：根据定义的映射关系，或者清除各带头下的当前方格中原有的带符号并写上新的带符号，或者保留原有的带符号。
- 移动带头：每一条带的带头，根据定义的映射关系，或者向左移动一格（L），或者向右移动一格（R），或者停在当前方格不动（S）。

可以形式地将一台 K 带图灵机描述为一个七元组：

$$TM=(Q, T, I, b, \delta, q_0, q_f)$$

其中，

Q 是一个有穷状态的集合；

$q_0 \in Q$ 称为初始状态；

$q_f \in Q$ 称为终止状态或接受状态。

T 是一个有穷符号集合；

I 是输入字符集，且 $I \subset T$；

b 是 T 中的唯一空符，有 $b \in T-I$；

δ称作图灵机的**下移函数**或有限状态控制函数，是从 $Q\times T^k$ 的某一个子集到 $Q\times(T\times\{L, R, S\})^k$ 的映射函数。即对于由一个当前状态和 k 条带上扫描到的当前符号所构成的一个 $k+1$ 元组，它唯一地给出一个新的状态和 k 个序偶，而每一个序偶由一个新的带符号和带头移动方向组成。

假定某台图灵机的下移函数表中有一个定义式为

$\delta(q, a_1, a_2, \cdots, a_k)=(q', (a_1', d_1), (a_2', d_2), \cdots, (a_k', d_k))$，当图灵机处于状态 q 且对一切 $1 \le i \le k$，第 i 条带的带头扫描着的当前方格中的符号正好是 a_i 时，图灵机就按这个下移函数定义式所规定的内容进行如下工作：

- 把第 i 条带头下当前方格中的符号 a_i 清除并写上新的带符号 a_i'，$1 \le i \le k$。
- 按 d_i 指出的方向移动各带的带头。这里，d_i=L 表示带头往左移一格，d_i=R 表示带头往右移一格，d_i=S 表示带头不动。
- 将图灵机的当前状态 q 改为 q'。

这样，图灵机就完成了一步计算。一台图灵机的全部工作是从初始状态 q_0 开始，按下移函数一步一步进行计算。如果通过若干步计算后，机器状态变成终止状态，图灵机就自动停止工作。如果没有出现终止状态，它就不会停机。

一台图灵机可以用来识别一个语言。这样一台图灵机的带符号集 T 应当包括这个语言的字母表中的全体符号和一个空白符 b，也许还有其它符号。开始，第一条带上放有一个输入符号串（从最左的方格起每格放一个输入字符），这条带的其余方格都是空白。其它各带上的方格也全是空白。所有的带头都处在各带左端的第一个方格上。当且仅当图灵机从指定的初始状态 q_0 出发，经过一系列计算步后，最终进入终止状态（或接受状态）q_f 时，称图

灵机接受这个输入符号串。被这台图灵机所接受的所有输入符号串的集合，称作这台图灵机识别的一个语言。

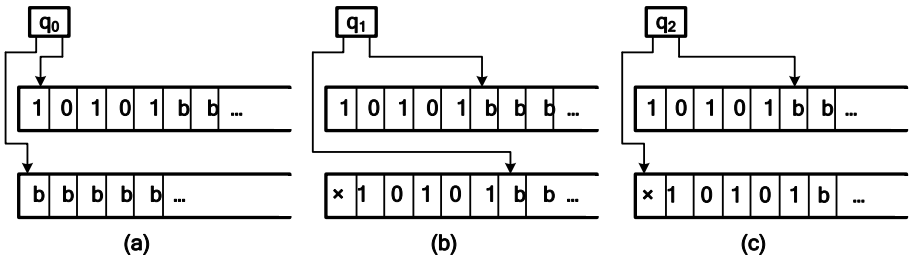

图 7.2 一台两带图灵机处理符号串 10101 的前几步

表 7.1 识别字母表{0，1}上正反读相同的串的双带图灵机下移函数

当前状态	带头扫描着的符号		新的带符号和带头移动方向		新的状态	说　　明
	带 1	带 2	带 1	带 2		
q_0	0	*b*	0，S	×，R	q_1	若输入非空，则在带 2 上印×，带头 2 右移一格，进入状态 q_1；否则进入状态 q_5。
	1	*b*	1，S	×，R	q_1	
	b	*b*	*b*，S	*b*，S	q_5	
q_1	0	*b*	0，R	0，R	q_1	将带 1 的符号依次往带 2 上抄写，状态 q_1 不变。直到带 1 遇到 *b*，才进入状态 q_2。
	1	*b*	1，R	1，R	q_1	
	b	*b*	*b*，S	*b*，L	q_2	
q_2	*b*	0	*b*，S	0，L	q_2	带头 1 不动，将带头 2 逐次向左移动，直到遇到 ×才脱离状态 q_2，进入状态 q_3。
	b	1	*b*，S	1，L	q_2	
	b	×	*b*，L	×，R	q_3	
q_3	0	0	0，S	0，R	q_4	当两个带头下读到的符号相同时，带头 2 向右移一格，进入状态 q_4。
	1	1	1，S	1，R	q_4	
q_4	0	0	0，L	0，S	q_3	如果带头 2 读到符号 *b*，则进入状态 q_5；否则，带头 1 左移一格，返回状态 q_3，继续往下比较。交替使用 q_3、q_4 是防止带头 1 从左端掉下。
	0	1	0，L	1，S	q_3	
	1	0	1，L	0，S	q_3	
	1	1	1，L	1，S	q_3	
	0	*b*	0，S	*b*，S	q_5	
	1	*b*	1，S	*b*, S	q_5	
q_5						接受。

例7.1图 7.2 是一台能识别字母表 Σ={0，1}上所有正反读相同的字的二带图灵机识别输入符号串 10101 的示意图。它的工作过程如下：

1. 当带 1 上有输入字符串 10101，带 2 上全是空白符，两个带头都扫描着各带左边的第一方格时，机器状态为初始状态 q_0（图 7.2 (*a*)）。

2. 在带 2 的第一方格中写入特殊符号 ×，且把带 1 上的这个符号串写到带 2 上，控制状态改为 q_1（图 7.2 (*b*)）。

3. 把带 2 的带头移到左边第一方格上，控制状态改为 q_2（图 7.2 (*c*)）。

4. 当带头 2 扫描着的当前符号不是空白符 *b* 时，带头 2 向右移动一格，带头 1 向左移动一格，转入状态 q_3；否则，进入接受状态 q_5，停机。

5. 判断两个带头下扫描着的符号是否相同。如果相同，重复步骤 4；否则，进入不接受状态且停机。

图灵机的第 4 步和第 5 步的工作过程，也就是控制状态 q_3 开始以后的转换过程没有画出来。但表 7.1 给出了这台图灵机的全部下移函数。从表 7.1 可以看出，这台图灵机对于字母表{0，1}上的任何长度的正反读相同的字符串，它都会进入接受状态 q_5，而对任何正反读不同的字符串，图灵机不可能进入接受状态，因而一概不接受。

用类似表 7.1 的形式给出图灵机的全部下移函数的好处是直观、清晰，但是当图灵机很复杂时，表格非常庞大。

就像写程序一样，我们也可以直接给出 δ 函数。δ 函数的任何部分集构成一段图灵机程序。为了描述的简单，我们甚至可以用自然语言给出一段图灵机程序要完成的事情。例如，后面 Cook 定理的证明就是这样的。写图灵机程序同写文章一样，需要考虑整体布局，需要段落清晰。结构程序设计的思想是值得借鉴的。

例如，对于表 7.1 定义的图灵机，我们给出 δ 函数如下：

若输入不为空，则在带 2 上印刷×，带头 2 右移一格，进入状态 q_1，否则，进入 q_5：

$$\delta(q_0, 0, b)=(q_1, (0, S), (\times, R))$$

$$\delta(q_0, 1, b)=(q_1, (1, S), (\times, R))$$

$$\delta(q_0, b, b)=(q_5, (b, S), (b, S))$$

将带 1 的符号依次往带 2 上复制，状态 q_1 不变。直到带 1 遇到 *b*，进入 q_2 状态：

$$\delta(q_1, 0, b)=(q_1, (0, R), (0, R))$$

$$\delta(q_1, 1, b)=(q_1, (1, R), (1, R))$$

$$\delta(q_1, b, b)=(q_2, (b, S), (b, L))$$

带头 1 不动，将带头 2 逐次向左移动，并保持状态不变。直到遇到 ×，进入状态 q_3：

$$\delta(q_2, b, 1)=(q_2, (b, S), (1, L))$$

$$\delta(q_2, b, 0)=(q_2, (b, S), (0, L))$$

$$\delta(q_2, b, \times)=(q_3, (b, L), (\times, R))$$

当两个带头指示的符号相同，带头 2 右移一格，进入状态 q_4（注意，此时带头 1 不能冒然左移，因为带头可能已经达到边界，需要进一步判断带头 2 指示的符号，才能决定带头 1 的动作）：

$$\delta(q_3, 1, 1)=(q_4, (1, S), (1, R))$$

$$\delta(q_3, 0, 0)=(q_4, (0, S), (0, R))$$

如果按照带头 2 的指示，读到符号 b，进入 q_5。否则，带头 1 左移一格，返回状态 q_3：

$$\delta(q_4, 1, 1)=(q_3, (1, L), (1, S))$$

$$\delta(q_4, 1, 0)=(q_3, (1, L), (0, S))$$

$$\delta(q_4, 0, 1)=(q_3, (0, L), (1, S))$$

$$\delta(q_4, 0, 0)=(q_3, (0, L), (0, S))$$

$$\delta(q_4, 0, b)=(q_5, (0, S), (b, S))$$

$$\delta(q_4, 1, b)=(q_5, (1, S), (b, S))$$

图灵机的工作过程可以用格局序列描述。一台图灵机 M 的某一格局是一个 k 元组$(a_1, a_2, \cdots, a_k)$，其中 a_i 是一个形如 xqy 的字符串，这里 q 是 M 的当前状态，xy 是在当前状态 q 下第 i 条带上的符号串（不计右端空白符），第 i 个带头指向 y 的第一个符号，当 y 为空串时，第 i 个带头指向空白符。如果图灵机计算一步后，它的格局由 D_1 变成了 D_2，就记作$D_1 \vdash_M D_2$（$\vdash$——读做“进入”），对于 $n \geq 2$，如果有

$$D_1 \vdash_M D_2 \vdash_M D_3 \vdash_M \cdots \vdash_M D_n$$

就记做

$$D_1 \vdash_M^+ D_n$$

对 $D=D'$或$D \vdash_M^+ D'$，都可以记做$D \vdash_M^* D'$。

如果对于某一 k 元组$(a_1, a_2, \cdots, a_k)$和某一输入串 $c_1c_2\cdots c_n$，$c_i \in$ I，i=1，2，…，k，有

$$(q_0c_1c_2\cdots c_n，\ q_0，\ \cdots，\ q_0) \mid \overset{*}{\underset{M}{-}} (a_1，\ a_2,，\ \cdots，\ a_k)$$

并且 q_f在$(a_1, a_2, \cdots, a_k)$中间，则称图灵机 M 接受输入串 $c_1c_2\cdots c_n$。

例7.2 对于表 7.1 定义的图灵机，假定输入带 1 的符号串为 0110，图灵机初始状态为 q_0。它识别这个符号串的格局序列记录在图 7.3 中。因为 q_5是接受状态，所以该图灵机接受符号串 0110。

$(q_00110，\ q_0)$ ⊢$(q_10110，\ \times q_1)$
⊢$(0q_1110，\ \times 0q_1)$
⊢$(01q_110，\ \times 01q_1)$
⊢$(011q_10，\ \times 011q_1)$
⊢$(0110q_1，\ \times 0110q_1)$
⊢$(0110q_2，\ \times 011q_20)$
⊢$(0110q_2，\ \times 01q_210)$
⊢$(0110q_2，\ \times 0q_2110)$
⊢$(0110q_2，\ \times q_20110)$
⊢$(0110q_2，\ q_2\times 0110)$
⊢$(011q_30，\ \times q_30110)$
⊢$(011q_40，\ \times 0q_4110)$
⊢$(01q_310，\ \times 0q_3110)$
⊢$(01q_410，\ \times 01q_410)$
⊢$(0q_3110，\ \times 01q_310)$
⊢$(0q_4110，\ \times 011q_40)$
⊢$(q_30110，\ \times 011q_30)$
⊢$(q_40110，\ \times 0110q_4)$
⊢$(q_50110，\ \times 0110q_5)$

图 7.3 图灵机识别字符串 0110 的格局序列

与 RAM 模型一样，除了将图灵机理解为语言接受器之外，还可以将其理解为函数计算装置。一个函数 f 的多个自变量可以当做一个符号串 X 编码到一条输入带上，并用一特殊符号来隔开这些不同的自变量。如果一台图灵机在读入这个输入串并经过有限步计算后，在一条指定的带上输出一个整数 y 并停机，则可以说图灵机计算出了 f(X)＝y。

例7.3 表 7.2 给出了计算函数 $f(x)=3x+1$ 的一台图灵机的下移函数。不妨设所有的带上最左一格中有一个特殊符号 ×，防止带头从左端失落。如果某输入带上最左一格没

有 ×，而且最左一格的符号为空白符，我们可以在该带上印刷 ×并让带头右移。如果某输入带上最左一格没有 ×，而且最左一格的符号不是空白符，要在该带的左端印刷 ×并且不破坏带上的信息有些麻烦。一种方案是，设需要在左端印刷 ×的带为 A，另外找一根输入带 B，先将 B 的带头移动到右端空白符处并印刷×，然后将 A 的内容搬移到 B，这个搬移直到 A 带上读到右端空白符为止。A 到 B 的搬移完成以后，再将 B 的从 ×往右的内容搬移到 A，搬移必须包括 ×。搬移过程中，清除 B 上新增加内容。这样一个过程完成以后，A 的左端就有了 ×。

图 7.4 给出了这台图灵机计算 $f(0)$，$f(2)$，$f(5)$时的各格局序列。自变量的初值在带 1 上输入，函数值产生在带 2 上（都以二进制形式表示）。结果的最高位有效数字前面最多可能有一个零（如果初始输入不是 0 且最高位为 1）。当然，也可以在进入接受状态前做点工作，把有效数字前面的零去掉。

图灵机的时间复杂性 $T(n)$，是它处理所有长度为 n 的输入所需的最大计算步数。当然，这是最坏情况下的时间复杂性。如果对于某个长为 n 的输入，图灵机不停机，则 $T(n)$对这个 n 无定义。图灵机的空间复杂性 $S(n)$是处理所有长度为 n 的输入时，在 k 条带上所使用过的方格的总和。如果某个带头无限地向右移动而不停机，$S(n)$也无定义。可以用 $O_{TM}(f(n))$来表示在图灵机模型下计算复杂性的量级。

表 7.2　计算函数 $f(x)=3x+1$ 的图灵机下移函数

当前状态	带上符号		新符号与带头移动		新状态	说　明
	带 1	带 2	带 1	带 2		
q_0	×	×	×，R	×，R	q_0	带 2 上先写一个 0，准备做进位用。
	0	b	0，S	0，R	q_1	
	1	b	1，S	0，R	q_1	
q_1	0	b	0，R	0，R	q_1	把带 1 上的数复写到带 2 上，并在带 2 末位加一个 0，得 $2x$ 的值。
	1	b	1，R	1，R	q_1	
	b	b	b，L	0，S	q_2	
q_2	0	0	0，L	1，L	q_3	带 1、带 2 的个位相加，再加 1，无进位时到 q_3，否则 q_4。
	1	0	1，L	0，L	q_4	
q_3	0	0	0，L	0，L	q_3	带 1、带 2 的数字做前次进位为 0 的加法，进位为 0 状态不变，进位为 1 进入状态 q_4。
	0	1	0，L	1，L	q_3	
	1	0	1，L	0，L	q_3	
	1	1	1，L	0，L	q_4	
	×	0	×，S	0，L	q_3	
	×	1	×，S	1，L	q_3	
	×	×	×，S	×，S	q_5	
q_4	0	0	0，L	1，L	q_3	带 1、带 2 的数字做前次进位为 1 的加法，本次进位为 1 时状态不变，本次进位为 0 时进入状态 q_3。
	0	1	0，L	0，L	q_4	
	1	0	1，L	0，L	q_4	

	1 × ×	1 0 1	1，L ×，S ×，S	1，L 1，L 0，L	q_4 q_3 q_4	
q_5						接受状态。

例7.4 表 7.1 定义的图灵机的时间复杂性 $T(n)=4n+3$；空间复杂性 $S(n)=2n+3$；它们都是 $O_{TM}(n)$。这里 n 是输入字符串的长度。表 7.2 定义的图灵机的时间复杂性 $T(n)=2n+6$；空间复杂性 $S(n)=2n+5$。这里，n 是指初始输入的二进制位数，而不是自变量的值大小。

$(q_0\times 0，\ q_0\times)$	$(q_0\times 10，\ q_0\times)$	$(q_0\times 101，\ q_0\times)$
$\vdash(\times q_0 0，\ \times q_0)$	$\vdash(\times q_0 10，\ \times q_0)$	$\vdash(\times q_0 101，\ \times q_0)$
$\vdash(\times q_1 0，\ \times 0 q_1)$	$\vdash(\times q_1 10，\ \times 0 q_1)$	$\vdash(\times q_1 101，\ \times 0 q_1)$
$\vdash(\times 0 q_1，\ \times 00 q_1)$	$\vdash(\times 1 q_1 0，\ \times 01 q_1)$	$\vdash(\times 1 q_1 01，\ \times 01 q_1)$
$\vdash(\times q_2 0，\ \times 00 q_2 0)$	$\vdash(\times 10 q_1，\ \times 010 q_1)$	$\vdash(\times 10 q_1 1，\ \times 010 q_1)$
$\vdash(q_3\times 0，\ \times 0 q_3 01)$	$\vdash(\times 1 q_2 0，\ \times 010 q_2 0)$	$\vdash(\times 101 q_1，\ \times 0101 q_1)$
$\vdash(q_3\times 0，\ \times q_3 001)$	$\vdash(\times q_3 10，\ \times 01 q_3 01)$	$\vdash(\times 10 q_2 1，\ \times 0101 q_2 0)$
$\vdash(q_3\times 0，\ \ q_3\times 001)$	$\vdash(q_3\times 10，\ \times 0 q_3 111)$	$\vdash(\times 1 q_4 01，\ \times 010 q_4 10)$
$\vdash(q_5\times 0，\ \times q_5 001)$	$\vdash(q_3\times 10，\ \times q_3 0111)$	$\vdash(\times q_4 101，\ \times 01 q_4 000)$
$x=0$	$\vdash(q_3\times 10，\ \ q_3\times 0111)$	$\vdash(q_4\times 101，\ \times 0 q_4 1000)$
$f(0)=1$	$\vdash(q_5\times 10，\ \times q_5 0111)$	$\vdash(q_4\times 101，\ \times q_4 00000)$
	$x=2$	$\vdash(q_3\times 101，\ q_3\times 10000)$
	$f(2)=7$	$\vdash(q_5\times 101，\ q_5\times 10000)$
		$x=5$
		$f(5)=16$

图 7.4 计算 $f(0)$、$f(2)$、$f(5)$的格局序列

§7.2 图灵机模型和 RAM 模型的关系

由于图灵机模型比 RAM 模型难以构造，所以人们通常愿意使用 RAM 这一模型来分析算法。然而同一个问题在不同模型下的求解复杂性是不同的。图灵机模型与 RAM 模型的关系是指同一个问题在两个模型下复杂性之间的关系。

定义7.1 对于函数 $f_1(n)$和 $f_2(n)$，如果存在两个多项式 $p_1(x)$和 $p_2(x)$，使得除去有限个 n 值外，有不等式

$$f_1(n)\le p_1(f_2(n)) \quad 和 \quad f_2(n)\le p_2(f_1(n))$$

成立，则说函数 $f_1(n)$和 $f_2(n)$是**多项式相关**的。

例如，$f_1(n)=2n^2$ 和 $f_2(n)=n^5$ 是多项式相关的。只要令 $p_1(x)=2x$，$p_2(x)=x^3$，则对于一

切 $n \geq 0$ 都有

$$2n^2 \leq 2n^5 \quad 和 \quad n^5 \leq (2n^2)^3。$$

函数 $f_1(n)=n\log_2 n$ 和 $f_2(n)=3n^3$ 也是多项式相关的。只要令 $p_1(x)=x$，$p_2(x)=3x^3+3$，对于一切 $n \geq 1$，有不等式

$$n\log_2 n \leq 3n^3 \quad 和 \quad 3n^3 \leq 3(n\log_2 n)^3+3。$$

但是，函数 n^2 和 2^n 就不是多项式相关的。因为不存在这样的多项式 $p(x)$，使得除去有限个 n，不等式 $2^n \leq p(n^2)$成立。

定理7.1 设 TM 是求解某一问题 P 的图灵机。对问题 P 的任何长度为 n 的输入，TM 处理它的时间复杂性是 $T(n)$，那么，存在一个求解问题 P 的 RAM 程序，它的时间复杂性不超过 $O(T^2(n))$。

证明： 显然只要证明存在一个求解问题 P 的 RAM 程序，对问题 P 的任何长度为 n 的输入，其时间复杂性为 $O(T^2(n))$就可以了。所以，我们通过构造一个 RAM 程序模拟求解问题 P 的 k 带图灵机 TM 的工作来证明本定理。

首先，我们将 TM 的第 j 条带上第 i 个单元与 RAM 的第 $k(i-1)+j+c$ 单元对应起来，这就建立了带方格与 RAM 内存单元的一一对应关系。这里 c 是一个常数，它给 RAM 内存留下了 c 个工作单元。这些工作单元中有 k 个用于存放 TM 的 k 个带头的当前位置。于是，RAM 程序可以借助于这 k 个单元，用间接寻址读出任何带头下的当前符号或改写任何带头下的符号。于是图灵机的带头移动在 RAM 程序中变成对记录带头的当前位置的那个单元的内容进行适当修改。

在均匀耗费下，对于图灵机 TM 的任何计算步（设时间耗费为 1），RAM 程序模拟它的时间耗费不超过 ck，所以一个 RAM 程序能在 $O(T(n))$的时间内完成对 TM 的模拟。

在对数耗费下，RAM 处理一个大小为 n 的整数需要的时间不超过 $O(\log_2 n)$。所以 RAM 模拟 TM 的时间耗费不超过 $c_1T(n)\log_2 n$，这里 c_1 是一个常数。因为 $n \leq T(n)$（对于图灵机，这是必然的），所以 RAM 模拟 TM 所需时间不超过 $O(T(n)\log_2 T(n))$。对一切自然数 n，函数 $T(n)\log_2 T(n)$囿于 $T^2(n)$，定理得证。 ■

这个定理的逆命题不成立。对于任意给定的充分大的整数 n，存在一个 n 条指令的 RAM 程序，不需要输入，能在均匀耗费标准下的 $O(n)$时间内产生 2^{2^n} 这样大的数。若用图灵机来模拟这个 RAM 程序，仅仅为了存取这个数，就需要至少 2^n 个方格和计算步。因此，在均匀耗费标准下，本定理的逆命题不成立。

在对数耗费标准下，关于 RAM 模型和 TM 模型的时间复杂性的关系，有以下定理。

定理7.2 设 L 是一个 RAM 程序所接受的一个语言。在对数耗费下，这个 RAM 程序的时间复杂性是 $T(n)$。存在一台接受同一个语言 L 的多带图灵机 TM，其时间复杂性为 $O(T^2(n))$。

证明： 构造 5 带图灵机 TM 模拟 RAM 程序的工作。

除了用做累加器的 0 号单元外，对于 RAM 的所有单元号以及该单元中的内容，我们用图灵机的第一条带依次存放它们，形如图 7.5 所示。这条带上的内容是由一系列(i_j，C_j)所组成的，其中每个 i_j 是二进制形式的 RAM 的单元号码。每个 C_j 是单元 i_j 中的二进制形式的内容。i_j 和 C_j 之间，用特殊符号“#”分开。

RAM 累加器中的内容，以二进制形式存放于 TM 的第二条带最左端的一些方格内。

TM 的第三条带作为暂存工作带。

TM 的第四条带作为 RAM 的输入带。

TM 的第五条带当作 RAM 的输出带。

用 TM 的某个有限状态控制集模拟 RAM 程序的每一条指令。

#	#	i_1	#	C_1	#	#	i_2	#	C_2	#	#	…	i_k	#	C_k	#	#	b	…

图 7.5 将 RAM 寄存器号及其单元中的内容表示在一条带上

下面以 ADD *20 和 STORE 30 为例，讨论模拟过程。

模拟 ADD *20 的过程如下：

(1) 在带 1 上寻找 RAM 寄存器号为 20 的存储单元，即在带 1 上寻找符号串##10100#。如果找到了这个单元号，就把它随后的那个整数，即 20 单元中的内容 C_{20} 复制到带 3 上。如果找不到这个单元，就停机。

(2) 在带 1 上寻找其单元号码等于带 3 上的数值的那个 RAM 的单元号数。如果找到，则把这个单元的内容复制到带 3 上。如果找不到，就停机。

(3) 将带 2 和带 3 上的数相加，并将结果写在带 2 上。

模拟 STORE 30 的过程如下：

(1) 在带 1 上寻找 RAM 的 30 号单元，即寻找符号串##11110#。

(2) 如果找到了符号串##11110#，就把 C_{30} 之外到尾端空白之前的全部内容复制到带 3 上去。否则跳转步骤（4）。

(3) 将带 2 上的内容（即累加器中的数）复制到带 1 的符号串##11110#的右边，紧接着将带 3 上的串复制到带 1 的右边（即 C_{30} 的后面），完成模拟。

(4) 从带 1 的尾部第一个空白符处开始写入符号串 11110#，接着将带 2 的内容复制到带 1 的右边，再写入两个符号##，完成模拟。

对于 RAM 的输入指令的模拟可以这样进行。设 RAM 程序的各个输入量是按输入的顺序事先置好在 TM 的第四条带上，并且彼此之间有一特殊符号分隔开。当 TM 模拟 RAM 指令 READ i 时，先在带 1 上寻找单元号 i（即寻找符号串##i#），然后把带 4 上的当前输入串复制到这个单元中去。写的方法可以依照模拟指令“STORE i”的某些步骤进行。不过前者是写带 2 上的内容到带 1 上，后者却是写带 4 上的一个输入串到带 1 上而已。

对于输出指令，TM 依次将 RAM 的输出写在第五条带上，并且不同输出量用一个特殊符号分隔开即可。

显然，这样一台图灵机将忠实地模拟一个 RAM 程序的工作。下面只要证明，如果一个 RAM 程序的对数耗费为 m 时，这台图灵机的计算步骤至多为 $O(m^2)$。

在对数耗费下，一个 RAM 程序的时间复杂性为 m 的实际意义是：其步数与数据的最大

长度均不超过 m。在 TM 模型下，把 C_j 存入 i_j 的耗费是 $l(C_j)+l(i_j)$（$l(C_j)$表示 C_j 的长度，$l(i_j)$表示 i_j 的长度），这个耗费与串$\#\#i_j\#C_j\#$的长度成正比，两者仅差一个常数。对给定的输入 n，RAM 涉及的各内存单元所使用的长度总和，按假设不超过 $O(m)$。故带 1 上非空部分的长度为 $O(m)$。除开乘、除法指令外，TM 模拟其它任何一条 RAM 指令的时间耗费，显然与带 1 上非空符号的长度有相同的量级，因为最大的耗费是搜索带上的全部非空符号。由 RAM 程序的指令被执行的次数（可能某些指令被多次执行）不超过 m 可得，一台图灵机模拟一个 RAM 程序的时间耗费不超过 $O(m^2)$。

一个 RAM 程序中如果有乘法和除法指令，可以编出用加法和减法来实现乘、除运算的 TM 子程序，且不难证明这两个子程序的对数耗费，不大于它们所模拟的指令的对数耗费的平方。 ■

由定理 7.1 和定理 7.2，有以下定理。

定理7.3 在对数耗费下，对于同一个算法，采用 RAM 模型和图灵机模型的时间耗费是多项式相关的。

证明： 根据定理 7.1 和定理 7.2 以及对乘、除法指令的分析，可得本定理的结论。■

建立 RAM 和 TM 两个计算模型之间多项式相关的几个定理是十分必要的。因为图灵机模型比较原始，故要构造一台图灵机来描述一个算法是十分困难的工作。在大多数情况下，我们采用 RAM 模型描述算法。在 RAM 模型下，如果一个算法的复杂性囿于多项式（按对数耗费），这个算法在图灵机模型下也必囿于多项式。除非有特殊需要，我们总是避免直接使用图灵机模型。

§7.3 非确定型图灵机

为讨论 NP 问题，再引进一种计算模型，即非确定型图灵机。

定义7.2 一台非确定型 k 带图灵机（简称 NDTM）M 由一个七元组

$$M = (Q，T，I，\delta，b，q_0，q_f)$$

构成。其中，Q，T，I，δ，b，q_0，q_f的定义与 §7.1 中的定义相同，δ 是从 $Q\times T^k$ 到 $Q\times(T\times\{L，R，S\})^k$的一个映射，而且其中至少有一个映射是一对多的映射。

非确定图灵机与确定图灵机的根本不同在于下移函数 δ。注意，一个一对多的映射意味着，从当前状态 q 和当前扫描的 k 个带符号 x_1，x_2，…，x_k，可以选择新状态、新的带符号和带头移动的多种组合，即可定义

$$\delta(q,x_1,x_2,\cdots x_k)=\begin{cases}(q_1,(a_{11},d_{11}),(a_{12},d_{12}),\cdots(a_{1k},d_{1k}))\\(q_2,(a_{21},d_{21}),(a_{22},d_{22}),\cdots(a_{2k},d_{2k}))\\\cdots\cdots\quad\cdots\cdots\\(q_r,(a_{r1},d_{r1}),(a_{r2},d_{r2}),\cdots(a_{rk},d_{rk}))\end{cases}$$

式中，$r\geq 2$。图灵机执行时，每次可以选择（猜测）这 r 种新状态、新带符号与带头移动的某一固定的组合。

例如，如果$(q', (a_1', d_1), (a_2', d_2), \cdots, (a_k', d_k)) \in \delta(q, a_1, a_2, \cdots, a_k)$，非确定型图灵机 M 正处在状态 q，且第 i 个带（$1 \le i \le k$）正扫描着第 i 条带上符号 a_i，则机器的下一动作可以进入状态 q'，并把 a_i 变为 a_i'，而各带头的动作由 d_i'指定。

同确定型图灵机一样，非确定型图灵机也有格局的概念。与确定型图灵机不同，对于非确定型图灵机 M，从当前格局 D 可导致多于一个的下一格局（但仅有穷多个），若 D'是其中之一，则记为$D \vdash_M D'$（或 D ⊢ D'，若不引起混淆），称为 D 进入 D'。

若对某个 $k > 1$，有$D_1 \vdash_M D_2 \vdash_M \cdots \vdash_M D_k$，或者$D_1 = D_k$，则可记作$D_1 \vdash_M^* D_k$。

非确定型图灵机 M 可以用作一种语言 L 的识别器。对于语言 L，我们可以构造一台非确定型图灵机 M，让机器的带符包括该语言的字母表（输入字母表）以及空白符 b 和其它一些特定符号（辅助符号）。机器处于开始状态时，将输入 w 打印在第一条带的最左边部分上，此外全为空白，而其他的带此时全为空白；把各条带的带头放在该带最左边的方格上。称输入 w 被这台机器接受，仅当：

$$<q_0 w, q_0, q_0, \cdots, q_0> \vdash_M^* < a_1, a_2, \cdots, a_k >$$

其中 a_1（因此一切 $a_2, \cdots, a_k$）中有停机状态 q_f。

表 7.3 等分划问题非确定图灵机 M 的下移函数

当前状态	当前带符			新带符及带头移动			新状态	说　明
	带 1	带 2	带 3	带 1	带 2	带 3		
q_0	1	b	b	1，S	\$，R	\$，R	q_1	第 1 带不变，2、3 带之左端打印\$，进入状态 q_1。
q_1	1	b	b	1，R 1，R	b，S b，S	b，S b，S	q_2 q_3	开始选择：是把下一段 0 记在带 2（q_2）上呢？还是记在带（q_3）上？
q_2	0 1 b	b b b	b b b	0，R 1，S b，S	0，R b，S b，L	b，S b，S b，L	q_2 q_1 q_4	把所扫描的一段 0 复制到 2 上。当带 1 已扫描到 1 时便回到 q_1；若在带 1 上扫描到 b，则去进行 2、3 带上 0 的个数的比较（q_4）。
q_3	0 1 b	b b b	b b b	0，R 1，S b，S	b，S b，S b，L	0，R b，S b，L	q_3 q_1 q_4	与 q_2 中在带 2 上的动作相似，但这次是在带 3 上进行。

q_4	b b	0 \$	0 \$	b，S b，S	0，L \$，S	0，L \$，S	q_4 q_5	比较带 2 与带 3 上 0 的个数。
q_5								接受。

例7.5 试设计一台 NDTM，它接受形如

$$1\quad 0^{i_1}\quad 1\quad 0^{i_2}\quad 1\cdots 1\quad 0^{i_k}$$

的字，其中 i_1，i_2，…，i_k 为非负整数满足下述要求：有 $I\subseteq\{1，2，\cdots，k\}$使得

$$\sum_{j\in I} i_j = \sum_{j\notin I} i_j$$

换言之，字 w 被接受当且仅当用字 w 所表现的数列 i_1，i_2，…，i_k，可以被分割为两个子序列，两子序列的各数之和相等。这个问题就是所谓等分划问题。

下面设计一台三带 NDTM 来接受所描述的语言。这台 NDTM 的工作情况是：对输入带 1 从左到右进行扫描，每次从带 1 上读入连续的 i_j 个 0，并且不确定地在第 2 或第 3 带上增补进 i_j 个 0；当扫描到输入末端时，机器便核对第 2 条带与第 3 条带上 0 的个数，若相等则接受（注意，可能有许多情况导致不相等，但我们关心的是：有一种选择导致相等）。设 NDTM = $<\{q_0, q_1, \cdots, q_5\}$，$\{0, 1, b, \$\}$，$\{0, 1\}, \delta, b, q_0, q_5>$，函数 δ 如表 7.3 所示。

下面给该机器输入 1010010，我们从许多可能选择的计算中，列出两个可能的计算，其中的第一个导致了对该输入是接受的，而第二个计算不接受该输入。因此，按我们的定义，该机器接受 1010010。

$(q_0$1010010，q_0，$q_0)$	$(q_0$1010010，q_0，$q_0)$
⊢$(q_1$1010010，\$$q_1$，\$$q_1)$	⊢$(q_1$1010010，\$$q_1$，\$$q_1)$
⊢$(1q_2$010010，\$$q_2$，\$$q_2)$	⊢$(1q_3$010010，\$$q_3$，\$$q_3)$
⊢$(10q_2$10010，\$0q_2，\$$q_2)$	⊢$(10q_3$10010，\$$q_3$，\$0$q_3)$
⊢$(10q_1$10010，\$0q_1，\$$q_1)$	⊢$(10q_1$10010，\$$q_1$，\$0$q_1)$
⊢$(101q_3$0010，\$0q_3，\$$q_3)$	⊢$(101q_3$0010，\$$q_3$，\$0$q_3)$
⊢$(1010q_3$010，\$0q_3，\$0$q_3)$	⊢$(1010q_3$010，\$$q_3$，\$00$q_3)$
⊢$(10100q_3$10，\$0q_3，\$00$q_3)$	⊢$(10100q_3$10，\$$q_3$，\$000$q_3)$
⊢$(10100q_1$10，\$0q_1，\$00$q_1)$	⊢$(10100q_1$10，\$$q_1$，\$000$q_1)$
⊢$(101001q_2$0，\$0q_2，\$00$q_2)$	⊢$(101001q_3$0，\$$q_3$，\$000$q_3)$
⊢$(1010010q_2$，\$00q_2，\$00$q_2)$	⊢$(1010010q_3$，\$$q_3$，\$0000$q_3)$
⊢$(1010010q_4$，\$0$q_4$0，\$0$q_4$0)	⊢$(1010010q_4$，\$$q_4$，\$000$q_4$0)
⊢$(1010010q_4$，q_4\$00，q_4\$00)	停止，因无下一格局。
⊢$(1010010q_5$，q_5\$00，q_5\$00)	
接受。	

图 7.6 关于 NDTM M 的两个移动序列

下面定义 NDTM 的时间复杂性与空间复杂性。

定义7.3 称一台 NDTM 的时间复杂性是 $T(n)$，假若对于任何长为 n 的可接受的输入 w，都存在着一条导致接受状态的计算序列，该序列至多有 $T(n)$步。带头移动过程中任何一条带上被扫描到的不同方格数的总和不超过 S(n)，则 S(n)定义为该台 NDTM 的空间复杂性。

例7.6 对于例 7.5 所设计的机器 M，其时间复杂性为 $2n+2$。因为当对输入扫描结束（共用 $n+1$ 步）后，回头要对 2、3 带的内容进行比较（不超过 $n+1$ 步）。

原则上，对于每一台 NDTM 机，都可以设计一台 DTM 机来模拟它，使得两者都接受同一语言。然而 DTM 的时间耗费要大得多，可以证明，这种模拟的时间耗费下界是指数型的。确切地说，有以下定理：

定理7.4 设 L(M)是一台非确定型图灵机 M 所接受的语言，M 的时间复杂性是 $T(n)$。那么，必存在一台确定型图灵机 M'，它所接受的语言 L(M') = L(M) 且时间复杂性是 $O(C^{T(n)})$，其中 C 是某个正常数。

可以通过构造一台确定型机器 M'来摸拟 M 的工作，从而证明本定理，这里略去。

像定义非确定型图灵机一样，也可以定义其它的非确定型计算模型(比如非确定型的 RAM 模型等)。这些非确定型模型原则上只要在原有的确定型模型基础上，增加一条特殊指令

$$\text{CHOICE}(L_1，L_2，\cdots，L_k)$$

即可。这条指令的功能是，每当执行到它时，或许以不确定的方式选择转向这 k 个标号语句之一，或许不确定地从 k 个数中任取一个（与寻址方式有关）。除此之外，还可以考虑增加两个表示不同停机状态的语句：

1．success (成功)；

2．failure (失败)。

success 表示接收后停机；failure 表示不接受停机。

一台非确定型机器对应着一个非确定型算法。一个非确定型算法中可能有多个 CHOICE 语句。给定一个输入 I，在这些语句的各种不同选择的一切组合中，只要有某个可能的组合使算法达到 success，就说接受输入 I。这里输出（也可能没有）就可以认为是该算法对输入 I 的计算结果。由于使用 NDTM 定义非确定型算法很麻烦，我们往往采用类似于非确定型 RAM 模型，来描述各种非确定型算法。例如，下面的过程 CHOICESORT 描述了一个非确定型排序算法。

```
//非确定型排序算法//
procedure CHOICESORT
  begin
```

```
1.        for i = 1 to n do s[i] ← 0;
2.        for i = 1 to n do
            begin1
3.            j ← CHOICE (1: n);        //从1,2,…,n中任取一数//
4.            if S[j]=0 then S[j] ← A[i];
5.            else failure
            end1
6.        for i = 1 to n-1 do
7.          if S[i]>S[i + 1] then failure;
8.        print(S[1], S[2], … , S[n]);
9.        success;
        end.
```

例7.7 非确定型排序算法

输入 n 个实数 a_1，a_2，…，a_n。它们放在数组 A[1: n]中。

输出 将 a_1，a_2，…，a_n，按非递减序排列置于数组 S[1: n]中，并输出 S。

方法 见过程CHOICESORT。因为在语句3中，总存在一种选择序列使得S[1] ≤ S[2] ≤… ≤ S[n]，所以这个过程是对 n 个元素排序的非确定型算法。例如，设(a_1，a_2，a_3，a_4，a_5)=(8，7，3，9，1)。如果在执行语句3时，选取的整数序列正好是5，3，2，1，4，就能产生正确的输出序列1，3，7，8，9，并达到语句success。

对于各种不同的非确定型计算模型，同样可以证明它们之间的多项式相关性。这里就不详述了。

下面我们再叙述一个关于多带与单带的 NDTM 机的时间复杂性的一个定理，也不加证明。这个定理使我们在讨论与非确定型图灵机有关的问题时，可以只对单带机而言，从而使讨论简单一些。

设语言 L 可以为时间复杂性为 $T(n)$的 k 带 NDTM 机所接受，则 L 也可以被一台时间复杂性为 $O(T^2(n))$的单带 NDTM 机所接受。

§7.4 P 和 NP 问题类

现在我们将讨论局限于判定问题的求解。这种问题只有两个可能的解，或者回答“是”，或者回答“否”。抽象地说，判定问题 Π 由实例集合 D_Π 和回答为“是”的实例子集 $Y_\Pi \subseteq D_\Pi$ 组成。

我们感兴趣的多数判定问题具有附加结构，在描述判定问题时，要强调这些附加结构。因此，一般~~用来~~规定问题的标准格式由两部分组成。第一部分用各种分量，如集合、图、函数、数字等规定该问题的一般实例。第二部分陈述根据这个一般实例提出的是－否问题。于是，一个实例属于 D_Π 当且仅当它可以通过用规定类型的具体对象替换一般实例中的所有分

量得到，而这个实例属于 Y_Π 当且仅当具体到这个实例时，对所陈述的问题的回答为“是”。

例如货郎担问题，其对应的判定问题如下。

实例：一个有穷个“城市”的集合 C = {c_1，c_2，…，c_m}，对于每一对城市 c_i，$c_j \in C$ 有“距离” $d(c_i, c_j) \in Z^+$，以及界限 $B \in Z^+$（这里 Z^+表示正整数集合）。

问：是否有经过 C = {c_1，c_2，…，c_m}中所有城市的“旅行路线”其全长不超过 B，即是否有 C = {c_1，c_2，…，c_m}的一个排列次序<$c_\pi(1)$，$c_\pi(2)$，…，$c_\pi(m)$>使得

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B?$$

我们只考虑判定问题的原因是因为它们有一个非常自然的、适合在计算理论中研究的形式对应物。这个对应物叫做“语言”，其定义如下：

对于任意有穷符号集合Σ，我们用Σ*表示所有Σ的有穷符号串（包括空串）组成的集合。如果 L 是Σ*的一个子集，称 L 是字母表Σ上的语言。

例如，设Σ = {0, 1}，那么，Σ*由空字符串“ε”，字符串 0、1、00、01、10、11、000、001 以及所有其他由 0 和 1 构成的有穷字符串组成。于是{01，001，111，0010101}是{0，1}上的一个语言，由所有完全平方数的二进制表示组成的集合是{0, 1}上的一个语言，甚至，{0, 1}*本身是{0, 1}上的一个语言。

将判定问题的每一个实例编码成一个符号串。这样，一个判定问题就变成一个语言识别问题，这个语言由对应的判定问题中回答为“是”的一切实例编码的串组成。

当然选择编码方法时，必须慎重，因为一个问题的复杂性可能与编码的方法有关。由于问题的难度在本质上不依赖于用来决定时间复杂性的具体编码方法和计算机模型，因此一个“合理的”编码方法产生的串长度与标准的编码方法产生的串长度是多项式相关的。虽然不可能把我们在这里用的“合理的”这个词表示的含义形式化，但是任何“合理的”编码应该满足下面两个条件，这两个条件“抓住”了这个概念的主要内容：

(1) 实例的编码必须是简洁的，不能“填塞”不必要的信息或符号。

(2) 实例中出现的数字必须用十进制（或二进制、八进制、以及以任何不等于 1 的数为基的进制）表示。

如果我们规定只使用满足这些条件的编码方法，那么具体使用什么编码方法将不会影响关于一个给定问题的难度的判断。

作为描述问题的共同标准，可以对编码作如下一些规定：

(1) 一切整数都采用二进制数表示；

(2) 一个图的 n 个顶点总是使用整数 1，2，…，n 来表示（当然这些数是二进制形式），而一条边(i, j)则使用两个二进制数表示；

(3) 可以使用少数简洁的特殊符号；

(4) 常用的字母和符号也采用二进制编码。等等。

当把整数及其他符号都采用二进制编码后，一个问题的判定过程就可以形式化地描述如下：

已知 $L \subseteq \{0, 1\}^*$，对于 $x \in \{0, 1\}^*$，若 $x \in L$ 则回答“是”；若 $x \notin L$，则回答“非”。

这里，{0, 1}*是指由有限个 0 和 1 组成的串的集合。

因为“问题”和“语言”的这种关系，我们常常对“语言”和“问题”不加区别。在许多情况下，只要讨论“语言”就行了。

容易理解，一般来说，一个问题的“解的存在性判定”比“问题求解”要容易，至少不会更难。进一步分析可以发现，一个问题的求解与同他相对应的判定问题在多项式意义下有相同的求解难度。例如，给定问题如下：给定一个图，如果其中存在哈密顿回路，我们要从中找到一条哈密顿回路。这个求解问题同判定一个给定的图是否哈密顿图具有相同的求解难度。

事实上，如果存在多项式时间算法求解一条哈密顿回路，那么，利用这个算法可以构造算法判定给定的图是否哈密顿图，这只要在求解算法找到解的情况下，回答“是”，在求解算法找不到解的情况下，回答“非”即可。

反过来，如果存在多项式时间算法判定一个给定的图是否哈密顿图，那么，设判定算法为 A，给定的 n 阶图为 G，利用算法 A，可以构造多项式时间求解算法如下：

1．调用 A 判定 G 是否哈密顿图。若 G 不是哈密顿图，算法终止，否则继续 2。

2．任取 G 的边 e，调用 A 判定 G-{e}是否哈密顿图，如果是，则执行 G←G－{e}

3．如果 G 中多于 n 条边，转 2 继续去边。否则 G 中剩余的 n 条边就是 G 的哈密顿回路。

设 A 的时间复杂性为 $p(n)$，它是关于 n 的一个多项式函数。则求解算法的时间复杂性不超过 $n^2p(n)$。因为语句 2 和语句 3 组成的去边过程最多执行 $\frac{n(n-1)}{2}-n$ 次。

不同判定问题的难度是不同的。例如，对于图的顶点着色问题，一个图是否二色可染的判定问题是多项式可解的。但是一般的着色问题，判定非常困难。又如，对给定的货郎担问题，问是否有一条代价小于 B（B 已给定）的周游路线，这个判定问题就没有判定一个数是不是某数的真因子那么容易。如果 B 给得特别大，大到比代价矩阵中 n 个最大元素的和还大，而且给定的图是完全图，能很快回答“有”；或者 B 给得特别小，小到比代价矩阵中最小的元素还小，能很快地回答“没有”。除了这样一些个别的极端情况，至今还没有人能给出一个算法，对任意的代价矩阵和 B，都能在多项式时间内给出“有”或“否”的回答。

我们希望按问题的计算难度把各方面的问题分成不同的类，以便开展讨论。

定义7.4 由确定型图灵机在多项式时间内可识别的一切语言的集合称为 P 类语言。广义地，由确定型图灵机在多项式时间内可解的一切判定问题的集合称为 P 类问题。

定义7.5 由非确定型图灵机在多项式时间内可识别的一切语言类称为 NP 类语言。由非确定型图灵机在多项式时间内可计算的判定问题类称为 NP 类问题。

习惯上，人们均简称 P 类问题和 NP 类问题为 P 问题和 NP 问题。

显然，如果一个问题 $Q \in P$，就有 $Q \in NP$。因为任何确定型图灵机不过是非确定型图灵机的一个特例。因此有 $P \subseteq NP$。然而，NP 问题类所包含的问题真的比 P 问题类更广泛吗？是否有某些问题 Q，$Q \in NP$，但 $Q \notin P$ 呢？这是当今计

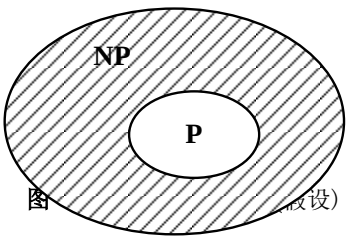

图 （假设）

算机科学中具有代表性的难题之一。至今，人们既没有证明 P ≠ NP，也没有证明 P = NP。

因为有定理 7.4，有人猜想 P ≠ NP。但是，迄今为止，它仍然没有被证明。因此，关于 P 和 NP 的关系，人们暂时假定如图 7.7 所示，图中，P 是 NP 的一个子集。但是，阴影部分是否非空，即 NP－P 是否非空，目前还不能确定。

§7.5 NP 完全性和 COOK 定理

定义7.6 设 $L_1 \subseteq \Sigma_1^*$和 $L_2 \subseteq \Sigma_2^*$，如果存在一个函数 *f*，满足

(1) 存在一个确定型算法，它在多项式时间内计算函数 *f*；

(2) 对一切 $x \in \Sigma_1^*$，$x \in L_1$ 当且仅当 $f(x) \in L_2$，

则说 *f* 是从 L_1 到 L_2 的一个多项式变换。简记为 $L_1 \propto L_2$，读作“L_1 多项式变换到 L_2”。

更一般地，有

定义7.7 对于问题 Q_1 和 Q_2，如果

(1) 对 Q_1 的任何具体问题 I，存在一个多项式时间的确定型算法，它计算出 *f*(I)，且 *f*(I) $\in Q_2$；

(2) 对于 *f*(I) $\in Q_2$ 的解，存在一个多项式时间的确定型算法，得到 I $\in Q_1$ 的解，

则说 Q_1 多项式归结为 Q_2，记作 $Q_1 \propto Q_2$。

多项式归结显然有如下性质：

性质 1：如果 $Q_1 \in P$ 且 $Q_2 \propto Q_1$，则 $Q_2 \in P$；

性质 2：如果 $Q_1 \propto Q_2$，$Q_2 \propto Q_3$，则 $Q_1 \propto Q_3$。

定义7.8 对于问题 Q 及任意判定问题 $Q_1 \in NP$，都有 $Q_1 \propto Q$，则称 Q 是 NP 困难的（NP-Hard）。

所谓问题 Q 是 NP 困难的，是指问题 Q 不比 NP 中的任何问题容易，至少是同样难或者更难。例如货郎担问题的求解或货郎担“判定问题”都是 NP 困难的。

定义7.9 对于问题 Q，若满足：

(1) Q ∈ NP；

(2) Q 是 NP 困难的；

则称 Q 为 NP 完全的（NP-Complete）。所有 NP 完全问题构成的集合记作 NPC。

直观地说，所谓 NP 完全问题是 NP 问题类中最困难的问题，它们彼此之间都可以多项式归结。

因此有

定理7.5 设 Q ∈ NPC。P = NP 成立当且仅当 Q ∈ P。

证明： 由 NP 完全性的定义，显然。 ■

具有 NP 完全性质的问题，现在至少已经发现了几

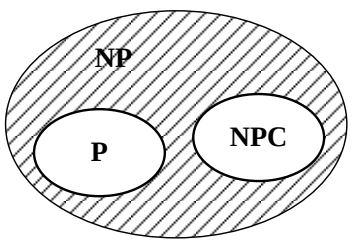

图 7.8 P, NP 和 NPC 的关系(假设)

千个甚至更多，他们分别属于很多领域。迄今为止还没有发现其中任何一个是属于 P 的。这一事实，增强了人们相信 P ≠ NP 的猜测。如果事实果真如此，P、NP 和 NPC 三者之间的关系或许如图 7.8 所示。若 P = NP 的假设成立，局面就另当别论了。

S. A. COOK 对于 NP 完全问题进行了许多研究，并于 1971 年证明了一个划时代的结果。这就是以下定理。

定理7.6 （COOK）布尔表达式的可满足性问题是 NP 完全的。

证明： 可满足性问题 SAT∈NP 是明显的。因为对于任何给定的布尔表达式，不妨设它有且仅有 K 个变量。我们可设计一个非确定型机器，先猜测这 K 个变量的真值赋值，并代入布尔表达式检验。如果布尔表达式取真值“真”，就是可满足的。只要一个布尔表达式至少存在一组真值赋值使它取真，一台 NDTM 机器总可以猜中这一组真值（这是由 NDTM 机器的定义决定的），并在多项式时间内完成检验。所以这一问题是属于 NP 的。

现在，只要证明对任何语言 L∈NP，都能多项式归结为这一问题，整个证明就告完成。设 M 是一台能在多项式时间内识别 L 的非确定型图灵机，而 w 是对 M 的一个输入。由 M 和 w 我们能构造一个布尔表达式 w_0，使得 w_0 是可满足的当且仅当 M 接受 w。

不妨假定 M 是一台单带机，设 M 有 s 个状态 q_1，q_2，…，q_s，它的带符号是 X_1，X_2，…，X_m。P(n)是 M 的时间复杂性（对于 M 不是单带 NDTM 的情形，我们不难证明，属于 NP 的任何语言能由一台单带的 NDTM 识别，也就是可以证明任何 k 带的 NDTM 可以用一台单带 NTDM 模拟，留给读者练习）。

设 w 是给 M 的一个长度为 n 的输入。如果 M 接受 w，只需要 P(n)次移动。也就是说至少存在一个 M 的格局序列 Q_0，Q_1，…，Q_q，它使 M 从初态 q_0 达到接受状态 q_f。q_f 是格局 Q_q 中的状态，q_0 是格局 Q_0 中的状态，$q \le P(n)$且没有任何格局会使用多于 P(n)个带方格。q_0、$q_f \in \{q_0, q_1, q_2, \cdots, q_s\}$。

我们构造一个布尔式 w_0，用它“模拟”M 中所能进入的格局序列：即 w_0 的变元的每个真（1）假（0）指派至多对应 M 的一条计算路（也可能不是 M 的合法的计算路），w_0 取值 1 当且仅当指派对应了导致接受格局的计算路 Q_0，Q_1，…，Q_q。换言之，w_0 可满足当且仅当 M 接受 w。我们先介绍构造 w_0 需要用到的各个变量。

(1) C<i, j, t> = 1 当且仅当在时刻 t，M 的输入中第 i 个带符号为 X_j。这里 $1 \le i \le P(n)$，$1 \le j \le m$，$0 \le t \le P(n)$。

(2) S<K, t> = 1 当且仅当在时刻 t，M 的状态为 q_k。这里 $1 \le K \le s$ 和 $0 \le t \le P(n)$。

(3) H<i, t> = 1 当且仅当在时刻 t 带头扫描着带上的第 i 个方格。这里 $1 \le i \le P(n)$，$0 \le t \le P(n)$。

上述变量共有 $m \cdot P(n)^2 + s \cdot P(n) + P^2(n)$个，故阶为 $O(P^2(n))$。如果要用二进制数来编码表示这些命题变元，则至多用到 $c\log_2 n$ 位的二进数即可，c 为与 P(n)有关的常数。为了方便起见，对上述的每个变量，我们可以用一个简单符号来代替长为 $c\log_2 n$ 的符号，这样丢掉一个因子 $c\log_2 n$ 并不影响对问题的讨论，因为我们只需要以多项式时间为界的函数。

现在定义谓词函数 U(x_1, x_2, …, x_r)如下：U(x_1, x_2, …, x_r)取值 1 当且仅当变量 x_1，x_2，…，x_r 中只有一个取值 1。因此，U(x_1，x_2，…，x_r)的布尔表达式可以写成如下形式：

$$U(x_1,x_2,\cdots,x_r)=(x_1+x_2+\cdots+x_r)\left(\prod_{\substack{i,j\\ i\neq j}}(\neg x_i+\neg x_j)\right)$$

上式的第一个因子断言 x_1，x_2，…，x_r 中至少有一个 x_i 取 1，而后面的 $r(r-1)/2$ 个因子断言没有两个 x_i 取 1。从$(x_1+x_2+\cdots+x_r)\left(\prod_{\substack{i,j\\ i\neq j}}(\neg x_i+\neg x_j)\right)$还可以看出，U($x_1$，$x_2$，…，$x_r$)的长度最多为 $O(r^3)$。

如果 M 接受 w，那么在 M 处理 w 时存在一个导致接受的格局序列 Q_0，Q_1，…，Q_q。为了使讨论简单而又不失普遍性，我们假定所有格局序列的长度都是 P(n)。对于长度小于 P(n)的情形，可以修改 M，使得无论 M 在何处达到接受状态，它都执行带头不移动的动作或执行状态不变的一些动作，直到它的格局序列的长达到 P(n)为止。

于是断言一个格局序列 Q_0，Q_1，…，Q_q 是一个接受序列，相当于断言 7 条事实：

(1) 在每个格局中带头实际上只能扫描一个方格；

(2) 在每一个格局中，每个方格的带符号是唯一确定的；

(3) 在每一个格局中，机器只有一个当前状态；

(4) 从一个格局到下一个格局，每次最多有一个方格（被带头扫描着的那个方格）的符号被修改；

(5) 两个连续格局之间的状态的改变、带头位置的改变和带上符号的改变都是 M 的下移函数所允许的。

(6) 第一个格局是初始状态下的格局；

(7) 最后一个格局中的状态为终结状态。

现在，我们构造出如下的 7 个布尔表达式 A、B、C、D、E、F、G，它们分别和以上 7 个句子一一对应。

1、A 断言：在 M 的每一个时间单位中，带头正好只扫描着一个方格。设 A_t 是断言时间 t 实际上被扫描的带方格，则

$$A=A_0A_1\cdots A_{P(n)}$$

其中，

$$\neg A_t=U(H<1,t>,H<2,t>,\cdots,H<P(n),t>)。$$

A_t 的涵义是说：在时刻 t，M 的磁头恰好扫描着某一磁带方格。注意，如果把 U 展开，表达式 A 的长度最多是 $O(P^4(n))$，并且在这个时间内可以写完。

2、B 断言：在每一个单位时间内，每一个方格中只有一个带符号。设 B_{it} 断言在时刻 t，第 i 个方格中只含有一个带符号，则

$$B = \prod_{i,t} B_{it}$$

其中，

$$B_{it}=U(C<i, 1, t>, C<i, 2, t>,\cdots, C<i, m, t>)。$$

式中 B_{it} 的长度与 n 无关，而 m 是带符号集的长度，它只与 M 有关而与 n 无关。因此 B 的长度是 $O(P^2(n))$。

3、C 断言：在每个时刻 t，M 只有一个确定的状态：

$$C = \prod_{0\le t\le P(n)} U(S(1,t),S(2,t),\cdots S(s,t))$$

因为 s 是 M 的状态数，它是一个常数，所以 C 的长度为 $O(P(n))$。

4、D 断言：在时刻 t 最多只有一个方格的内容被修改：

$$D = \prod_{i,j,t}[(C\langle i,j,t\rangle \equiv C\langle i,j,t+1\rangle) + H\langle i,t\rangle]$$

其中，表达式$(C\langle i,j,t\rangle \equiv C\langle i,j,t+1\rangle) + H\langle i,t\rangle$断言以下二者之一：

（1） 在时刻 t 带头扫描着第 i 格，或者，

（2） 在时刻 $t+1$，方格 i 中是符号 X_j，当且仅当在时刻 t，方格 i 中是符号 X_j。

因为 A 和 B 断言在时刻 t 带头只能扫描着一个带方格和方格 i 上仅有一个符号，所以，在时刻 t，或者带头扫描着方格 i（这里的符号可能被修改），或者方格 i 的符号不变。

注意，由于 $1 \le i \le P(n)$，$1 \le j \le m$ 且 $1 \le t \le P(n)$，m 为常量，故 D 的长为 $O(p^2(n))$。

5、E 断言：根据 M 的下移函数δ，从一个格局一定可以成功地转向下一个格局。设 E_{ijkt} 断言下列四种情形之一：

在时刻 t 第 i 格的符号不是 X_j；

在时刻 t 带头没有扫描着方格 i；

在时刻 t，M 的状态不是 q_k；

按 M 的下移函数δ，从前一格局能获得下一个格局；

即

$$E_{ijkt} = \neg C\langle i,j,t\rangle + \neg H\langle i,t\rangle + \neg S\langle k,t\rangle + \sum_l (C\langle i,f_l,t+1\rangle S\langle k_l,t+1\rangle H\langle i_l,t+1\rangle)$$

于是，

$$E = \prod_{i,j,k,t} E_{ijkt}$$

Σ_l 中的 l 遍历当机器 M 扫描着 x_j 且处于状态 q_k 时所有可能的下一动作，即对每一$<q, x, d> \in \delta(q_k, x_j)$，有一 l 值使 $x_j = x$，$q_k = q$，且根据 d 为 L，S，或 R，i_l 分别为 $i-1$，i，$i+1$。δ为机器 M 的下移函数。

因为 M 是不确定的，故可能有 $l>1$，但在任何情况下却一定有某个常数 c，使得 $l<c$。故 E_{ijkt} 的长度以常数为界而与 n 无关。注意到 i, j, k, t 的变化区域，可知 E 的长度是 $O(P^2(n))$。

6、F 断言 M 满足初始条件：

$$F = S\langle 1,0\rangle H\langle 1,0\rangle \prod_{1\le i\le n} C\langle i, j_i, 0\rangle \prod_{n\le i\le P(n)} C\langle i, 1, 0\rangle$$

其中$S\langle 1,0\rangle$断言在时刻 $t = 0$，M 处于状态 q_1 下，我们总可以取这个状态为初始状态；$H\langle 1,0\rangle$断言在时刻 $t = 0$，M 的带头扫描着最左边的带方格；$\prod_{1\le i\le n} C\langle i, j_i, 0\rangle$断言在时刻 $t = 0$,带上最前面的 n 个方格中放有串 w 的 n 个符号；而$\prod_{n\le i\le P(n)} C\langle i, 1, 0\rangle$断言带上的其余各方格中开始都是空白符。这里不妨假定 X_1 就是空白符。显然，F 的长度是 $O(P(n))$。

7、G 断言 M 最终将进入终止状态。因为我们已经对 M 进行过修改，一旦 M 在某个时刻 t 进入终止状态（$1 \le t \le P(n)$）,它将始终停在这个状态。所以我们有 $G = S(s, P(n))$。不妨认为 q_s 是 M 的终结状态。

现在定义 w_0＝ABCDEFG。它就是我们所要构造的布尔表达式。w_0 可满足的充分必要条件是 w 被 M 所接受。

因为 w_0 的每一个因子最多需要 $O(P^4(n))$个符号，它一共只有 7 个因子，从而 w_0 的符号长度不过是 $O(P^4(n))$。即使用长度为 $O(\log_2 n)$的符号串来取代描述各个变量的简单符号，w_0 的长度也不过是 $O(P^4(n)\log_2 n)$。或者说，存在一个常数 c，w_0 的长度不超过 $cnP^4(n)$。因此可以肯定，对给定的 w 和 $p(n)$，w_0 的长度是 w 的长度的多项式函数。

这里并没有对语言 L 加任何限制，也就是说，对属于 NP 的任何语言，都能在多项式时间内，将其转换为布尔表达式的可满足性问题。所以我们可以断定布尔表达式的可满足性问题是 NP 完全的。 ■

COOK 定理的重要性是明显的，它实际上给出了第一个 NP 完全问题。对于任何问题 Q，只要能证明（1）$Q \in NP$；（2）$SAT \propto Q$，则 $Q \in NPC$。于是，COOK 定理之后，证明一个问题 Q 的 NP 完全性由下述三步组成：

(1) 证明问题 Q 属于 NP。

(2) 选择一个已知的 NP 完全问题 Q'。

(3) 构造从 Q'到 Q 的多项式变换函数 f。

就是根据这样的思路，在 COOK 证明了这一结果后，人们很快地证明了其它许多问题

的 NP 完全性。

§7.6 若干 NP 完全问题及证明

本节讨论 NP 完全性证明。所采用的思路是上节末尾介绍的思路：对任意问题 Q，

(1) 证明问题 Q 属于 NP。

(2) 选择一个已知的 NP 完全问题 Q'。

(3) 构造从 Q'到 Q 的多项式变换函数 Q。

下面是几个已知的 NP 完全问题。

1．合取范式的可满足性问题

实例：有穷的变量集合 U 上的子句集 C = {c_1，c_2，…，c_m}，$1 \le i \le m$（变量集合 U 上的子句是 U 中部分变量构成的文字的析取。文字的定义是，设 x 是 U 中的变量，x 和$\neg x$ 都是文字）。

问：对于 U 是否存在满足 C 中所有子句的真值赋值？

证明要点：可由布尔表达式可满足性问题直接证明。

2．三元可满足性

实例：有穷的变量集合 U 上的子句集 C = {c_1，c_2，…，c_m}，其中$|c_i| = 3$（c_i 仅含 3 个文字），$1 \le i \le m$。

问：对于 U 是否存在满足 C 中所有子句的真值赋值？

证明要点：可由合取范式可满足性问题变换而来。

3．三维匹配

实例：集合 $M \subseteq W \times X \times Y$，这里 W，X 和 Y 是三个不相交的集合，且有相同的元素个数 q。

问：M 是否包含一个匹配，即是否有子集 $M' \subseteq M$ 使得$|M'| = q$ 且 M'中任何两个元素的任何坐标都不相同？

证明要点：可由三元可满足性问题变换而来。

4．顶点覆盖

实例：图 G = (V，E)和正整数 $k \le |V|$。

问：G 是否有大小不超过 k 的节点覆盖，即是否有子集 V'使得$|V'| \le k$ 并且对每一条边(u，v) ∈ E，u 和 v 中至少有一个属于 V'?

证明要点：可由三元可满足性问题变换而来。

5．团

实例：图 G = (V，E)和正整数 $J \le |V|$。

问：G 是否包含大不小于 J 的团，即是否有子集 $V' \subseteq V$，使得$|V'| \ge J$ 并且 V'中每两个节点都由 E 中的一条边连接着。

证明要点：可由顶点覆盖变换而来。

6．哈密顿回路

实例：图 G = (V，E)。

问：G 是否包含一条哈密顿回路，即是否有 G 的节点排列次序<v_1，v_2，…，v_n>使得(v_n，v_1) ∈ E 和(v_i，v_{i+1}) ∈ E，$1 \le i \le n$-1？这里 n = | V |。

证明要点：可由顶点覆盖变换而来。

7．划分

实例：有穷集合 A 以及每一个 $a \in A$ 的“大小”$S(a) \in Z^+$。

问：是否有子集 A' ⊆ A 使得$\sum_{a\in A'} S(a) = \sum_{a\in A-A'} S(a)$？

证明要点：可由三维匹配变换而来。

8．背包问题

实例：有限集合 U，每个 u ∈ U 的大小为 S(u) ∈ Z^+且值为 v(u) ∈ Z^+，B，K ∈ Z^+。

问：是否有子集 U' ⊆ U 使得$\sum_{u\in U'} S(u) \le B$和$\sum_{u\in U'} v(u) \ge K$?

证明要点：可由划分变换而来。

9．流水作业车间调度

实例：处理机数目 $m \in Z^+$，任务集 J，每个任务 j 可由 m 个子任务 $t_1[j]$，$t_2[j]$，…，$t_m[j]$ 组成（$t_i[j]$表示由处理机 i 执行），每个任务 t 的长度 $l[t] \in Z^+$,总的截止时间 D ∈ Z^+。

问：是否有 J 的流水作业车间时间表适合总的截止时间，这里所要求的时间表就是开放式车间时间表加上附加限制：对于每个 $j \in J$ 和 $1 \le i \le m$，有 $\sigma_{i+1}(j) \ge \sigma_i(j) + l(t_i[j])$?

证明要点：可由划分变换而来。

10．子集的和

实例：有限集 A，每个 $a \in A$ 的“大小”$S(a) \in Z^+$，以及正整数 B ∈ Z^+。

问：是否有子集 A' ⊆ A，使得 A'中元素的大小之和恰好为 B，即使得$\sum_{a\in A'} S(a) = B$？

证明要点：可由划分变换而来。

NP 完全问题还有很多，不胜枚举。我们下面给出前两个问题的证明。关于其余的问题的证明，有兴趣的读者可以查阅计算复杂性理论或者 NP 完全理论的相关书籍。

如果一个布尔表达式是一些文字的和之积，则称该布尔表达式为合取范式，简称 CNF。这里的文字或者是变量 x，或者是$\neg x$。例如，$(x_1+x_2)(x_2+x_3)(x_3+\neg x_2+\neg x_1)$就是一个合取范式，$x_1x_2+x_3$ 不是合取范式。

定理7.7　合取范式的可满足性问题是 NP 完全的。

证明　如果在 Cook 定理中定义的 7 个布尔表达式 A、B、C、D、E、F、G 或者本身已经是合取范式，或者有的虽然不是合取范式，但可以应用布尔代数中的定律将它们化成合取范式，而且合取范式的长度与原表达式的长度只差一个常数因子，证明即告完成。

因为$U(x_1,x_2,\cdots,x_r)=(x_1+x_2+\cdots+x_r)\left(\prod_{i\neq j}(\neg x_i+\neg x_j)\right)$已经是一个合取范式，所以，A，B，C 都是合取范式。按照 F 和 G 的定义，他们都是文字的积，所以，他们都是合取范式。

D 是形如$(x\equiv y)+z$的表达式的积，如果我们以$xy+\neg x\neg y$替换$x\equiv y$，就得到

$$(x\equiv y)+z=xy+\neg x\neg y+z=(x+\neg y+z)(\neg x+y+z)$$

以$(x+\neg y+z)(\neg x+y+z)$替换 D 中的一切形如$(x\equiv y)+z$的项，得到的布尔表达式同原式等价，但替换后的布尔表达式是一个合取范式，而且表达式的长度最多是原式长度的两倍。

最后，由于表达式 E 是 E_{ijkl}的积，每个 E_{ijkl}的长度与 n 无关且能展开成一个合取范式（其长度也与 n 无关）。因此将 E 变换成合取范式后 E 的长度与原式长最多只差一个常数因子。

这样，我们就证明了 Cook 定理中定义的布尔表达式 w_0 变换成一个等价的合取范式后，其长度只相差一个常数因子。常数时间内可以实现的转换，当然是多项式变换。 ■

上面已经证明了合取范式的可满足性问题是 NP 完全的。甚至对问题加以更多的限制后，我们还能证明布尔表达式的可满足性仍然是 NP 完全的。如果一个布尔表达式的合取范式的每一个乘积项最多是 k 个文字的析取式，就称之为 k 元合取范式（k-CNF）。一个 kSAT 问题是确定一个 k-CNF 中的表达式是否可满足。对于 $k=1$ 或 $k=2$，人们已经找到了确定型的多项式时间算法。对 $k=3$，有如下定理。

定理7.8 3SAT（三元可满足性问题）是 NP 完全的。

证明 因为非确定型算法只需要猜想一个对变量的真值赋值，并且在多项式时间内检查这个真值赋值是否满足所有给定的三文字子句，所以容易看到 3SAT ∈ NP。

我们证明合取范式的可满足性问题可以多项式变换为 3SAT，即可把 SAT 变换到 3SAT。

给定一个合取范式，其中每一个合取项具有形式

$$(x_1+x_2+\cdots+x_k),\quad k\geq 4$$

添加 $k-3$ 个新变元 y_1，y_2，…，y_{k-3}。做一个三变元合取范式

$$(x_1+x_2+y_1)\,(x_3+\neg y_1+y_2)\,(x_4+\neg y_2+y_3)\,\cdots\,(x_{k-2}+\neg y_{k-4}+y_{k-3})\,(x_{k-1}+x_k+\neg y_{k-3}) \tag{7.1}$$

例如,当 $k=4$ 时,上式的形式是

$$(x_1+x_2+y_1)\,(x_3+x_4+\neg y_1) \tag{7.2}$$

可以证明(7.1)与$(x_1+x_2+\cdots+x_k)$真值相同。事实上，如果有指派满足$(x_1+x_2+\cdots+x_k)$，这个指派可以延拓成满足(7.1)的指派。例如，设 $x_i=1$,那么对 $j\leq i-2$ 置 y_j 为 1；对于 $j>i-2$ 置 y_j 为 0。这时，式(7.1)取值 1。反之，如果有指派满足(7.1)，则该指派必然满足$(x_1+x_2+\cdots+x_k)$。理由如下：如果满足(7.1)的指派给某个 x_i 指派为 1，该指派当然满足$(x_1+x_2+\cdots+x_k)$。如果满足(7.1)的指派给所有 x_i 指派为 0，必然给 y_1 指派 1。于是 y_2 指派 1，…，y_{k-3}

指派 1，$\neg y_{k-3}$ 指派 1，这是不可能的。

对于每一个形如$(x_1+x_2+\cdots+x_k)$，$k \geq 4$ 的项，我们总可以用一个式(7.1)这样的式子替换它，而且替换式的长度与原式长度只差一个常数因子。实际上，给定任何一个合取范式 E，都能把它变成一个 3 元合取范式 E'，使得 E'是可满足的当且仅当 E 是可满足的，而且计算两者的时间只差一个常数因子。 ■

现在已经证明的 NP 完全问题至少几千个，不可能一一列举。图 7.9 中给出了一部分 NP 完全问题及它们之间的可能的归结关系。这些问题涉及的面很广，它包括图论、网络设计、集合与划分、存贮与检索、排序与调度、数学规划、代数与数论、游戏与智力测验、逻辑学、自动机与形式语言理论、程序的优化及其他许多问题。这些问题既有实用价值，又有理论意义，因此吸引着大批计算机科学家和数学家进行深入的研究。

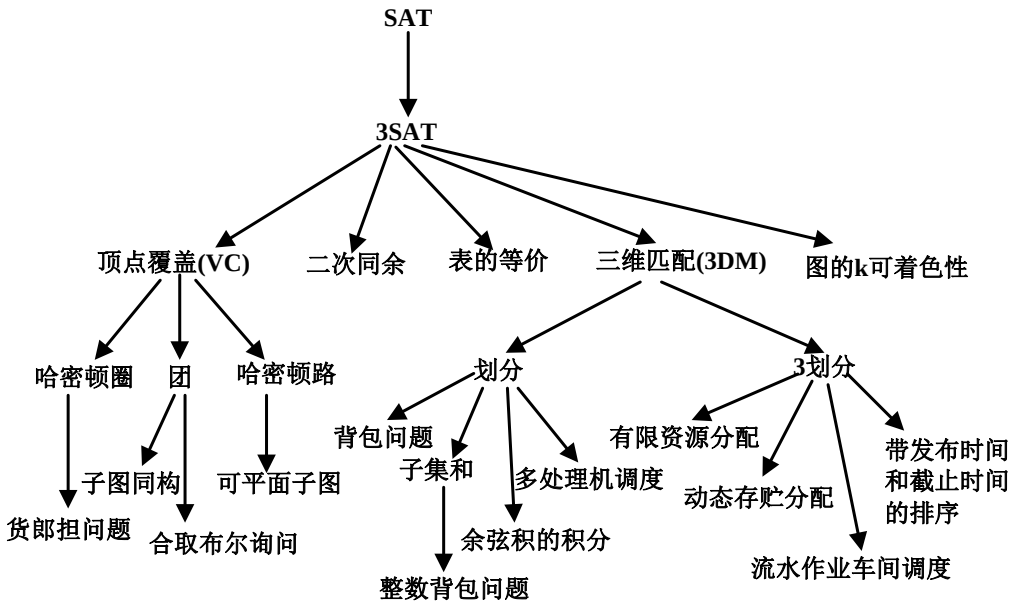

图 7.9 某些 NP 完全问题及归结顺序

§7.7 Co-NP 类问题

定义7.10 对任一判定问题 Π = ($D_Π$，$Y_Π$)，其中，$D_Π$ 为所有实例的集合，$Y_Π$ 为所有回答为“是”的实例集合，定义 $\overline{\Pi} = (D_\Pi$为 Π = ($D_Π$，$Y_Π$)的补或余，其中 $Y_{\overline{\Pi}} = D_\Pi$ 。

例如：对于哈密顿回路问题，它的补可如下定义。

实例：图 G = (V，E)。

问：G 是否不存在哈密顿回路。

记哈密顿回路问题为 HC，它的补记为$\overline{HC}$。

为了证明 G 不存在哈密顿回路，需要给出 G 中节点所有可能的排列，并且验证这些排列都不是 G 的哈密顿回路。这和验证 G 存在哈密顿回路是不同的，验证没有排列是 G 的哈密顿回路必须检查所有的排列，而验证 G 存在哈密顿回路只需检查一个排列就可以了。检查 G 的所有排列是不可能在多项式时间内完成的。事实上，至今仍然不知道$\overline{HC}$是否在 NP 中。

定义7.11 定义

$$Co-NP=\{\overline{\Pi}|\Pi\in NP\};$$

$$Co-P=\{\overline{\Pi}|\Pi\in P\}。$$

关于 P 与 Co－P 的关系有以下定理。

定理7.9 如果 是 P 类问题，那么 的补 也是 P 类问题。

证明：因为Π是 P 类问题，因此存在多项式时间算法求解Π。利用求解Π的多项式时间算法，可以构造求解$\overline{\Pi}$的多项式时间算法，这只要将求解Π的多项式时间算法中回答“是”的时候，改成回答“否”，回答“否”的时候改成回答“是”即可。 ■

因此，若判定问题$\Pi\in P$，则有$\Pi\in NP\cap Co-NP$即$P\subseteq NP\cap Co-NP$。

定理7.10 如果一个 NP 完全问题的补是 NP 的，那么 $\mathbf{NP=Co}$-。

证明：假设有一个 NP 完全问题Π_0，它的补$\overline{\Pi_0}$是 NP 的，我们证明，任何 NP 问题Π的补$\overline{\Pi}$也是 NP 的。

因为Π_0是 NP 完全问题，故Π可多项式变换到Π_0，而这个变换也构成从$\overline{\Pi}$到$\overline{\Pi_0}$的多项式变换。于是能给出$\overline{\Pi}$的任意回答为“是”的实例的简单检验，即可在多项式时间内完成两件事情：生成$\overline{\Pi_0}$的回答为是的实例的多项式变换的运算过程和$\overline{\Pi_0}$的该实例的证明过程。因为$\overline{\Pi_0}$是 NP 的，这样的多项式时间内可检验的论证存在，且因变换是多项式时间的，故整个证明就是多项式时间的，所以$\overline{\Pi}$是 NP 的。因Π是任意的 NP 问题，故推出$NP=Co-NP$。 ■

NP 和$Co-NP$的关系，也是一个没有解决的难题。即使$\Pi \in NP$，也不一定有$\overline{\Pi} \in Co-NP$

人们一般猜想$NP \neq Co-NP$。

类似地可以证明：若$NP \cap Co-NP \neq \emptyset$，则亦有$NP = Co-NP$。

于是，NP 完全问题是那样一些问题，它们的补很可能不是 NP 的。反之，如果一个 NP 问题的补也是 NP 的，就表明该问题不是 NP 完全的。

综合以上各节的讨论，我们可以用图 7.10 来表示各类问题之间的关系。

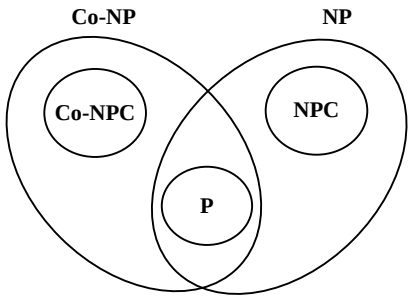

图 7.10 一些问题类之间的一种可能的关系（假设）

无论是 NP 类还是$Co-NP$类，都是多项式空间可解的。即 NP⊆P-SPACE，$Co-NP$⊆P-SPACE。在 P-SPACE 之外，还有更复杂的问题类。人们现在猜想的各类问题的计算难度大体如图 7.11 所示。现实世界中还存在大量的比 NP 类、Co－NP 类更加难解的问题！

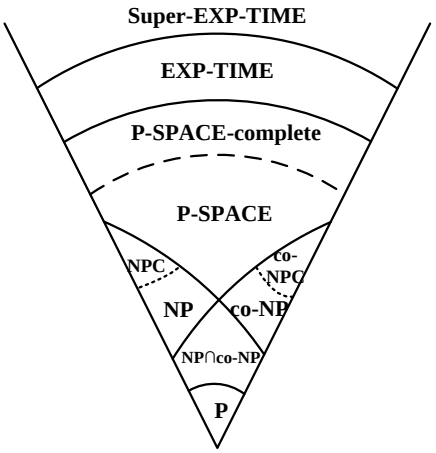

图 7.11 复杂性的分级结构

本章小结

本章定义了确定型图灵机和非确定型图灵机，讨论了图灵机模型与 RAM 模型的关系。在此基础上定义了 P 和 NP 两个问题类，讨论了 NP 完全性，证明了 COOK 定理，讨论了 Co-NP 类问题及其性质。

COOK 定理的重要性是巨大的，它实际上给出了第一个 NP 完全问题。对于任何问题 Q，只要能证明（1）$Q \in NP$；（2）$SAT \propto Q$，则 $Q \in NPC$。于是，COOK 定理之后，证明一个问题 Q 的 NP 完全性由下述三步组成：

(1) 证明问题 Q 属于 NP。

(2) 选择一个已知的 NP 完全问题 Q'。

(3) 构造从 Q'到 Q 的多项式变换函数 *f*。

就是根据这样的思路，人们很快地证明了其它许多问题的 NP 完全性。

习 题

10.1 对于表 7.1 定义的图灵机，给出输入串是：（*a*）0010，（*b*）01010 时的格局序列。

10.2 设计一台三带图灵机，在带 1 和带 2 上输入两个二进制正整数，在带 3 上输出这两个整数的和。假定各带的左端有一个特殊符号“τ”做标记。

10.3 设计一台多带图灵机，给它输入两个二进制正整数时，它计算出这两个数的积。

10.4 设计做下述工作的图灵机：

(1) 当带 1 上输入0^n时，在带 2 上输出0^{n^2}；

(2) 接受一切形如$0^n1^{n^2}$的输入串。

10.5 对于例 7.5 给出的非确定型图灵机 M，输入符号串 10101，给出一切合法的格局序列。M 能接受这个输入吗？

10.6 用类高级语言设计非确定型算法，使得它们接受一切形如 $10^{i_1}10^{i_2}\cdots10^{i_k}$ 的串，串满足的条件是：只要对某个 r，s，$1\le r<s\le k$，有 $i_r=i_s$。

10.7 证明定理 7.4。

10.8 证明一台多带确定型图灵机能由一台单带确定型图灵机模拟，而且二者之间的时间复杂性是多项式相关的。[提示：可以只考虑由单带机来模拟双带机的工作。]

10.9 写一个 RAM 程序，它能在 $O(n)$的时间内求一个布尔合取范式的值。

10.10 给出一个 RAM 程序，它能在 $O(n)$的时间内求出一个布尔表达式的值。

10.11 证明团（clique）问题是 NP 完全的。[提示：把 CNF 可满足性问题多项式归结为

团问题。]

10.12 证明顶点覆盖（VC）问题是 NP 完全的。[提示：由 SAT∝VC 或 clique∝VC。]

10.13 证明哈密顿回路问题是 NP 完全的。

10.14 设 $P_1(x)$和 $P_2(x)$都是多项式。证明存在一个多项式，使得对任意 x，它的值超过 $P_1(P_2(x))$。

10.15 给出一个解 2 满足性问题的多项式时间算法。