中国科学院大学教材送审稿

计算复杂性导引

中国科学院大学网络空间安全学院

2022 年 2 月 21 日

目　录

第一章　引言 1

第一节　计算机与可计算理论 1

第二节　计算复杂性及应用 3

第二章　计算问题的算法实例 11

第一节　图论中问题与算法 12

第二节　逻辑中问题与算法 17

第三节　格中问题与算法 26

习题 27

第三章　计算模型 28

第一节　图灵机基础 28

第二节　多带图灵机、时间与空间 32

第三节　非确定图灵机 37

第四节　通用图灵机 42

第五节　递归语言与递归可枚举语言 43

第六节　不可判定性 45

习题 46

第四章　计算复杂类 49

第一节　复杂类 49

第二节　分离定理 51

第三节　可达性方法 53

习题 55

第五章　归约和完备性 57

第一节　归约 57

第二节　完备性 61

第三节　$\mathcal{NP}$ 关系 65

第四节　Oracle 图灵机 66

第五节　自归约 69

习题.....71
第六章　$\mathcal{NP}$-完备问题、$co\mathcal{NP}$.....73
第一节　搜索问题与判断问题.....73
第二节　$\mathcal{P}$与$\mathcal{NP}$、高效归约.....75
第三节　$\mathcal{NP}$-完备性.....76
第四节　$\mathcal{P}$与$\mathcal{NP}$续、$co\mathcal{NP}$.....80
第五节　$\mathcal{P}$与$\mathcal{NP}$的一般化：$\mathcal{P}$/*poly*与多项式时间分层.....82
第七章　概率算法.....85
第八章　交互证明与零知识证明.....87
第一节　基本概念.....87
第二节　交互证明系统与零知识证明.....88
第三节　$\mathcal{NP}$问题的零知识证明.....90
第九章　密码学的计算复杂性视角.....92
第一节　平均困难问题和单向函数.....92
第二节　随机串.....95
第三节　伪随机生成器.....98
第四节　密码应用与新进展.....99

第一章 引 言

第一节 计算机与可计算理论

每位能操作计算机的人都知道，我们让计算机执行某项任务其实就是，打开存储在计算机存储器中的应用程序。但是，十九世纪三四十年代的计算机却不是如此的。最早的计算机是大型电子数字“计算机”，如，英国的科洛萨斯（1943）与美国的埃尼阿克（1945），并没有存储程序，因此，如果让这些机器执行一项新的任务就需要修改其线路，通过手动装置修改电缆和开关。

计算机发展面临着很多基本问题，如，计算机的基本模型是什么？计算机能否计算一切事情？计算时间的有效性如何衡量？计算机能计算那些问题？等等。面对这些问题， 1935 年图灵提出新想法，即，在计算机存储器中存储一系列指令程序（符号或数字编码）来控制计算机器的功能。这就是图灵抽象的“通用计算机器”，简称为“通用图灵机”，1937 年丘奇（Church，代数和逻辑学家）将其称之为图灵机。1936 年，图灵在其论文“论可计算数在判定问题中的应用”中提出抽象模型“计算机器”。该文中的“可计算数”开拓了计算研究的新领域，即，现代计算机和不可计算性研究领域，这是现代计算机科学的奠基性著作。其另一重要贡献是证明，并非每个实数都是可计算的。由此可见，数学范畴超出了通用计算机领域，即，并非所有明确阐述的数学问题均能被图灵机解决，如，打印问题和停机问题就是典型的不可计算问题。所谓打印问题就是判定“一段程序在某阶段打印 0 后，余下的程序就不再打印 0”；而停机问题就是判定“一台图灵机从空白带开始执行是否会停机”。图灵利用打印问题证明，一阶命题函数演算是不可判定的，即，如果图灵机能够辨别出任一给出的命题能否被一阶命题函数演算所证明，那么图灵机就能够辨别出任一给出的图灵机是否曾经打印过 0。不可计算问题的存在导致了原有数学和哲学观点的混乱。

在二十世纪初，希尔伯特认为，数学应该具有完备性、一致性以及判定性的形式系统（公理体系），即，数学的整个思想内容具有同一性。这里，完备性是指每个真命题都可被证明；一致性是指所有真命题可以被统一证明；而判定性则是指有一个有效方法判别每一个数学命题。这就是著名的希尔伯特纲领（Hilbert's Programme）。1930 年代，Gödel 证明不完备性定理，即，任何一个公理体系，一定存在一个真命题不能被证明。这说明希尔伯特纲领是不正确的，但是该纲领的提出促使可计算理论（Computability Theory）的发展。

如果不可判定性失败了，那么从今天的意义上来讲，数学将不复存在；它的位置将被完备的机械规则所替代，如此以来，任何人都可以利用这个规则去判定某给定的命题能否被证明。

-------von Neumann,1927

对于计算有效性，Kolmogorov 建议由求解问题的最短程序的大小做度量。此外，人们给出很多非正式描述，这些描述都难以应用。但是，借助一台图灵机使其明确易用，即，定义其时间为图灵机关于输入的运行时间为输入长的多项式。这可以使我们将“探讨是否存在遍及数学和逻辑的有效方法”代替为“是否存在图灵机的程序”。既然任何图灵机能解决问题的方法都是有效的，图灵机的有效性就决定可计算的数是可数的，即，图灵机没有足够的程序去证明每个实数都可计算的。

图灵的思想很快被冯诺依曼（Von Nuemann）和纽曼（M.Newman）分别传到美国和英国。1945 年，美英两国的研究机构开始硬件实现通用图灵机。在这场研制电子存储程序式计算机的竞赛中，曼彻斯特大学胜出，于 1948 年 6 月 21 日纽曼的计算机器实验室研制出“曼彻斯特婴儿”并执行它的第一个程序。1951 年，电子存储程序式计算机开始上市，第一个上市的模型是曼彻斯特计算机，又被称为曼彻斯特马克 I，共计有九台被分别卖到英国、加拿大、荷兰和意大利，其中第一台机器于 1951 年 2 月被安装在曼彻斯特大学。美国也在同一年稍晚时期售出第一台通用电子计算机。1951 年第一台商用计算机问世，1953 年，基于冯诺依曼的原型 IAS 计算机，IBM701 诞生，这是首次公司大规模生产存储程序式计算机。由此，现代计算机的时代开启。但是随着现代计算机的产生，问题也随之产生。

什么是图灵机？在图灵写出“可计算数”的时代，计算机器根本不是指一台机器而是指人，是凭记忆和根据计算之前学到的一些“有效方法”进行计算的助手。恰如 Wittgenstein 所言，图灵的机器就是能够进行计算的人。按照图灵的描述，图灵机由存储器和扫描器两部分组成。其中，存储器由可无限存储符号的被划分为许多单元格的磁带组成，每个单元格存一个符号，如，0/1，或空白。扫描器则是，可在存储器中前后移动，每次扫描一个单元格，一个接一个地读取-写符号。除了擦除、写入、移动、停止等功能外，还可以改变状态，其本身有一个指针和一个刻度盘（简单的存储器），每个位置代表一个状态，随刻度盘上标识位置的改变而改变状态；它能够存储少量信息，以至于通过改变状态，机器可以记住先前的符号。

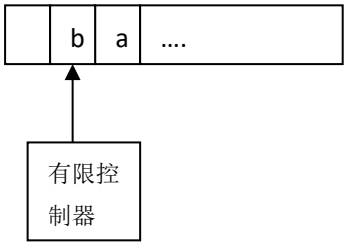

图 1-1 图灵机

图灵机是一种抽象的机器，因此我们无需为了让机器服从指令而制定一些特殊的机制。下面通过例子说明图灵机如何运行。

简单记这台机器为 M，在一条无限长空白磁带上开始工作。任务是在带上打印出 0□1□0□1□0□1…。我们需要解决的问题是，如何对机器设置使得，如果扫描器可以定位在磁带的任何一个单元格上，并可以进行移动，那么它可以在磁带上交叉打印出 010101…，并且从起始单元格向右移动，并在每两个数字之间留下一个空格。

为了执行此任务，我们定义四个分别标识为 a、b、c、d 的状态，令 R 表示指针向右移动，其中，当处于状态 a 时，表示 M 开始工作。具体执行如下：

1） a：表示 M 开始工作；若被扫描的是空白格□，则写 0 并向右移动，进入下一个状态 b。简写为(a, □)→(b, 0, R)。

2） b：若被扫描的是空白格□，则向右移动，进入下一个状态 c。简写为(b, □)→(c, □, R)。

3） c：若被扫描的是空白格□，则写 1 并向右移动，进入下一个状态 d。简写为(c, □)→(d, 1, R)。

4） d：若被扫描的是空白格□，则向右移动，进入状态 a。简写为(d, □)→(a, □, R)。

直观地，我们可以想象为扫描器有一组开关和插头，类似于一台老式电话交换台，通过特殊方式排列插头并调整开关，使机器按照如上指令执行。当然，随着交换台设定方式的不同，机器可以按照不同指令执行相应的操作。

既然类似交换台的装置可以使机器进入指令操作，我们可以设置一个单独固定的指令桌面，通过交换台进入机器内部，再根据指令桌面上的指令进行操作。由此，我们得到通用图灵机。通用图灵机能够执行原来机器在指令桌面上写下的任意指令。我们可以采取如下办法通过指令桌面把指定任务写入通用图灵机的磁带：桌面上的第一行首先占用磁带上的一些单元格，第二行接着第一行占据磁带的另一些单元格等，以此类推；然后，通用图灵机读取指令并在磁带上执行操作。因此，通用图灵机就是利用存储概念的数字计算机，这种想法是计算机发展的基础。Turing 同时证明无论哪种图灵机所能执行的计算，用上述方法，一台固定结构的单机都能够执行。于是，我们得到如下论断，虽然无法证明，但是被普遍接受。

Church-Turing 论题：通用图灵机可以执行任一人工计算者执行的计算。即，任何有效的或机械的方法均能被通用图灵机执行。

由此可见，所有合理的计算模型可以互相模拟，因而，计算可解问题的集合与计算模型无关。注意，“合理”计算模型没有准确的定义，因此，上述命题无法被证明。但是，普遍被认为合理计算模型具有三个最基本条件：1)计算一个函数只要有有限条指令；2)每条指令可由模型中的有限个计算步骤完成；3)执行指令的过程是确定的。到目前为止，人们认为合理的计算模型均被证明符合这些命题，因而得到广泛认可。

我们进一步也有如下广义 Church-Turing 命题：

广义 Church-Turing 命题： 对于任意两个合理的计算模型 $\boldsymbol{R}_1$ 和 $\boldsymbol{R}_2$，存在一个多

项式 p ,使得关于长为 n 的输入， $\boldsymbol{R}_1$ 的 t 步计算可以被 $\boldsymbol{R}_2$ 在 $p(t,n)$ 步内模拟。

但近年来广义 Church-Turing 命题受到挑战，主要因为量子图灵机的出现使很多在确定有效时间内没有找到算法的数论问题可在量子图灵机上找到有效求解算法。我们选取图灵机作为复杂性理论的计算模型，并以其基本的计算步数来衡量计算时间。由于其计算步数仅对局部变化有影响，因此选取它可以为我们带来方便。

第二节　计算问题

作为计算机科学的一个活跃领域，计算复杂性理论主要研究（数学）问题的内在难度，反映算法可行性及其所用合理资源的下界。其常用的资源包括计算所需要的时间、空间（用于存储）等，这也是计算机的基本资源。计算复杂性理论的一个主要目标是确定任何有定义的计算问题的复杂度，这是要求对特定计算问题给予明确回答；另一个主要目标是理解不同计算问题之间关系的（如，不同计算问题之间的复杂性比较）。有趣的是，目前计算复杂性理论在后一目标的研究成就更显著。可以认为，恰恰是解决确定的问题的失败导致了研究问题之间相互关系的方法的繁荣。事实上，从某种意义上说，建立问题之间的关系要比解决某类问题更有启发作用，如，**NP**-完备理论。目前计算复杂性理论虽然不能完全确定问题的内在复杂度，如，有效找到给定的（可满足）公式的成真赋值或者已知可 3-染色图的 3-染色，但是，我们可以证明这两个看似不同的计算问题是计算等价的。类似的复杂性理论的其它方面的问题也很有趣。下面我们先简单概述一下复杂性理论及其问题。

贯穿全书，对于任意正整数n，我们记$\log n = \log_2 n$。另外，除了特殊的说明外，本书中所有的图都是有限的有向图。令$\mathbb{N}$表示所有非负整数的集合，除非特殊说明外，在本书中假定所涉及的函数都是从$\mathbb{N}$到$\mathbb{N}$的函数，并且对于一般函数 f ，定义 $f(n) = \max\{\lceil f(n) \rceil, 0\}$。我们使用标准渐进符号来表示函数增长的阶：对于任何正实数函数$f(n)$和$g(n)$有

- $f = O(g)$当存在两个常量a, b使得对于所有$n > b$有$f(n) \leq a \cdot g(n)$。即，f 的增长不会比 g 快。
- $f = o(g)$，如果$\lim_{n\to\infty} f(n)/g(n) = 0$。
- $f = \Omega(g)$，如果$g = O(f)$。即， f 的增长不会比 g 慢。
- $f = \omega(g)$，如果$g = o(f)$。
- $f = \Theta(g)$，如果$f = O(g)$和$g = O(f)$。即， f 和 g 具有相同的增长比例。
- $f = \tilde{O}(g)$，如果对某常数 c 和所有充分大的 n，有 $f(n) \leq \log^c g(n) \cdot g(n)$。

例如，若 $p(n)$ 为 d 次多项式，则 $p(n) = O(n^d)$，即 $p(n)$ 的增长比例由第一个非零项决定。又 $p(n) = \Omega(n^d)$，从而 $p(n) = \Theta(n^d)$。若 $c > 1$ 是一个整数，则

$p(n)=O(c^n)$，但是 $p(n)\neq\Omega(c^n)$， $p(n)=o(c^n)$，即，任何多项式的增长严格慢于指数函数。

类似地，$\log n=O(n)$， $\log^k n=O(n)$，其中 k 为任意数。

称函数f是可忽略的，如果对于任何多项式$g(n)=n^c$有$f=o(1/g)$。

假设字母表Σ是符号的有限非空集，其中包含空符号$\sqcup$。称这些符号为字符，则$\sqcup$为空字符。字符串是由Σ产生的有限字符序列。我们称没有字符的串为空串，记为ε。字符串y的长度是字符串y中包含符号的个数，用$|y|$表示。显然，$|\varepsilon|$=0。所有Σ上的字符串的集合用Σ^*表示，所有长度为 n 的字符串集合用Σ^n表示。对于字符串组成集合 $L\subseteq(\Sigma\setminus\{\sqcup\})^*$，称 L 为语言。称由语言组成的族为语言类。由于每个问题实例都可以描述成字符串，因此语言即为问题，其中元素即为该问题的实例。

称图灵机$\mathcal{M}$在时间$t(n)$内运行，如果对于任何长度为 n 的输入字符串 w（基于某固定输入字母表Σ），$\mathcal{M}(n)$在至多$t(n)$步内停机。我们用在输入长度的多项式时间内停机的图灵机来区分有效计算的概念，即对于某些独立于 n 的常量a, b，图灵机在时间$t(n)=a+n^b$内运行。

判定问题是指判定输入的字符串是否满足特定的语言，即，是否是满足一定条件的问题实例。形式上，判定问题是要指定语言，即，字符串集合$L\subset\Sigma^*$，对于给定一个输入串$w\in\Sigma^*$，判断是否有$w\in L$。

P *vs* NP 问题　能被确定性图灵机在多项式时间内解决的判定问题类称为 **P**。能被非确定性图灵机在多项式时间内解决的判定问题称为 **NP**。显然，**P**⊆**NP**，但是普遍认为 **P** 与 **NP 不等**，即存在不能在确定多项式时间内解决的 **NP** 问题。

日常经验告诉我们解决一个问题要比判定解的正确性更困难（如，思考一个谜语或者做数独游戏）。这是巧合还是客观事实？我们能想象到解决问题和验证解答正确性的难度没有显著区别的世界吗？在这样一个假想的世界中，解决问题这个词语就失去它的意义。这个困惑的否定就是“**P≠NP**”，其中 **P** 代表能有效解决的问题，而 **NP** 代表解答能被有效验证的问题。

喜欢理论的学者也许会考虑证明定理和验证证明正确性之间的关系。实事上，找到证明是解决问题的一种特殊方法，而验证证明则对应于验证证明正确性。“**P≠NP**”意味着“证明定理”要比“验证定理证明的正确性”困难。此时，**NP** 代表能在适当证明（亦称为“证书”）帮助下被有效验证的断言的集合，而 **P** 代表能被无条件验证的断言集合。等价地，**NP** 问题可被表述为所有语言L的集合，满足存在关系$R\subseteq\Sigma^*\times\Sigma^*$使得$(x,y)\in R$在以|x|的多项式时间内检验，并且$x\in L$当且仅当存在满足$(x,y)\in R$的字符串 y。这样的字符串 y 称为L中 x 成员关系的 **NP** 证据或者 **NP** 证书。

P *vs* **NP** 问题似乎超越我们现有能力，但它导致了 **NP**-完备理论的发展。该理论的研究确定了一个难度等同于整个 **NP** 的计算问题集合，**P** *vs* **NP** 问题的发展与每一个这样问题相关，即，如果任何一个这样的问题容易求解，那么所有问题都在 **P** 中。称此

性质为 **NP**-完备性。因此，在“**P≠NP**”假定下，要证明一个问题是 **NP**-完备的就要提供其困难性证据，**NP**-完备性可以作为刻画计算问题困难性的一个中心工具。

另外，**NP**-完备性也说明所讨论问题的内容非常丰富，足以对 **NP** 中其它问题“编码表示”。一般来讲，复杂性理论是研究在问题描述中解答不明显的那类问题，即，这个问题包含所有必要信息，需要仅仅通过处理这些信息来获得答案。因此，复杂性理论也是关于信息处理的理论，是从一种表示(已知信息)到另一种表示(希望得到的信息)的转化。实事上，一个计算问题的解可以被视为已知信息的不同表示，而在该表示中答案是明确的。如，一个布尔公式是否是可满足的问题的答案隐含在公式本身里，求解的目的是让它明确。因此，复杂性理论就是通过表示来展示明确信息和隐含的信息之间的区别。此外，它还对信息明确程度给予量化。总之，复杂性理论为各种被前人思考过的问题提供了新视角，这不仅包括前面提及的证明和表示观点，也包括随机性、知识、交互、保密性等概念。

为了理解不同计算问题之间关系的（如，不同计算问题之间的复杂性比较），我们需要引入比较工具，称之为归约。

对于从$(\Sigma \setminus \{\sqcup\})^*$到$\Sigma^*$的函数$f$，称$\mathcal{M}$计算$f$，如果对于任意串$x \in (\Sigma \setminus \{\sqcup\})^*$，$\mathcal{M}(x) = f(x)$。令$A$和$B$是两个判定问题。从$A$到$B$的（Karp）归约是一个多项式时间可计算函数$f : \Sigma^* \to \Sigma^*$使得$x \in A$当且仅当$f(x) \in B$。显然，如果$A$可归约到$B$并且$B$可在多项式时间内解决，那么$A$也能在多项式时间内求解。

（判定）问题A是 **NP**-难的，如果任何其他 **NP** 问题B可归约到A。如果A也在 **NP** 中，那么称A为 **NP** 完备的。显然，如果问题A是 **NP**-难，除非 **NP**=**P**，否则A不能在多项式时间内解决。证明问题A是 **NP** 难的一个标准方法（因此A的多项时间解法不可能存在）是将其他 **NP**-难的问题B归约到A上。

另一个比较的工具是 Cook 归约。从A到B的 Cook 归约是一个多项式时间带 Oracle 的图灵机$\mathcal{M}$，接入以 B 的实例作为输入的 oracle。称$\mathcal{M}$归约A到 B，如果，已知正确解答B的 oracle，$\mathcal{M}$正确解决问题A。问题A称为 Cook 归约下 **NP**-难的，如果对于任何 **NP**-问题 B，存在 B 到A的一个 Cook 归约。如果 A 是在 **NP** 中的，那么我们称 A 是 Cook 归约下 **NP** 完备的。Cook 归约下的 **NP**-难度也给出了问题内在复杂性的证据，因为如果 A 能在多项式时间内被求解，那么 **NP**=**P**。后面，我们也将介绍其他复杂类和不同的归约概念，例如，随机复杂类或者非一致归约。

随机性、知识与交互系统　在复杂性理论中以随机性为中心（如，在伪随机性、概率证明系统和密码系统的研究中显而易见），人们会问，在各种应用中所需要的随机性能否在现实生活中获得？一个受到很多关注的具体问题是得到“纯”随机的可能性，即，我们能否用“有缺陷”随机性资源得到几乎完美的随机资源（从坏的资源中抽取好的随机性）。当然，答案依赖于有缺陷资源的模型。这项研究被证明与复杂性有关，与随机抽取器和伪随机发生器紧密关联。

随机性的概念曾长期困惑着人们，因为以前所描述的随机性观点是存在性的：人们会问“什么是随机的”并且想知道它是否存在于所有事情中(否则，这个世界是确定的)。

而复杂性理论的观点是行为性的，其实质是：如果对象不能被任何有效程序区分，就认为它们是等价的。如，如果预测掷币结果是不可能的，那么掷币是随机的（即，相信世界是非确定的）。在密码学中，如果我们有足够多的随机掷币，即，随机比特，那么一次一密的应用将使一切将变得相当容易。但是一次一密不适合应用，因为高度随机字符串难于生成，高度随机的比特产生过程非常慢，往往依赖于量子现象。密码学的解决方法是利用伪随机生成器代替，即，只要有足够随机性的字符串即可。

何谓一个足够随机的字符串？Kolmogorov 定义为：称一个 n 长字符串为随机的，如果任何描述长度<0.99n 且初始状态为空带的图灵机都无法输出该字符串。该定义在某些哲学和技术意义上来说是“正确的”定义，但是其在复杂性设置不是很有用。统计学家们也尝试着给出自己的定义，他们将定义归结为利用统计规律来检验一个字符串是否具有某个期望的的“正确的次数”， 如，11100 次可以看作一个子字符串。结果是这种定义对密码学设置来说太弱了，事实上，我们可以找到一个可以满足该统计检验的分布，但该分布用于产生一次一密中的填充则在密码学中会导致不安全。

在 1980 年代初期 Blum-Micali 和 Yao 分别给出的复杂性理论上的伪随机性的定义。对于字符串 $y\in\{0,1\}^n$，我们用 $y|_{[1,\ldots,i]}$表示 y 的前 i 比特。Blum-Micali 的定义来自观察：对于一个统计随机的字符串 y，如果给定 $y|_{[1,\ldots,i]}$，那么无论具有多么强大的计算能力，我们都无法以优于 1/2 的概率要预测 y_{i+1}。因此我们可以通过考虑一个预测器来定义“伪随机字符串”，该预测器具有有限的计算能力并且无法从 $y|_{[1,\ldots,i]}$中以优于 1/2 的概率要预测 y_{i+1}。即，比特是不可预测的。该定义是具有缺点的，因为当任意的单个有限字符串与一个作为预测机的图灵机相连时其可以被预测。为了克服该缺点，Blum-Micali 定义字符串分布的伪随机性，而不是定义单个字符的伪随机性。即，该定义关注于分布的无限个序列，每个对应一个输入长度。如果我们允许检验器为任意的多项式时间的图灵机将使得在密码学中只限制敌手为有界的计算能力的假设具有意义。

复杂性理论早期提出的伪随机生成器，如，线性或者二次同余数生成器是不满足 Blum-Micali 的伪随机性定义的，因为它们的比特预测可以在多项式时间内实现。

Yao 给出了另一个定义。该定义允许检验图灵机一次访问整个字符串，给出的随机性的检验器与人工智能中的图灵检验器类似。检验图灵机从两种渠道得到一个字符串 y，即，其或为一致随机分布的字符串，或为将某个短随机字符串作为某确定性的函数 G 的输入产生的字符串。如果检验图灵机认为字符串看起来是随机的，那么就输出“1”，否则输出“0”。我们称 G 为一个伪随机生成器，如果没有多项式时间的检验图灵机具有很大优势判定这两个分别利用某种方法产生的。

在不考虑安全参数的微小变化的情形下上述两定义是等价的。在应用中，依据 Blum-Micali 的定义设计伪随机生成器似乎更容易；而 Yao 的定义在应用中更有利于为我们所要证明和所需要的命题之间建立了联系，因为其允许敌手以不受限的方式访问伪随机字符串。

伪随机数生成器是否存在？令人奇怪的是，伪随机生成器存在当且仅当单向函数存在。这表明，随机性与难解（困难）性之间有着紧密的关系。我们目前具有一些令人满意的候选的陷门单向函数，这个结论可以帮助我们设计伪随机生成器。如果伪随机生成器被证明是不安全的，那么候选的陷门单向函数事实上为非单向的，因此我们可以得到对其求逆的有效算法。

复杂性理论提供了知识（不同于信息）的概念。它把知识视为一个难以计算的结果，因此一切能被有效实现的对象都不能被认为是知识。如，应用于公共信息的难于计算的函数值是知识，而简单计算结果则不被认为是知识。特别地，如果有人给予你这样函数值，那么他给了你知识。这就又涉及到一个新的概念，即，零知识交互系统（没有增加知识的交互系统）。这样的交互是有用的，因为它可以使对方相信事先提供的特定知识的正确性。因此，对于 NP-完备问题而言，证书就是知识，而且每个 NP-完备问题都可以用于构造零知识交互系统。

交互的动机之一是获得知识。事实表明，交互在很多背景下是有用的。如，当与一个证明者交流时，验证一个断言的正确性要比得到证明容易得多。这样的交互证明的能力源于其随机性（即，验证程序是随机的）。如果验证者的问题能被提前确定，那么证明者只需在交互中提供证明的复制品就可以。另一个关于知识的概念是保密性。知识是一部分人知道而另一部分人不知道的东西（至少仅靠自己不容易得到），因此，在某种意义上讲，知识就是秘密。

密码学　一般而言，理想的密码函数是可以抵抗任意的恶意攻击的系统。密码学中典型的问题： 通信双方通信的内容不被非授权用户获取，即，秘密通信。这个目标可以达到，如果双方同享一个随机密钥并且只有他们知道。发送者用密钥对欲发送的信息编码，再通过公共网络传递给接受者。接受者收到信息后，通过用共享的密钥进行译码得到原始信息。原始信息、加密后信息、编码、译码通常称为原文、密文、加密、解密。一个加密模式是安全，如在没有密钥情况下，从密文中获取原文是计算不可行的。因此，窃取者从密文中不会获得任何信息，除了知道密文长度、双方通信（由于技术原因，在不严重影响通信效率下隐藏发送信息长度是不可能的。鉴于效率和安全的因素，泄露信息长度是可以接受的）。密码学最重要的任务是设计和分析加密方案或密码函数。一个首要任务是对“安全性”和“敌手”这两个直观的术语给出正确的数学定义。目前已知有两种定义安全性的方法。

一是香农在 1949 年给出的信息论方法。它与密文中有关明文的信息有关。通俗地说，如果密文中不存在有关明文的信息，则加密方案是安全的。在该理论中，香农表明，假设敌手拥有无限的计算资源，只有当使用的密钥至少与使用加密系统要交换的信息一样大时，安全的加密系统才存在。

二是基于计算复杂性的现代方法。它放弃了敌手拥有无限的计算资源的假设，并假设敌手的计算以某种合理的方式有界，通常假设敌手有概率多项式计算能力。在这种情况下，重要的问题不是密文中是否存在明文信息，而是该信息是否可以被有效计算出来。此时，安全性是基于为合法用户保证的有效算法与为敌手寻找信息的计算不可行性之间的差距。设计这样的系统需要存在具有某种计算困难性质的计算工具，其中最基本的是单向函数。

单向函数是密码学中基础概念。称一个函数 f 是单向的，如果从其定义域任意选取一个元素 x，$f(x)$是容易求出，但是求 f 逆是困难的，即，一个容易计算但很难求逆的函数。单向函数在密码学中有非常多的应用，如数字签名、伪随机生成器、承诺方案等等。这些的安全性都是在平均困难的假定下定义的。

例 1.1（大数分解问题）对于整数$n \in \mathbb{N}$，目前用于分解n最佳算法的运行时间为$2^{\mathcal{O}(\sqrt{\log p \log\log p})}$，其中 p 是 $n = pq$ 中的第二大素因子。因此我们可以推测函数 $f_{mult}(p,q) = n$ 是单向的。假设大数分解的复杂性和使用素数的密度，可以构造一个基于

f_{mult} 的单向函数。如，RSA 或 RABIN-SQUARE 都与整数分解有关。

例 1.2（离散对数）设p为素数，g 为乘法群$\mathbb{Z}_p^*$的生成元。定义函数$\mathrm{EXP}_{g,p}$为$\mathrm{EXP}_{g,p}: x \to (g^x \bmod p)$，称计算$\mathrm{EXP}_{g,p}$的逆的问题为离散对数问题(DLP)。目前已知的最快的随机算法在次指数运行时间内工作。我们认为函数$\mathrm{EXP}_{g,p}: x \to (g^x \bmod p)$是单向的。与 DLP 相关的一个有趣问题是找到一种算法，该算法将生成一个素数 p 和一个$\mathbb{Z}_p^*$的生成元 g。对此，目前还不知道确定性多项式时间算法，只有期望多项式运行时间的随机算法是已知的。著名的 ElGamal 密码系统、DDH 问题等与 DLP 密切相关。

在密码学中平均困难的定义即是，对一个加密方案，若其密钥是随机选取，则不存在多项式概率算法以不可忽略的概率攻破该体制。这不同于计算复杂性里面的最坏情况，如，**NP**-完备性。证明一个问题是 **NP**-难的需要说明：不存在一个多项式算法能正确解决该类所有问题，除非 **NP**=**P**。换言之，任意的多项式运行的程序，对该类问题的某些例子都会给出错误的回答。通常地，我们在设计一个加密方案时，如果密钥空间中有不可忽略的密钥不安全，则认为该加密体制不安全。因此，在密码学中我们往往采用的问题是，对任意的多项式概率算法，只能以可忽略的概率成功解决。

形式上，称一个函数是可忽略的，如果对所有足够大的 n，该函数都小于任意的$1/n^c$。类似地，称一个加密方案是安全的，如果，对任意的$1/n^c$和任意的多项式概率算法，存在n_0，所有大于n_0的 n，破译该方案的概率小于$1/n^c$，这里 n 为方案的安全参数。

现代密码学的终极目的是构造可证明安全的密码函数，即，在平均情况下很难攻破。遗憾的是，根据当前计算复杂性的研究结果，暂没有这样的构造。如果 P=NP，则大部分的密码问题将会可解，因为攻击者可以非确定的猜测密钥。由此可见，一个无条件安全的密钥必要条件是 **P≠NP**。在现阶段，我们最满意的目的是，在某些困难假设下，构造可证明安全的密码函数。如，假设不存在多项式算法，随意输入整数 n，输出 n 的所有因子。我们在此假设下，可以构造一个安全的加密算法。但是，目前还没有一个基于大数分解问题的密码方案，这里要求难分解不仅是在最坏情况下，而且也在平均情况下。这个情况对用在密码学中的其他数学难问题也一样存在。

最近，格受到了很多关注，被认为在密码学中有巨大的潜力。格之所以受到众多关注，关键点是把其平均情况和最坏情况联系起来了（Ajtai，1996）。在这篇突破性文章中，Ajtal 证明，对任意格，近似因子为$\gamma(n) = n^c$，如果没有算法可以近似解决（判断）最短向量问题（SVP），则从某些样本空间里随机选取一个格，解决其 SVP 是难的，其中这个样本空间是容易找到。基于这个结果，Ajtai 提出了一个基于格的单向函数。

Ajtai 发现格问题平均情况下和最坏情况下的联系后，许多科研者建议利用格去解决密码学中一些问题，如，抗碰撞哈希函数和公钥体制。构造抗碰撞哈希函数类似于 Ajtai 的单向函数，很好说明如何利用格去构造密码函数且难度等价于解决格的某些近似问题。应当注意，基于某种数学难题构造的密码函数，如果攻破此密码函数与攻破该数学难问题最坏情况难度类似，则是非常好的。对于格问题，已知的所有近似算法（如 LLL 算法）在解决平均难问题上比解决最坏情况更加有效。因此，我们可以合理假设，没有多项式概率算法能近似解决在最坏情况下的格问题，且近似因子很小。

值得注意是，有些假设是不合理的，如，当随机从一个样本空间里选取一个格，

不存在多项式概率算法以不可忽略的概率解决。原因在于，选取能否成功很大程度上取决于样本分布。但是，我们不知道有算法能近似解决格最坏情况下问题，其近似因子不大于$\sqrt{n}$，因此我们可以假设没有多项式概率算法能近似解决在最坏情况下的格问题，且近似因子很小。然而，如果我们考虑随机选取的格，那么作为输入的 n 个不相关的基向量是随机选取，有很大的概率到这些基向量距离最短的向量在近似因子$\sqrt{n}$内。针对这种情况，随机输入一个格，平均而言是能近似解决其 SVP 问题。Ajtal 所研究结果令人惊叹的是，针对格，他提供了一个详细的概率分布，使得从这个分布里随机抽取一个格，基于假设解决某些格问题最坏情况下是没有有效算法，即，可证明解决其是困难的。

综上，复杂性理论与密码学密切相关。密码学是研究“易于使用但难以破坏”的系统理论。这样的系统理论的典型性质不仅包含保密性、随机性和交互性，也包括与“易于正确使用而难以使系统偏离规则”的行为相关的复杂性差距。因此，密码更多地是基于复杂性理论假设得到一个典型变换结果，该变换将相对简单的计算基础工具（如，单向函数）转变为更复杂的密码系统(如，安全加密方案)。过去几十年里，密码学的最大进展是将密码学建立在计算复杂性的基础上，也恰是计算复杂性理论在密码学中的应用促成了密码学的飞跃，使密码学从一门艺术发展为一门严格的科学。二者相互交织，相互促进，恰如 S.Goldwasser 所说，“Cryptology and Complexity Theory: A Match Made in Heaven.(密码学和复杂性理论：天作之合)”。

计算解的数目和近似解　前面我们已经提及复杂性理论的两种基本类型的任务“搜索求解（或找到解）”和“做出判定（如，断言的真假判定）”，也说明在某些场合二者是相关的。现在我们考虑另外两种类型的任务：计算解的数目和近似求解。这两个问题都与寻找相应问题的任意解一样困难，但事实表明，对于某些问题而言它们实质上不会更难，甚至在某些关于问题的自然条件下，近似计算解的数目和生成一个近似随机解不会比寻找任意解更难。

寻找近似解的复杂性研究已经得到很多关注。一类近似问题是定义在可能解集合上的目标函数，除了寻找达到最优值的解，近似任务还包括寻找“几乎最优”解，其中“几乎最优”的概念可以理解为以不同方式产生不同近似程度的解。有许多例子表明，找到近似解与找到最优解一样难；在其它一些情况，一定程度的合理近似比解决原始（精确）搜索问题容易。令人惊奇的是，近似结果的困难性与概率可检验证明相关，这是一种允许存在快速概率验证的证明；每一个证明都能被有效转变为一个允许基于常数多个比特探测的概率验证的证明。

显然，近似是多种计算问题要求的自然放宽。另一种自然放宽是平均复杂性的研究。尽管未明确说明，前面讨论的问题都涉及对算法的最坏情况分析，但是，我们认为平均复杂性是比最坏情况复杂性更强的概念，但它的发展远不如最坏情况复杂性。1996 年，Ajtai 等人开创性地给出了格中困难问题最坏情况到平均情况的归约证明，这一工作是格复杂性领域中的重大突破，打破了 **NP**-问题的困难性只是针对最坏情况的限制；并于 1997 年基于最坏情况下格问题构造出密码系统，得到安全性依赖于格中最难问题的密码体制，使得格密码具有了可证明安全的性质。

Promise 承诺问题是互不相交语言对(Π_{YES},Π_{NO})，即，$\Pi_{YES},\Pi_{NO}\in\Sigma^*$并且$\Pi_{YES}\cap\Pi_{NO}=\emptyset$。称算法解决承诺问题$(\Pi_{YES},\Pi_{NO})$，如果输入实例

$I \in \Pi_{YES} \cup \Pi_{NO}$，它能正确判定$I \in \Pi_{YES}$或者$I \in \Pi_{NO}$。在$I \notin \Pi_{YES} \cup \Pi_{NO}$（即当$I$不满足承诺时）并未指定，即对于超出承诺的实例，算法允许返回任何回答。

判定问题是承诺问题的一个特例，其中集合$\Pi_{NO} = \Sigma^* \setminus \Pi_{YES}$是没有明确指定的并且承诺$I \in \Pi_{YES} \cup \Pi_{NO}$是显然为真。格近似问题的计算复杂性可以很容易地用承诺问题阐述。这些是判定问题适用于研究近似困难性的推广。下面说明格的 Promise（承诺）问题。

令$\mathbb{R}^m$是m - 维欧式空间，给定n个线性无关向量$\mathbf{b}_1, \ldots, \mathbf{b}_n$，$\mathbb{R}^m$（$m > n$）生成的格定义为$\mathcal{L} = \{\sum_{i=1}^{n} x_i \mathbf{b}_i | x_i \in \mathbb{Z}, 1 \le i \le n\}$. 这里$\mathbf{b}_1, \ldots, \mathbf{b}_n$称为格基，我们可以把基向量作为列表示成$m \times n$ 矩阵的形式为$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$,由$\mathbf{B}$生成的格可以写为$\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\}$,我们称$m$是格的维数，$n$是格的秩。若$m = n$,则这个格称为满秩格或满维格。格$\mathcal{L}(\mathbf{B}) \subseteq \mathbb{R}^m$ 是满秩的当且仅当由基向量张成的线性扩张$\mathrm{span}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{R}^n\}$等于整个空间$\mathbb{R}^m$。

格$\mathcal{L}$中第一个连续最小量是格中非零最短格向量的长度，记为$\lambda_1(\mathcal{L})$，即$\lambda_1(\mathcal{L}) = \min\{\|\mathbf{x}\| : \mathbf{x} \in \mathcal{L} \setminus \{0\}\} = \min_{\mathbf{x} \ne \mathbf{y} \in \mathcal{L}} \|\mathbf{x} - \mathbf{y}\|$. 格$\mathcal{L}$中第$i$个连续最小量$\lambda_i(\mathcal{L})$是以原点为圆心中包含$i$个线性无关格向量的最小的球半径，即$\lambda_i(\mathcal{L}) = \min\{r : \dim(\mathcal{L} \cap \mathcal{B}(0, r)) \ge i\}$，其中$\mathcal{B}(0, r)$是以原点$0$为圆心半径为$r$的开球。

最短向量问题（SVP）即，给定一个格$\mathcal{L}$，找到一个非零格向量$\boldsymbol{v}$满足$\|\boldsymbol{v}\| \le \gamma \cdot \lambda_1(\mathbf{L})$。最近向量问题（CVP）即，给定一个格$\mathcal{L}$和一个目标向量$\mathbf{t}$，找到一个格向量 $\boldsymbol{v}$满足$\mathrm{dist}(\boldsymbol{v}, \mathbf{t}) \le \gamma \cdot \mathrm{dist}(\mathcal{L}, \mathbf{t})$。我们现在定义与近似 SVP 和 CVP 相关的承诺问题。用$GAPSVP_\gamma$和$GAPCVP_\gamma$表示。

承诺问题$GAPSVP_\gamma$，其中γ（差距函数）是秩的函数，定义如下：

- YES 实例是这样的对$(\mathbf{B}, r)$，其中$\mathbf{B} \in \mathbb{Z}^{m \times n}$是格基和$r \in \mathbb{Q}$为有理数使得对于某$\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$，有$\|\mathbf{Bz}\| \le r$。
- NO 实例是这样的对$(\mathbf{B}, r)$，其中$\mathbf{B} \in \mathbb{Z}^{m \times n}$是格基和$r \in \mathbb{Q}$为有理数使得 对于所有$\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$都有$\|\mathbf{Bz}\| > \gamma r$。

承诺问题$GAPCVP_\gamma$，其中近似因子γ（差距函数）是向量阶的函数，定义如下：

- YES 实例是三元组$(\mathbf{B}, \mathbf{t}, r)$，其中$\mathbf{B} \in \mathbb{Z}^{m \times n}$是格基，$\mathbf{t}$是一个向量以及$r \in \mathbb{Q}$为有理数使得对于某$\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$，有$\|\mathbf{Bz} - \mathbf{t}\| \le r$。
- NO 实例是这样的对$(\mathbf{B}, \mathbf{t}, r)$，其中$\mathbf{B} \in \mathbb{Z}^{m \times n}$是格基，$\mathbf{t}$是一个向量以及$r \in \mathbb{Q}$为有理数使得对于所有$\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$都有$\|\mathbf{Bz} - \mathbf{t}\| > \gamma r$。

注意到当近似因子$\gamma = 1$时，$GAPSVP_\gamma$和$GAPCVP_\gamma$等价于精确 SVP 和 CVP 的判定问题。这里符号有点混淆，我们考虑实例$(\mathbf{B}, r)$（或者$(\mathbf{B}, \mathbf{t}, r)$）其中 r 是一个实数，如$r = \sqrt{2}$。这不是一个实际问题，因为 r 总能被合适的有理逼近所取代，如在ℓ_2范

数中，如果 **B** 是整数格，那么 r 可以被区间$[r, \sqrt{r^2+1})$中的任何有理数所代替。承诺问题$GAPSVP_\gamma$和$GAPCVP_\gamma$在下面的意义下捕获了因子为γ的近似SVP和CVP的计算任务。

假定算法$\mathcal{A}$近似解决因子为γ的 SVP，即对于输入的一个格Λ，找到向量$\mathbf{x} \in \Lambda$使得$\|\mathbf{x}\| \leq \gamma\lambda_1(\Lambda)$。那么算法$\mathcal{A}$可以用来解决$GAPSVP_\gamma$如下：对于输入$(\mathbf{B}, r)$，在格$\mathcal{L}(\mathbf{B})$上运行算法$\mathcal{A}$来获得最短向量长度的估计$r' = \|\mathbf{x}\| \in [\lambda_1, \gamma\lambda_1]$。 如果$r' > \gamma r$，那么$\lambda_1 > r$，即，$(\mathbf{B}, r)$不是一个 YES 实例。因为$(\mathbf{B}, r) \in \Pi_{YES} \cup \Pi_{NO}$，所以$(\mathbf{B}, r)$必是一个 NO 实例。反之，如果$r' < \gamma r$，那么$\lambda_1 < \gamma r$并且由承诺$(\mathbf{B}, r) \in \Pi_{YES} \cup \Pi_{NO}$，可以得出$(\mathbf{B}, r)$是一个 YES 实例。另一方面，假定有一个判定 oracle $\mathcal{A}$能解决$GAPSVP_\gamma$。（根据定义，当输入不满足承诺时，oracle 返回任何答案。）令$u \in \mathbb{Z}$是$\lambda^2(\mathbf{B})$的上界（如，令 u 是任何一个基向量长度的平方）。注意到$\mathcal{A}(\mathbf{B}, \sqrt{u})$总是返回 YES，同时$\mathcal{A}(\mathbf{B}, 0)$总是返回 NO。使用二分搜索可找到一个整数$r \in \{0, \ldots, u\}$使得$\mathcal{A}(\mathbf{B}, \sqrt{r})$ =YES 并且$\mathcal{A}(\mathbf{B}, \sqrt{r-1})$ =NO。那么$\lambda_1(\mathbf{B})$必须位于区间$[\sqrt{r}, \gamma\sqrt{r})$。对于最近向量问题有相同的讨论。

NP 类很容易被扩展到包含承诺问题。我们说承诺问题(Π_{YES}, Π_{NO})是在 **NP** 中的，如果存在一个关系$R \subset \Sigma^* \times \Sigma^*$使得$(x, y) \in R$能在以|x|为变量的多项式时间内判定并且对于任何$x \in \Pi_{YES}$，存在一个 y 使得$(x, y) \in R$，而对于任何$x \in \Pi_{NO}$，不存在一个 y 使得$(x, y) \in R$。如果 x 不满足承诺时，R 可能包含也可能不包含对(x,y)。

承诺问题(Π_{YES}, Π_{NO})的补问题是承诺问题(Π_{NO}, Π_{YES})。对于判定问题，这与在Σ^*中取一个语言的补集相同。补集在 NP 中的判定问题类表示为 coNP。同样 coNP 也可以扩展为包含所有 NP 承诺问题的补。

承诺问题之间的归约以直接的方式定义。函数$f : \Sigma^* \to \Sigma^*$是从$(\Pi_{YES}, \Pi_{NO})$到$(\Pi'_{YES}, \Pi'_{NO})$的归约，如果它把 YES 实例映射到 YES 实例，NO 实例映射到 NO 实例，即$f(\Pi_{YES}) \subseteq \Pi'_{YES}$和$f(\Pi_{NO}) \subseteq \Pi'_{NO}$。显然，任何可以解决$(\Pi'_{YES}, \Pi'_{NO})$的算法$\mathcal{A}$也能用来解决$(\Pi_{YES}, \Pi_{NO})$如下：

对于输入$I \in \Pi_{YES} \cup \Pi_{NO}$，运行算法$\mathcal{A}$于$f(I)$并输出结果。注意到$f(I)$总是满足承诺$f(I) \in (\Pi'_{YES}, \Pi'_{NO})$并且$f(I)$是一个 YES 实例当且仅当 I 是一个 YES 实例。称一个承诺问题 A 是 **NP**-难的，如果任何 **NP** 语言 B（或者更一般地将任何承诺问题）可以有效归约到 A。通常，证明一个承诺问题是 NP 难的要说明除非 NP=P，否则这个问题的多项式时间算法不存在。以 Cook 归约为例，在已知任何解决问题(Π'_{YES}, Π'_{NO})的 oracle 下，解决问题(Π_{YES}, Π_{NO})的带 oracle 图灵机$\mathcal{A}$也能工作。特别地，不管承诺之外的问题 oracle 如何回到，$\mathcal{A}$也能正常工作。

目前为止，我们一直关注计算问题的时间复杂度，并用时间刻画效率。但是，如前面所述，时间不是我们关注的唯一资源。另一个重要资源是空间：计算所消耗的（暂时）存储量。空间复杂性的研究也让人们发现了不同于时间复杂性的性质。例如，在空间复杂性的背景下，验证（任意指定）断言成真的证明与验证同类断言的非真的证明的复杂度是相同的。

后面我们将从算法开始逐步详细地介绍复杂性理论的基本内容，但我们将不涉及

近似计算（求解）、随机抽取器、伪随机发生器以及密码学的内容，有兴趣的读者可以参考相关文献中的内容。

说明：本讲义主要讨论问题的计算复杂度，即，时间和空间资源的消耗，所用算法仅用于对资源占用的估计，因此对于算法叙述更侧重于思想，非严谨论述和分析。

第二章 计算问题的算法实例

计算复杂性理论与算法的设计与分析是计算机科学的两个领域，都是极力估计算法复杂度，衡量在现实资源要求下能做什么和不能做什么，但它们是从两个相反的方向研究算法问题的。算法是一个详细的逐步解决问题的方法。自然地，证明一个问题有有效可解的算法是一个更受欢迎结果，因为一个有效算法不仅可以直接求解问题，也是该问题有效可解的一个证明；然而，每个问题都有确定的算法复杂度。复杂性理论就是要弄清楚求解问题所需要的最小资源，证明求解困难问题不能有更节省资源的办法，这是问题本身的固有难度。

当设计一个算法时，我们只需要形成算法并分析，这将得到求解问题所需要的极小资源上界； 而复杂性理论则是提供下界，即，每个求解该问题的算法都必需的资源下界。对于上界的证明，设计和分析一个有效算法是足够的，而每个下界则是要研究求解一个特定问题的所有算法。对于这些算法组成的集合，我们能够掌握的性质就是它们能够求解该问题。因此，为了证明一个特定问题的求解要求一定的极小资源，必须被考虑到关于该问题的所有算法，这恰是计算复杂性的难点所在。

当然，二者是相互联系，相辅相成的。当证明一个问题没有有效解时，我们会发现问题的一些本质，通常能够揭示问题的困难性所在，有助于得到问题困难性证明的突破口。此外，当我们发现所研究的问题不是有效可解的时候，首先得到的好处是“放弃寻找该问题的有效算法”，避免浪费精力于一个不可达到的目标（如，理智的人不可能去设计永动机），其次，通过加强限制条件等，可以找到有效可解的问题（所谓加强限制条件是指问题的特殊化，或要求更弱的解形式）。因此，“证明一个问题不是有效可解的”与“设计有效算法”一样都有意义，这便于人们设计更高效的算法和评价某个算法的优劣。

什么是问题呢？一般性的问题的定义太宽泛，难以形式化。为了能够研究更多可处理的问题，我们仅考虑需要解决且有价值的计算问题。这些问题可以通过计算机处理，且其正确结果的集合是确定的。但是，就复杂性理论而言，并不是所有计算机可处理的问题都是计算问题，计算问题具体定义如下：

定义2.1 一个计算问题如下组成：

1） 对可允许的输入集合的描述：在计算机的有限字母表上，每个输入可以表示为一个字符串；

2） 对映射每个输入到正确输出（回答或结果）的函数的描述：每个函数亦是一个有限字母表上的有限序列。

由定义可见，一个问题不是由单个的询问构成，而是一族要被回答的询问构成。对于一个给定的问题，这些询问都是相关的，而且有一个简单的“填空”式结构。每个输入就是一个实例。对于不同的询问，每个不同的输入被填入空格，如下列的形式：

实例: 一个正整数 n。

问题: n 是否为素数？

对于问题的回答，则需要寻找算法。尽管精确的形式化不重要，但是通常问题与算法都需要分别作为语言和图灵机（定义见第三章）被形式化并被分析。根据实际可解性的直观意义，多项式时间可计算性是计算问题重要的理想性质。另外，亦存在一些问题，不管效率如何，它们没有求解算法。

根据对问题回答的不同要求，问题可分为以下几类：

（1）搜索性问题：对于给定的问题，找到一个可能的答案。如例 1.1，对于图的可达性问题，则是给定两点 u.v，找到 u.v 间的一条通路。

（2）最优化问题：对于给定的问题，找出最佳的解答。如：上例中找出从 u 到 v 的最短路径。若仅要求一个最优解的值，则称问题为最值问题。

（3）验证问题：对于给定的问题及一个解，验证其是否为该问题的解。如上例中，给定一条通路，验证是否是最短的。

（4）判定性问题：给定一个问题，判定是否有解。如上例中，判定是否有从 u 到 v 的一条通路。

如图 2-1 中，有向图 $G=(V,E)$ 顶点 $V=\{1,2,3,4,5\}$，边为 $E=\{(1,4),(4,3),(2,1),(2,3),(5,4),(3,5)\}$。于是，对于图 2-1，对于 u.v∈$V$，上述四类问题具体为

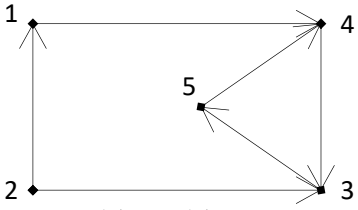

图 2-1 图 G

1) 搜索性问题：给定两顶点 u.v，找到 u.v 间的一条通路。

2) 最优化问题：找出从 u 到 v 的最短路径。若仅要求一个最优解的值，则称问题为最值问题。

3) 验证问题：对给定 u 到 v 的一条通路，验证是否是最短的。

4) 判定性问题：判定是否有从 u 到 v 的一条通路。

显然，搜索性问题比判定性问题更难，而判定性问题比验证性问题更难。即，

$$\text{搜索性问题} \xrightarrow{\text{求解}} \text{判定性问题} \xrightarrow{\text{求解}} \text{验证性问题。}$$

对于最优化问题，多数情况下可以利用判定性问题求解。

第一节 图论中问题与算法

例 2.1 图的可达性(GRAGH REACHABILITY).

一个图 $G=(V,E)$ 由一个顶点集 V 和边集 E 组成，其中 $V=\{1,2,\ldots,n\}$，$E=\{(i,j):i,j\in V\}$ 均为有限集。图有许多计算问题，但最基本即是图的可达性（GRAGH REACHABILITY）：给定一个图 G 和两个顶点 $1,n\in V$，是否有一条从 1 到 n 的通路（Path）？称此问题为可达性问题，记为 REACHABILITY。

如大多数有趣的问题一样，REACHABILITY 有如下特点：

①无限多个可能的实例。

②每个实例是一个数学对象，这里对象是一个希望得到回答的具体问题，并，其回答的是 yes 或 no，即是否有通路。

③它是判定性问题。

在本书中判定性问题发挥重要的作用。为了统一和简便，我们只考虑判定性问题。

下面做为例子，我们给出 REACHABILITY 的一个求解问题的算法，在后面引入图灵机的概念后，我们会给出正式的描述。对于 REACHABILITY，我们有如下搜索算法，整个算法中，我们保存一个顶点集合 S：

初始，$S=\{1\}$，每个顶点或者已被标记或者未被标记，这里点 i 被标记意味着在过去的某一时刻曾在 S 中或者目前正在 S 中。因此，初始只有 1 被标记。

重复执行如下：

从 S 中选择一个顶点 $i\in S$，并且 $S\leftarrow S\setminus\{i\}$，然后找出从 i 出发的所有的边 $E_i=\{(i,j):(i,j)\in E\}$，逐条检查。如果 j 未被标记，则标记，并且 $S\leftarrow j$；

如此下去直到 $S=\varnothing$ 停止。此时，若顶点 n 被标记，则输出“yes”，否则，输出“no”。

显然，一个顶点被标记当且仅当存在从 1 到它的通路，因此该算法可以解决 REACHABILITY。自然地有一个问题，即，如何将图表示为算法的输入？不同的模型可以选择不同的表示，但是后面将会发现，不同表示的选择并不是关键。因此，这里我们可以设图是由邻接矩阵表示，而且矩阵中所有赋值可由算法随机访问。

另外，该算法对于“如何选取 S 中的元素 i”是模糊的。比如，若我们总是选择 S 中停留时间最长的点，则所进行的搜索是宽度优先，而且可找到最短的通路；若 S 被视作堆栈处，则得到深度优先的搜索。因此，不同的顶点选择方式将导致不同的搜索方式，但是算法对于这些选则都是有效的。

我们需要重点研究的是算法的效率。以邻接矩阵表示为例，当对应于行的顶点被选择时，邻接矩阵的每个赋值仅被访问一次。此时，需要花费大约 n^2 次操作去处理从选择的顶点出发的边（注意：至多有 n^2 条边）。若假定其它的操作，如从 S 中选择元素、标记一个顶点以及判断一个顶点是否被标记等，可在常数时间内完成，则该搜索算法至多可在与 n^2 成比例的时间内判定两个顶点是否连通。于是，该算法所用的时间资源为 $O(n^2)$。

下面我们分析其所用的空间资源。首先，需要贮存集合 S 和顶点的标记。显然，

$|S| \le n$，即至多有 n 个标记，因此，该算法需要用 $O(n)$ 个空间。注意，对于空间考虑，我们的要求要比时间更严格，后面我们会看到，如果我们首先利用深度优先法，再利用宽度优先法，可使所用空间资源降为 $O(\log^2 n)$。

在问题的研究中，确定我们满意的资源的标准是很重要的，即确定算法运行时间的增长率。在本书中，我们将关于输入长的多项式时间视为可接受的时间，或者作为一个标记，称有这样算法的问题为有效可解的，而所用算法为多项式算法。相反，当时间超越多项式甚至指数时，如 2^n 或 $n!$，我们不能在多项式时间内解决该问题，称该问题为困难的。

值得注意的是，不能将多项式时间算法等同于实际可行性算法，因为存在有效地计算不是多项式时间的，而且多项式时间亦不一定实用，特别地，当 n 很小的时候，如 n^{80}，$2^{\frac{n}{100}}$。既然如此，为何还要用多项式时间作为有效计算的标准呢？在理论研究中，主要原因有以下两方面：

1） 除了有限的具体实例外，任何多项式的增长率要小于指数。

2） 我们考虑的是问题的最坏情形。也许问题的最坏情形在其实例中所占的比例是可忽略的，但由于我们研究的是资源的下界，更重要的是无法知道实例的分布情形，因此，尽管许多问题其平均情形是多项式的，但我们仍以其最坏情形为准。这也将为我们带来更多的方面。至于空间资源，我们亦类似的考虑。

例2.2 极大流量（MAX FLOW） 这是 REACHABILITY 的一个推广。

一个网络 $N = (V, E, s, t, c)$ 是一个带有两个特定顶点 s 和 t 的图 (V, E)，其中称 s 为源，t 为接收器。并且没有进入源 s 的边，也没有离开接收器 t 的边。对每条边 (i, j)，给定一个称为容量的正整数 $c(i, j)$。

N 中的一个流量 f 是对每条边 (i, j) 的一个非负整数赋值 $f(i, j) \le c(i, j)$，满足，除了 s 和 t 外，每个顶点的进入边的 f 值的和等于其离开边的 f 值的和。一个流量 f 的值则是离开 s 的（或到达 t 的）边的流量的和。如图 2.1-2 即为一个网络。

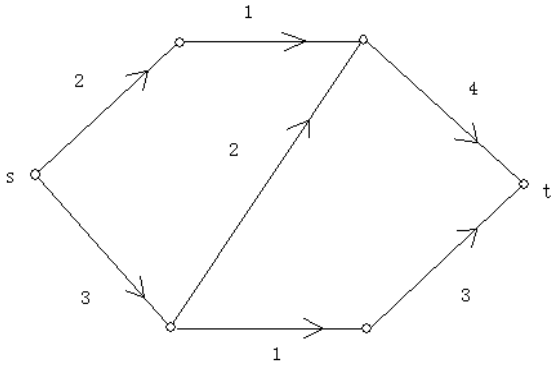

图 2.1-2 网络

极大流量问题（MAX FLOW）：*给定一个网络 N，找到一个最大可能值的流量。*下面我们讨论极大流量问题的多项式时间求解算法。

假定有一个流量 f，要判定 f 是否最优。若存在一个流量 f' 使得 $f' \geq f$，则 $\Delta f = f' - f$ 亦是一个流量，但可能在某边 (i,j) 上 $\Delta f(i,j) < 0$，此时，我们视其为边 (j,i) 上的流量。该反向流量至多为 $f(i,j)$，正的流量 $\Delta f(i,j) \leq c(i,j) - f(i,j)$。利用 c 与 f 构造导出网络 $N(f) = (V, E', s, t, c')$，满足 $E' = E \setminus \{(i,j): f(i,j) = c(i,j)\} \cup \{(i,j): (j,i) \in E, f(j,i) > 0\}$，而且对于 $(i,j) \in E$ 有 $c'(i,j) = c(i,j) - f(i,j)$，对于 $(i,j) \in E' \setminus E$ 有 $c'(i,j) = f(j,i)$。

如图 2.1-3 和 2.1-4 所示，给出了网络 N 和 $N(f)$ 的构造：由于 (s,c)、(c,b) 和 (c,d) 均有 $f = c$，将这些边去掉，添加 (c,s)、(b,c)、(d,c)、(t,b) 和 (t,d)，则 $c'(c,s) = 3$，$c'(b,c) = 2$，$c'(d,c) = 1$，$c'(b,t) = 2 = c(b,t) - f(b,t)$，$c'(t,b) = f(b,t) = 2$，$c'(d,t) = c(d,t) - f(d,t) = 2$，$c'(t,d) = f(d,t) = 1$。

于是，f 是否为极大等同于判定 $N(f)$ 没有正的流量。但是如何在一个网络中判定没有正的流呢？主要思想是将 MAX FLOW 问题归约为 REACHABILITY。

在一个网络中，找一个有正容量的正的流量，即，判定 $N(f)$ 中是否有从 s 到 t 的通路，这即为一个 REACHABILITY 的实例。于是有下列算法：

对于网络 N，开始设处处流量为 0，构造 $N(f)$，并在 $N(f)$ 中应用 REACHABILITY 的求解算法找到从 s 到 t 的通路。若这样的通路存在，则沿此通路找到最小的容量 c'，并将 c' 加到该通路上的所有边的 f 值上，得到一个更大的流量 f。然后，针对新的 f，再构造 N(f) 重复上述过程，直到有 f 使得 $N(f)$ 找不到通路，所得 f 即为极大流量。

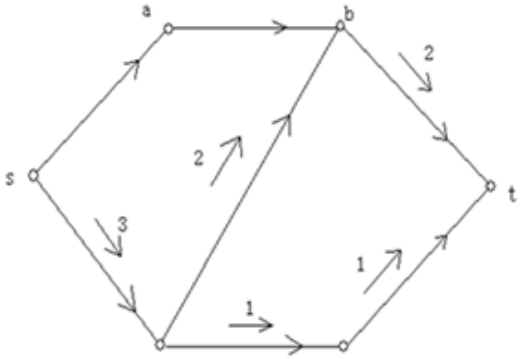

图 2.1-3　网络 N,c=4

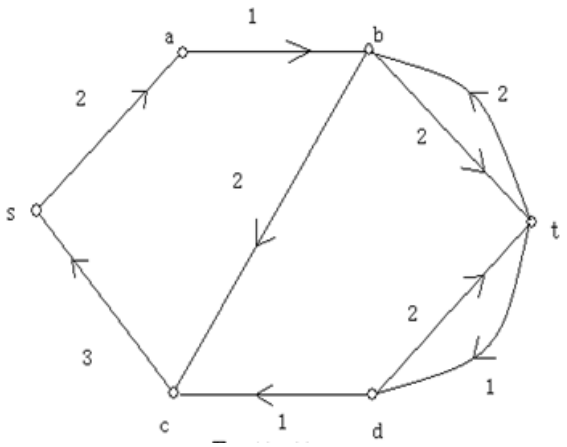

图 2.1-4 N(f)

如上例中，下面分析该算法的效率。算法每次重复地找到一条通路，并增加该通路上的流量。每执行一次其所用的时间如 REACHABILITY 一样为$O(n^2)$。若 C 为网络中边的最大容量，则至多需要 nC 个阶段。事实上，每次重复后，流量至少增加 1，且最大流量应不超过 nC，故要有 nC 个阶段，即有 nC 次重复。因此，总的时间为$O(n^3C)$。注意，虽然看起来是个多项式，但由于 C 的存在，只能是个伪多项式，原因是 C 可能是指数的（如，图 2.1-5）。

为克服该缺陷，在上述算法中，我们不沿着通路增加流量，而是先用 REACHABILITY 的宽度优先的搜索算法找到最短通路，然后沿最短通路增加流量。一条通路上有最小容量的边，称之为瓶颈。由于 N 至多有 n^2 条边，且每次重复至少有一个瓶颈，故重复应有 $\le n^3$ 次，于是可在时间 $O(n^5)$ 内算法求解 MAX FLOW。

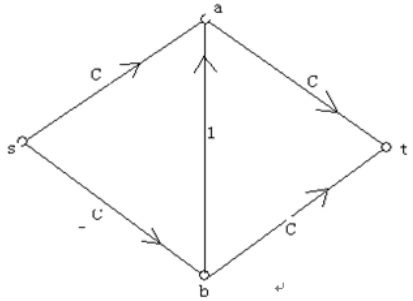

图 2.1-5　特例

现在分析该算法的空间使用情况。尽管算法的每次重复需要空间很小，类似于 REACHABILITY，大约 $O(\log^2 n)$，但需要额外的 $O(n^2)$ 空间贮存当前的流。

另一个值得注意的是效率 $O(n^3C)$ 是可能达到的，如图 1-4 所示的例子。网络中四条边容量均为 C，对角线的容量为 1。利用 REACHABILITY 时，若每次的运气不好，总是选取通路 (s,b,a,t) 或 (s,a,b,t)，则算法需要重复 $2C$ 次。于是总的时间为 $O(n^3C)$。

显然，MAX FLOW 是一个最优化问题，而不是一个判定性问题。但是我们可以将

任意最优化问题转变为一个近似等价的判定性问题。具体方法是先为最优化的量选取一个目标值，然后询问是否可以达到该值。于是 MAX FLOW 可以转化为判定问题 MAX FLOW（D）,即，*给定一个网络N 和一个整数K，问是否存在不小于K 的流量。*可以证明该问题等价于 MAX FLOW。根据求解 MAX FLOW 的算法，我们可以得到下列定理。

MAX-FLOW MIN-CUT 定理：*在任何网络中，极大流量的值等于极小割集的容量。*

注：这里的割集是指边割集，即，对于一个连通图来说，删除这些边后连通图变为不连通。割集一般不是唯一的，含有最小边数的割集称为最小边割集。所谓的极小割集S 的容量是指离开S 的边的容量和，这里要求源$s \in S$ 。

这里我们不证明该定理，有兴趣的读者可以参考相关文献。

例2.3　二分图匹配（Bipartite Matching）

所谓二分图是一个三元组$B = (U, V, E)$，其中$U = \{u_1, \mathrm{L}\ , u_n\}$，称为 boy 顶点集，且$V = \{v_1, \mathrm{L}\ , v_n\}$，称为 girl 顶点集，而$E \subseteq U \times V$ 为边的集合。

在二分图中一个完美匹配（或简单匹配）是一个n 边集合$M \subseteq E$，满足，对于任意两条边，$(u,v),(u',v') \in M$，$u \neq u'$ 且$v \neq v'$，即M 中没有两条边邻接。直观的，一个匹配即是将u 分配一个v，使得$(u,v) \in E$，如图 2.1-6。

于是有下列问题，称之为匹配问题(MATCHING)：*给定一个二分图，它是否有匹配？*

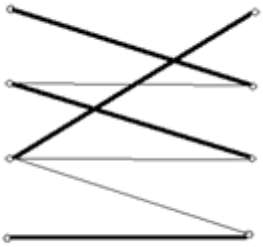

图 2.1-6 二分图完美匹配

我们将利用算法的一个中心概念-------归约。所谓归约就是一个算法，为了对问题 *A* 求解，通过将 *A* 的任何实例转变为已经解决的问题 *B* 的等价实例，从而求解 *A*。这里我们将匹配问题归约到 MAX FLOW(D)。

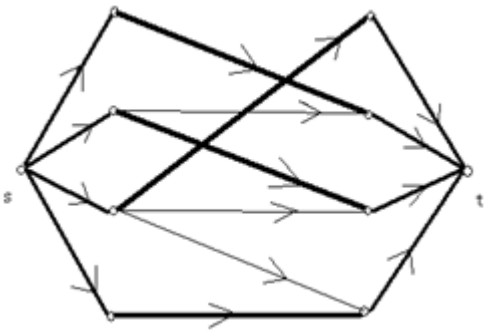

图 2.1-7 匹配归约到极大流量

对于二分图$B=(U,V,E)$，以顶点$U \cup V \cup \{s,t\}$构造网络，这里 s 为源，t 为接收器，边为$\{(s,u):u\in U\}\cup E\cup\{(v,t):v\in V\}$(如图 2.1-7)，且所有边的容量均为 1。易见，B 有一个匹配当且仅当所得网络有值为 n 的流量，从而完成归约。利用 MAX FLOW(D)的算法，可知 MATCHING 可在多项式时间$O(n^3)$内求解。我们后面会详细介绍归约。

第二节 逻辑中问题与算法

本部分将简单介绍 Boolean 逻辑、一阶逻辑和二阶逻辑中的计算问题。这些问题在计算复杂性中具有中心地位，以逻辑为中介，我们可以将复杂性概念等同于实际的计算问题。

1 Boolean 逻辑

设 Boolean 变量$X=\{x_1,\cdots,x_n,\cdots\}$组成一个可数的无限字母集，这些变量取真值 true 或 false，简记为 1 或 0，可以通过∨（析取，或），∧（合取，且）和¬（否）作为运算符号连接起来可得到 Boolean 表达式。对于变量 x，称 x 或$\neg x$为文字。

一个赋值 T 是从 Boolean 变量的子集 $X'\subset X$ 到真值集合$\{0,1\}$的一个映射。对于一个Boolean表达式ϕ，记 $X(\phi)$ 表示ϕ中Boolean变量的集合。称一个赋值 T 适合ϕ，如果 $X(\phi)$ 在 T 的定义域内。设赋值 T 适合ϕ，称 T 满足ϕ，记为$T\models\phi$，如果 $X(\phi)$ 中元在 T 下的取值使得ϕ的真值为 1。否则，称 T 不满足ϕ，记为$T\not\models\phi$。显然，$T\not\models\phi$当且仅当$T\models\neg\phi$。显然，根据 Boolean 变量真值的取值，Boolean 表达式可以取 0 或 1。称一个 Boolean 表达式ϕ是可满足的，如果存在适合ϕ的赋值 T，有$T\models\phi$；否则，称之为不可满足。称一个 Boolean 表达式ϕ为永真式或重言式，如果对于所有适合ϕ的赋值 T，有$T\models\phi$，记为$\models\phi$。可满足和永真性是 Boolean 表达式的重要性质，显然，一个 Boolean 表达式是不可满足的当且仅当其否是永真式。关于可满足性有如下问题，简记为 **SAT**：

给定一个合取范式形式的Boolean 表达式ϕ，问ϕ是否可满足？

显然，该问题利用穷搜索可在$O(2^n n^2)$时间内求解，后面将会看到，该问题可用非确定多项式时间算法求解，且是一个 ***NP*** 完备问题。

为了方便，我们引进另外两个连接词蕴含“$\Rightarrow$”和等价“$\Leftrightarrow$”。对于 Boolean 表达式ϕ_1和ϕ_2，定义$\phi_1\Rightarrow\phi_2$为$\neg\phi_1\vee\phi_2$。若$\phi_1\Rightarrow\phi_2$并且$\phi_2\Rightarrow\phi_1$，则定义为$\phi_1\Leftrightarrow\phi_2$。称两个表达式$\phi_1$与$\phi_2$等价，如果对于任何适合于$\phi_1$与$\phi_2$的赋值 T，有$T\models\phi_1$当且仅当$T\models\phi_2$，即$T\models(\phi_1\Leftrightarrow\phi_2)$，记为$\phi_1\equiv\phi_2$。若两个 Boolean 表达式等价，则将其视为同一对象的不同表达形式，可以互相代替。

称一个 Boolean 表达式ϕ为合取范式形式，如果$\phi=\bigwedge_{i=1}^{n}\phi_i$，其中$n\geq 1$，且每个$\phi_i$

为一些文字的析取。称ϕ_i为分句。类似的，可定义析取范式形式为$\phi=\bigvee_{i=1}^{n} D_i$，其中$n\geq 1$，且每个$D_i$为一些文字的合取。易证，每个 Boolean 表达式等价于一个合取范式，而且也等价于一个析取范式。虽然我们目前没有找到 **SAT** 的确定的有效算法，但是有一些特殊的 **SAT 问题**容易求解。

称一个分句为 Horn 分句，如果它至多有一个正的文字，如，$\neg x_1\vee\neg x_2\vee x_3$和$x_1\vee\neg x_2$等。显然，这类分句等价于$(x_1\wedge\cdots\wedge x_n)\Rightarrow y$。我们有 **HORNSAT** 问题，即，*设$\phi$为Horn 分句的合取，问$\phi$是否为可满足的*？下面算法可判定$\phi$是否可满足：

*这里不妨设赋值*T *是使得取值为*1 *的变量的集合。初始*$\mathrm{T}:=\{\ \}$*（空集）,即所有变量取*0*，然后重复下列步骤，直到所有蕴含式都被满足：挑选任意不满足的蕴含式*$(x_1\wedge x_2\wedge\cdots\wedge x_m)\Rightarrow y$*，此时有*$T(x_1)=\cdots=T(x_m)=1$*，则*$\mathrm{T}\leftarrow y$*（即，令*$y$ *取*1*）。*

由于每一步后T 都增大，因此算法最终会停止，可判定ϕ是否可满足。事实上，易证，若存在T′满足ϕ，则$\mathrm{T}\subseteq\mathrm{T}'$。进而有$\phi$是可满足的当且仅当由算法所得的赋值T 满足$\phi$。事实上，若$\mathrm{T}\not\models\phi$，则$\phi$一定有分句$\neg x_1\vee\neg x_2\vee\cdots\vee\neg x_m$，且$\{x_1,x_2,\cdots x_m\}\subseteq\mathrm{T}$，于是对于任意$\mathrm{T}'\supseteq\mathrm{T}$，亦有$\mathrm{T}'\not\models\phi$。此时$\phi$为不可满足的。另外，算法中在添加$y$对应变元($y$或$\neg y$）到T 时，若发现其已经在其中，则输出拒绝并停机，说明ϕ不是可满足的。所述算法在多项式时间内完成，因此判定 Horn 分句的 Boolean 表达式需要多项式时间。

定义2.2　一个 Boolean 电路是一个图$C=(V,E)$，其中顶点集$V=\{1,2,\cdots,n\}$中的顶点被称为门，满足：

（1）　图C中没有圈，故可设边为(i,j)，$i<j$。

（2）　所有顶点的入度分别为 0，1 或 2。

（3）　每个门$i\in V$都有一个分类，$s(i)\in\{0,1,\neg,\wedge,\vee\}\cup\{x_1,x_2,\mathrm{L}\}$。若$s(i)\in\{0,1\}\cup\{x_1,x_2,\cdots\}$，则$i$的入度为 0，即，没有进入$i$的边；若$s(i)=\neg$，则$i$的入度为 1；若$s(i)\in\{\wedge,\vee\}$，则$i$的入度为 2。

（4）　顶点n为电路的输出门。

对于每个合适的赋值，电路描述了一个真值。令$X(C)$为出现在C中的所有 Boolean 变量的集合，即$X(C)=\{x\in X:\text{有}i\in V\text{使得}s(i)=x\}$。称一个赋值T 是适合$C$，如果T 对于$X(C)$中所有变量都有定义。给定这样一个T，门$i\in V$的真值$\mathrm{T}(i)$可归纳定义如下：若$s(i)=1$，则$\mathrm{T}(i)=1$；若$s(i)=0$，则$\mathrm{T}(i)=0$；若$s(i)\in X$，则$\mathrm{T}(i)=\mathrm{T}(s(i))$；若$s(i)=\neg$，则存在唯一门$j<i$，使得$(j,i)\in E$。由归纳可知$\mathrm{T}(j)$，进而可知$\mathrm{T}(i)=\neg\mathrm{T}(j)$；若$s(i)=\vee$，则存在两条边$(j,i)$和$(j',i)$进入$i$，$\mathrm{T}(i)=\mathrm{T}(j)\vee\mathrm{T}(j')$；若$s(i)=\wedge$，则$\mathrm{T}(i)=\mathrm{T}(j)\wedge\mathrm{T}(j')$，

最后电路值$\mathrm{T}(C)=\mathrm{T}(n)$。

由定义可以看出，电路和 Boolean 表达式是一致的，可以互相构造。相应于 Boolean 电路，亦有一个计算问题，称为 **CIRCUITSAT**，即，*给定一个电路C，是否有适合C的赋值*T，*满足*$\mathrm{T}(C)=1$？易知，**CIRCUITSAT** 与 **SAT** 是计算等价的（即，相互归约）。

若一个电路C中没有变量门，则对应问题为 **CIRCUIT VALUE**，只需要按顺序计算所有门即可，这是多项式时间可求解的。

2 一阶逻辑

一阶逻辑在语义上将句子和其应用的数学对象视为等同，具有强大综合性的原始证明系统，为我们提供用于详细表达数学命题的语法。与 Boolean 逻辑相比，它能表达更广泛的数学思想和事实；这些表达涉及了数学各领域的常数、函数和关系。当然，每个表达式仅涉及这些中的一个小部分，而且每个表达将含有取自一个词汇表的特定有限的包含函数、常数和关系。因此，利用一阶逻辑，我们可以统一地研究定义域上的所有推理。

定义2.3 一个词汇表$\Sigma=(\Phi,\Pi,r)$由两个不交的可数集合组成：函数符号集合Φ和关系符号集合Π。r为一个计数函数，将$\Phi\cup\Pi$映到非负整数，即说明每个函数和关系符号取多少个变量。称一个符号f是一个k-元函数，如果$f\in\Phi$且$r(f)=k$；称一个符号R为k-元关系，如果$R\in\Pi$且$r(R)=k$。常数为0-元函数。这里我们假定Π总含有二元相等关系“=”，且有一个固定的可数变量集$V=\{x,y,z,\cdots\}$，其中所有变量取值于由特定表达式所讨论事物全体组成的定义域。

基于词汇表Σ的一阶逻辑表达式中，如 Boolean 逻辑一样，可以用“$\Rightarrow$”和“$\Leftrightarrow$”；对于全称量词$\forall$和存在量词$\exists$，用$\exists x\phi$代表$\neg(\forall x\neg\phi)$。对于一阶逻辑，赋值的分析需要一个更复杂的数学对象-------模型。

定义2.4 词汇表Σ，适合于Σ的模型是一个对$M=(U,\mu)$，其中U为一个集合，称之为M的定义域，μ是给$V\cup\Phi\cup\Pi$中的每个变量、函数符号和关系符号分配一个U中对象的函数。即，对于每个变量x，μ赋值$x^M\in U$给它，对每个k-元函数符号$f\in\Phi$，在μ下有一个实际函数$f^M:U^k\to U$；若$c\in\Phi$为常数，则$c^M\in U$；对每个k-元关系$R\in\Pi$，则在μ下的像为$R^M\subseteq U^k$。对于相等关系“=”，μ下的像为$=^M$，即为$\{(u,u):u\in U\}$。

在模型M下，设ϕ是一个原子表达式，$\phi=R(t_1,\cdots,t_k)$，则若$(t_1^M,\cdots,t_k^M)\in R^M$，则称$M$满足$\phi$，若$\phi$不是原子表达式，下面归纳定义$M$满足$\phi$，记为$M\models\phi$：

对于$\phi=\neg\psi$，若$M\not\models\psi$，则$M\models\phi$；若$\phi=\psi_1\vee\psi_2$，当$M\models\psi_1$或者$M\models\psi_2$时，则$M\models\phi$；若$\phi=\psi_1\wedge\psi_2$，当$M\models\psi_1$和$M\models\psi_2$时，则$M\models\phi$。

设 $\phi = \forall x\psi$，则 $M \models \phi$，若对于任意的 $u \in U$，令 $M_{x=u}$ 是一个除了 $x^{M_{x=u}} = u$ 之外均等同于 M 的模型，满足对于所有的 $u \in U$ 有 $M_{x=u} \models \psi$。

我们将主要研究没有自由变元的表达式，即句子，有时亦考虑一个句子的子表达式。“模型适合一个表达式”将意味着函数、关系和自由变量在一个模型上的部分取值。在句子和模型之间存在一个有趣的对偶性：一个模型满足或不满足一个句子；而一个句子可以被视为一个模型集合的描述，即，满足该句子的模型的集合。由此可见，一个句子描述了模型的一个性质。

例2.4　图论模型　为表达图的性质，定义图论的词汇 $\Sigma_G = (\Phi_G, \Pi_G, r_G)$。令 Φ_G ={空集}，且除了“=”外，有一个二元关系 G。图论中有典型的表达：$G(x,x)$；

$\exists x\left(\forall y G(y,x)\right)$；$\forall x\left(\forall y\left(G(x,y) \Rightarrow G(y,x)\right)\right)$；

$\forall x\left(\forall y\left(\exists z\left(G(x,z) \wedge G(z,y)\right) \Rightarrow G(x,y)\right)\right)$。适合于 Σ_G 的任何模型是一个图，这里我们仅考虑有限图。

$\phi_1 = \left(\forall x \exists y G(x,y) \wedge \forall x \forall y \forall z\left(\left(G(x,y) \wedge G(x,z)\right) \Rightarrow y = z\right)\right)$，有模型 Γ。Γ 的定义域是图 2.2-1 中 7 个顶点的图的集合。$G^{\Gamma}(x,y) = 1$ 当且仅当在图中存在从 x 到 y 的边。易见，$\Gamma \models \phi_1$。自然地问，是否有其它图满足 ϕ_1？事实上，*所有表示一个函数的图（即，所有顶点的出度为 1）都满足 ϕ_1 且仅有这些图*。句子 ϕ_1 完全说明性质“G 是一个函数”。

$\phi_2 = \forall x\left(\forall y\left(G(x,y) \Rightarrow G(y,x)\right)\right)$。如图 2-7 所示模型满足 ϕ_2，但图 2.2-1 不满足 ϕ_2，ϕ_2 说明性质“G 为对称图”。

$\phi_3 = \forall x\left(\forall y\left(\forall z\left(G(x,z) \wedge G(z,y)\right) \Rightarrow G(x,y)\right)\right)$，描述性质：“G 为传递图”。

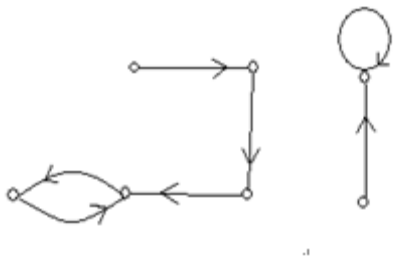

图 2.2-1 对称图

因此，在图论中每个句子描述了图的一个性质。反过来，图的任何性质对应于一个计算问题：*给定一个图，问该图是否具有该性质 ϕ？*称此问题为 **ϕ-GRAPHS** 问题，即，

设 ϕ 为 Σ_G 上的一个表达式，定义下列问题：给定一个关于 ϕ 的模型 Γ，是否有 $\Gamma \models \phi$？。

如，若$\phi=\forall x\left(\forall y\left(G\left(x,y\right)\Rightarrow G\left(y,x\right)\right)\right)$，则$\phi$-GRAPHS 为判定图的对称性问题。也有涉及一个图和一部分顶点的问题，如以前所知的 REACHABILITY，但下面结论说明，ϕ-GRAPHS 与 REACHABILITY 具有一样的重要性质。

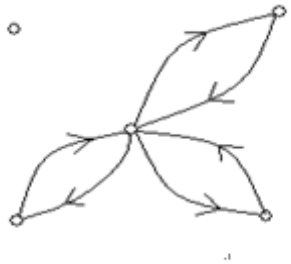

图 2.2-2 传递图

定理2.1　对于任何Σ_G上的表达式ϕ，问题ϕ-GRAPHS 是确定的多项式时间可求解的。

证明：对ϕ的结构做归纳：当ϕ为形如$G\left(x,y\right)$和$G\left(y,x\right)$的原子表达式，则可检查邻接矩阵得到结论。若$\phi=\neg\psi$，则由归纳法知，存在一个求解ϕ-GRAPHS 的多项式算法（只需要将“yes”和“no”互换即可）。若$\phi=\psi_1\vee\psi_2$，由归纳法可知，存在多项式算法A_1和A_2分别求解ψ_1-GRAPHS 和ψ_2-GRAPHS。对于ϕ-GRAPHS，我们构造算法 A 先后分别执行A_1和A_2。若其中之一回答“yes”，则回答“yes”，否则回答“no”。显然，A 的运行时间是A_1和A_2的时间之和。类似，可得$\phi=\psi_1\wedge\psi_2$。

最后，假设$\phi=\exists x\psi$，则有一个多项式时间算法A_1求解ψ-GRAPHS。于是ϕ-GRAPHS 的算法 A 可如下构造：

对于G 的每个顶点v*，对于给定的*ϕ*的模型*Γ*，将*$x=v$*附加于*Γ*得到*ψ*的模型。然后检验是否*$\Gamma_{x=v}\models\psi$*。对于所有顶点*v*，若有一个回答是 yes，则 A 输出 yes；否则，“no”。*

显然，A 的运行时间是（顶点数）×（A_1的时间），因此 A 是确定多项式算法。

类似方法，可得到关于空间的结果。

推论2.1　对于Σ_G上任意表达式ϕ，ϕ-GRAPHS 问题可在空间$O\left(\log n\right)$内求解。

由上述讨论可见，对于一个表达式，我们不得不努力寻找满足它的模型。若这样的模型存在，则称该表达式是可满足的。也有一些表达式可被任意模型满足，称为永真式。对于永真式ϕ，记为$\models\phi$。永真式是一个命题，其成真仅是因为关于函数、量词、等式等的一般性质的基本推理，而不是特殊的数学领域。类似于 Boolean 逻辑，可以知道，一个表达式ϕ是不可满足的当且仅当$\neg\phi$是永真式。

问题：*给定*ϕ*，它是否是永真式*？称此问题为 VALIDITY。

定理2.2 (Löwenheim-Skolem) 若句子ϕ有任意大规模（cardinality）的有限模型，则ϕ有一个无限模型。

我们可以利用定理 2.2 说明一阶逻辑表达式的局限性。这里我们略去这些结论的证明，有兴趣的读者可以参考相关逻辑的文献。

在例 2.4 中，我们用一阶逻辑表达了图的性质：出度为 1、对称性、传递性等；并在定理 2.1 中证明可由一个句子ϕ表达的图的性质是容易验证的，即ϕ-GRAPHS 问题有多项式算法。那么，*是否所有的多项式可验证的图的性质可用一阶逻辑表达？*答案是否定的。下列结论说明 REACHABILITY 不能用一阶逻辑表示。

定理2.3 没有（有两个自由变量x和y的）一阶逻辑表达式ϕ使得ϕ- GRAPHS 与 REACHABILITY 相同。

证明：假设存在一个表达式ϕ使得ϕ-GRAPHS 与 REACHABILITY 相同。那么句子$\psi_0 = \forall x \forall y \phi$表明图 G 是强连通的，即，从任何顶点可以到达任何顶点。而句子

$\psi_1 = \left(\forall x \exists y G(x,y) \wedge \forall x \forall y \forall z \left(\left(G(x,y) \wedge G(x,z)\right) \Rightarrow y = z\right)\right)$ 说明每个顶点出度为 1；

$\psi_1' = \left(\forall x \exists y G(y,x) \wedge \forall x \forall y \forall z \left(\left(G(y,x) \wedge G(z,x)\right) \Rightarrow y = z\right)\right)$ 说明每个顶点的入度为 1。

令$\psi = \psi_o \wedge \psi_1 \wedge \psi_1'$，则$\psi$说明图是强连通的，且所有顶点的入度和出度均为 1，即为圈（如，图 2.2-3）。显然给定顶点数，则有有限多个圈。于是，ψ有任意大的有限模型。由推论 2.3 可知，ψ有一个无限模型G_∞。

考虑G_∞的一个顶点v，则G_∞有顶点序列$v_0, v_1, \cdots$，其中(v_i, v_{i+1})为边，且v_i有出度和入度均为 1。由于G_∞为强连通的，故这些顶点包含了G_∞的所有顶点。但是，v_0有入度为 1，于是存在v_j，使得(v_j, v_0)为边，于是G_∞有有限个顶点，矛盾。故ϕ不存在。

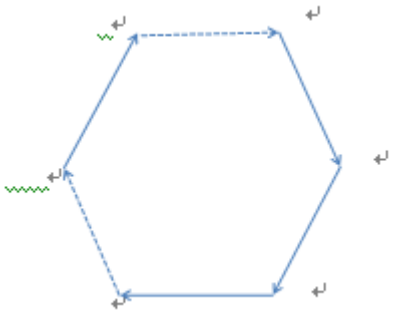

图 2.2-3 无限模型G_∞

由此可见，REACHABILITY 是一阶逻辑不可表达的。但该结果是很有趣的，主要原因是：

（1）这是一个非平凡的不可能结论。该结论恰是复杂性理论中所要研究的结果类型。

（2）这种不可能结果促使我们对一阶逻辑的推广研究，最终导致复杂性和逻辑之间相似性的另一层面。

下面我们分析需要给一阶逻辑附加什么“特征”，才能使 REACHABILITY 可被表达。首先分析 REACHABILITY：是否存在从 x 到 y 的通路。这里“存在”似乎可以用量词“∃”在一阶逻辑中恰当表示。但是量词“∃”之后要紧接一个代表单个的顶点变量，如 x，那么，一阶逻辑仅说明“存在顶点 x …”；然而“从 x 到 y 的通路”则是说的是一个顶点集合的性质。因此，我们需要引入更复杂的对象上的存在量词。

3 二阶逻辑

基于词汇表 $\Sigma=(\Phi,\Pi,r)$，一个存在二阶逻辑的表达式形如 $\exists P\phi$，其中 ϕ 为基于词汇表 $\Sigma'=(\Phi,\Pi\cup\{P\},r)$ 的一个一阶逻辑，即，$P\notin\Pi$ 是一个新的 $r(P)$-元关系符号。自然地，在 ϕ 中一定会涉及到 P。简单地说，表达式 $\exists P\phi$ 即为存在关系 P 使得 ϕ 成立。

称适合 Σ 的一个模型 $M=(U,\mu)$ 满足 $\exists P\phi$，如果存在一个关系 $P^M\subseteq(U)^{r(P)}$，使得 P^M 增广 M 后得到一个适合 Σ' 的模型，且该模型满足 ϕ。

通过下面的例子说明存在二阶逻辑具有强大功能，不仅可用于表达有多项式时间算法的图论性质，也可用于表达目前还不知道是否有多项式时间算法的图论性质。

例 2.5 在图论词汇表中，句子 $\exists P\forall x\forall y(P(x,y)\Rightarrow G(x,y))$ 断言图 G 的子图的存在性。显然，这是一个成真的句子。

例2.6 图的可达性。为了简便，只描述不可达性，其补即为可达性。

令 $\phi(x,y)=\exists P(\forall u\forall v\forall w((P(u,u))\wedge(G(u,v)\Rightarrow P(u,v))\wedge((P(u,v)\wedge P(v,w))\Rightarrow P(u,w))\wedge\neg P(x,y)))$。

在图论词汇表中，$\phi(x,y)$ 表明存在一个图 P 以 G 为子图，具有自反性和传递性，而且没有从 x 到 y 的边。显然，任何满足前三个条件的图一定含有 G 的自反和传递闭包。$\neg P(x,y)$ 表明 G 中没有从 x 到 y 的边。于是 $\phi(x,y)$-GRAPHS 恰恰描述了 REACHABILITY 的补。

注意 $\neg\phi(x,y)$ 不是 REACHABILITY，原因在于存在二阶逻辑在 $\neg$（否）下不封闭。要确切的表达 REACHABILITY，需要更多工作。有兴趣的读者可参考文献[9]和[10]。

例 2.7 Hamilton 通路问题（HAMILTON PATH）

给定一个图，是否有一条通路通过每个顶点恰好一次?

目前，没有多项式时间算法判定一个图是否有一条 Hamilton 通路。下面给出描述有 Hamilton 通路的图的句子：

$\psi=\exists P\chi$，这里 χ 要求 P 为 G 的顶点上的一个线性序，使得先后相邻点在 G 中是连通的。另外 χ 还要求：

（1）G 的所有不同顶点可由 P 比较：$\forall x\forall y\left(P\left(x,y\right)\vee P\left(y,x\right)\vee x=y\right)$。

（2）P 是传递的，但不是自反的：$\forall x\forall y\forall z\left(\left(\neg P\left(x,x\right)\right)\wedge\left(\left(P\left(x,y\right)\wedge P\left(y,z\right)\right)\Rightarrow P\left(x,z\right)\right)\right)$。

（3）P 中任意先后相邻两点在 G 中一定邻接：

$$\forall x\forall y\left(\left(P\left(x,y\right)\wedge\forall z\left(\neg P\left(x,z\right)\vee\neg P\left(z,y\right)\right)\right)\Rightarrow G\left(x,y\right)\right)。$$

易证，ψ -GRAPHS 与 HAMILTON PATH 相同。事实上，满足这三个性质的 P 一定是线性序，而且任意两个先后相邻的元素在 G 中邻接，故 P 即是一个 Hamilton 通路。

由上面的例子可以看出，存在二阶逻辑的强大功能，后面其功能会进一步体现。现在我们再分析(un)REACHABILITY 和 HAMILTON PATH 的表达式。已知前者是多项式时间可求解的。而后者被相信没有确定多项式时间算法可解。原因在于，对于 UNREACHABLITY，表达式$\phi\left(x,y\right)$是一个前缀范式，其中仅有全称量词，而且是以合取范式形式组成的列表。该表达式有分句列表 $P\left(u,u\right)$，$G\left(u,v\right)\Rightarrow P\left(u,v\right)$，$\neg P\left(x,y\right)$，$\neg P\left(u,v\right)\vee\neg P\left(v,w\right)\vee P\left(u,w\right)$。从这些分句中删除不涉及 P 的原子表达式，则有 $P\left(u,u\right)$，$\neg P\left(x,y\right)$，$\neg P\left(x,y\right)\vee\neg P\left(y,z\right)\vee P\left(x,z\right)$。这三个分句中，每个都至多有一个非负的涉及 P 的原子公式。如前，称为 Horn 表达式。称存在二阶逻辑中一个表达式为一个 Horn 表达式，如果在其前缀形式中仅有一阶全称量词，而且其分句列表中每个分句至多含有一个非负的关于 P 的原子公式。UNREACHABILITY 是一个 Horn 表达式。对于 HAMILTON PATH 而言，表达式中有分句$P\left(x,y\right)\vee P\left(y,x\right)\vee x=y$不是 Horn 的；并且将$\psi$ 化为前缀范式时，有存在量词出现。下列结果说明ϕ 与ψ 不同的原因。

定理2.4　对于任意存在二阶逻辑的 Horn 表达式$\exists P\phi$，问题$\exists P\phi$-GRAPHS 可由确定的多项式时间算法求解。

证明：设$\exists P\phi=\exists P\forall x_1\cdots\forall x_k\eta$，其中$\eta$ 为 Horn 分句的合取，且 P 为 r-元关系。给定图 G 有 n 个顶点$\left\{1,2,\cdots,n\right\}$，问是否 G 是$\exists P\phi$-GRAPHS 的一个“yes”实例？即，是否存在一个关系$P\subseteq\left\{1,2,\cdots,n\right\}^r$使得$\phi$ 可满足。

对于$x_i\in\left\{1,2,\cdots,n\right\}$，由于$\eta$ 必须对 x_i 的值的所有组合成立，故可将$\exists P\phi$ 重写为

$$\exists P\phi=\bigwedge_{v_1,\cdots,v_k=1}^{n}\eta\left[x_1\leftarrow v_1,\cdots,x_k\leftarrow v_k\right]\qquad\text{（式 1）}$$

此式中含有hn^k 个分句，其中h 为η 中分句的个数。式 1 中至多有三种原子表达式，分别为$G\left(v_i,v_j\right)$，$v_i=v_j$ 或$P\left(v_{i_1},\cdots,v_{i_r}\right)$。前两种可以容易地被计算是 0 或 1。对（2.1）做如下处理：

如果一个文字取值为 0，则将该文字从其分句中删除；若一个文字取值为 1，则所在分句被删除。如此下去，若最终得到含有空分句的表达式，则ϕ 为不可满足，且 G 不满足ϕ。

经如此处理后，留下至多 hn^k 个分句。每个分句均为形为 $P\left(v_{i_1},\cdots,v_{i_r}\right)$ 和 $\neg P\left(v_{i_1},\cdots,v_{i_r}\right)$ 的原子表达式的析取。由于每个 $P\left(v_{i_1},\cdots,v_{i_r}\right)$ 独立的取值 0 或 1，故可以用不同的 Boolean 变量 $x^{v_{i_1},\cdots,v_{i_r}}$ 代替它的每次出现，得到新的表达式 *F*。于是，*F* 是可满足的当且仅当存在 *P* 使得 *G* 满足带有 *P* 的 ϕ。由于 η 为 Horn 二阶逻辑分句，故 *F* 是一个有至多 hn^k 个分句和至多 n^r 个变量的 Horn 形 Boolean 表达式。于是，利用关于 HORNSAT 的多项式算法，可以求解 *F* 的满足问题，进而得到求解 $\exists P\phi$-GRAPHS 的实例 *G*。

第三节一些格问题

到目前为止，我们不知道任何解决 SVP 的多项式时间算法。实际上，我们甚至不知道如果找到长度在 Minkowski 边界$\|\mathbf{Bx}\| < \sqrt{n}\det(\mathbf{B})^{1/n}$内的非零格向量。另一个没有已知多项式算法的相关问题是最近向量问题。对于 CVP 和 SVP，可以考虑不同的算法任务，这些是（以困难性的阶递减）：

- 搜索问题：找到（非零）格向量$\mathbf{x} \in \Lambda$使得$\|x - \mathbf{t}\|$（相应地，$\|\mathbf{x}\|$）是最小的。
- 最优问题：在$\mathbf{x} \in \Lambda$（相应地，$\mathbf{x} \in \Lambda \setminus \{\mathbf{0}\}$）找到$\|x - \mathbf{t}\|$（相应地，$\|\mathbf{x}\|$）的最小值。
- 判定问题：已知有理数$r > 0$，判定是否存在一个（非零）格向量$\mathbf{x}$使得$\|\mathbf{x} - \mathbf{t}\| \leq r$（相应地，$\|\mathbf{x}\| \leq r$ ）。

我们注意到到目前为止，几乎所有已知 SVP 和 CVP 的（指数时间）算法解决搜索问题（因此也包括相关联的最优问题和判定问题），同时所有困难性结果都对判定问题成立（因此这表明了搜索问题和最优问题的困难性）。这表明解决 SVP 和 CVP 问题的困难性已经被判断在已知阈值下是否存在解的判定任务所捕获。我们将在第三章看到与 CVP 相关的判定问题是NP完备的，因此不存在算法能在多项式时间内解决 CVP 问题，除非 **NP=P**。同样的结果（在随机归约下）对 SVP 也成立。

解决 SVP 和 CVP 问题的困难性导致计算机学家开始考虑这些问题的近似版本。近似算法返回的结果可以保证在与最优结果相差某特定因子γ范围内。SVP 和 CVP 搜索问题的近似版本定义如下：

- **近似 SVP**：已知一组基$\mathbf{B} \in \mathbb{Z}^{m\times n}$，找到一个非零格向量$\mathbf{Bx}$（$\mathbf{x} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$）使得对于任何$\mathbf{y} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$有$\|\mathbf{Bx}\| \leq \gamma\|\mathbf{By}\|$。

在近似 SVP 的最优版本中，只需要找到$\|\mathbf{Bx}\|$，即，找到值 d 使得$\lambda_1(\mathbf{B}) \leq d < \gamma\lambda_1(\mathbf{B})$。

- **近似 CVP：**已知一组基$\mathbf{B} \in \mathbb{Z}^{m \times n}$和目标向量$\mathbf{t} \in \mathbb{Z}^m$，找到一个非零格向量$\mathbf{Bx}$（$\mathbf{x} \in \mathbb{Z}^m$）使得对于任何$\mathbf{y} \in \mathbb{Z}^m$有$\|\mathbf{B}\ \mathbf{x} - \mathbf{t}\| \leq \gamma \|\mathbf{By} - \mathbf{t}\|$。

在近似 CVP 的最优版本中，只需计算$\|\mathbf{Bx} - \mathbf{t}\|$，即找到值 d 使得$dist(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq d < \gamma dist(\mathbf{t}, \mathcal{L}(\mathbf{B}))$。

无论是在近似 SVP 还是在近似 CVP 中，近似因子γ可以是任何格的相关参量的函数，典型地有与阶 n 相关，这与该参量增加时问题变得更难的事实一致。到目前为止，最著名的 SVP 和 CVP 的多项式近似算法（可能是概率的）取得最坏情况的（基于输入的选择）近似因子$\gamma(n)$为 n 的指数阶。找到获得近似因子$\gamma(n) = n^c$为多项式（对于独立于 n 的某常量 c）的算法仍是这一领域的主要开放性问题。仍有许多有关格可以有效解决（在确定多项式时间内）的计算问题。这里让我们回忆其中的一些。找到这些问题的多项式时间解留给读者作为练习。

- 成员关系：已知基 **B** 和向量 **x**，判断 **x** 是否属于格$\mathcal{L}(\mathbf{B})$。这个问题本质上等价于解整数上的线性方程组。这可以在多项式次算术运算下实现，但需要确保其中的数字不会变成指数大的。

- 核：已知整矩阵$\mathbf{A} \in \mathbb{Z}^{m \times n}$，计算格$\{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Ax} = \mathbf{0}\}$的基。一个相似的问题是，已知模数 M 和矩阵$\mathbf{A} \in \mathbb{Z}_M^{m \times n}$，找到格$\{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Ax} = \mathbf{0} \pmod{M}\}$。同样地，这个等价于解（齐次）线性方程组。

- 基：已知可能相互独立的向量集合$\mathbf{b}_1, \cdots, \mathbf{b}_n$，找到它们生成格的一组基。这问题能以多种方式解决，例如使用艾尔米特正规形式。

- 并：已知整数格$\mathcal{L}(\mathbf{B}_1)$和$\mathcal{L}(\mathbf{B}_2)$，计算包含$\mathcal{L}(\mathbf{B}_1)$和$\mathcal{L}(\mathbf{B}_2)$的最小格的基。这个问题可以直接归约到计算可能独立向量组生成格的基问题上。

- 对偶：已知格$\mathcal{L}(\mathbf{B})$，计算$\mathcal{L}(\mathbf{B})$对偶的基，即$span(\mathbf{B})$中所有向量 **y** 组成的集合，该集合满足对于每个格向量$\mathbf{x} \in \mathcal{L}(\mathbf{B})$有$\langle \mathbf{x}, \mathbf{y} \rangle$是整数。易知对偶格的基是由$\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}$给出。

- 交集：已知两个整数格$\mathcal{L}(\mathbf{B}_1)$和$\mathcal{L}(\mathbf{B}_2)$，计算交集$\mathcal{L}(\mathbf{B}_1) \cap \mathcal{L}(\mathbf{B}_2)$的基。易知$\mathcal{L}(\mathbf{B}_1) \cap \mathcal{L}(\mathbf{B}_2)$总是格。这个问题使用对偶格可以很容易解决。

- 等价；已知两组基$\mathbf{B}_1$和$\mathbf{B}_2$，检验它们是否生成相同的格$\mathcal{L}(\mathbf{B}_1) = \mathcal{L}(\mathbf{B}_2)$。可以通过检验每一个基向量是否属于由另一个矩阵生成的格解决这个问题，但是，更有效的解法也存在。

- 循环：已知一个格$\mathcal{L}(\mathbf{C})$，检验$\mathcal{L}(\mathbf{C})$是否是循环的，即如果对于格向量$\mathbf{x} \in \mathcal{L}(\mathbf{C})$，通过循环旋转$\textbf{x}$的坐标得到的所有向量都属于这个格。这个可以通过循环旋转基矩阵 **C** 某位置的坐标，并检验所得到的基是否与原来的基等价。

习题

（1）在 REACHABILITY 问题的搜索算法中，证明如下：

a. 若点 v 是第 i 个添加到 S 中的点，则存在从1到 v 的一条通路。

b. 若从1出发通过 l 条边可到达 v，则 v 一定能被添加到 S 中。

c. 图 $G=(V,E)$ 的每条边被处理至多一次，从而算法至多运行 $O(|E|)$ 步。

（2） 若 $A=(A_{ij})$ 是一个 $m\times m$ 的 Boolean 矩阵，定义 A 的传递闭包 *Atr-cl* 为矩阵积 *Atr-cl* $=A^1A^2\ldots A^m\ldots$，其中 A^k 为 A 的 k 次 Boolean 矩阵乘积，并且 $A\vee B$ 为矩阵对应每个(i,j)处赋值单独做比特的 OR 运算。

a. 假设A为有向图G的邻接矩阵。

(i) 证明A^2的（i,j)处赋值为1当且仅当在G 中存在从顶点i到顶点j的长为2的通路。于是， A^2是图G'的邻接矩阵，其中G'中有从顶点i到k的边当且仅当G中有从顶点i到k的长为2的通路。

(ii) 类似地证明A^l 的（i,k)处赋值为1当且仅当G中存在从顶点i到k的长为l的通路。

(iii) 综上可知，PATH问题(输入为一个图G和两个顶点i与j)可以通过计算图G的传递闭包求解。

b. 证明以下结论

(i) 证明A^k的计算可被组织为一个大小为$O(k)$、深为$O(\log k)$的二元树，其每个节点计算两个输入矩阵的积。

(ii) 证明计算A^m 的电路大小为$O(n^2)$ 深度为$O(\log_2 n)$ ，其中$n=m^2$ 为A的大小。

(iii) 推导计算A的传递闭包的大小-深度复杂度为$(O(n^{5/2});O(\log^2 n))$。

（3）如果一个无向连通有限图有一条通路通过每条边恰好一次，则称该图为欧拉图，称该通路为欧拉通路。给定无向连通有限图G，试给出一个算法判定G是否为欧拉图。

（4） 一个二人游戏是由一个顶点被划分为两个集合 $V=V_1\cup V_2$ 的有向图 $G=(V_1,V_2,E)$ 和获胜条件组成。假定 G 的每个顶点 $v\in V$ 都有一个后继。为方便不妨分别称两个参与者为 Bob 和 Alice，他们的一次对决是 G 中的任意有限或无限通路 $v_1v_2v_3\mathrm{L}$ ，其中 v_1 是起始点。若该次对决当前在顶点 v_i 并且 $v_i\in V_0$ ，则轮到 Bob 从 v_i 的后继中选择 v_{i+1}；若 $v_i\in V_1$，则由 Alice 来决定下一次移动。获胜条件由一个子集 $T\subseteq V_0\cup V_1$ 定义：如果 Bob 在 $n-1$ 步内访问到 T，则 Bob 获胜；如果在至少 $n-1$ 步内不能访问到 T，则 Alice 获胜。这里 $|G|=n$。称参与者 Bob/Alice 赢得顶点 s，如果他从 s 出发选择移动使得在任何对决都获胜。证明，可以在关于 $|\langle G,s,T\rangle|$ 的多项式时间内判定是否 Bob 赢得顶点 s。（提示：从 T 开始找出一个顶点集，使得不管 Alice 如何选择，Bob 总可以通过这些顶点到达 T。）

（5）证明极大流量-极小割集定。

第三章　计算模型

算法和复杂性的概念必须建立在一定的计算模型上。本章将介绍我们采用的计算模型：确定图灵机、非确定图灵机和通用图灵机。尽管它们看起来很笨拙，但它们以有限的效率损失为代价却可以模拟任何算法。

设Σ是字母表，它是一个所给定的有限字符的全体组成的集合，称由Σ上字符串组成的集合为语言，并称由语言组成的族为语言类。约定不含任何字符的字符串为空字符串，记为ε，有$|\varepsilon|=0$。令Σ^*表示Σ上所有的字符串。为了方便起见，通常只考虑$\{0,1\}$上的字符串，即二元字符串。

对于一个计算问题的复杂性定义，直观的想法是“计算问题的最优算法所需要的计算时间”，这即为 Kolmogorov 的观点。但是，该观点无法回答如下问题：

1） 是否总存在最优算法？

2） 一个算法所需要的计算时间是指什么？

3）能否明确什么是算法？

直观地说，算法就是一个明确的命令的集合，这些命令描述了一系列要执行的步骤，通过执行这些步骤生成一个特定的输出。称一个算法是确定的，如果在计算过程中任何时刻，每步计算都是明确指定的。对于一个计算问题的算法$\mathbf{A}$，其计算时间t至少依赖于输入x、选择的计算机、选择的程序语言以及算法的实现。计算时间对于输入x的依赖性显然是不可避免，但是，若它还必须至少依赖于其它三者，则算法就无法比较好坏。因此，计算时间的定义必须简化为只依赖于算法及其输入。对于不同的模型，所得到的效果会如何呢？计算问题算法对模型的依赖性如何？下列命题给出了回答。

Church-Turing 命题：所有合理的计算模型可以互相模拟，因而，计算可解问题的集合与计算模型无关。

注意，“合理”计算模型没有准确的定义，因此，上述命题无法被证明。但是，普遍被认为合理计算模型具有三个最基本条件：1)计算一个函数只要有有限条指令；2)每条指令可由模型中的有限个计算步骤完成;3)执行指令的过程是确定的。到目前为止，人们认为合理的计算模型均被证明符合这些命题，因而得到广泛认可。我们进一步也有如下广义 Church-Turing 命题，但近年来广义 Church-Turing 命题受到挑战，主要因为量子图灵机的出现使很多在确定有效时间内没有找到算法的数论问题可在量子图灵机上找到有效求解算法：

广义 Church-Turing 命题：对于任意两个合理的计算模型$\boldsymbol{R}_1$和$\boldsymbol{R}_2$，存在一个多项式p,使得关于长为n的输入，$\boldsymbol{R}_1$的t步计算可以被$\boldsymbol{R}_2$在$p(t,n)$步内模拟。

我们选取图灵机作为复杂性理论的计算模型，并以其基本的计算步数来衡量计算时间。由于其计算步数仅对局部变化有影响，因此选取它可以为我们带来方便。

第一节　图灵机基础

与程序语言一样，图灵机存在唯一的数据结构：字符串。任何图灵机都有两个最基本的单元（如图 3.1-1）：控制单元和存储单元。控制单元通常被成为有限控制器，它有有限个状态。存储单元由一条（或数条）无限带组成(不妨设，这些带是左端界定，右端无限延伸)，每条带上被分成无限个小方格，每格可以存储一个符号，由字符串组成带。有限控制器和带之间通过指针来联系。每个时刻，指针只移动一个方格。

图灵机的运算由一系列的移动组成：

a. 程序在带上左或右移动指针；
b. 指针在当前位置读写、擦除字符；
c. 改变控制器的状态。

这里，每次都是由当前状态和指针所指位置的值决定下次移动，状态描述了机器内部当前环境。

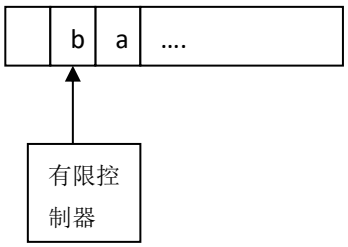

图 3.1-1 图灵机

总之，图灵机是一个很弱的基本程序语言，但能够表达任意算法，模拟任意程序语言。形式的定义如下：

定义3.1 图灵机是一个四元组 $\mathbf{M}=(K,\Sigma,\delta,s)$，满足

1) K 为一个有限状态集合（即为上面提到的指令），$s\in K$ 为初始状态。
2) Σ 为 $\mathbf{M}$ 的字母表，是一个与 K 无关的有限符号集合。Σ 总含有两个元素：空格∪与带上第一个符号▷（即，初始状态下▷只出现在带的第一个位置）。
3) h，"yes"与"no"是另外三个不在 $K\cup\Sigma$ 中的状态，分别定义如下：
   h 是停机状态，表示机器进入该状态后停机；"yes"是接受状态，表示机器进入该状态后停机并接受输入；"no"是拒绝状态，表示机器进入该状态后停机并拒绝输入。
4) 指针有三个移动方向符号：“$\rightarrow$”表示指针要向右移动；“$\leftarrow$”表示指针要向左移动；“–”表示指针不动。
5) δ 为转移函数，是一个从 $K\times\Sigma$ 到 $(K\cup\{h,"yes","no"\})\times\Sigma\times\{\rightarrow,\leftarrow,-\}$ 的映射，即，

$$\delta:\mathbf{K}\times\Sigma\longrightarrow(\mathbf{K}\cup\{\mathbf{h},"\mathbf{yes}","\mathbf{no}"\})\times\Sigma\times\{\rightarrow,\leftarrow,-\}$$

$$(\mathbf{q},\sigma)\longrightarrow(\mathbf{p},\rho,\mathbf{D})=\delta(\mathbf{q},\sigma)$$

   这里函数 δ 即为 $\mathbf{M}$ 的“程序”，对于控制器中当前状态 q 和指针所指符号 σ，机器进入下一状态 p，并将 σ 改写为 ρ，沿着方向 D 移动一次指针，其中 $D\in\{\rightarrow,\leftarrow,-\}$。特别地，当 $\sigma=\triangleright$ 时，一定有 $\rho=\triangleright$ 且 $D="\rightarrow"$，即，当指针指着▷时，总是要向右移动，即，指针不能移动到▷的左边，且▷不能被改写。
6) $\mathbf{M}$ 的程序运行：初始，$\mathbf{M}$ 处于状态 s，带上的符号串为▷x，$x\epsilon\Sigma^*$，称 x 为 $\mathbf{M}$ 的输

入，并且指针总是指着第一个符号▷；然后，机器根据δ执行一步：改变状态，改写符号并移动指针；接下来再根据δ执行另一步，如此下去，直到进入状态$(h,"yes","no")$之一时停机。此时，关于输入x，机器$\mathbf{M}$的输出记为$\mathbf{M}(x)$。当$\mathbf{M}$进入状态yes时，$\mathbf{M}(x)=yes$，表示接受x；当$\mathbf{M}$进入状态no时，$\mathbf{M}(x)=no$，表示拒绝x；当$\mathbf{M}$进入状态h时，若$\mathbf{M}$的带上串为$\triangleright y$，其中y的最后一个符号可能不为$\cup$，y亦可能为一串$\cup$，则$\mathbf{M}(x)=y$，即输出带上的内容。若$\mathbf{M}$不停机，则记为$\mathbf{M}(x)=\nearrow$，表示关于输入x，$\mathbf{M}$不停机。

例3.1 考虑图灵机$\mathbf{M}=(K,\Sigma,\delta,s)$，其中$K=\{s,q,q_0,q_1\}$，$\Sigma=\{0,1,\cup,>\}$，转移函数$\delta$定义如下（图 3.1-2）：

$p\in K,\sigma\in\Sigma$	$(s,0)$	$(s,1)$	$(s,\cup)$	$(s,\triangleright)$	$(q,0)$	$(q,1)$	$(q,\cup)$	$(q,\triangleright)$
$\delta(p,\sigma)$	$(s,0,\rightarrow)$	$(s,1,\rightarrow)$	$(q,\cup,\leftarrow)$	$(s,\triangleright,\rightarrow)$	$(q_0,\cup,\rightarrow)$	$(q_1,\cup,\rightarrow)$	$(q,\cup,-)$	$(h,\triangleright,\rightarrow)$
	$(q_0,0)$	$(q_0,1)$	$(q_0,\cup)$	$(q_1,0)$	$(q_1,1)$	$(q_1,\cup)$	$(q_1,>)$	
	$(s,0,\leftarrow)$	$(s,0,\leftarrow)$	$(s,0,\leftarrow)$	$(s,0,\leftarrow)$	$(s,1,\leftarrow)$	$(s,1,\leftarrow)$	$(h,>,\rightarrow)$	

关于输入$x=010$，$\mathbf{M}$执行如下（图 3.1-3），输出$M(x)=\cup 010$。由此可见，$\mathbf{M}$该机器所做的运算是在字符串与▷之间加个空格。值得注意的是，命令$\delta(q,\cup)=(q,\cup,-)$将使机器无法停机，我们通过令输入$x$中无$\cup$避免执行此命令。

步数	0	1	2	3	4	5	6
瞬时像	$s,\geq 010$	$s,>\ \underline{0}10$	$s,>\ 0\underline{1}0$	$s,>\ 01\underline{0}$	$s,>\ 010\underline{\cup}$	$q,>\ 01\underline{0}\cup$	$q_0,>\ 01\underline{\cup}\cup$
转移函数	$\delta(s,>)=(s,>,\rightarrow)$	$\delta(s,0)=(s,0,\rightarrow)$	$\delta(s,1)=(s,1,\rightarrow)$	$\delta(s,0)=(s,0,\rightarrow)$	$\delta(s,\cup)=(q,\cup,\leftarrow)$	$\delta(q,0)=(q_0,\cup,\rightarrow)$	$\delta(q_0,\cup)=(s,0,\leftarrow)$

步数	7	8	9	10	11	12	13
瞬时像	$s,>\ 01\underline{\cup}0$	$q,>\ 0\underline{1}\cup 0$	$q_1,>\ 0\cup\underline{\cup}0$	$s,>\ 0\underline{\cup}10$	$q,>\ \underline{0}\cup 10$	$q_0,>\ \cup\underline{\cup}10$	$s,>\ \underline{\cup}010$
转移函	$\delta(s,\cup)=$	$\delta(q,1)=$	$\delta(q_1,\cup)=$	$\delta(s,\cup)=$	$\delta(q,0)=$	$\delta(q_0,\cup)=$	$\delta(s,\cup)=$
步数	14	15					
瞬时像	$q,\geq\cup 010$	$h,>\underline{\cup}010$					
转移函数	$\delta(q,>)=(h,>,\rightarrow)$						

图 3.1-3 图灵机$\mathbf{M}$的关于 010 的运行

下面我们形式地定义图灵机的计算-----瞬时像。瞬时像表示一个时刻计算的当前状态、字符串及指针位置。于是，图灵机$\mathbf{M}$的一个瞬时像被定义为一个三元组(q,w,u)，其中$q\in K$为当前状态，$w,u\in\Sigma^*$且指针恰指着w的最后一个字符，u为指针右边的串，亦可能为空串，如，例 3.1 中对应瞬时像为：$(s,>,010)$，$(s,>0,10)$，$(s,>010,\varepsilon)$，$(s,>010\cup,\varepsilon)$等。

瞬时像(q,w,u)可在一步内得到瞬时像(q',w',u')，记为$(q,w,u)\xrightarrow{M}(q',w',u')$，此时，设$w=w_0\mu$，$u=\tau u_0$，其中，$w_0,u_0\in\Sigma^*$，$\mu,\tau,\sigma\in\Sigma$，对于$\delta(q,\sigma)=(p,\rho,D)$，则有$q'=p$，而且，若$D=\leftarrow$，则$w'=w_0$，$u'=\rho\tau u_0=\rho u$；若$D=\rightarrow$，则$w'=w_0\rho\tau$，$u'=u_0$；若$D=-$，则$w'=w_0\rho$，$u'=u$.归纳地，可定义由瞬时像$(q,w,u)$在$k$步得到瞬时像$(q',w',u')$，记为$(q,w,u)\xrightarrow{\mathbf{M}^k}(q',w',u')$，$k\geq 0$。更一般的，称由瞬时像$(q,w,u)$得到瞬时像$(q',w',u')$，记为$(q,w,u)\xrightarrow{\mathbf{M}^*}(q',w',u')$，若存在$k$，使得$(q,w,u)\xrightarrow{\mathbf{M}^k}(q',w',u')$。如，例 3.1，有$(s,>,010)\xrightarrow{\mathbf{M}^*}(h,>\cup,010)$。

例3.2 对于正整数n，设计一个图灵机，计算$n+1$。

令$\Sigma=\{0,1,>,\cup\}$，$K=\{s,q,h\}$，定义转移函数δ如图 3.1-4，将 *n* 表示为二进制 *x*，然后计算 *n*+1 的二进制 *x*+1，如，令 *x*=11。

$p\in K,\sigma\in\Sigma$	$(s,0)$	$(s,1)$	$(s,\cup)$	$(s,>)$	$(q,0)$	$(q,1)$	$(q,\cup)$	$(q,>)$
$\delta(p,\sigma)$	$(s,0,\rightarrow)$	$(s,1,\rightarrow)$	$(q,\cup,\leftarrow)$	$(s,>,\rightarrow)$	$(q_0,\cup,\rightarrow)$	$(q_1,\cup,\rightarrow)$	$(q,0,\leftarrow)$	$(p',>,\rightarrow)$
	$(q_0,\cup)$	$(q_1,\cup)$	$(q_0,1)$	$(q_0,0)$	$(q_1,0)$	$(q_1,1)$	$(q_0,>)$	$(q_1,>)$
	$(s',0,\leftarrow)$	$(s',1,\leftarrow)$	$(s',0,\leftarrow)$	$(s',0,\leftarrow)$	$(s'',1,\leftarrow)$	$(s'',1,\leftarrow)$	$(s',>,\rightarrow)$	$(s',>,\rightarrow)$
	$(s',>)$	$(s',0)$	$(s',1)$	$(s',\cup)$	$(q',0)$	$(q',1)$	$(q',>)$	$(q',\cup)$
	$(s',>,\rightarrow)$	$(s',0,\rightarrow)$	$(s',1,\rightarrow)$	$(q',\cup,\leftarrow)$	$(h,1,-)$	$(q',0,\leftarrow)$	$(h,>,\rightarrow)$	$(q',\cup,\leftarrow)$
	$(p,\cup)$	$(p,0)$	$(p,1)$	$(p,>)$				
	$(s',0,\rightarrow)$	$(s',0,\rightarrow)$	$(s',1,\rightarrow)$	$(s',>)$				

$$(s,>,11)\rightarrow(s,>1,1)\rightarrow(s,>11,\varepsilon)\rightarrow(s,>11\cup,\varepsilon)\rightarrow(q,>11,\cup\varepsilon)\rightarrow(q_1,>1\cup\cup,\varepsilon)$$
$$\rightarrow(s',>1\cup,1\varepsilon)\rightarrow(q,>1,\cup1\varepsilon)\rightarrow(q_1,>\cup\cup,1\varepsilon)\rightarrow(s,>\cup,11\varepsilon)\rightarrow(q,>,\cup11\varepsilon)$$
$$\rightarrow(p,>\cup,11\varepsilon)\xrightarrow{M^2}(s',>01,1\varepsilon)\rightarrow(s',>011,\varepsilon)\rightarrow(s',>011\cup,\varepsilon)\rightarrow(q',>011,\varepsilon)$$
$$\rightarrow(q',>01,0\varepsilon)\rightarrow(q',>0,00\varepsilon)\rightarrow(h,>,100\varepsilon)$$

故$\mathbf{M}(x)=100$。

机器**M**主要工作如下：首先利用例 3.1 在$>$与输入之间插入 0，然后以 x 的最后一个字符开始往前，若遇到 1，则将其利用$\delta(q',1)=(q',0,\leftarrow)$将其变为 0，继续左移；若遇到 0，则将 0 变为 1 并停机，输出带上内容即可。

以上例子是计算串到串的函数，其均进入状态 h 终止，输出为带上的内容。下面，我们给出一个例子，没有计算输出结果，仅是在状态“yes”或“no”停机，来表示接受或拒绝输入。

例3.3 判定一个字符串是否为回文。

定义具有如下转移函数 δ 的机器 **M**。（如图 3.1-5）

$p\in K$ $\sigma\in\Sigma$	$(s,0)$	$(s,1)$	$(s,>)$	$(s,\cup)$	$(q_0,0)$	$(q_0,1)$	$(q_0,\cup)$
$\delta(p,\sigma)$	$(q_0,>,\rightarrow)$	$(q_1,>,\rightarrow)$	$(s,>,\rightarrow)$	$(yes,\cup,-)$	$(q_0,0,\rightarrow)$	$(q_0,1,\rightarrow)$	$(q_0',\cup,\leftarrow)$
$p\in K$ $\sigma\in\Sigma$	$(q_1,0)$	$(q_1,1)$	$(q_1,\cup)$	$(q_0',0)$	$(q_0',1)$	$(q_0',>)$	$(q_1',0)$
$\delta(p,\sigma)$	$(q_1,0,\rightarrow)$	$(q_1,1,\rightarrow)$	$(q_1',\cup,\leftarrow)$	$(q,\cup,\leftarrow)$	$(no,1,-)$	$(yes,\cup,\rightarrow)$	$(no,1,-)$
$p\in K$ $\sigma\in\Sigma$	$(q_1',1)$	$(q_1',>)$	$(q,0)$	$(q,1)$	$(q,>)$		
$\delta(p,\sigma)$	$(q,\cup,\leftarrow)$	$(yes,>,\rightarrow)$	$(q,0,\leftarrow)$	$(q,1,\leftarrow)$	$(s,>,\rightarrow)$		

图 3.1-5 回文图灵机

若输入为回文，则 **M** 进入状态"yes"停机，并接受；否则进入状态"no"并拒绝。具体操作如下：在状态 s，机器 **M** 寻找输入的第一个符号 σ_1，将其改为$>$，并将 σ_1 存入状态，从而进入状态 q_{σ_1}，然后 **M** 指针向右移动直到遇到第一个 $\cup$，然后进入状态 q_{σ_1}'，并向左移动扫描输入的最后一个符号 σ_n，若 $\sigma_1=\sigma_n$，则用 $\cup$ 代替 σ_n；然后用新状态 q 向右移动直到$>$。若在某一点最后文字与所计入状态的文字不同，则进入"no"状态，并停机。重复上述过程，直到进入停机状态，并输出"yes"或"no"。

如 $x=101$,则计算如下：

$$(s,\triangleright,101)\xrightarrow{M}(s,\triangleright 1,01)\xrightarrow{M}(q_1,\triangleright\triangleright 0,1)\xrightarrow{M^3}(q_1',\triangleright\triangleright 01,\cup)\xrightarrow{M}(q,\triangleright\triangleright 0,\cup\cup)$$
$$\xrightarrow{M}(q,\triangleright\triangleright,0\cup\cup)\xrightarrow{M}(s,\triangleright\triangleright 0,\cup\cup)\xrightarrow{M}(q_0,\triangleright\triangleright\triangleright,\cup\cup)\xrightarrow{M}(q_0,\triangleright\triangleright\triangleright\cup,\cup)$$
$$\xrightarrow{M}(q_0',\triangleright\triangleright\triangleright,\cup\cup)\xrightarrow{M}(yes,\triangleright\triangleright\triangleright,\cup\cup).$$

对于输入 ε，$(s,>,\varepsilon)\xrightarrow{M}(s,>\cup,\varepsilon)\longrightarrow(yes,>\cup,\varepsilon)$。

对于输入 $x=100$，我们有

$$(s,\triangleright,100)\xrightarrow{M}(s,\triangleright 1,00)\xrightarrow{M}(q_1,\triangleright\triangleright 0,0\varepsilon)\xrightarrow{M^3}(q_1',\triangleright\triangleright 00,\cup)\longrightarrow(no,\triangleright\triangleright 01,\cup\varepsilon)。$$

现在我们分析该机器的效率。对于回文 $x=\sigma_1\sigma_{2\ldots}\sigma_n$，在机器执行期间，作为串的两个界$>$与$\cup$在不断的内缩，所剩余的串恰好为要证明是回文的串。如此执行一次，串长缩短 2，机器至多需要执行$\lceil n/2\rceil$次，每次机器指针要扫描$2(n-2(i-1))+1$次符号，即移动$2(n-2(i-1))+1$次，故机器需要移动$\sum_{i=1}^{\lceil n/2\rceil}(2(n-2(i-1))+1)=\lceil n/2\rceil\frac{2n+1+1}{2}$，即，机器指针移动次数为$O(n^2)$。

第二节　多带图灵机、时间与空间

下面，我们进一步推广图灵机的定义得到多带图灵机。

定义3.2　对于整数$k \geq 1$，一个k-带图灵机是一个四元组$\mathbf{M}=(K,\Sigma,\delta,s)$，其中$K,\Sigma,s$如通常的图灵机，如以前定义，转移函数$\delta$可以决定下一个状态，也决定每条带上哪个符号被改写和指针移动方向，是一个从$K\times\Sigma^k$到$(K\bigcup\{h,"yes","no"\})\times(\Sigma\times\{\rightarrow,\leftarrow,-\})^k$的映射，即，对于$(q\ \sigma_1\cdots\sigma_k$，有$\delta(q\ \sigma_1\cdots\sigma_k)=p\ \rho_1D_1\cdots\rho_kD_k$。即，若$\mathbf{M}$在状态$q$，第$i$条带上指针指着符号$\sigma_i$，$i=1,\cdots,k$，则接下来进入状态为$p$，在第$i$条带上将$\sigma_i$改写成$\rho_i$，然后指针沿方向$D_i$移动一次，$i=1,\cdots,k$。这里$>$仍然不能被改写或指针移动到其左边。若$\sigma_i=>$，则$\rho_i=>$且$D_i=\rightarrow$。初始，所有带上都以$>$开始，第一条带上含有输入$x$，除了机器在计算函数时，输出可以在机器停机时从第$k$条带上输出外，$k$-带图灵机的计算与通常机器一样。

单带图灵机的性质，在多带图灵机上的每条带上都成立。我们定义k-带图灵机的瞬时像为一个$(2k+1)$元组$(q,w_1,u_1,\cdots,w_k,u_k)$，其中$q$为当前状态，在第$i$条带上有字符串$w_i,u_i$，且指针指向$w_i$的最后一个字符。称$(q,w_1,u_1,\cdots,w_k,u_k)\xrightarrow{M}(q',w_1',u_1',\cdots,w_k',u_k')$为由$(q,w_1,u_1,\cdots,w_k,u_k)$一步得到$(q',w_1',u_1',\cdots,w_k',u_k')$。类似可定义"在$n$步内得到"。对于一个$k$-带图灵机$\mathbf{M}$，关于输入$x$，从瞬时像$(s,>,x,>,\varepsilon,\cdots,>,\varepsilon)$开始，即$x$在第一条带，且所有的带都以$>$开始。

若$(s,\triangleright,x,\triangleright,\varepsilon,\cdots,\triangleright,\varepsilon)\xrightarrow{\mathbf{M}^*}("yes",w_1,u_1,\cdots,w_k,u_k)$，则称$\mathbf{M}(x)="yes"$。

若$(s,\triangleright,x,\triangleright,\varepsilon,\cdots,\triangleright,\varepsilon)\xrightarrow{\mathbf{M}^*}("no",w_1,u_1,\ldots,w_k,u_k)$，则称$\mathbf{M}(x)="no"$。

若$(s,\triangleright,x,\triangleright,\varepsilon,\cdots,\triangleright,\varepsilon)\xrightarrow{\mathbf{M}^*}(h,w_1,u_1,\ldots,w_k,u_k)$，则称$\mathbf{M}(x)=y=w_ku_k$。即，若$\mathbf{M}$在状态 h 停机，则该计算的输出为最后一条带上的串，这是一个在$>$后开始并除掉了后面多余"$\cup$"的串。

显然，通常的图灵机就是$k=1$时的k-带图灵机。一旦定义了$\mathbf{M}(x)$，可以简单的将函数计算、语言的判定和接受的定义推广到多带图灵机。从此我们将以多带图灵机作为定义图灵机计算的时间（空间）的基础。对于一个k-带图灵机$\mathbf{M}$和输入x，若有$(s,\triangleright,x,\triangleright,\varepsilon,\cdots,\triangleright,\varepsilon)\xrightarrow{\mathbf{M}^t}(H,w_1,u_1,\ldots,w_k,u_k)$，$H\in\{h,"yes","no"\}$，则称关于输入$x$，$\mathbf{M}$需要时间$t$，即所需的时间仅仅是到停机的步数。若$\mathbf{M}(x)=\nearrow$，时间被认为是$\infty$。

例3.4　利用 2-带图灵机判定回文，定义图灵机如下（图 3.2-1）。

多带图灵机

单带是n方
2带是n

开始，机器先将输入复制到第二条带，接下来，令第一条带的指针指向输入的第一个符号，第二条带的指针指向输入的最后一个符号。然后，两指针以相对方向同时移动，检查所指两个符号是否相同。若相同，则擦去第二条带上所复制的符号；若不相同，则进入状态 *no*，并停机。

机器需要移动的次数：将输入复制到了第二条带，需要指针移动 2(*n*+1)，然后，两指针移动，检查符号是否相同，并擦去复制品，需要 *n*+1，因此共需要 3(*n*+1)次，即

用 2-带图灵机判定回文需要 $O(n)$次移动。这与前面单带图灵机比较，时间由 2 次将为 1 次。

$(p,\sigma_1,\sigma_2)\in K\times\Sigma^2$	$(s,0,\cup)$	$(s,1,\cup)$	$(s,>,>)$	$(s,\cup,\cup)$	$(q,0,\cup)$	$(q,1,\cup)$
$\delta(p,\sigma_1,\sigma_2)$	$(s,0,\rightarrow,0,\rightarrow)$	$(s,1,\rightarrow,1,\rightarrow)$	$(s,>,\rightarrow,>,\rightarrow)$	$(q,\cup,\leftarrow,\cup,\leftarrow)$	$(q,0,\leftarrow,\cup,-)$	$(q,1,\leftarrow,\cup,-)$
$(p,\sigma_1,\sigma_2)\in K\times\Sigma^2$	$(q,>,\cup)$	$(p,0,0)$	$(p,1,1)$	$(p,0,1)$	$(p,1,0)$	$(p,\cup,>)$
$\delta(p,\sigma_1,\sigma_2)$	$(p,>,\rightarrow,\cup,\leftarrow)$	$(p,0,\rightarrow,\cup,\leftarrow)$	$(p,1,\rightarrow,\cup,\leftarrow)$	$(no,0,-,1,-)$	$(no,1,-,0,-)$	$(yes,\cup,-,>,\rightarrow)$

图 3.2-1 判定回文的 2-带图灵机

前面仅完成了对于单个实例的计算时间的定义，我们需要进一步定义时间使得对一个问题的任何实例都可以反映求解的时间。由例 3.4 可以看出，实例的大小"n"可以用于对计算实例所用的时间（空间）进行刻画，即 n 的函数。

设 f 为非负整数到非负整数的函数，称 $\mathbf{M}$ 在时间 $f(n)$ 内运算，如果对于任意输入串 x， $\mathbf{M}$ 计算的时间$\leq f(|x|)$，即， $f(|x|)$是 $\mathbf{M}$ 的时间界。

设 $L\subseteq(\Sigma\setminus\{\cup\})^*$ 在时间 $f(n)$ 内被一个多带图灵机判定，则称 $L\in\mathbf{TIME}(f(n))$，这里 $\mathbf{TIME}(f(n))$ 是一个语言集合，其所含的每个语言可由一个多带图灵机在时间 $f(n)$ 内判定，因此 $\mathbf{TIME}(f(n))$ 为一个复杂类。显然，当 $f(n)<g(n)$ 时，有 $\mathbf{TIME}(f(n))\subseteq\mathbf{TIME}(g(n))$。特别的，定义$\boldsymbol{P}=\bigcup_{k\geq0}\mathbf{TIME}(n^k)$，这是最中心的复杂类。

在例 3.3 中，单带图灵机判定回文的时间为$O(n^2)$，而例 3.4 中 2-带图灵机判定回文的时间为 $O(n)$，由此可见，多带图灵机在效率上可节省 2 次方幂。这是一个一般性的事实。

2022年0307
下一节课证明

定理 3.1 设 $\mathbf{M}$ 为在时间 $f(n)$ 内运行的 k-带图灵机，则可以构造一个在时间 $O(f(n)^2)$ 内运行的图灵机 $\mathbf{M}'$，使得对于任意输入 x，有 $\mathbf{M}(x)=\mathbf{M}'(x)$。

证明：设 $\mathbf{M}=(K,\Sigma,\delta,s)$，去构造 $\mathbf{M}'=(K',\Sigma',\delta',s)$，使得单带 $\mathbf{M}'$ 可以模拟 k-带 $\mathbf{M}$。

直观的方法：将 $\mathbf{M}$ 的各条带上的串在 $\mathbf{M}'$ 的带上连缀起来。当然，串之间没有">"，以便指针可以前后移动。

但是，该方法存在如下问题：

1) 如何区分 $\mathbf{M}$ 的不同带上的串（即，串的右端的位置）？
2) 如何记住 $\mathbf{M}$ 的每个串上的指针的位置？

为此，我们引入新的字母 $\{>',<\}$ 用于表示 $\mathbf{M}$ 的一个串的开始和终点；$\underline{\Sigma}=\{\underline{\sigma}\mid\sigma\in\Sigma\}$表示指针所指符号集合。故令 $\Sigma'=\Sigma\cup\underline{\Sigma}\cup\{\triangleright',<,<'\}$，其中$>'$表示一个串的开始，指针可以通过并到其左端。$<$ 表示一个串的右端，指针可以通过。于是，$\mathbf{M}$ 的任意瞬时像 $(q,w_1,u_1,\cdots,w_k,u_k)$ 可由 $\mathbf{M}'$ 的瞬时像 $(q,\triangleright w_1'u_1\triangleleft w_2'u_2\triangleleft\cdots w_k'u_k\triangleleft\triangleleft)$ 模拟，其中，若 $w_i=>w_{i0}\sigma_i$，则 $w_i'=>'w_{i0}\underline{\sigma}$。最后，用两个$\triangleleft\triangleleft$表示 $\mathbf{M}$ 的串结束。

模拟开始：$\mathbf{M}'$将输入右移一位，在其前面加上$>'$；在输入后面写上$<(>'<)^{k-1}<$。这可以通过添加至多$2k+2$个新的状态到$\mathbf{M}'$的状态中实现该写入。

模拟$\mathbf{M}$的一次移动：$\mathbf{M}'$从左到右扫描串并返回，执行两次。第一次扫描，$\mathbf{M}'$收集关于$\mathbf{M}$中k个当前被指针所指符号，即k个有下划线的符号。要记住这些符号，$\mathbf{M}'$必须添加一些新的状态，每个状态对应于$\mathbf{M}$的一个状态和一个k-元符号组的特定组合$(q,\sigma_1,\cdots,\sigma_k)$，表示当$\mathbf{M}$处于状态$q$时，指针指着$(\sigma_1,\cdots,\sigma_k)$。

第二次扫描：根据第一次扫描收集的信息，$\mathbf{M}'$知道需要执行哪些操作才能反映出$\mathbf{M}$的移动。于是，$\mathbf{M}'$从左向右扫描串，在每个有下划线的符号处改写附近的一个或两个符号，并移动指针。这反映了$\mathbf{M}$在相应带上符号的改写。根据$\mathbf{M}'$的得到的信息，这些很容易实现。

但是，有一点值得注意：若$\mathbf{M}$的一条带上指针在串的右端，并需要继续向右移动，则$\mathbf{M}'$的带上，对应的部分串必须为新符号创造一个空间（比如$\cup$）。于是执行如下：

先用一个特殊符号标记当前的$<$（如记为$<'$），移动$\mathbf{M}'$的指针到右末端的$<<$，向右移动所有符号一个位置（如例 3.1）。当遇到$<'$时，将其右移一个位置，并改写为$<$，并在$<'$的原位置处写$\cup$，然后执行$\mathbf{M}$的下一个串的变化。

该模拟继续进行直到$\mathbf{M}$停止，此时，$\mathbf{M}'$擦掉$\mathbf{M}$的串，仅留下$\mathbf{M}$的最后一条串，即为输出$\mathbf{M}'(x)=\mathbf{M}(x)$。

现在分析$\mathbf{M}'$模拟$\mathbf{M}$所需要的时间。关于输入x，由于$\mathbf{M}$在$f(|x|)$步内停机，故$\mathbf{M}$各带上串的长$\le f(|x|)$。于是$\mathbf{M}'$的串的总长$\le k(f(|x|)+1)+1$。模拟$\mathbf{M}$的一次移动要有至多两次来往扫描，则至多有$4k(f(|x|)+1)+4$步。在模拟$\mathbf{M}$的每个串时，为了在某个位置添加一个"$\cup$"，需要将此位置后面的所有字符右移一格，这需要至多$3k(f(|x|)+1)+3$步。于是，至多需要$(f(|x|)+1)(4k(f(|x|)+1)+4+k(3k(f(|x|)+1)+3))$，即$\boldsymbol{O}(k^2 f(|x|)^2)$。由于$k$为固定的，且与$x$无关，故有$\boldsymbol{O}(f(|x|)^2)$。

注意，关于带数的依赖性可以通过不同的模拟避免，但f的次数不能改变。另外，作为一个计算模型，添加有限多条带不能增加其计算能力，对效率影响仅为多项式。

空间：设$k\ge 2$是一个整数，一个有输入-输出带的k-带图灵机是一个通常的k-带图灵机，其转移函数δ满足，若$\delta(q,\sigma_1,\ldots,\sigma_k)=(p,\rho_1,D_1,\cdots,\rho_k,D_k)$，则有 a）$\rho_1=\sigma_1$；b）$D_k\ne\leftarrow$； c）若$\sigma_1=\cup$，则$D_1=\leftarrow$。这里，a）表明第一条带（即，$k=1$）是一条只读带，称之为输入带；b）表明最后一条带（即，第k条带）的指针只能向右移动，是一条只写带，称之为输出带；c）确保输入带上的指针不会离开输入进入"$\cup$"中。除了第一条带和第k条带外，称其余的带为工作带。下列命题说明，条件 a），b），c）并没有改变图灵机的能力。

命题 3.1 对于在时间界 $f(n)$ 内运算的任意 k-带图灵机 $\mathbf{M}$，存在一个带有输入和输出带的 $(k+2)$-带图灵机 $\mathbf{M}'$，并在时间界 $O(f(n))$ 内运行。

证明： $\mathbf{M}'$ 首先将输入复制到第二条带，然后在第 2 至第 $k+1$ 条带上模拟 $\mathbf{M}$ 运行。当 $\mathbf{M}$ 停机时，$\mathbf{M}'$ 复制其输出到第 $k+2$ 条带上并停机。

基于限制 a），b），c），我们可以得到空间的定义：

定义3.3 对于一个 k-带图灵机 $\mathbf{M}$ 和一个输入 x。设 $H \in \{h,"yes","no"\}$ 为一个停机状态，若 $(s,\triangleright,x,\triangleright,\varepsilon,\cdots,\triangleright,\varepsilon) \xrightarrow{\mathbf{M}^t} (H,w_1,u_1,\ldots,w_k,u_k)$，则关于输入 x，$\mathbf{M}$ 所要求空间为 $\sum_{i=1}^{k}|w_i u_i|$；但是对于带有输入-输出带的 $\mathbf{M}$，则其所要求的空间为 $\sum_{i=2}^{k-1}|w_i u_i|$。

设 $f:\mathbb{N}\to\mathbb{N}$ 为一个函数，称图灵机 $\mathbf{M}$ 在空间界 $f(n)$ 内运作，若对于任意输入 x，$\mathbf{M}$ 所要求的空间至多为 $f(|x|)$。令 L 为一个语言，称 L 属于空间复杂类 $\boldsymbol{SPACE}(f(n))$，若存在一个带有输入-输出带的图灵机 $\mathbf{M}$ 在空间 $f(n)$ 内判定 L。定义语言类 $\boldsymbol{PSPACE} = \bigcup_{k\geq 0} \boldsymbol{SPACE}(n^k)$，$\mathbb{L} = \boldsymbol{SPACE}(\log n)$。

例3.5 回文的判定：考虑 3-带图灵机 $\mathbf{M}$（如图 3.2-2）：第一条带含输入 $x=\sigma_1\cdots\sigma_k$，并且不能被改写；第二条带含数 i 的二进制形式；第三条带含数 j 的二进制形式。

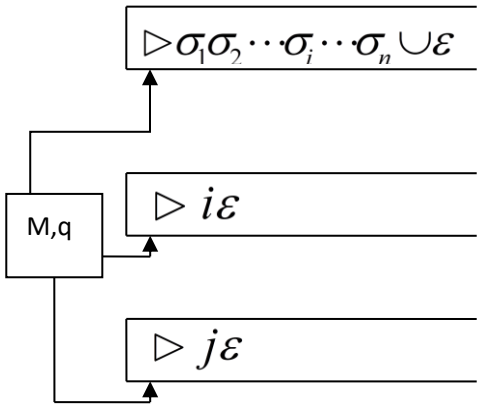

图 3.2-2 判定回文的 3-带图灵机

在第一阶段，$i=1$，比较 i 与 j，若 $i=j=1$，则将 σ_1 存入状态，然后将输入带的指针移动到最后一个符号 σ_n 处，比较 x 的最后一个符号与 σ_1 是否相等。若不等，则以状态“no”停机；若相等，则 $i\leftarrow i+1$，而且 $j=1$。并开始下一个阶段。

第 i 阶段，此时 $i\geq 2$。比较 i 与 j（这可以通过移动第 2，3-条带的指针即可）。若 $j<i$，则 $j\leftarrow j+1$，并将第一条带的指针从 σ_1 处向右移动，扫描下一个输入的符号，如此运行直到 $i=j$。则将 σ_i 存入状态，然后将输入带的指针移动到最后一个符号 σ_n 处，再向

左移动输入带指针；同时，每向左移动一次输入带指针，则 $j \leftarrow j-1$，直到 $j=1$。此时，输入带指针恰好指在 x 的倒数第 i 个符号处。再检查 σ_i 是否与 x 的倒数第 i 个符号相等。若相等，则 $i \leftarrow i+1$， j=1并且第一条带指针回到 σ_1 处；若不等，则以状态“no”停机。

最后，若有个阶段 i_t， x 的第 i_t 个符号为∪，则 $i_t > n$。可断定输入是一个回文，则以状态“yes”停机。

该机器运行所需要的空间即为第 2，3 条带上字符串 i 与 j 所占的空间，为 $|i|+|j| \le 2\log n$。于是，回文组成语言类在空间复杂类 $\boldsymbol{PSPACE}(\log \boldsymbol{n})$。

对于我们定义复杂类而言，一旦我们采用多带图灵机作为时间与空间的复杂性定义的模型，在时间和空间上增加或减少一个常数因子不会给它们带来任何变化。具体的说，有下面两个定理，其证明技巧类似于定理 3.1。

定理 3.2 (线性加速定理) 对于任意 $\varepsilon > 0$，$\boldsymbol{TIME}(f(n)) = \boldsymbol{TIME}(\varepsilon f(n) + n + 2)$。

由此可见，对于任意多项式 $f(n) \ge n$，若 $f(n)$ 为线性，则 $TIME(f(n)) = TIME(n)$；若 $f(n)$ 是非线性，则首项系数可以任意小而且时间界可以包含任意线性界，因此可以忽略低阶项，即，若 $f(n) = n^t + c_1 n^{t-1} + \mathrm{L} + c_t$，则 $TIME(f(n)) = TIME(n^t)$，因此复杂类

$$\boldsymbol{P} = \bigcup_{\substack{f\text{为非负}\\ \text{多项式}}} TIME(f(n)) = \bigcup_{t \ge 0} TIME(n^t)$$

显然，有如下结论：

1）$\mathbf{HORNSAT} \in \boldsymbol{P}$

2）$\mathbf{CIRCUIT\ VALUE} \in \boldsymbol{P}$

3）对于任意 HORN 存在二阶逻辑表达式 $\exists P\phi$，问题 $\exists P\phi\text{-GRAPHS} \in \boldsymbol{P}$。

定理 3.3 (空间压缩定理) 对于任意 $\varepsilon > 0$，有 $\boldsymbol{SPACE}(f(n)) = \boldsymbol{SPACE}(\varepsilon f(n) + 2)$。特别地，对于 k 次多项式 f，有 $\boldsymbol{SPACE}(f(n)) = \boldsymbol{SPACE}(n^k)$。于是，$\mathbf{PSPACE} = \bigcup_{\substack{f\text{为非负}\\ \text{多项式}}} SPACE(f(n))$。

第三节 非确定图灵机

下面介绍一种非现实的计算模型----非确定图灵机。顾名思义，它的移动不能被转移函数唯一确定，即，非确定图灵机的转移函数是一个多值函数。以指数效率为代价，它可以被确定图灵机模拟。这例证了 Church-Turing 命题。然而，该指数代价是问题固有的还是由于我们理解的局限性造成的，目前还没有明确的答案，此即为著名的问题

$\boldsymbol{P}\overset{?}{=}\boldsymbol{NP}$。

一个非确定图灵机是一个四元组 $\boldsymbol{N}=(K,\Sigma,\Delta,s)$，其中，$K,\Sigma,s$ 如前，但是每次移动后的下一次移动不是唯一确定的，而是在一组移动中选择一个。因此，Δ 不再是 $K\times\Sigma$ 到 $\left(K\cup\{"h","yes","no"\}\right)\times\Sigma\times\{\rightarrow,\leftarrow,-\}$ 的函数，而是一个关系

$$\Delta\subseteq(K\times\Sigma)\times\left[\left(K\cup\{"h","yes","no"\}\right)\times\Sigma\times\{\rightarrow,\leftarrow,-\}\right]$$

即，对于每个状态-符号组合 (q,σ)，或存在不止一个下一步移动的合适选择或者没有下一步移动。

与确定图灵机一样，我们定义非确定图灵机的瞬时像，但“得到”不再是一个函数，而是一个关系。称由非确定图灵机的瞬时像 (q,w,u) 一步得到瞬时像 (q',w',u')，记为 $(q,w,u)\xrightarrow{N}(q',w',u')$，如果存在一个移动 $\left((q,\sigma),(q',\rho,D)\right)\in\Delta$ 使得：对于 $w=w_0\sigma$，$u=\tau u_0$，有 a）$D=\rightarrow$，$w'=w_0\rho\tau$，$u'=u_0$；或 b）$D=\leftarrow$，$w'=w_0$，$u'=\rho u$；或 c）$D=-$，$w'=w_0\rho$，$u'=u$ 发生。

类似的，可定义一般情形 $\xrightarrow{N^k}$，$\xrightarrow{N^*}$，但需要注意这些都是关系，而不是函数。

设 L 为一个语言，$\boldsymbol{N}$ 为一个非确定图灵机，称 $\boldsymbol{N}$ 判定 L，如果对于任意 $x\in\Sigma^*$，有 $x\in L$ 当且仅当 $(s,>,x)\xrightarrow{N^*}("yes",w,u)$，这里 w、$u\in\Sigma^*$。

由该定义可见，如果存在一系列非确定选择，使得以“yes”状态结束，则称输入被接受；而对于其他的选择也许都被拒绝。若没有选择序列使得机器进入接受，则称输入被拒绝。这种非对称性与图灵机接受（见定义 3.2）很类似。

称非确定图灵机 $\boldsymbol{N}$ 在时间 $f(n)$ 内判定语言 L，如果 $\boldsymbol{N}$ 判定 L，而且对于任意 $x\in\Sigma^*$，若 $(s,\triangleright,x)\xrightarrow{N^k}(q,w,u\ )$，则 $k\le f\left(|x|\right)$，这里的 $f\left(|x|\right)$ 是 $\boldsymbol{N}$ 的计算活动的最长路径。显然，所有计算活动（即，所有路径）的总和将是指数大。类似于确定图灵机，可以直接定义带有输入-输出带的多带非确定图灵机。定义复杂类 $\boldsymbol{NTIME}\left(f(n)\right)$ 为在时间 $f(n)$ 内由非确定图灵机判定的语言类的集合。类似于确定性图灵机，对于非确定图灵机有线性加速定理：

定理 3.4 对于任意的 $\varepsilon>0$，有 $\boldsymbol{NTIME}(f(n))=\boldsymbol{NTIME}(\varepsilon f(n)+n+2)$。

于是，对于任意 k -次多项式 $f(n)$，有 $\boldsymbol{NTIME}(f(n))=\boldsymbol{NTIME}(n^k)$。定义 $\boldsymbol{NP}=\bigcup_{k\ge0}\boldsymbol{NTIME}(n^k)$。显然，$\boldsymbol{P}\subseteq\boldsymbol{NP}$。

例3.6 $TSP(D)$ 问题这是一个经典的问题。假设有 n 个城市，分别记为 1,2,…,n。任意两个城市 i 和 j 之间有距离 d_{ij} 且 $d_{ij}=d_{ji}$，问能否找到一个最短的旅行路线，即，

找到一个{1,2,…,n }上的置换π使得$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)}$尽可能地小，称之为 TSP。类似于 MAX FLOW(D)，可以定义其判定性问题 TSP(D)：*给定非负整数B及矩阵(d_{ij})，其中$d_{ij}>0$，问是否有一个长度小于等于B的旅行线路？*

显然，穷举所有可能可以解此问题，但代价是所用时间与$n!$成比例（即，$\frac{1}{2}(n-1)!$）；由于需要存储当前的置换和最佳路线，故需要空间与n成比例。

目前，我们不知道$TSP(D)$是否在$\boldsymbol{P}$中，但易知$TSP(D)\in \boldsymbol{NP}$，因为这可以在时间$O\left(n^2\right)$内有一个非确定图灵机$\boldsymbol{N}$判定。事实上，设$x$为$TSP(D)$的一个实例表示，关于含$x$的输入，$\boldsymbol{N}$继续写一个长度不超过$|x|$的任意符号序列 y，然后$\boldsymbol{N}$返回并检查是否$y$为城市的一个置换的表示。若是一个置换，则检查是否该置换的旅行代价≤B。这两项任务可在时间$\boldsymbol{O}\left(n^2\right)$内完成（必要时可用另外一条带）。若该串的确编码了一个代价比 B 小的旅行，则机器接受；否则，拒绝。

注意，这里需要有一个“猜测”置换的机器运算，然后检查代价。由这个例子可以看出，若输入是一个“*yes*”实例，则机器猜测一个恰当的置换，并验证代价≤B；也许其它的猜测都被拒绝。反过来，若发现所有计算的猜测都是错的，输入将被拒绝。因此，就非确定机器而言，只要有一个选择序列接受，则机器接受输入。

目前，我们不知道如何将非确定思想转变为一个确定多项式算法，但下列定理为我们提供了一个转变为确定的指数时间的方法。

定理 3.5 设非确定图灵机$\boldsymbol{N}$在时间$f(n)$内判定语言L，则存在一个 3-带确定图灵机$\mathbf{M}$在时间$\boldsymbol{O}(c^{f(n)})$内判定L，其中$c>1$是某依赖于$\mathbf{N}$的常数。

证明：设$N=(K,\ \Sigma,s)$，对于每个$(q,\sigma)\in K\times\Sigma$，令$C_{q,\sigma}=\{(q',\sigma',D):\sigma(q,\ \ ,)\ni(q',\ \ ,D)\}$，则$C_{q,\sigma}$为有限集。令$d=\max_{q,\sigma}|C_{q,\sigma}|$，（称$d$为$\mathbf{N}$的非确定度），即，$\boldsymbol{N}$的计算通路的每个节点的出度至多为$d$。不妨设每个节点出度均为$d>1$，否则，$\boldsymbol{N}$为确定图灵机。于是，可以将$\boldsymbol{N}$的每步运行的非确定选择$C_{q,\sigma}$中元（即，从各个节点出发的边）从左往右分别编码为$\{1,\cdots,\mathrm{d}\}$，从而$\boldsymbol{N}$的每次计算（即，如图 3.3-1 非确定模型中从开始到停机的一条计算通路）都被唯一编码为$\{1,\cdots,d\}$上的序列。下面模拟$\boldsymbol{N}$（如图 3.3-1）。

由于$\boldsymbol{N}$的任意计算是一组由$\{1,\cdots,d\}$中的t个整数组成对应于$C_{q_1,\sigma_1},\cdots,C_{q_t,\sigma_t}$中非确定选择的序列，称为非确定选择序列，因此，$\boldsymbol{N}$的任意一个$t$-步计算即为$t$个元的非确定选择序列；每个这样的非确定选择序列$(c_1,\cdots,c_t)$可以看作一个$d$-进制数，并按照数的增长排序。模拟器$\mathbf{M}$去考虑所有这样的选择序列，并在每条序列上模拟$\boldsymbol{N}$的运行（注意：由于不知道$f(n)$，故只能按照长度递增顺序逐条序列考虑）。

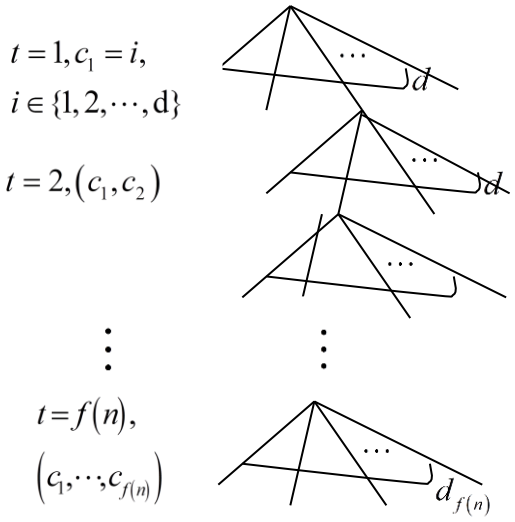

图 3.3-1 非确定计算模型

对于非确定选择序列$(c_1,\cdots,c_t)$， **M** 将其保存在第二条带，然后模拟 ***N*** 的行为。如图 3.3-2，其中$c_i\in\{1,\cdots,d\}$，$1\leqq i\leqq t$，

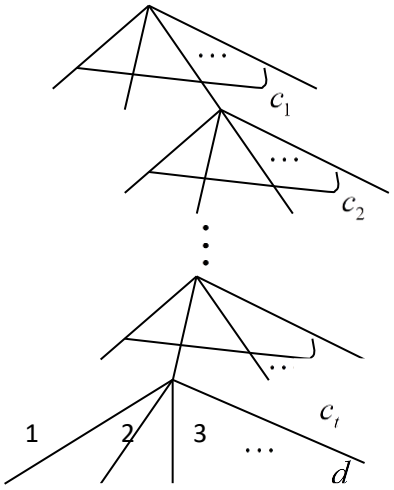

图 3.3-2　序列$(c_1,\cdots,c_t)$的模拟

t=1 步时，选择为 c_1，其按照从小到大顺序取遍{1,2,…,*d*}，共有 *d* 个选择。

t=2 步时，选择的是(c_1,c_2)，对应的计算从c_1到c_2，其中c_1与c_2均取遍按照从小到大取遍{1,2,…,*d*}。

…….

$t=i$ 步时，选择的是$(c_1,\cdots,c_i)$，对应的计算从c_1到c_i。

……

$t=f(n)$步时，选择的是（$c_1,\ldots,c_{f(n)}$），对应的计算从c_0到$c_{f(n)}$。

注意，以上各步中c_i均取遍按照从小到大顺序取遍{1,2,…,*d*}，通过计算d 进制下的整数办法生成每个非确定选择$(c_1,\cdots,c_i)$。

对于 $\boldsymbol{N}$ 的前 t 步，若在第 i 步选择是 c_i，而且这些选择可使得 $\boldsymbol{N}$ 以一个“*yes*”停机（可能早于第 t 步），则 $\mathbf{M}$ 以“*yes*”停机；否则，$\mathbf{M}$ 执行下一个序列。注意，生成下一个序列即是计算一个 d 进制下的整数（用例 3.2 可以实现）。

由于不知道 $f(n)$，$\mathbf{M}$ 如何能知道 $\boldsymbol{N}$ 拒绝？事实上，若 $\mathbf{M}$ 模拟所有长为 t 的选择序列，并且发现其中没有再继续进行的计算，即，均进入“*no*”或“*h*”状态，则 $\mathbf{M}$ 可知 $\boldsymbol{N}$ 拒绝输入。

下面分析效率：为了完成模拟，$\mathbf{M}$ 需要的时间由“需要走遍的序列数 $\sum_{t=1}^{f(n)} d^t = \boldsymbol{O}\left(d^{f(n)+1}\right)$”与“生成和考虑每条序列的时间”的积界定，而后者可由 $\boldsymbol{O}(d^{f(n)})$ 界定。故结论成立。

类似于确定图灵机，我们也可以定义带有输入-输出带的多带非确定图灵机的空间复杂度。一个 k -带非确定图灵机 $\boldsymbol{N}$ 和一个输入 $x \in \left(\Sigma \setminus \{\cup\}\right)^*$，设 $H \in \{"yes","no",h\}$ 为一个停机状态。若 $(s,>,x,\mathrm{L},>,\varepsilon) \xrightarrow{N^*} (H,w_1,u_1,\mathrm{L},w_k,u_k)$，则关于输入 x，$\boldsymbol{N}$ 所要求的空间为 $\sum_{i=1}^{k}|w_i u_i| \le f(|x|)$。但是，对于带有输入-输出带的非确定图灵机，所要求空间定义为工作带上所用空间的和 $\sum_{i=2}^{k-1}|w_i u_i|$（不计输入-输出带）。

设 $f:\mathbb{N}\to\mathbb{N}$ 为一个函数，称非确定图灵机 $\boldsymbol{N}$ 在空间界 $f(n)$ 内运行，若对于任意输入 x，$\boldsymbol{N}$ 运行到停机所需要的空间至多为 $f(|x|)$。给定一个带有输入-输出带的 k -带非确定图灵机 $\boldsymbol{N}=(K,\Sigma,\Delta,s)$，称在空间 $f(n)$ 内 $\boldsymbol{N}$ 判定语言 L，如果 $\boldsymbol{N}$ 判定 L 所需要的空间至多为 $f(|x|)$。令 $\boldsymbol{NPSPACE}(f(n))$ 为带有输入-输出带的非确定图灵机 $\boldsymbol{N}$ 在空间 $f(n)$ 内判定的语言类。对于非确定图灵机有下列空间压缩定理。

定理 3.6 对于任意 $\varepsilon>0$，有 $\boldsymbol{NSPACE}(f(n)) = \boldsymbol{NSPACE}(\varepsilon f(n)+2)$。特别地，对于一个 k 次多项式 f，有 $\boldsymbol{NSPACE}(f(n)) = \boldsymbol{NSPACE}(n^k)$。

定义语言类 $\boldsymbol{NPSPACE} = \bigcup_{k\ge 0} \boldsymbol{NPSPACE}\left(n^k\right)$，$\mathbb{NL} = \boldsymbol{NPSPACE}(\log n)$，$\boldsymbol{EXP}(f(n)) = \bigcup_{c>1} \boldsymbol{TIME}\left(c^{f(n)}\right)$。显然，$\mathbb{L}\subseteq\mathbb{NL}$，$\boldsymbol{PSPACE}\subseteq\boldsymbol{NPSPACE}$，$\boldsymbol{NTIME}(f(n))\subseteq\boldsymbol{EXP}(f(n))$。

例3.7 可达性问题（REACHABILITY）

在前面我们曾经说明，可达性问题的确定性算法要求线性空间复杂度。后面我们将证明对于确定算法仅要求空间 $\boldsymbol{O}\left(\log^2 n\right)$。这里我们说明，在空间 $\boldsymbol{O}(\log n)$ 内，非确定图灵机可求解可达性问题。具体算法如下：

利用一个 3-带非确定图灵机 $\boldsymbol{N}$ （如，图 3.3-3），第一条带为输入带。

1) 初始，在第二条带上以二进制写下当前顶点 i；在另一条带上“猜测”一个整数 $j \le n$。
2) 在输入中检查邻接矩阵 (i, j) 处赋值是否为 1。若不是 1，则停机并拒绝（显然，这对 $\boldsymbol{N}$ 的计算没有影响）；若是 1，检查 j 是否为 n：若 j 是 n，则接受并停机；否则，$i \leftarrow j$。

重复上述过程 n 步，若 $j = n$ 且邻接矩阵 (i, n) 处赋值，则接受并停机；否则，拒绝（显然，这对 $\boldsymbol{N}$ 的计算没有影响）。

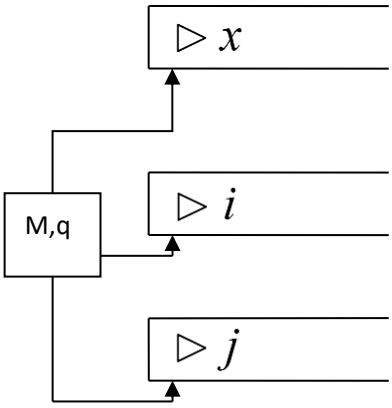

图 3.3-3 计算 REACHABILITY 的机器

易知，$\boldsymbol{N}$ 可解可达性问题；在任何时刻，两带所用空间为 $\boldsymbol{O}(\log n)$。

注意，确定算法所用空间为 $\boldsymbol{O}(\log^2 n)$，显然这比时间差距小的多。事实上，这是目前所知道的空间复杂性的最大差距，同时亦说明空间复杂性更精确的刻画了非确定性。

第四节　通用图灵机

通过前面对计算模型的介绍，似乎令人感到它们比实际的计算机功能弱。在现实中，一台计算机配以适当的程序就可以解决广泛的问题；相反，图灵机似乎仅局限于解决单个问题。但事实并非如此。本节将指出图灵机的强大功能，具有通用性，即，介绍通用图灵机。事实上，*任何有效的或机械的方法均能被通用图灵机执行*。

所谓通用图灵机 $\mathbf{U}$，当给定一个输入时，它将该输入解释为另外一个图灵机 $\mathbf{M}$ 的描述与其输入 x 的连缀，然后 $\mathbf{U}$ 将模拟以 x 为输入的 $\mathbf{M}$ 的运行。将此表示为 $\mathbf{U}(\mathbf{M}; x) = \mathbf{M}(x)$。

为了能将图灵机作为 $\mathbf{U}$ 的输入，首先要将 $\mathbf{M}$ 编码成符号串。由于 $\mathbf{U}$ 要模拟任意的图灵机 $\mathbf{M}$，故 $\mathbf{U}$ 要利用的状态和符号的个数没有预知的界。因此，将所有图灵机的状态与符号都编码为整数，进而可以用 $\{0,1\}^*$ 中的串编码。具体如下：

对于任意图灵机$\mathbf{M}=(K,\Sigma,\delta,s)$，令$\Sigma=\{1,2,\mathrm{L}\ ,|\Sigma|\}$，$K=\{|\Sigma|+1,|\Sigma|+2,\mathrm{L}\ ,|\Sigma|+|K|\}$，其中，起始状态$s$总是编码为$|\Sigma|+1$。另外，对于$\rightarrow$，$\leftarrow$，$-$，$h$，“*yes*”与“*no*”，做如下编码：

M 状态	$\rightarrow$	$\leftarrow$	$-$	h	*yes*	*no*
对应编码	$\|K\|+\|\Sigma\|+1$	$\|K\|+\|\Sigma\|+2$	$\|K\|+\|\Sigma\|+3$	$\|K\|+\|\Sigma\|+4$	$\|K\|+\|\Sigma\|+5$	$\|K\|+\|\Sigma\|+6$

图 3.4-1 方向与停机状态的编码

所有数都可以用$\lceil\log(|K|+|\Sigma|+6)\rceil$个比特表示为二元数，为了便于**U**处理，可以在数前面适当添加0使之等长。

于是，一个图灵机$\mathbf{M}=(K,\Sigma,\delta,s)$的描述将数$|K|$紧接着连缀数$|\Sigma|$，再连缀转移函数$\delta$的描述$desc(\delta)$组成，即，形为$(|K|,|\Sigma|,desc(\delta))$，其中$\delta$的描述是由形如$((q,\sigma),(p,\rho,D))$的对组成的序列。假定这里所用的“（”、“）”、“，”等均在**U**的字母表中。**M**的一个瞬时像(q,w,u)被编码为(w,q,u)存贮，即用串w的编码，接着一个“，”，再接状态q的编码，然后是一个“，”，再接u的编码，那么起始瞬时像为$>,s,x$，其中x为输入。

用**M**仍表示图灵机**M**的描述。**U**的输入表示为$\mathbf{M};x$，其中仍用x表示**M**的输入x的编码，而且x中符号的编码用“，”分开。

由此可见，每个图灵机对应一个字符串，而对应不太长的字符串的图灵机的每一步行为都可以被确定并被模拟；反过来，若令非图灵机描述的字符串对应于平凡图灵机（如任何输入永不停机），则任意字符串亦编码一个图灵机及其输入。

关于输入$\mathbf{M};x$，我们用 2-带通用图灵机**U**模拟“关于输入x，**M**的运行”。**U**用第二条带存贮当前瞬时像。开始U在第二条带上贮存$\triangleright,s,x$；然后如下模拟**M**的一步（如图 3.4-2）：

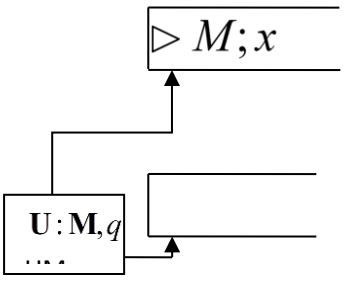

图 3.4-2 **U** 模拟 **M**

1) **U**扫描第二条带，直到找到一个对应于状态q的整数$q\in\left[|\Sigma|+1,|\Sigma|+|K|\right]$；

2) 在第一条带上寻找与状态q匹配的规则δ。若该δ规则被找到，则**U**在第二条带上左移一位，比较**M**指针所扫描符号是否与所找到的δ符号一致。若不一致，则寻

找另外的规则；若一致，则实施该规则，即，改变第二条带上的符号和状态，并将状态向左或右移动一个符号的位置或不动。

3) 当$\mathbf{M}$停机时，$\mathbf{U}$亦停机并输出$\mathbf{M}$的输出，从而完成$\mathbf{U}$关于输入$\mathbf{M};x$的运算。注意一点，若$\mathbf{U}$发现其输入不是一个图灵机及其输入的编码，此时$\mathbf{U}$进入一个状态，使之永远向右移动。

第五节　递归语言与递归可枚举语言

由前面的例子可以看出，对于一些特殊的串问题，如串函数计算，接受或判定一个语言，图灵机是一个理想的工具。下面对于这些任务给予正式定义。

定义3.4　设$L\subseteq(\Sigma\setminus\{\cup\})^*$为一个语言，$\mathbf{M}$为一个图灵机，满足：对于任意串$x\in(\Sigma\setminus\{\cup\})^*$，若$x\in L$，则$\mathbf{M}(x)=\text{"}yes\text{"}$；若$x\notin L$，则$\mathbf{M}(x)=\text{"}no\text{"}$，则称$\mathbf{M}$判定$L$。若$L$由某图灵机$\mathbf{M}$判定，则称$L$为一个递归语言。记所有递归语言类为$\mathbf{R}$。

对于任意串$x\in(\Sigma\setminus\{\cup\})^*$，如果$x\in L$，则有$\mathbf{M}(x)=\text{"}yes\text{"}$；若$x\notin L$，则$\mathbf{M}(x)=\nearrow$，此时称$\mathbf{M}$接受$L$。若$L$被某图灵机接受，则称$L$为递归可枚举的。记所有递归可枚举语言类为$\mathbf{RE}$。

假设f为从$(\Sigma\setminus\{\cup\})^*$到$\Sigma^*$的函数，且$\Sigma$为$\mathbf{M}$的字母表，称$\mathbf{M}$计算$f$，如果对于任意串$x\in(\Sigma\setminus\{\cup\})^*$，$M(x)=f(x)$。若这样的$\mathbf{M}$存在，则称$f$为一个递归函数。

显然，回文为递归语言，$f(x)=\cup x$及$f(x)=x+1$均为递归函数。注意，图灵机的“接受”定义不像可判定那样对称。对于$x\notin L$，机器永远运行下去。因此，“接受”仅是一个有理论意义的定义。下面命题说明“接受”与“判定”的关系，即“递归可枚举”与“递归”的关系，从而可见“递归可枚举”的语言是非常广泛的。

命题 3.2　若L为递归语言，则L一定为递归可枚举的。

证明：由于L为递归语言，则存在图灵机M判定L，即对于任意$x\in(\Sigma\setminus\{\cup\})^*$，$\mathbf{M}(x)=yes$，若$x\in L$；否则，no。于是有状态$q$和某个字符$\sigma$在转移函数$\delta^{\mathbf{M}}$下的象为$\delta^{\mathbf{M}}(q,\sigma)=(no,\rho,-)$。

下面构造$\mathbf{M}'$，使得$\mathbf{M}'$接受L：$\mathbf{M}'$的执行如$\mathbf{M}$一样，但是当$\mathbf{M}$即将停机并进入状态no时，$\mathbf{M}'$永远向右移动下去，即引入新的状态p，定义$\delta^{\mathbf{M}'}(q,\sigma)=(p,\sigma,\rightarrow)$，进而对于任意的$\sigma\in\Sigma$，$\delta^{\mathbf{M}'}(p,\sigma)=(p,\sigma,\rightarrow)$即可。

于是，递归语言是递归可枚举语言的一个子集，即，$\mathbf{R}\subseteq\mathbf{RE}$。有如下关系知递归语言$\mathbf{R}$及其补$co\mathbf{R}$相同，即，$co\mathbf{R}=\mathbf{R}$：

命题 3.3（对称性）若L是递归语言，则$\overline{L}$亦然。

证明：由于 L 是递归语言，故存在图灵机 $\mathbf{M}_L$ 判定 L，即，对于任意 $x\in(\Sigma\setminus\{\cup\})^*$，若 $x\in L$，则 $\mathbf{M}_L(x)="yes"$；否则，$\mathbf{M}_L(x)="no"$。于是我们可以构造 $\bar{\mathbf{M}}$ 如下：对于任意 $x\in(\Sigma\setminus\{\cup\})^*$，若 $\mathbf{M}(x)="yes"$，则 $\bar{\mathbf{M}}(x)="no"$；否则，$\bar{\mathbf{M}}(x)="yes"$。于是，$\bar{\mathbf{M}}$ 判定 $\bar{L}$。

命题 3.4 L 是递归语言当且仅当 L 与 $\bar{L}$ 都是递归可枚举的。

证明：若 L 是递归语言，则由命题 4.3 可知，$\bar{L}$ 亦然。从而 L 与 $\bar{L}$ 都是递归可枚举的。

若 L 和 $\bar{L}$ 都是递归可枚举的，则存在图灵机 $\mathbf{M}$ 和 $\bar{\mathbf{M}}$ 分别接受 L 与 $\bar{L}$。于是可构造判定 L 的图灵机 $\mathbf{M}_L$：

令 $\mathbf{M}_L$ 是 3-带图灵机。关于输入 $x\in(\Sigma\setminus\{\cup\})^*$，$\mathbf{M}_L$ 在两条工作带上穿插模拟 $\mathbf{M}$ 与 $\bar{\mathbf{M}}$ 关于 x 的运行。即在一条带上模拟 $\mathbf{M}$ 一步，然后在另一条带上模拟 $\bar{\mathbf{M}}$ 一步，然后在模拟 $\mathbf{M}$，如此下去，直到有一条带进入停机状态。

不管是否有 $x\in L$，总有一条带进入停机状态，并接受。若 $\mathbf{M}$ 接受，则 $\mathbf{M}_L$ 在状态 $"yes"$ 停机；若 $\bar{\mathbf{M}}$ 接受，则 $\mathbf{M}_L$ 在状态 $"no"$ 停机。于是 $\mathbf{M}_L$ 判定 L，即 L 为递归语言。

于是，$\mathbf{R}\subseteq co\mathbf{RE}\cap\mathbf{RE}$，如图 3.5-1。

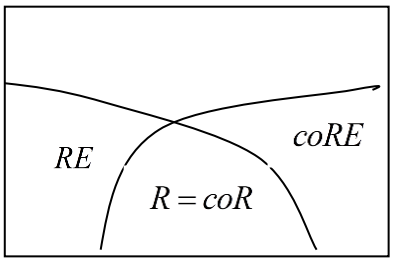

图 3.5-1　R，RE 与 $coRE$ 的关系

约定：本课程着重讨论递归语言。

第六节　不可判定性

不可判定问题是没有算法的问题或不递归的语言，这一直是计算机科学研究的重要课题。停机问题是一个典型的例子，其研究中对角化方法起到了重要的作用。对角化方法的核心就是一个图灵机对另一个图灵机的模拟：模拟机器能够确定另一个机器的行为，从而以不同的方式动作。我们将问题等同于语言，将算法等同于图灵机。

在实分析等理论学习中一个常用技巧——对角化方法，该方法源于 Cantor 关于实数不可数性的证明。

命题 3.5　令 $\Omega=\left\{f\,\middle|\,f:\{0,1\}^*\to\{0,1\}\right\}$，则存在 $UC\in\Omega$ 是不可计算的，即，UC 非递归函数。

证明：定义函数UC如下：对于每个$\alpha \in \{0,1\}^*$，令$\mathbf{M}$为α所描述的图灵机。关于输入α，若$\mathbf{M}$在有限步内停机并输出 1，则$UC(\alpha)=0$；否则，$UC(\alpha)=1$。若存在$\mathbf{M}_{UC}$计算UC，即，对于任意串$\alpha \in \{0,1\}^*$，有$\mathbf{M}_{UC}(\alpha)=UC(\alpha)$，则$UC(\mathbf{M}_{UC})=\mathbf{M}_{UC}(\mathbf{M}_{UC})$。

若$UC(\mathbf{M}_{UC})=1$，则$\mathbf{M}_{UC}(\mathbf{M}_{UC})=0$或 Z，故$UC(\mathbf{M}_{UC})\neq 1$，矛盾。若$UC(\mathbf{M}_{UC})=0$，则$\mathbf{M}_{UC}(\mathbf{M}_{UC})=1=UC(\mathbf{M}_{UC})$，矛盾，故$UC$不可计算。

所谓**停机问题（HALTING）**是指，*给定一个图灵机*$\mathbf{M}$*和其输入*x*的描述，问关于*x，$\mathbf{M}$*是否停机？*

利用通用图灵机$\mathbf{U}$的字母表，可将该问题编码为语言：$\mathbf{H}=\{\mathbf{M};x:\ \mathbf{M}(x)\neq\nearrow\}$，即，$\mathbf{H}$由满足如下条件的所有串组成：

每个串编码一个图灵机和一个输入，使得该图灵机关于该输入停机。

命题 3.6 $\mathbf{H}$是递归可枚举的语言。

证明：根据定义，需要构造图灵机$\mathbf{M}_{\mathbf{H}}$，使得对于任意x，有：若$x\in\mathbf{H}$，则有$\mathbf{M}_{\mathbf{H}}(x)=yes$；否则，$\mathbf{M}_{\mathbf{H}}(x)=\nearrow$。由于$u\in\mathbf{H}$时$u=\mathbf{M};x$，于是可以利用通用图灵机$\mathbf{U}$定义$\mathbf{M}_{\mathbf{H}}$。事实上，对于任意的$u\in(\Sigma\setminus\{\cup\})^*$，若$u\in\mathbf{H}$，则$u=\mathbf{M};x$，其中$\mathbf{M}$为一个图灵机的描述，$x$为$\mathbf{M}$的输入，而且$\mathbf{M}(x)\neq\nearrow$。于是，$\mathbf{U}(u)=\mathbf{U}(\mathbf{M};x)=\mathbf{M}(x)\neq\nearrow$。若$u\notin\mathbf{H}$，则$u$不能分解为一图灵机及其输入的编码，或者$u=\mathbf{M};x$但$\mathbf{M}(x)=\nearrow$。此时有$\mathbf{U}(u)=\nearrow$。

因此，我们定义$\mathbf{M}_{\mathbf{H}}$为：对于任意$x\in(\Sigma\setminus\{\cup\})^*$，若$\mathbf{U}(x)\neq\nearrow$，则$\mathbf{M}_{\mathbf{H}}(x)=yes$；否则，$\mathbf{M}_{\mathbf{H}}(x)=\nearrow$。显然，$\mathbf{M}_{\mathbf{H}}$接受$\mathbf{H}$，故$\mathbf{H}$为递归可枚举的。

$\mathbf{H}$不仅是一个递归可枚举语言，还具有完备性，即，若我们有一个判定$\mathbf{H}$的算法，则可以得出判定任何递归可枚举语言的算法。事实上，设L是由图灵机$\mathbf{M}$接受的任意递归可枚举语言。给定输入x，我们只需要判定是否$\mathbf{M};x\in\mathbf{H}$，就可以判定是否$x\in L$。于是，所有递归可枚举语言都可以归约到$\mathbf{H}$，称 HALTING 为递归可枚举问题类的完备问题。归约和完备是复杂性理论研究中两个重要的基本工具，后面将会有更多的讨论和应用。完备是问题最难求解部分的标志；而归约则是问题之间困难性的比较。$\mathbf{H}$的完备性为我们利用归约研究不可判定语言提供了方便。

命题 3.7 $\mathbf{H}$不是递归语言。

证明：若否，则存在一个图灵机$\mathbf{M}_{\mathbf{H}}$判定$\mathbf{H}$。利用$\mathbf{M}_{\mathbf{H}}$构造图灵机$\mathbf{D}$满足，关于输入$\mathbf{M}$，$\mathbf{D}$首先模拟$\mathbf{M}_{\mathbf{H}}$关于输入$\mathbf{M};\mathbf{M}$的行为直到$\mathbf{M}_{\mathbf{H}}$即将停机。若$\mathbf{M}_{\mathbf{H}}$将要接受，则$\mathbf{D}$进入一个状态，使得指针一直向右移动；若$\mathbf{M}_{\mathbf{H}}$将要拒绝，则$\mathbf{D}$停机并接受。

于是，关于输入$\mathbf{M}$，$\mathbf{D}$运行为：$\mathbf{D}(\mathbf{M})$：若$\mathbf{M}_{\mathbf{H}}(\mathbf{M};\mathbf{M})="yes"$，则↗；否则，$"yes"$。

若$\mathbf{D}(\mathbf{D})=$↗，有$\mathbf{M}_{\mathbf{H}}(\mathbf{D};\mathbf{D})="yes"$，则$\mathbf{D};\mathbf{D}\in\mathbf{H}$，于是$\mathbf{D}(\mathbf{D})=$↗，矛盾。若$\mathbf{D}(\mathbf{D})="yes"$，有$\mathbf{M}_{\mathbf{H}}(\mathbf{D};\mathbf{D})="no"$，则$\mathbf{D};\mathbf{D}\notin\mathbf{H}$，即，$\mathbf{D}(\mathbf{D})=$↗，矛盾。故没有图灵机判定$\mathbf{H}$。

在该定理证明中，我们研究的关系是$\{(\mathbf{M};x):\mathbf{M}$为图灵机，$x\in(\Sigma\setminus\{\cup\})^*,\mathbf{M}(x)\neq Z\}$。对角线上元素为$\{(\mathbf{M};\mathbf{M}):\mathbf{M}$为图灵机$\}$。即行对应图灵机，列对应输入，对角线上即为以自己为输入的图灵机。于是可以构造图灵机$\mathbf{D}$，与对角线元素相悖，从而得到矛盾。这是对角化方法的应用。

习题

(1) 一个二维图灵机（2D-TM）$\mathbf{M}$是一个四元组$\mathbf{M}=(K,\Sigma,\delta,s)$，其中

- K为有限状态集；
- Σ为有限字母表，且$\Sigma\cap K=\varnothing$，$\{\cup,>,\Delta,\Diamond\}\subset\Sigma$；
- 转移函数$\delta:K\times\Sigma\to(K\cup\{h,yes,no\})\times\Sigma\times\{\to,\leftarrow,\uparrow,\downarrow,-\}$，其中$\{\to,\leftarrow,\uparrow,\downarrow,-,h,yes,no\}\cap(K\cup\Sigma)=\varnothing$。
- $s\in K$为初始状态。

转移函数δ满足：

a）$\delta(q,>)\in K\times\{>\}\times\{\to\}$；

b）$\delta(q,\Diamond)\in K\times\{\Diamond\}\times\{\to\}$；

c) $\delta(q,\Delta)\in K\times\{\Delta\}\times\{\uparrow\}$。

于是，存在存储单元（称为“黑板”）是一个二维的仅向右和向上无限延伸的无界表格，即，对应于标准坐标系的右上象限。试完成如下问题：

A） 定义 2D-TM 的瞬时象和计算（即，“一步得到”函数）。

B）用一个 3-带图灵机模拟 2D-TM $\mathbf{M}$。□

（2） 设函数$f:\{0,1\}^*\to\{0,1\}$在时间$T(n)$内可由带有输入-输出带的k-带图灵机$\mathbf{M}$计算。则f可在时间$O((k-2)T(n)^2)$内由一个带有输入-输出带的3-带图灵机$\tilde{\mathbf{M}}$（即，仅有一条工作带）计算。

说明：这里的$\tilde{\mathbf{M}}$具有性质：$\tilde{\mathbf{M}}$的指针移动与其带的内容无关，仅与输入长度有关，即，不管输入是什么，$\tilde{\mathbf{M}}$总是执行从左到右并返回的同样形式的扫描。称具有该性质的机器为健忘的（Oblivious）。事实上，每个图灵机都可以由一个健忘图灵机模拟。□

（3）定义双向图灵机 TM 是一个带在两个方向上都是无限的图灵机。对于每个函数$f:\{0,1\}^*\to\{0,1\}^*$，若f在时间$T(n)$内可由一个双向图灵机$\mathbf{M}$计算，则f可在时间$O(T(n))$内由一个标准图灵机$\tilde{\mathbf{M}}$（即，单向带）计算。

（4）证明线性加速定理和空间压缩定理，即，定理 3.2 和 3.3。

（**5**）假定语言 L 可由一个 2-带图灵机 M 在时间 $t_{\mathrm{M}}(n)=\sqrt{n}$ 内判定。能否设计一个标准图灵机 N 在时间 $t_{\mathrm{N}}(n)=O(n)$ 内判定 L？是给出合理说明。

（**6**）考虑一个 1-带确定图灵机，满足，在每一步后其指针或者右移一步或者跳到带的第一个位置，但不能一步一步的左移。证明该图灵机可以在时间 $O(t(n)^3)$ 内模拟通常的时间复杂度为 $t(n)$的 1-带确定图灵机。

（**7**）设 **M** 是一个图灵机，带有一条只读的输入带和一条工作/输出的混合带。假定 **M** 判定一个语言 $L\subseteq\{0,1\}^*$，并且 **M** 永远不会写任意空格□。记关于输入 x，**M** 所用空间为 $s(|x|)$。

a) 定义约化瞬时像为由 M 的任意瞬时像略掉输入带所得的多元组，即，一个约化瞬时像只记录控制状态和 k -条工作带的内容和指针位置。给定输入 x，令 $C_i(x)$ 表示当输入带指针读第 i 个符号 x_i 时 **M** 的计算的所有瞬时像集合。令 $R_i(x)$ 表示从 $C_i(x)$ 得到的约化瞬时像集合。

令 $x=x_1\cdots x_n$ 为任意长为 n 的输入，满足对于任意长至多为 $n-1$ 的输入 y 有 $s(|y|)<s(|x|)$。证明，对于 $1\le i<j\le n$，$R_i(x)=R_j(x)$ 蕴含 $x_i\neq x_j$。（提示：假定对于某 $1\le i<j\le n$ 有 $R_i(x)=R_j(x)$ 且 $x_i\neq x_j$，考虑输入 $y=x_1\mathrm{L}\ x_ix_{j+1}\mathrm{L}\ x_n$，即，从 x 中去掉位置 $i+1,\mathrm{L}\ ,j$ 上的符号得到的。尝试证明，当 $1\le k\le i$ 时 $R_k(y)\subseteq R_k(x)$；当 $i<k\le n-(j-i)$ 时 $R_k(y)\subseteq R_{k+(j-i)}(x)$。从而可以证明关于输入 x，M 需要少于 $s(|x|)$ 的空间，矛盾。）

b）令 $f(n)=\max\{s(|x|)\,|\,x\in\{0,1\}^n\}$ 并且 $f(n)$ 是无界的。证明 $f(n)\notin o(\log\log n)$。（提示：利用 a）的结果得到仅依赖于 $f(n)$ 的 n 的上界。）

（**8**）对于 $L=\{uu\,|\,u\in\{a,b\}^*\}$，试分别构造判定 L 的图灵机 M 和非确定 2-带图灵机 N，并比较哪个计算更快。

（**9**）对于字母表 $\sum$ 上的任意两个语言 L_1 和 L_2，定义其并为 $L_1\cup L_2=\{x\in\Sigma^*\,|\,x\in L_1\text{或}x\in L_2\}$，其交为 $L_1\cap L_2=\{x\in\Sigma^*\,|\,x\in L_1\text{且}x\in L_2\}$。称复杂类 C 在并（或交）下封闭的，如果对于 C 中任意语言 L_1 和 L_2 有 $L_1\cup L_2$（或 $L_1\cap L_2$）也在 C 中。证明：

a）递归可枚举语言对于并和交的封闭性。即，若 L_1 和 L_2 是{0，1}上的两个递归可枚举语言，则 $L_1\cup L_2$ 和 $L_1\cap L_2$ 都是递归可枚举的。

b）递归语言对于并和交的封闭性。即，如果 $L1$ 和 $L2$ 都是递归语言，则 $L_1\cup L_2$ 和 $L_1\cap L_2$ 都是递归。

（**10**）设 $L1$、$L2$ 和 $L3$ 是满足如下条件的三个语言：

(a) 这三个语言中每个都是递归可枚举的。

(b) $L_1\cup L_2\cup L_3=\{0,1\}^*$。

(c) 这三个语言两两不交，即，$L_1\cap L_2=\varnothing$，$L_2\cap L_3=\varnothing$，$L_1\cap L_3=\varnothing$。

证明这三个语言都是递归的。

（**11**）定义空串停机问题（EMPTYHALTING）为：给定图灵机 $\mathbf{M}$，以空串 ε 为输入，

问 **M** 是否停机？记 EMPTYHALTING $=\{\mathbf{M}:\mathbf{M}(\varepsilon)\neq Z\}$。证明：EMPTYHALTING 是非递归语言。

（12）定义有限语言问题为：给定图灵机 **M**，问 $L(\mathbf{M})$ 是否有限？证明：有限语言问题不是递归的。

（13）给定串 x 和一个图灵机 **M** 的描述（仍记为 **M**）。如果 **M** 接受语言 $\{x;y:(x,y)\in R\}$，这里 R 为一个二元关系。试构造一个图灵机 $\mathbf{M}_x$ 接受 $\{y:(x,y)\in R\}$。

（14）证明如下语言是不可判定的：

a）$\{\mathbf{M}:\mathbf{M}$ 关于所有输入都停机 $\}$

b）$\{\mathbf{M};x:$存在 y 使得 $\mathbf{M}(x)=y\}$

c）$\{\mathbf{M};x:$关于输入 x，**M** 的计算利用了 **M** 的所有状态 $\}$

d）$\{\mathbf{M};x;y:\mathbf{M}(x)=y\}$

第四章 计算复杂类

前面我们介绍了图灵机及其功能，同时介绍了一些复杂类，如 **P**、**NP**、**PSPACE**、**NL**、**L** 等，以及类与类之间的关系。对于多数复杂类的性质，我们仍知之甚少。

第一节 复杂类

一般复杂类的定义可概括为：在适当的计算模式下，对于任意输入 x，一个多带图灵机 **M** 花费至多 $f(|x|)$ 个特定的资源单位判定的所有语言组成的集合，即为复杂类。其刻画主要由如下参数：

①基本的计算模型：多带图灵机；

②计算模式：确定性和非确定性；

③所用的计算资源：基本的时间和空间；

④非负整数到非负整数的函数 f，用于界定计算资源。

为了方便，我们定义一类广泛的函数，我们将用这类函数界定计算资源。设 f 是一个从非负整数到非负整数的非递减函数，称 f 是一个完全可构造函数，如果存在一个有输入-输出带的 k-带图灵机 $\mathbf{M}_f=(K,\Sigma,\delta,s)$ 满足：对于任意的整数 n 和任意长为 n 的输入 x，有 $(s,\triangleright,x,\triangleright,\varepsilon,\ldots,\triangleright,\varepsilon)\xrightarrow{\mathbf{M}_f^t}(h,\triangleright,x,\triangleright,\cup^{j_2},\triangleright,\cup^{j_3},\ldots,\triangleright,\cup^{j_{k-1}},\triangleright,\cap^{f(n)})$，使得

$t = O(n + f(n))$，并且 $j_i = O(f(n))$，$i = 2,3,\mathrm{K}\ ,k-1$。这里 t 和 j_i 只依赖于 n，$\cap$ 是一个“假空格”符号。即，所谓完全可构造函数 f 就是：关于输入 x，经图灵机 $\mathbf{M}_f$ 计算后输出串 $\cap^{f(|x|)}$，其中 $\cap$ 是“假空格”符号；而且对于任意输入 x，在 $O(|x| + f(|x|))$ 步后，$\mathbf{M}_f$ 所用空间为 $O(f(|x|))$。

例4.1　如下函数是完全可构造函数。

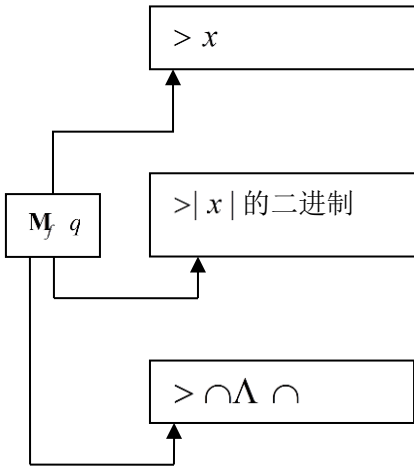

图 4.1-1 计算 $\lceil \log n \rceil$ 的图灵

①常值函数 $f(n) = c$。事实上，构造 $\mathbf{M}_f$ 如下：不管输入是什么，只在最后带上写 $\cap^c$。

② $f(n) = n$，定义 $\mathbf{M}_f$ 如下：在最后带上重新写入，同时将每个字符改写为 $\cap$ 即可。

③ $f(n) = \lceil \log n \rceil$，定义如图 4.1-1 的 3-带图灵机 $\mathbf{M}_f$，运行如下：

第一个指针在输入带上从左到右慢慢移动读取输入；同时，在第二条带上以二进制的方式对输入进行计数。当在输入带上指针遇到第一个“∪”时，第二条带上串长为 $\lceil \log n \rceil$。然后，$\mathbf{M}_f$ 擦去第二条带上的内容，将其所有符号在第三条带上复制为 $\cap$。

该计算所用的时间为 $O(n)$，$|x| = n$。

类似可以证明，$\log n^2,\ n\log n,\ n^2,\ 2^n,\ \sqrt{n},\ \ n^3 + 3n,\ n!$ 均是完全可构造的。

易证，若 f 与 g 都是完全可构造函数，则 $f + g$，$f \cdot g$ 与 2^g 亦然。

完全可构造函数是一类很广泛的函数，包含了算法分析中用的所有函数，排除了许多畸形函数，利用它可以为我们带来很多方便，亦使我们的讨论更标准化。

称一个图灵机 $\mathbf{M}$（不管是否有输入-输出带，确定或非确定）是精确的，如果存在函数 f 和 g，使得对于每个 $n \geq 0$、每个长为 n 的输入 x 和（非确定）$\mathbf{M}$ 的每个计算，恰在 $f(n)$ 步后 $\mathbf{M}$ 停机，并且所有带都在长为 $g(n)$ 处停机（对于有输入-输出带的 $\mathbf{M}$，第一条带和最后一条带除外）。简单地说，精确图灵机就是给定输入后，其运行所用的时间和空间都是可计算的函数。

分别考虑确定的时间或空间、非确定的时间或空间，如果一个（确定或非确定）图灵机 $\mathbf{M}$ 判定语言 L 所用的时间或空间是关于输入长 n 的完全可构造函数 $f(n)$，那么一定存在一个精确图灵机 $\mathbf{M}'$ 在时间（或空间）$O(f(n))$ 内判定同一语言 L。

对于完全可构造函数 f，有下列复杂类：

- $\boldsymbol{TIME}(f)$：确定的时间复杂类
- $\boldsymbol{NTIME}(f)$：非确定的时间复杂类
- $\boldsymbol{SPACE}(f)$：确定的空间复杂类
- $\boldsymbol{NSPACE}(f)$：非确定的空间复杂类

更多地，我们会用由参数 $k>0$ 所描述的函数类代替单个函数，则有

“复杂类”=$\bigcup_{k}$ “单个复杂类(k)”，

于是可定义复杂类：

$\mathbb{L}=\boldsymbol{SPACE}(\log n)$； $\boldsymbol{N}\mathbb{L}=\boldsymbol{NSPACE}(\log n)$； $\boldsymbol{P}=\boldsymbol{TIME}(n^k)=\bigcup_{j>0}\boldsymbol{TIME}(n^j)$；

$\boldsymbol{NP}=\boldsymbol{NTIME}(n^k)=\bigcup_{j>0}\boldsymbol{NTIME}(n^j)$； $\boldsymbol{PSPACE}=\boldsymbol{SPACE}(n^k)=\bigcup_{j>0}\boldsymbol{SPACE}(n^j)$；

$\boldsymbol{NPSPACE}=\boldsymbol{NSPACE}(n^k)=\bigcup_{j>0}\boldsymbol{NSPACE}(n^j)$； $\boldsymbol{EXP}=\boldsymbol{TIME}(2^{n^k})=\bigcup_{j>0}\boldsymbol{TIME}(2^{n^j})$；

$\boldsymbol{NEXP}=\boldsymbol{NTIME}(2^{n^k})=\bigcup_{j>0}\boldsymbol{NTIME}(2^{n^j})$

根据 Church-Turing 命题可知，所有这些复杂类与所选模型无关。

设 L 为一个语言，则 $L\subseteq\Sigma^*$，L 的补定义为 $\Sigma^*\setminus L$，记为 $\overline{L}$，对于判定性问题 A，我们可以判定 A 的补 $\overline{A}$，即，*若输入为 A 接受，则输出*"no"；*否则，为*"yes"。

例4.2 1）**SAT** 的补 $\overline{\mathbf{SAT}}$ 问题定义为：给定 Boolean 表达式 ϕ（合取范式），ϕ 是否是不可满足的？这显然等价于判断 $\neg\phi$ 是否为永真式。

2) HAMILTON PATH 的补 $\overline{\text{HAMILTON PATH}}$ 定义为：给定一个图 G，是否 G 没有 HAMILTON PATH？值得注意的是，HAMILTON PATH $\overline{\text{UHAMILTON PATH}}$ 仅由所有编码图的串组成，而不是 Σ^*。

一般地，对于任意复杂类 C，定义 $coC=\{\overline{L}:L\in C\}$ 为 C 的补。易知，若 C 为确定的时间或空间复杂类，则 $C=coC$。但是，对于非确定复杂类如 NP，仍是公开问题。

第二节　分离定理

对于一个复杂类，是否可以进行层次划分？如，TIME(n)$\subseteq$TIME(n^2)$\subseteq$…TIME(n^k)$\subseteq$…之间是否有真包含关系。直观上，随着时间分配的增多，图灵机计算能力亦加强，

从而对同一复杂类可进行层次划分。

设 $f(n)\geq n$ 为完全可构造函数，考虑停机问题的一类特殊情形：$\mathbf{H}_f=\{\mathbf{M};x:\mathbf{M}$ 在至多 $f(|x|)$ 步后接受 x，这里 $\mathbf{M}$ 为任意确定多带图灵机的描述。$\}$。为了方便，我们所讨论的语言及输入均是由图灵机的符号编码组成。

定理 4.1 (时间分层定理) 若 $f(n)\geq n$ 为完全可构造函数，则 $\boldsymbol{TIME}(f(n))\subsetneq\boldsymbol{TIME}\left((f(2n+1))^3\right)$。

该定理的证明需要下面两个引理：

引理 4.1 $\mathbf{H}_f\in TIME\left(f(n)^3\right)$。

证明：我们要构造一个 4-带图灵机 $\mathbf{U}_f$（见图 4.2-1），在时间 $f(n)^3$ 内判定 $\mathbf{H}_f$，这是前面用过的如下几个图灵机的有机结合：

- 通用图灵机 $\mathbf{U}$；
- 单带模拟多带图灵机模拟器；
- 线性加速机器（用于去掉时间界的常数）；
- $\mathbf{M}_f$：精确计算长为 $f(n)$ 的界标。

$\mathbf{U}_f$ 运行如下：

首先，$\mathbf{U}_f$ 在其输入的第二部分 x 上，利用四条带模拟 $\mathbf{M}_f$，从而在第四条带上初始化一个适当的界标 $\cap^{f(|x|)}$，所用时间为 $O\left(f(|x|)\right)$。这将用于 $\mathbf{M}$ 的模拟。

然后，$\mathbf{U}_f$ 将 $\mathbf{M}$ 复制到第三条带上，并将第一条带上 x 变为 $>x$；第二条带上是 $\mathbf{M}$ 当前状态的编码。初始为 $\mathbf{M}$ 的初始状态 s，同时亦验证 $\mathbf{M}$ 的确是一个图灵机的描述；否则，拒绝。到目前为止所用的总的时间是 $O\left(f(|x|)+n\right)=O\left(f(n)\right)$。

现在 $\mathbf{U}_f$ 模拟 $\mathbf{M}$ 的运行：关于输入 x，如定理 3.1 的证明，模拟仅在第一条带上运行，并保存 $\mathbf{M}$ 的所有带内容的编码。

为了模拟 $\mathbf{M}$ 的一步运行，$\mathbf{U}_f$ 先后扫描第一条带两次，在第一次扫描期间，$\mathbf{U}_f$ 收集当前状态下 $\mathbf{M}$ 指针所指的符号的所有相关信息，并在第二条带上记录这些信息，注意第二条带亦含有当前状态的编码；然后，$\mathbf{U}_f$ 将第二条带的内容与第三条匹配，找到 $\mathbf{M}$ 合适的转移函数，再进行对第一条带的第二次扫描，执行适当的变化；并向前移一步界标上的指针。令 k_M 为 $\mathbf{M}$ 的带数，l_M 为 $\mathbf{M}$ 的每个符号和状态编码的长。第一次扫描时间为 $O\left(k_M f(|x|)\right)$，第二次扫描时间为 $O\left(l_M k_M\right)O\left(k_M f(|x|)\right)$，故总时间为 $O\left(l_M k_M^2 f(n)\right)$。由于 l_M, k_M 由 $\mathbf{M}$ 的描述长 $|K|(|\Sigma|^{f(n)+1})^k$ 的对数界定，故 $\mathbf{U}_f$ 模拟 $\mathbf{M}$ 一步所用时间共计为 $O\left(f(n)^2\right)$。若 M 在 $f(|x|)$ 步内接受 x，则 $\mathbf{U}_f$ 接受 $\mathbf{M};x$；否则，拒绝。因此，所用总的时间为 $O(f^3(n))$。注意，$\mathbf{U}_f$ 可利用线性加速器的办法将常数系数变为 1，使得时间为 $f^3(n)$。

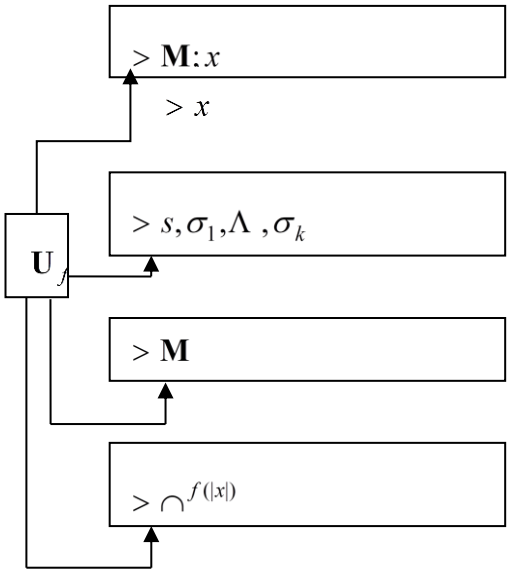

图 4.2-1 $\mathbf{U}_f$

引理 4.2 $H_f \notin TIME\left(f\left(\lfloor n/2 \rfloor\right)\right)$。

证明：若否，存在图灵机 M_{H_f} 在时间 $f\left(\lfloor n/2 \rfloor\right)$ 内判定 H_f。构造图灵机 D_f：以图灵机描述 M 为输入，运行 $D_f(M)$：若 $M_{H_f}(M;M) = yes$，则输出"no"；否则，"yes"。

因此，D_f 关于输入 M 的运行时间与 M_{H_f} 关于输入 $M;M$ 的运行时间相同，均为 $f\left(\lfloor 2n+1/2 \rfloor\right) = f(n)$。于是，对于输入 D_f，若 $D_f(D_f) = yes$，则 $M_{H_f}(D_f;D_f) = no$，故 $D_f;D_f \notin H_f$，即在时间 $f(n)$ 内 $D_f(D_f) \neq yes$。由 D_f 的定义知，$D_f(D_f) = no$，矛盾。若 $D_f(D_f) = no$，则 $M_{H_f}(D_f;D_f) = yes$，即 $D_f;D_f \in H_f$，于是 $D_f(D_f) = yes$，矛盾。故 $H_f \notin TIME\left(f\left(\lfloor n/2 \rfloor\right)\right)$。

推论4.1 $\boldsymbol{P} \subsetneq \boldsymbol{EXP}$。

证明：利用定理 4.1 可知，$\boldsymbol{P} \subseteq \boldsymbol{TIME}(2^n) \subsetneq \boldsymbol{TIME}((2^{2n+1})^3) \subseteq \boldsymbol{TIME}(2^{n^2}) \subseteq \boldsymbol{EXP}$。

时间分离定理可进一步推广：设 $f_1(n)$ 与 $f_2(n)$ 是完全可构造函数且 $f_2(n) \geq f_1(n) \geq n$，有 $\liminf_{n\to\infty} \dfrac{f_1(n)\log\left(f_1(n)\right)}{f_2(n)} = 0$，则 $\boldsymbol{TIME}(f_1(n)) \subsetneq \boldsymbol{TIME}(f_2(n))$。

对于空间复杂类，有类似的结果。

定理 4.2 （空间分离定理）若 $f(n)$ 是完全可构造函数，则

$$\boldsymbol{SPACE}(f(n)) \subsetneq \boldsymbol{SPACE}(f(n)\log f(n))。$$

更一般地，设 $f_1(n)$ 与 $f_2(n)$ 是完全可构造函数，$\liminf_{n\to\infty} \dfrac{f_1(n)}{f_2(n)} = 0$，并且有

$f_2(n) \geq f_1(n) \geq \log n$，则 $\boldsymbol{SPACE}(f_1(n)) \subsetneq \boldsymbol{SPACE}(f_2(n))$。

第三节　可达性方法

为研究复杂类之间的关系，本节引入一个新的研究空间复杂度的方法，称为可达性方法，主要思想是将计算过程转化为图，再利用 REACHABILITY 的算法得到结论。下面我们定义瞬时像图。

设 $M=(K,\Sigma,\delta,s)$ 为一个有输入-输出带的 k -带非确定图灵机，判定语言 L。关于输入 x，运行时间为 $f(|x|)$。M 的瞬时像即为 $2k+1$ 元组 $(q,w_1,u_1,\mathrm{K},w_k,u_k)$。由于第二、三个分量总是 $>x$，而对于判定机器而言，最后带为只写带，其它带的长不超过 $f(n)$，于是，可将瞬时像简记为 $(q,i,w_2,u_2,\mathrm{K},w_{k-1},u_{k-1})$，其中 i 记录第一条带上指针的位置，$0 \leq i \leq n = |x|$。定义 $\mathbf{M}$ 的瞬时像图 $G(M,x)$ 为

- 顶点为所有可能的瞬时像组成的集合。
- 定义边为：瞬时像 C_1 和 C_2（两顶点）之间有一条边当且仅当 $\boldsymbol{C}_1 \xrightarrow{\mathbf{M}} \boldsymbol{C}_2$。

由于 $\mathbf{M}$ 的瞬时像个数为 $|\boldsymbol{K}|(\boldsymbol{n}+1)(|\Sigma|^{f(n)})^{(k-2)} \leq \boldsymbol{n}\boldsymbol{c}_1^{f(n)} = \boldsymbol{c}_1^{\log n + f(n)}$，其中 c_1 仅依赖于 $\mathbf{M}$，故 $G(M,x)$ 的顶点个数 $\leq c_1^{\log n + f(n)}$。x 是否属于 L 当且仅当在图 $G(M,x)$ 中是否有一条从 $C_0=(s,0,>\varepsilon,\mathrm{K},>\varepsilon)$ 到 $C=(yes,i,\mathrm{K})$ 的通路。

对于完全可构造函数 $f(n)$，可以证明：

1）$SPACE(f(n)) \subseteq NSPACE(f(n))$，并且 $TIME(f(n)) \subseteq NTIME(f(n))$；

2）$NTIME(f(n)) \subseteq SPACE(f(n))$；

3）$NSPACE(f(n)) \subseteq TIME(k^{\log n + f(n)})$。

由此易知，$\boldsymbol{L} \subseteq \boldsymbol{NL} \subseteq \boldsymbol{P} \subseteq \boldsymbol{NP} \subseteq PSPACE$。值得注意的是，利用空间分离定理，有 $\boldsymbol{L} \subset PSPACE$，因此，上述包含关系应有一个真包含，但我们不知道是哪一个。根据 $NSPACE(f(n)) \subseteq SPACE(k^{\log n + f(n)})$，问，*是否有更好的方式使得确定性空间模拟非确定性空间？或者是否非确定性空间的确比确定性空间有更强大的功能？*下面我们进一步利用可达性方法实现对非确定空间的模拟，从而得到答案，说明确定空间模拟非确定性空间仅需要二次方代价。

定理 4.3　（Savitch 定理）$REACHABILITY \in SPACE(\log^2 n)$。

证明：设图 $G=(V,E)$，$|V|=n$，$x,y \in V$，令 $i \geq 0$。定义断言 $PATH(x,y,i)=1$，如果“*G 中存在一条长不超过 2^i 的从 x 到 y 的通路*”。

由于 G 任意通路的长不超过 n，因此，给定 G 的两点 x,y，只要计算 $PATH(x,y,\lceil \log n \rceil)$ 就可以解决 G 中的可达性问题。

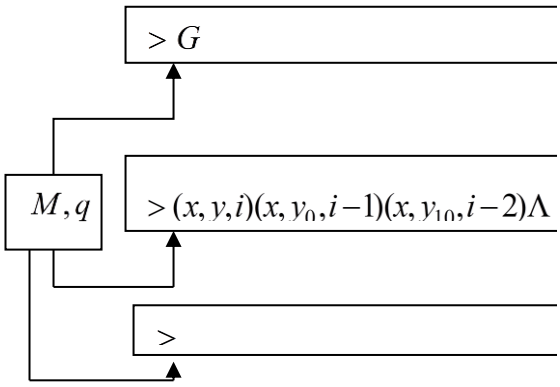

图 4.3-1 计算 $PATH$ 的图灵机

下面设计一个计算 $PATH(x,y,i)$ 的两条工作带图灵机（除输入带外）（如图 4.3-1）：

- 输入串为 G 的邻接矩阵；
- 第一条工作带： x,y 为顶点， i 为整数，该带含有一系列三元组， (x,y,j) ， $0\le j\le i$ ，而且 (x,y,i) 为最左边一个。
- 第二条工作带：用作计算需要的零散空间， $\boldsymbol{O}(\log n)$ 足够。

若 $i=0$ ，检查输入可知是否 $x=y$ 或 x 与 y 邻接。若 $i\ge 1$ ，用递归算法计算 PATH(*x*,*y*,*i*) ：“*对于所有顶点 z ，检验是否* $PATH(x,z,i-1)=1$ *和* $PATH(z,y,i-1)=1$ ”。

思路：任何从 x 到 y 的长为 2^i 的通路有一个中间点 z ，且 x 到 z 与 z 到 y 的长不超过 2^{i-1} （如图 4.3-2）。为节省空间，重复利用空间逐个生成所有顶点 z 。一旦生成一个新的 z ，就将 $(x,z,i-1)$ 添加到主要工作带上，并就此新问题开始递归。

若 $PATH(x,z,i-1)=0$ ，则擦去该三元组，生成一个新的 z' ，在主要工作带上写上 $(x,z',i-1)$ ，进行递归。

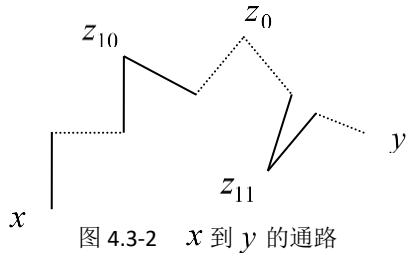

图 4.3-2 x 到 y 的通路

若 $PATH(x,z,i-1)=1$ ，则擦去该三元组，写上 $(z,y,i-1)$ ，继续计算 $PATH(z,y,i-1)$ 。若 $PATH(z,y,i-1)=0$ ，则擦去该三元组，生成一个新的 z' ，再如上计算 $PATH(x,z',i-1)$ 和 $PATH(z',y,i-1)$ ；若 $PATH(z,y,i-1)=1$ ，则与其左边的 (x,y,i) 比较，可得 $PATH(x,y,i)=1$ 。

易知，该算法求解 $PATH(x,y,i)$ 。第一条工作带至多有 $\lceil\log n\rceil$ 个三元组，而每个三元组长至多为 $3\lceil\log n\rceil$ ，于是可在空间 $\boldsymbol{O}\left(\left\lceil\log^2 n\right\rceil\right)$ 内求解 REACHABILITY。

注意，对于求解 REACHABILITY，以前一直用确定性算法，分别用深度优先和宽度优先法，所占空间为 $\boldsymbol{O}(n)$ 。这里利用“中间优先”搜索法，尽管浪费时间，但是节省

了空间。

推论4.2 设 f 为完全可构造函数， $f(n) \geq \log(n)$，则

$$\boldsymbol{NSPACE}\left(\boldsymbol{f}(\boldsymbol{n})\right) \subseteq \boldsymbol{SPACE}\left(\boldsymbol{f}^{2}(\boldsymbol{n})\right)$$，进而 $PSPACE = NPSPACE$。

证明：关于输入 x， $|x| = n$。为了模拟 $f(n)$ 空间界定的非确定图灵机 M，构造 (M, x) 的瞬时像图，然后运用定理 5.5 的算法找出接受计算。唯一不同的是在检查两个点是否相邻，即 $i = 0$ 的情形。此时，除了检查输入外，还要根据转移关系才能确定是否邻接。由于模拟图 $G(M, x)$ 有至多 $c^{f(n)}$ 个顶点，其中 c 为常数，则有 $\boldsymbol{O}(f^{2}(n))$ 空间足够。

对于时间情形目前不知道是否有 $NP = coNP$。对于 $\boldsymbol{coNPSPACE}$ 与 $\boldsymbol{NPSPACE}$，对于完全可构造函数 $\boldsymbol{f}(\boldsymbol{n}) \geq \log \boldsymbol{n}$，$\boldsymbol{NSPACE}(\boldsymbol{f}(\boldsymbol{n})) = \boldsymbol{coNPSPACE}(\boldsymbol{f}(\boldsymbol{n}))$。

习题

(1) 设 f 与 g 都是完全可构造函数，证明 $f + g$， $f \cdot g$， $f\left(g\right)$ 与 2^{g} 为完全可构造函数。

(2) 设 f 为一个完全可构造函数。若(确定或非确定)图灵机 M 在时间（或空间）$f(n)$内判定语言 L，则存在一个精确图灵机 $\mathbf{M}'$，在时间（或空间）$O\left(f(n)\right)$ 内判定同一语言 L。

(3) 设 $f(n)$ 为完全可构造函数，证明：

a） $SPACE(f(n)) \subseteq NSPACE(f(n))$，并且 $TIME(f(n)) \subseteq NTIME(f(n))$；

b） $NTIME(f(n)) \subseteq SPACE(f(n))$；

c） $NSPACE(f(n)) \subseteq TIME(k^{\log n + f(n)})$。

(4) 对于字母表 $\sum$ 上的任意两个语言 L_1 和 L_2，定义其并为 $L_1 \cup L_2 = \{x \in \Sigma^* \mid x \in L_1 \text{或} x \in L_2\}$，其交为 $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \text{且} x \in L_2\}$。称复杂类 C 在并（或交）下封闭的，如果对于 C 中任意语言 L_1 和 L_2 有 $L_1 \cup L_2$（或 $L_1 \cap L_2$）也在 C 中。证明 P 与 NP 对于并和交的封闭性。

(5) 定理：存在一个语言 L 满足，对于任意在时间 $t_{\mathrm{M}}(n)$ 内接受 L 的图灵机 M，存在一个图灵机 N 在时间 $t_{\mathrm{N}}(n) \in O(\log_2 t_{\mathrm{M}}(n))$ 内接受 L。

a.对于定理中的语言 L，下列是否成立，并说明理由：

b.如果存在一个图灵机 M 在时间 $t_{\mathrm{M}}(n) \in O(2^{n})$ 内接受 L，则存在一个图灵机 N 在时间 $t_{\mathrm{N}}(n) \in O(n)$ 内接受 L。

(6) 定义语言 L 的 Kleene 星为 $L^* = \left\{x_1 \mathrm{L}\ x_k \middle| \exists k \geq 0 \text{并且} x_1, \mathrm{L}, x_k \in L\right\}$，$\overline{L} = \{x \in \Sigma^* \mid x \notin L\}$。称一个语言类 C 在 Kleene 星下封闭，如果 $L \in C$ 则 $L^* \in C$。讨论下列是否成立：

a. $\boldsymbol{P}$ 与 $\boldsymbol{NP}$ 在Kleene星下封闭。

b. $\overline{L}^{*} = \overline{L^{*}}$，进而coNP在该*作用下是否封闭。

第五章　归约和完备性

在对问题的研究中，通常对问题之间的困难性给予比较。一方面我们可以通过一个问题的求解找到另一个问题的解；另一方面，可以找到一些可以刻画整个复杂类的困难性的问题，从逻辑上而言，这些问题在复杂类研究中具有中心地位。

第一节　归约

在复杂类中，有些问题比其他问题能够更好地反映语言类的性质，如，SAT 似乎比 REACHABILITY 能更好反映 NP 的复杂性。为了更好比较这些问题的难度，精确地描述这些现象，我们需要引入归约。事实上，前面我们已经利用归约思想研究一些问题，如，将完美匹配问题转化成一个 MAXFLOW 问题，再利用 MAXFLOW 问题求解，从而得到完美匹配问题的解；在不可判定性的研究中亦有任何递归可枚举语言都可以归约到停机问题。

定义5.1　称一个语言 L_1 可以归约到语言 L_2，如果存在一个确定图灵机在多项式时间内计算一个串到串的函数 R，使得对于所有输入 x，

$$x \in L_1 \text{ 当且仅当 } R(x) \in L_2 \text{。}$$

称 R 为从 L_1 到 L_2 的归约。

定义说明，$\boldsymbol{x \notin L_1}$ 当且仅当 $\boldsymbol{R(x) \notin L_2}$。因此归约函数 R 提供了从语言 L_1 表示的判定问题的任何实例 x 到语言 L_2 表示的判定问题的实例的映射，如果能提供是否有 $R(x) \in L_2$，也就直接提供了是否有 $x \in L_1$ 的答案。从直觉上看，一个问题 A 可以被归约为另一个问题 B，就是 A 的任何实例都可以被“容易的表示（重新描述）为” B 的实例，而 B 的实例的解也是 A 的实例的解。此时我们称问题 A 至少与问题 B 一样难。有的文献将归约定义为确定图灵机在空间 $\boldsymbol{O}(\log n)$ 内所计算的函数 R，由推论 4.2 可知，这显然是更强的要求。另一方面，定义要求在归约中要求 R 的计算不会太难；否则，将会得到的奇怪的结果。如，给定 $TSP(D)$ 的一个实例 x，定义归约 R：*检查所有的旅行，若其中之一有比 D 更小的代价，则令 $R(x)$ 为一个从点1到点2的单边组成的图；否则，$R(x)$ 为没有边的点1与2组成的图。这就将 $TSP(D)$ 归约到 REACHABILITY*。但就目前我们的知识而言，REACHABILITY 不会比 TSP（D）难。造成这样结果的原因在于 R 的计算是指数时间算法。下面给出归约的几个例子：

例5.1 HAMILTON PATH 可归约到 SAT。

所谓 HAMILTON PATH 是指给定一个图 G，问是否有一条通路访问每个顶点恰好一次。于是，对于一个图 G，我们需要构造 $R(G)$，使得 G *有一条 Hamilton 通路当且仅当 $R(G)$ 可满足*。为了构造 $R(G)$，首先分析一条 Hamilton 通路的特点。

设 G 有顶点 $\{1,2,\cdots,n\}$，对于一条通路 $u_{i(1)}, u_{i(2)}, \text{L}\ , u_{i(n)}$，其中 $i(1), i(2), \cdots, i(n)$ 为

$\{1,2,\cdots,n\}$的一个置换。于是，点k可出现在位置j处。因此，要刻画每个点及其出现的位置，则需要$\{1,2,\cdots,n\}\times\{1,2,\cdots,n\}$的子集表示的关系，即，需要$n^2$个变量$\{x_{i,j}:1\le i,j\le n\}$，并定义$x_{i,j}=true$当且仅当点$j$出现在位置$i$处；否则，为$false$。

另外，每条 Hamilton 通路满足：每个点j只能出现在唯一的位置，且每个位置只能有一个点；并且不邻接的点一定不会出现在相邻的位置。下面利用 Boolean 表达式将该性质描述出来：

定义分句如下：

① 每个点j必须出现在某个位置i，$1\le i\le n$：$x_{1j}\vee x_{2j}\cdots\vee x_{nj}$，$1\le j\le n$；

② 每个点j不可能同时出现在两个位置：$\neg(x_{ij}\wedge x_{kj})$；$i\ne k$，$1\le i,j,k\le n$；

③ 每个位置i处必须有一个顶点：$x_{i1}\vee x_{i2}\cdots\vee x_{in}$，$1\le i\le n$；

④ 每个位置i只能有一个顶点：$\neg(x_{ij}\wedge x_{ik})$，$j\ne k$，$1\le i,j,k\le n$；

⑤ 对于任意两点i，j，若(i,j)不为G的边，则不会出现在通路中的相邻位置：

$\neg(x_{ki}\wedge x_{k+1,j})$，$1\le k\le n-1$。

将上述分句取合取即得$R(G)$，这即完成了$R(G)$的构造。下面证明R是一个归约，即证明：

a） 对于任意图G，G有 Hamilton 通路当且仅当$R(G)$可满足。

b） 可在多项式时间内计算R。

事实上，设$R(G)$有可满足的赋值 T。由$(x_{1j}\vee x_{2j}\cdots\vee x_{nj})\wedge(\wedge_{i\ne k}(\neg x_{ij}\vee\neg x_{kj}))=true$可知，对于每个点$j$，存在唯一位置$i$使得$T(x_{ij})=true$。类似的，由$(x_{i1}\vee x_{i2}\cdots\vee x_{ik})\wedge(\wedge_{j\ne k}(\neg x_{ij}\vee\neg x_{ik}))=true$可知，对于每个位置$i$，有唯一的点$j$使得$T(x_{ij})=true$，即，每个位置对应唯一顶点，反之亦然。

于是，每个 T 对应G的一个顶点置换$\pi(1),\cdots,\pi(n)$，其中$\pi(i)=j$当且仅当$T(x_{ij})=true$。再由分句$\neg(x_{ki}\wedge x_{k+1,j})=true$可知，对于所有的$k$，$(\pi(k),\pi(k+1))$为边。因此，$(\pi(1),\cdots,\pi(n))$为一条 Hamilton 通路。

反过来，设G有一条 Hamilton 通路$(\pi(1),\cdots,\pi(n))$，其中π为一个置换。在上面的构造中，令$T(x_{ij})=true$，如果$\pi(i)=j$；否则，$false$。则$T\models R(G)$。

下面说明R可在多项式时间内计算。

给定图G作为图灵机M的输入，则M可如下产生输出$R(G)$：

首先，在一条工作带上以二进制形式写出n。由于$R(G)$描述的前四类分句仅与图G的顶点数n有关，故可在输出带上逐个生成这些分句。这里，M需要三个计数器i,j,k帮助构造分句中的变量指标。然后，对于最后一组分句，M在其一条工作带上逐个生成分句$\neg x_{ki}\vee\neg x_{k+1,j}$，$k=1,2,\mathrm{L}\ ,n-1$。$M$在输入中检查$(i,j)$是否为$G$的一

条边。若不是，则在输出带上输出该分句；否则，生成下一个。

以上运算仅需要记录指标i, j, k，以及$\left(\neg x_{ki} \vee \neg x_{k+1,j}\right)$，因此需要空间为$O(\log n)$，由推论 5.2 知，可在 poly($n$) 内完成。

在以后的归约中，根据构造很容易知道计算所需时间为 poly(n) 的多项式，因此，我们不再具体分析归约的时间复杂性。

例5.2　REACHABILITY 归约到 CIRCUIT VALUE。

给定图G，1 与 n 为G 的两个顶点，构造一个没有变量的电路$R(G)$，使得$R(G)$的输出为*true* 当且仅当G 中有从 1 到 n 的通路。

我们考虑G 的任意两点i 与 j 之间的一条通路$w_0 = i, w_1, \Lambda, w_t, \Lambda, w_l = j$。在其中选取最大值顶点如$w_t = k$，则该通路是$w_0 = i, w_1, \Lambda, w_t$ 和$w_t, \Lambda, w_l = j$ 同时是通路的连接，因此是二者都同时成立的合取。于是，可得到一个 AND 门$h_{i,j,k}$，即$s\left(h_{i,j,k}\right) = \wedge$，其前继为$g_{i,k,k-1}$ 和$g_{k,j,k-1}$，这里$h_{i,j,k}$ 表示i 与 j 之间的一条通路，且所有中间点均不超出k，k 一定出现在其中。$g_{i,k,k-1}$ 或$g_{k,j,k-1}$表示i（或k）与k（或 j）之间的一条通路，且所有中间点均不超出$k-1$。

另外，两点i 与 j 之间或者有对应于$h_{i,j,k}$ 的一条通路$w_0 = i, w_1, \Lambda, w_t, \Lambda, w_l = j$ 外，或者还有另外一条对应于$g_{i,j,k-1}$ 的通路$w_0 = i, w'_1, \Lambda, w'_{t'}, \Lambda, w'_{l'} = j$。此时，关于$i$ 与 j 之间的连通性则只要二者之一成立即可，于是有 OR 门$g_{i,j,k}$，即$s\left(g_{i,j,k}\right) = \vee$，其前继为$g_{i,j,k-1}$ 和$h_{i,j,k}$。如此组合下去，就将各个通路组合起来，得到从 1 到 n 的通路，输出结果。因此我们可以如下定义$R(G)$：

定义$R(G)$的门：$g_{i,j,k}$，$1 \le i, j \le n$，$0 \le k \le n$，和$h_{i,j,k}$，$1 \le i, j, k \le n$。于是有$2n^3 + n^2$ 个门并满足

$g_{i,j,k} = true$ 当且仅当在G 中从i 到 j 的通路上所有中间点均不超出k。

$h_{i,j,k} = true$ 当且仅当在G 中从i 到 j 的通路上所有中间点均不超出k，而且k 一定出现在其中。

下面对这些门给予归类：

- $k = 0$时，定义$g_{i,j,0} = true$ 当且仅当$i = j$ 或(i, j)为一条边；否则为 *false*。令$g_{i,j,0}$ 为输入门。注意，$h_{i,j,0}$无定义。
- $k \ge 1$时，令$h_{i,j,k}$为 AND 门，即$s\left(h_{i,j,k}\right) = \wedge$，其前继为$g_{i,k,k-1}$ 和$g_{k,j,k-1}$。令$g_{i,j,k}$ 为 OR 门，即$s\left(g_{i,j,k}\right) = \vee$，其前继为$g_{i,j,k-1}$ 和$h_{i,j,k}$。令$g_{1,n,n}$ 为输出门。

这就得到电路$R(G)$。对电路的门按照第三个指标k 的非递减序重新命名，使得门有从低到高的排列顺序。对k 归纳可以证明：*G 中有从 1 到 n 的通路当且仅当*$R(G)$

的输出为true。易知，存在机器可在时间 poly(n) 内计算 $R(G)$。

例5.3　CIRCUIT SAT 归约到 SAT。

给定电路 C，构造一个 Boolean 表达式 $R(C)$，使得 $R(C)$ *可满足当且仅当* C *是可满足的*。由于二者是 Boolean 函数的不同表示形式，因此该归约应该容易。$R(C)$ 中变量应该表示 C 中的变量以及 C 的各个门。具体变量定义如下：

- C 的变量输入门 g：对应变量 x，则关于任何赋值 T，有 $T(g)=T(x)$，于是 $R(C)$ 中　需添加 $g \Leftrightarrow x$，即，$(\neg g \vee x)\wedge(g \vee \neg x)$。
- 常值输入门 $g = true$ 或 $false$，则分别添加分句 (g) 或 $(\neg g)$。
- NOT 门 g：　设其前继为 h，则 $g = \neg h$，于是应添加分句 $g \Leftrightarrow \neg h$，即，$(\neg g \vee \neg h)\wedge(g \vee h)$。
- OR 门 g：设其前继为 h 与 h'，则 $g \Leftrightarrow h \vee h'$，即，$(\neg h \vee g)\wedge(\neg h' \vee g)\wedge(h \vee h' \vee \neg g)$。
- AND 门 g：设其前继为 h 与 h'，则 $g \Leftrightarrow h \wedge h'$，即，$(\neg g \vee h)\wedge(\neg g \vee h')\wedge(\neg h \vee \neg h' \vee g)$
- 输出门 g：添加 (g)。

以上分句取合取，则得 $R(C)$。直接验证，C 可满足当且仅当 $R(C)$ 可满足。

由于每个 SAT 成员都可直接得到一个电路，因此，SAT 与 CIRCUITSAT 是互相归约的，二者一样困难。对于 $k \geq 1$，定义 $kSAT$ 为 SAT 的一个特殊情况，其中公式是合取范式而且所有分句至多有 k 个文字。由于每个分句可用增加文字的个数，如，$x_1 \vee x_2 \vee x_3 = (x_1 \vee x_2 \vee x_3)\vee(w \wedge \neg w) = (x_1 \vee x_2 \vee x_3 \vee w)\wedge(x_1 \vee x_2 \vee x_3 \vee \neg w)$，其中 w 为新文字；或重复一个文字使得 $kSAT \subseteq (k+1)SAT$。于是有序列 $2SAT \subseteq 3SAT \subseteq \cdots \subseteq kSAT \subseteq (k+1)SAT \subseteq \cdots \subseteq SAT$。由于在 CIRCUITSAT 到 SAT 的归约中所构造的分句至多有 3 个文字，因此可将 CIRCUITSAT 归约到 $kSAT$，$k \geq 3$，从而亦得到 SAT 到 $kSAT$ 的归约。即，对于 $k \geq 3$，SAT 与 $kSAT$ 可以相互归约，有一样的困难性。

由前面例子得到归约序列 $REACHABILITY \subseteq CIRCUITVALUE \overset{id}{\subseteq} CIRCUITSAT \subseteq SAT$，其中 id 为恒等归约。于是得到 REACHABILITY 到 SAT 的特殊归约，归约的像为 SAT 的一个特殊情形。由定义可知对于一般问题之间的归约具有传递性，甚至我们有更强的结论。

定理 5.1　对于语言 L_1，L_2 和 L_3，若 R 为 L_1 到 L_2 的归约，R' 为 L_2 到 L_3 的归约，并且 R 与 R' 均在空间 $\boldsymbol{O}(\log n)$ 内可计算，则 $R \circ R'$ 为 L_1 到 L_3 的归约且在空间 $\boldsymbol{O}(\log n)$ 内可计算。

证明：对于任意 x，有 $x\in L_1$ 当且仅当 $R(x)\in L_2$，而 $R(x)\in L_2$ 当且仅当 $R'(R(x))\in L_3$，因此只要证明在空间 $O(\log n)$ 内可以计算 $R\,\mathrm{o}\,R'$。设 R 与 R' 分别由机器 M_R 和 $M_{R'}$ 计算，构造计算 $R\,\mathrm{o}\,R'$ 的图灵机 $M_{R\mathrm{o}R'}$（如图 5.1-1）：

将 M_R 的输出带与 $M_{R'}$ 的输入带合并为一条工作带 **I**，用于记录 M_R 输出带当前所指的符号 σ_i 和 $M_{R'}$ 输入带指针的位置 i，即 (σ_i,i)，当指针向右移动一位时，将 (σ_i,i) 擦去，改写为 $(\sigma_{i+1},i+1)$；然后再添加一条工作带 **J**，用于 $M_{R'}$ 输入带上指针左移时的计算。

初始 $i=1$，有单独的带集合模拟 M_R 关于输入 x 的运算。由于输入带上指针开始扫描 $>$，故易模拟 $M_{R'}$ 的第一次移动，当 $M_{R'}$ 的输入带上指针右移一位时，$i\leftarrow i+1$，执行 M_R 的运算足够长时间以便能产生下一个输出符号 σ_{i+1}，将 σ_i 改写为 σ_{i+1}，此即为当前 $M_{R'}$ 的输入带指针扫描的符号，继续模拟 $M_{R'}$，如此下去，…。

若 $M_{R'}$ 的输入带上指针在某相同位置 i 不动，则以 (σ_i,i) 继续模拟 $M_{R'}$，不对 (σ_i,i) 做改写。若 $M_{R'}$ 的输入带指针向左移动一位，由于 M_R 输出的符号已经被忘记，则利用工作带 J 采取如下方法：

在 I 带上，令 $i\leftarrow i-1$；然后关于输入 x，重新开始运行 M_R，同时在带 J 上记录 M_R 的输出的符号 σ_j 及符号数 j，即记录 (σ_j,j)，并比较 I 带上的 i 与 J 带上的 j。当 $i=j$ 时，则 M_R 暂时停机，将 σ_{i+1} 改写为 $\sigma_i=\sigma_j$，然后以 σ_i 为输入继续模拟 $M_{R'}$。

如此下去，直到 M_R 与 $M_{R'}$ 停机，则 $M_{R\mathrm{o}R'}$ 停机并输出 $R'(R(x))$。易知，$M_{R\mathrm{o}R'}$ 确定的计算了归约 $R\,\mathrm{o}\,R'$，所用空间为 $O(\log n)$。

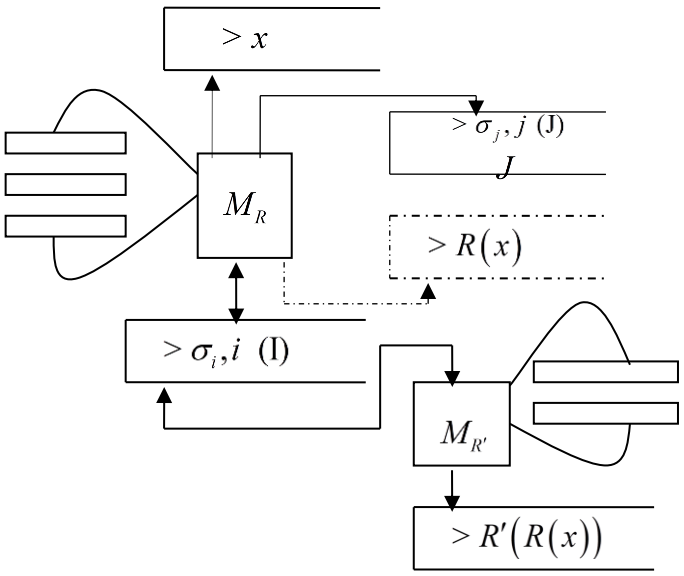

图 5.1-1　计算 $R\,\mathrm{o}\,R'$ 的图灵机 $M_{R\mathrm{o}R'}$

第二节　完备性

归约的传递性可依据困难性给问题排序，而该排序的极大元则是重点研究。

定义5.2　设C是一个复杂类，L为一个语言，称L为C-困难的，如果任何语言$L^{'} \in C$都可归约到L。若还有$L \in C$，则称L为C-完备的。

特别地，如果C=NP，则L是NP-难的。此外，如果L ∈NP，则L为NP-完备的，记为L ∈NPC。

注意，定义5.2提供了证明语言L是NP完全语言的步骤：

(1) 证明L∈NP；
(2) 选取一个已知的NP完备语言$\boldsymbol{L}'$；
(3) 设计一种算法来计算一个函数R，它把$\boldsymbol{L}'$中的每个实例x都映射为L中的一个实例R(x)；
(4) 证明对于所有x，函数R满足x∈$\boldsymbol{L}'$当且仅当R（x）∈L；
(5) 证明计算函数R的算法是多项式时间的。

称一个语言类C在归约下封闭，如果只要L可归约到L'，且$L' \in C$，则$L \in C$。显然，语言类P，NP，coNP，L，NL，PSPACE，和EXP都是归约下封闭的。完备性问题存在的好处之一是区分归约下封闭的语言类。对于两个语言类C和C'，有$C' \subseteq C$。若C中有一个完备问题A不属于C'，则$C' \subsetneq C$；而若$A \in C'$，则$C' = C$。如，对于语言类P与NP，只要找到一个NP-完备问题属于P，则有P=NP。显然，若两个语言类C与C'在归约下封闭，并且存在一个语言L对于C与C'都是完备的，则$C' = C$。这是复杂性研究中完备问题应用的一个基本方法。

为了说明CIRCUITVALUE、CIRCUIT SAT和SAT完备性，下面介绍一个理解时间复杂性的方法——计算表方法。

假设图灵机$M = (K, \Sigma, \delta, s)$在多项式时间内判定语言$L$。关于输入$x$，$M$的计算可被视为一个$|x|^k \times |x|^k$的表：不妨设$M$为单带机器，关于任意输入$x$，在至多$|x|^k - 2$步后停机。　定义表的行对应运行的时间，即计算步数，在0与$|x|^k - 1$之间；列对应M带中指针的位置，亦在0与$|x|^k - 1$之间。于是表中(i, j)处的赋值为M在i步后指针所指带的第j个位置处的内容。为了使计算表标准化，做如下改进：

① 用足够的∪填充串的右边，使其总长为$|x|^k$。于是实际计算不会到表的最右端。

② 若在时间i时状态为q，并且指针扫描第j个位置，则第(i, j)处赋值记为σ_q，即在时间i，位置j处的符号为σ，此时状态为q，若$q = yes$或no，则用yes或no代替σ_q。

>	0_s	1	0	∪	∪	∪	∪	∪
>	>	1_{q_0}	0	∪	∪	∪	∪	∪
>	>	1	0_{q_0}	∪	∪	∪	∪	∪
>	>	1	0	$\cup_{q_0}$	∪	∪	∪	∪
>	>	1	$0_{q_0'}$	∪	∪	∪	∪	∪
>	>	1_q	∪	∪	∪	∪	∪	∪
>	$>_q$	1	∪	∪	∪	∪	∪	∪
>	>	1_s	∪	∪	∪	∪	∪	∪
>	>	>	$\cup_{q_1}$	∪	∪	∪	∪	∪
>	>	$>_{q_1'}$	∪	∪	∪	∪	∪	∪
>	>	>	*yes*	∪	∪	∪	∪	∪

图 5.2-2　判定回文的计算表

③　设机器的指针不在 > 处，而是在输入的第一个符号处开始，即，指针不访问 > 。于是，计算表中每行的第一个符号是 > ，而不是 $>_q$ 。

④　若机器在时间界 n^k 内停机时， yes/no 出现在不是最后一行，则所有后面的行将等同于这一行。

称一个计算表 T 是接受的，如果对某个 j ，有 $T_{|x|^k-1,j}=yes$ 。于是，关于输入 x ，M 接受 x 当且仅当 M 的计算表是接受的。

例5.4　在 $O(n^2)$时间内判定回文。设输入为 $x=010$ 。取 $k=2$ ，$|x|^k=9$ ，计算表如图 5.2-2。由计算表知，(i,j) 处的赋值仅与 $(i-1,j-1)$ ，$(i-1,j)$ ，$(i-1,j+1)$ 处的赋值有关。

定理 5.2　CIRCUITVALUE 是 ***P***-完备的。

证明：由于 CIRCUITVALUE $\in \boldsymbol{P}$，故只需要证明完备性，即，对于任意语言 $L\in P$ ，找一个归约 R ，将 L 归约到 CIRCUITVALUE。具体地，给定 x ，有无变量的电路 $R(x)$ ，使得 $x\in L$ 当且仅当 $R(x)$ 值为 $true$ 。不妨设 M 在 n^k 时间内判定 L 。考虑 M 关于输入 x 的计算表 T：

当 $i=0$ 或 $j=0$ ，或 $j=|x|^k-1$ 时，T_{ij} 即为 x 的第 j 个符号，或 ∪ ，> 之一，这是事先知道的。

对于任意的 i 与 j ，在时间 i 带的第 j 个位置处的内容 T_{ij} 由时间 $i-1$ 时相同位置和相邻位置决定的，即，由 $T_{i-1,j-1}$ ，$T_{i-1,j}$ 和 $T_{i-1,j+1}$ 决定 T_{ij} 。若 $T_{i-1,j-1}$ ，$T_{i-1,j}$ 和 $T_{i-1,j+1}$ 均为 Σ 中的符号，则在时刻 $i-1$ 时，指针不在 j 的周围或 j 处，则 $T_{ij}=T_{i-1,j}$ 。若这三个位置之一为 σ_q ，则 T_{ij} 为 $T_{i-1,j}$ ，或者是一个形为 σ_q 的新的符号而且此时指针恰好移动到 j 处。

令 Γ 为出现在表中的所有符号的集合， $m=\lceil \log|\Gamma| \rceil$ 。将每个符号 $\sigma \in \Gamma$ 编码为 $(s_1,\mathrm{L}\ ,s_m)$， $s_i \in \{0,1\}$ 。于是计算表就可视为二元赋值 s_{ijl} 的表， $0 \le i \le n^k-1$， $0 \le j \le n^k-1$， $1 \le l \le m$ 。于是， $T_{i-1,j-1}=(s_{i-1,j-1,1},\cdots,s_{i-1,j-1,m})$， $T_{i-1,j}=(s_{i-1,j,1},\cdots,s_{i-1,j,m})$ 和 $T_{i-1,j+1}=(s_{i-1,j+1,1},\cdots,s_{i-1,j+1,m})$， $T_{i,j}=(s_{i,j,1},\cdots,s_{i,j,m})$ 。于是存在 m 个 Boolean 函数 $F_1,\cdots,F_m$，均有 $3m$ 个输入，1 个输出，使得对于所有 $i,j>0$， $s_{ijl}=F_l(T_{i-1,j-1},T_{i-1,j},T_{i-1,j+1})$ 。自然得到对应于 $F_1,\mathrm{L}\ ,F_m$ 的电路 C（图 5.2-3），其有 $3m$ 个输入， m 个输出，此即计算 T_{ij} 的二元编码， $i=1,2,\cdots,|x|^k-1$， $j=1,2,\cdots,|x|^k-1$ 。这里 C 是一个仅依赖于 M ，大小为固定的常数且与 x 无关的电路。下面描述归约 R ：

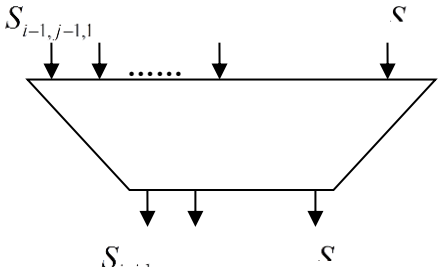

图 5.2-3　电路 C

对于每个输入 x， $R(x)$ 是一个由 $(|x|^k-1)(|x|^k-1)$ 个电路 C 的复制复合而成的大电路，其输入门是由分别对应于计算表第一行，第一列和最后一列的门组成，这些门的分类（*true* 或 *false* ）分别对应于这三条线上的内容，输出门是电路 $C_{|x|^k-1,1}$ 的第一个输出。

对于计算表中每个 $i \ne 0$， $j \ne 0$ 和 $j \ne |x|^k-1$ 处的赋值 T_{ij}，有一个 C 的复制 C_{ij} 。对于 $i \ge 1$， C_{ij} 的输入门为 $C_{i-1,j-1}$， $C_{i-1,j}$ 和 $C_{i-1,j+1}$ 的输出门（图 5.2-4）。这即完成了 $R(x)$ 的描述。

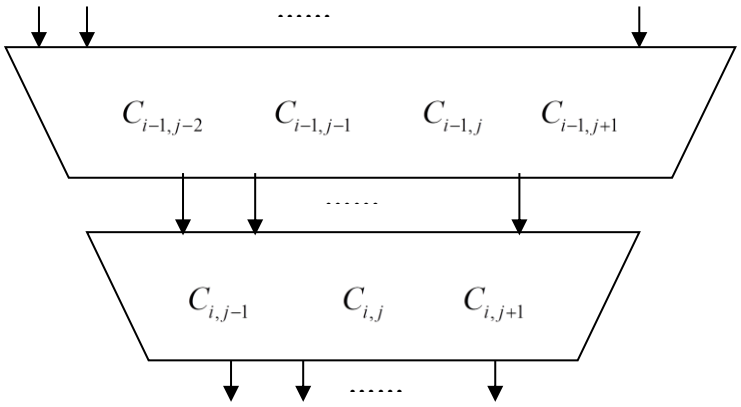

图 5.2-4　电路复合

下证： $x \in L$ 当且仅当 $R(x)=true$ 。

事实上，设 $R(x)=true$，通过对 i 的归纳易知，C_{ij} 的输出值给出了 M 关于输入 x 的计算表的二元形式赋值。于是，$T_{|x|^k-1,1}=yes$。这说明该表是接受的，从而 M 接受，即 $x\in L$。反过来，若 $x\in L$，则计算表接受。于是，$R(x)=true$。

下面说明可在多项式时间内完成，只需说明可在空间 $O(\log|x|)$ 内实现 R 即可。事实上，由于电路 C 是由 M 固定的，因此 R 的计算仅涉及如下运行：

1） 构造输入门。这可由检查 x，并计数到 $|x|^k$ 完成。

2） 生成电路 C 关于位置 (i,j)，$1\le i,j\le|x|^k-1$ 的复制。

3） 将以上所有这些输入与输出门分别确定并对应起来。

以上操作仅涉及 i， j 的运算，$0\le i,j\le|x|^k-1$，可在空间 $O(\log|x|)$ 内完成，因此可在多项式时间内完成。

在 CIRCUITVALUE 中，电路有 AND，OR，NOT 门。利用德·摩根律可将 NOT 略去，称这样的电路为单调电路（Monotone Circuit）。

推论5.1 单调电路的 CIRCUITVALUE 是 P-完备的。

定理 5.3 (Cook 定理) SAT 是 NP-完备的。

证明：显然，$SAT\in NP$。由于 CIRCUIT SAT 可归约到 SAT，故只需要证明 NP 中所有语言都可归约到 CIRCUIT SAT 即可，即，$L\in NP$，*对于任意* x，*构造输入为变量或常值的电路* $R(x)$，*使得* $x\in L$ *当且仅当* $R(x)$ *可满足。*

类似于定理 5.2 即可证明。仅有的不同在于非确定图灵机每步选择有非确定度 $|\Delta(q,\sigma)|=m>2$，于是我们可添加 $m-2$ 个状态，使得每步仅有两个选择，实现相同的效果。于是，机器对每个状态-符号组合恰有两种选择，即 0-选择和 1-选择。此时，只需要在电路构造中添加一列非确定选择的比特串 $(c_0,c_1,\mathrm{L}\ ,c_{|x|^k-1})\in\{0,1\}^{|x|^k-1}$，即可完成证明。

第三节 NP 关系

前面利用非确定图灵机定义了语言类 NP，现在从另一个角度观察 NP。一般地，对于计算问题 π 可以抽象出一个二元关系 $\boldsymbol{R}_\pi$，将问题转化为 $\boldsymbol{R}_\pi$ 上的搜索问题“*输入* x，*寻找* y *使得* $(x,y)\in\boldsymbol{R}_\pi$”。注意，我们在这里总假定 y 是存在的。

设 $\boldsymbol{R}\subseteq\Sigma^*\times\Sigma^*$ 为串上的二元关系，称 $\boldsymbol{R}$ 为多项式时间可判定的，如果存在确定图灵机在多项式时间内判定语言 $\{x;y:(x,y)\in\boldsymbol{R}\}$。记 $L(\boldsymbol{R})=\{x:$ 存在 y 使得 $(x,y)\in\boldsymbol{R}\}$。称 $\boldsymbol{R}$ 为多项式平衡的，如果 $(x,y)\in\boldsymbol{R}$，则对某个 $k\ge1$ 有 $|y|\le|x|^k$。若关系 R 既为多项式可判定的又为多项式平衡的，则称 $\boldsymbol{R}$ 为 NP -关系。NP -关系则是一

般关系 $\boldsymbol{R}_{\pi}$ 的一个子类，于是 $\boldsymbol{P}\overset{?}{=}\boldsymbol{NP}$ 问题可描述为问题：*对于每个 NP -关系* $\boldsymbol{R}$，*是否存在确定图灵机在多项式时间内判定* $L(\boldsymbol{R})$ *？*

定理 5.4 设 $L\in\Sigma^{*}$ 是一个语言，则 $L\in\boldsymbol{NP}$ 当且仅当存在 *NP* -关系 $\boldsymbol{R}$ 使得 $L=L(\boldsymbol{R})$。

证明：设 $\boldsymbol{R}$ 为一个 *NP* 关系且 $L=L(\boldsymbol{R})=\{x:$存在 y，使得 $(x,y)\in\boldsymbol{R}\}$。于是，存在确定的图灵机 $\mathbf{M}_{\boldsymbol{R}}$ 有效判定 $\{x,y:(x,y)\in\boldsymbol{R}\}$，且对于某个 k 有 $|y|\le|x|^{k}$。定义非确定图灵机 $\mathbf{N}$ 满足：关于输入 x，猜测一个 y 使得 $|y|\le|x|^{k}$；然后，以 x,y 为输入运行 $\mathbf{M}_{\boldsymbol{R}}$，检验是否 $(x,y)\in\boldsymbol{R}$。若 $(x,y)\in\boldsymbol{R}$，则输出 *yes* 并接受；否则，拒绝。于是，关于 x，$\mathbf{N}$ 输出 *yes* 当且仅当 $x\in L$。故 $L\in\boldsymbol{NP}$。

反过来，若 $L\in\boldsymbol{NP}$，则存在非确定图灵机 $\mathbf{N}$ 判定 L，运行时间为 $|x|^{k}$，$k\ge1$。定义关系 $\boldsymbol{R}$：$(x,y)\in\boldsymbol{R}$ 当且仅当，关于输入 x，y 为 $\mathbf{N}$ 的一个接受计算的编码。于是 $|y|\le|x|^{k}$ 且 $L=\{x:$存在 y 使得 $(x,y)\in\boldsymbol{R}\}$。对于 $\{x,y:(x,y)\in\boldsymbol{R}\}$，定义图灵机 $\mathbf{M}_{\boldsymbol{R}}$ 如下：

关于输入 x 和 y，以 x 为输入，以 y 为接受计算编码，运行 $\mathbf{N}$。若 $\mathbf{N}(x)=yes$，则 $\mathbf{M}_{\boldsymbol{R}}$ 输出 *yes*；否则，*no*。

于是，$\mathbf{M}_{\boldsymbol{R}}$ 判定 $\{x,y:(x,y)\in\boldsymbol{R}\}$ 且运行时间至多为 $|x|^{k}$，故 $\boldsymbol{R}$ 为一个 *NP* -关系。

这个等价的条件可以用作 *NP* 的定义，则 $\boldsymbol{NP}=\{L(\boldsymbol{R}):\boldsymbol{R}$ 为一个 *NP* -关系$\}$。这里我们称 y 为关于 x 的证书，它是一个关于 x 的知识。对于 *NP* 的任何 *yes* 实例 x，证书 y 一定存在，但是也许不知道如何在多项式时间内找到 y；而对于 *no* 实例，则没有这样的证书。后面我们会知道，对于 *NP* 语言的一个实例 x 及其证书 y，一定对应一个以 y 为知识的零知识证明系统。一旦给定 y 则可以有效验证 x 是否为 *yes* 实例。关于证书的寻找则是一个搜索问题，通常比判定问题更难。但是后面会介绍对于 *NP* 完备问题我们可以利用判定性问题有效求解搜索问题。

例5.5 给出下面一组例子，说明判定问题可以转化为搜索问题：

(1) **SAT** 对于 $\phi\in\mathbf{SAT}$，其证书是满足 ϕ 的赋值 T。于是对应的搜索问题是：
输入一个 Boolean 表达式 ϕ，去寻找赋值 T 使得 T 满足 ϕ。对应的关系为

$$\boldsymbol{R}_{\mathbf{SAT}}=\{(\phi,\mathrm{T}):\mathrm{T}\models\phi\}$$，而且 $L(\boldsymbol{R}_{\mathbf{SAT}})=\mathbf{SAT}$。

(2) **图的 3-染色问题** 给定一个无向图 $G=(V,E)$，V 为顶点集，E 为边集，G 是否有一个 3-染色？所谓一个 3-染色，即为一个映射 $\varphi:V\to\{1,2,3\}$ 使得，对于任意 $(u,v)\in E$ 有 $\varphi(u)\ne\varphi(v)$。

- 搜索问题：给定一个无向图 $G=(V,E)$，寻找 G 的一个 3-染色。
- 定义关系 $\boldsymbol{R}_{3col}=\{(G,\varphi):G$ 为无向图，φ 为一个 3-染色$\}$。

- $L(\boldsymbol{R}_{3col}) = \{G : G$ 为 3-染色图$\}$。

第四节　Oracle 图灵机

对于一个 Boolean 表达式$\phi = \phi(x_1, x_2, \ldots, x_n)$为 n 个变量的合取范式，我们可以想象有一个可以判定是否可满足的算法 D，在计算期间我们可以询问该算法是否是可满足的并得到即时正确回答，将该回答用于接下来的计算。事实上，要构造一个算法，利用 D，通过建立部分赋值逐步求解ϕ可满足赋值 T。具体为：用ϕ询问 D，若回答"*no*"，则拒绝；若回答"*yes*"，则执行：令 T(x_1)=1，用$\phi_1 = \phi(1, x_2, \ldots, x_n)$询问 D，若回答"*yes*"，则再选取 T($x_2$)；若回答"*no*"，则取 T($x_1$)=1。不妨设，　T($x_1$)=$\sigma_1$，然后再同样通过调用 D 选取 T($x_2$)= σ_2，如此下去，经过 n 步后可得到赋值 T 使得 T$\models\phi$。

这里实际上构思了一个 *SAT* 的理想世界能够正确的免费回答我们所有的关于 *SAT* 的询问，称之 Oracle，这当然是不现实的。当然，Oracle 可以对应于任何具体语言，而不仅仅是可满足性问题，也不一定对应于任何物理设备。

一旦我们定义了这样的 Oracle，就可以借用其强大的功能解决很多问题，特别是 *NP* 中的问题。但是 Oracle 的出现也使非确定图灵机的功能提到更神秘的高度。因此，对于 ***P.vs.NP*** 而言，我们可以构造两个 Oracle 使之有两个相反的回答。

定义5.3　一个带 Oracle 的图灵机$\mathbf{M}^?$是一个多带确定性图灵机，其有一条称之为询问带的特殊带和三个特殊状态：询问状态$q_?$，回答状态q_{yes}和q_{no}。

注意，这里定义的$M^?$与所用的 Oracle 无关，这里的？表明任何语言都可以被用作一个 Oracle。

设$A \subseteq \Sigma^*$是任意语言。除了询问状态引起的转移函数外，带有 Oracle A 的 Oracle 机器M^A的计算如通常图灵机一样运行，对 Oracle 询问写在询问带上。当机器进入询问状态$q_?$时，机器对 Oracle A 询问，即，询问串是否在 A 中，然后M^A从询问状态移动到回答状态q_{yes}/q_{no}，得到回答 yes/no，并允许机器在进一步计算中使用这个回答。关于输入x，M^A的计算输出表示为$M^A(x)$。带有 Oracle 的机器的时间复杂度定义方式如通常的图灵机，每步询问记作通常的一步计算。类似可定义带有 Oracle 的非确定机器。于是，若C是任意的确定或非确定时间复杂类，则可以定义C^A为相应机器所判定的语言类，而且时间界要加上对 Oracle A 的询问次数。

定理 5.5　1）设$\overline{\mathbf{SAT}}$为$\mathbf{SAT}$的补，则$\overline{\mathbf{SAT}} \in \boldsymbol{P}^{\mathbf{SAT}}$。

2）设$O \in C$，C为语言类，则$C^O = C$。

3）设$\mathbf{EXPCOM}$为语言$\{< M, x, 1^n >:$关于输入x，M在2^n步内输出1$\}$，则$\boldsymbol{P}^{\mathbf{EXPCOM}} = \boldsymbol{NP}^{\mathbf{EXPCOM}} = \mathbf{EXP}$。

证明：1）定义确定图灵机$\mathbf{M}^{\mathbf{SAT}}$，有 Oracle $\mathbf{SAT}$。对于任意 Boolean 表达式φ，询问 Oracle $\mathbf{SAT}$，$\mathbf{M}^{\mathbf{SAT}}$输出与 Oracle 回答相反即可。

2）显然，$C \subseteq C^O$。若$O \in C$，由于我们可以将任意判定C中语言的 Oracle 机器转变为一个标准的机器，仅需要用 Oracle O的计算替代每次 Oracle 调用即可。

3）显然，关于$\mathbf{EXPCOM}$的 Oracle 是允许以一次调用的代价执行指数时间的运算，因此，$\mathbf{EXP} \subseteq \boldsymbol{P}^{\mathbf{EXPCOM}}$，显然，$\boldsymbol{P}^{\mathbf{EXPCOM}} \subseteq \boldsymbol{NP}^{\mathbf{EXPCOM}}$且$\mathbf{EXPCOM} \in \mathbf{EXP}$。

另一方面，若N是一个非确定多项式时间 Oracle 图灵机，则可以在指数时间内模拟带有 Oracle $\mathbf{EXPCOM}$的$\mathbf{N}$的执行，故$\mathbf{EXP} \subseteq \boldsymbol{P}^{\mathbf{EXPCOM}} \subseteq \boldsymbol{NP}^{\mathbf{EXPCOM}} \subseteq \mathbf{EXP}$。

关于 Oracle 图灵机，值得注意的是，不管 Oracle 是什么，访问 Oracle O的所有图灵机都是具有两个性质：

I. 图灵机可以由字符串有效（不太长的）表示；

II. 一个图灵机无需更多的时间或空间代价模拟其它图灵机。

事实上，我们可以将$\mathbf{M}^O$表为字符串，并用通用图灵机$\mathbf{U}^O$以对数代价模拟每个这样的机器。

特别值得注意的是，Oracle 图灵机具有的这两个性质亦是我们的对角化方法的基础。因此，对于利用仅有性质 I 与 II 的图灵机或复杂类所得的任何结果对于带有 Oracle 的图灵机集合亦成立。即，假定 II 中两个图灵机带有同样的 Oracle，当被模拟的机器询问 Oracle 时，模拟机也询问，于是模拟可以如以前一样进行下去，因此凡是仅用利用性质 I 与 II（如，对角化方法）证明的结论在配以相同的 Oracle 时仍成立。称这样的结果为相对化的。事实上，关于通用图灵机的所有结果及用对角化方法的结果都是相对化结果。但是，下列定理却向我们表明，对角化方法似乎不能解决 *P.vs.NP* 问题。

定理 5.6 存在 Oracle A和B，使得$\boldsymbol{P}^A = \boldsymbol{NP}^A$和$\boldsymbol{P}^B \neq \boldsymbol{NP}^B$。

证明：由命题 6.6 可知，令$A = \mathbf{EXPCOM}$则有$\boldsymbol{P}^A = \boldsymbol{NP}^A$，下面构造$B_o$，使得$\boldsymbol{P}^{B_o} \neq \boldsymbol{NP}^{B_o}$。

对于任意语言B，定义$U_B = \{1^n : B \text{中有长为} n \text{的字符串}\}$。易知，$U_B \subseteq \boldsymbol{NP}^B$。（事实上，存在长为$n$的$x$，利用 Oracle 验证是否有$x \in B$，并以此作为输出即可。）下面构造$B_O$，使得$U_{B_O} \notin P^B$，则有$\boldsymbol{P}^B \neq \boldsymbol{NP}^B$。

设$\mathbf{M}_0^?$，$\mathbf{M}_1^?$…，为 Oracle 图灵机的一个字典排序。分阶段构造语言B，在阶段i，确保$\mathbf{M}_i^B$在$\frac{2^n}{10}$时间内不判定U_B。

初始，B为空集，每个阶段向B中添加一个串，而且每个阶段可以确定有限多个串的状态，即是否在B中。

在阶段i，我们已经确定了有限多个串是否在B中，即$B = B_{i-1}$。选择n使得对任意$x \in B$，有$|x| < n$。以1^n为输入，运行$\mathbf{M}_i^?$至多$\frac{2^n}{10}$步。只要$\mathbf{M}_i$用状态已确定的串询问 Oracle，回答将与状态一致。当询问状态未确定的串时，则声明该串不在B中。注意，此时B中没有任何长为n的串。

若$\mathbf{M}_i$在$\frac{2^n}{10}$步内停机，则关于1^n，令其回答不正确。具体的，若$\mathbf{M}_i$接受，则声

明所有长为n的串不在B中，即令$B_i = B_{i-1} = B$，从而确保$1^n \notin U_B$；若$\mathbf{M}_i$拒绝，则挑选一个未被询问的长为n的串x_i；并声明$x_i \in B$，从而确保$1^n \in U_B$。（注意，由于$\mathbf{M}_i$至多有$\frac{2^n}{10}$次询问 Oracle，故这样的串总存在）。总之，$\mathbf{M}_i$的回答总是不正确的，如此下去，可得语言B，并确保$U_B \notin P^B$。

事实上，上述证明中所用的$\boldsymbol{U_B}$不在$\boldsymbol{DTIME^B(f(n))}$中，这里$\boldsymbol{f(n) = O(2^n)}$为$2^n$的任意多项式。结合前一章我们可以看到任何试图用仅满足性质 I 与 II 的方法，如，对角化方法或任意模拟方法，去解决*P.vs.NP*的问题必须用到一些 Oracle 出现时不成立的结果。称这样的结果为非相对化的。目前遇到的一个简单例子为 Cook 定理，它对于一般 Oracle A 不是真的，即，存在语言$L \in \boldsymbol{NP}^A$不能多项式归约到$\mathbf{3SAT}$。后面还会遇到更多的例子，但是非相对化事实是必要性而不是充分的。如何利用非相对化事实求解*P.vs.NP*问题是一个有趣的公开问题。另外，只要我们证明一个复杂性理论的事实，就去检查是否可以用相对化技巧证明它，这样的思考将是很有启发性的。

第五节　自归约

虽然复杂性理论中更多的研究是围绕判定性问题进行的，但是人们自然地会问，*是否"解决判定性问题"的一个有效程序可以确保得到"解决搜索问题"的一个有效程序？*一搜索问题比相应的判定性问题更难，前者被解决，后者则必然可解。事实上，设$\boldsymbol{R}$为多项式可验证的关系， A 为关于$\boldsymbol{R}$的一个多项式时间的搜索算法，则可以构造对于$L(\boldsymbol{R})$的一个多项式时间判定算法：*关于输入*x*，模拟* A *的运行，回答*"yes" *当且仅当* A *停机并输出一个合适的*y *使得*$(x, y) \in R$。显然，该算法判定$L(R)$。

但是对于$\boldsymbol{NP}$-完备问题，"解决判定性问题"的一个有效程序可以确保得到"解决搜索问题"的一个有效程序。称从问题（语言）π_1到问题（语言）π_2的一个归约为 Cook 归约，如果该归约是带有一个关于π_2的 Oracle 的多项式时间机器$\mathbf{R}\pi_2$，关于输入x，只要得到关于π_2的 Oracle 回答，则可以求解（判定）π_1。由于 Oracle 机器是多项式时间的，因此这里对 Oracle 的询问至多多项式次。特别地，对于判定语言可叙述如下：

设π_1与π_2分别是语言L_1与L_2的判定问题，令χ_L是语言L的特征函数，即$\chi_L(x) = 1$，如果$x \in L$；否则，$\chi_L(x) = 0$。则称π_1可以 Cook 归约到π_2，如果存在一个 Oracle 机器$\mathbf{R}\pi_2$，关于输入x，询问关于L_2的 Oracle 多项式多次询问 q 并分别得到相应的回答$\chi_{L_2}(q)$，则输出$\chi_{L_1}(x)$。易知，Cook 归约具有传递性。在前面所定义的利用多项式时间对语言的归约通常被成为 Karp 归约，它一定是 Cook 归约。

定理 5.7　设L_1与L_2是两个语言。若L_1可 Cook 归约到L_2，并且$L_2 \in \boldsymbol{P}$，则$L_1 \in \boldsymbol{P}$。

证明：由于$L_2 \in \boldsymbol{P}$，则存在确定的多项式时间图灵机$\mathbf{M}_{L_2}$判定L_2。由L_1可 Cook

归约到L_2可得，存在一个Oracle机器$\mathbf{M}$，只要询问关于L_2的Oracle得到相应的回答，则可以判定L_1。下面构造判定L_1的机器$\mathbf{M}_{L_1}$：

关于输入x，运行$\mathbf{M}$，直到向Oracle提出询问q；将q输入给$\mathbf{M}_{L_2}$，运行$\mathbf{M}_{L_2}$得到输出$\mathbf{M}_{L_2}(q)$，并作为Oracle回答返回给$\mathbf{M}$；继续运行$\mathbf{M}$，并询问Oracle，直到没有更多的Oracle询问。运行$\mathbf{M}$到停机，输出$\mathbf{M}(x)$。

由于$\mathbf{M}_{L_2}$与Oracle给出相同的回答，故该算法的正确性是显然的。由于$\mathbf{M}$的运行时间为多项式的，故询问的次数和询问的长度是关于$|x|$的多项式。由于$\mathbf{M}_{L_2}$的运行时间是询问长的多项式，因此亦是$|x|$的多项式。于是，$\mathbf{M}_{L_1}$的运行时间为（$\mathbf{M}$的运行时间）+（$\mathbf{M}_{L_2}$的运行时间）×（询问次数），这是$|x|$的一个多项式。

由此可知，$\boldsymbol{P}$是Cook归约封闭的。对于$\boldsymbol{NP}$，$\boldsymbol{PSPACE}, \boldsymbol{NPSPACE}, \boldsymbol{EXP}$也有相同的结论。

定义5.4 设$\boldsymbol{R}$是一个二元关系，$\boldsymbol{R}$的相应语言为$L(\boldsymbol{R})=\{x:\exists y(x,y)\in \boldsymbol{R}\}$。称一个关系$\boldsymbol{R}$是自归约的，如果求解关于$\boldsymbol{R}$的搜索问题可以Cook归约到语言$L(\boldsymbol{R})$的判定问题。

于是，若一个关系是自归约的，则一定有一个多项式时间的Oracle图灵机求解搜索问题。对Oracle询问得到的回答是断言：*关于输入x'，是否存在y'，使得*$(x',y')\in R$。如，对于图的3-染色问题，定义Oracle为判定图G是否可以3-染色的图灵机$\mathbf{M}_1$。定义Oracle图灵机$\mathbf{M}$：给定图G作为输入，通过访问Oracle $\mathbf{M}_1$去寻找一个3-染色。于是，$\mathbf{M}$对关系$\boldsymbol{R}=\{(G,\varphi):\varphi$为$G$的3-染色$\}$的搜索问题求解过程中，其所做的询问不局限于$G$而是多项式个图，而且每个图又只依赖于前面的询问。

例5.6 **SAT**：设φ为变量集$\{x_1,x_2,\Lambda,x_n\}$上的合取范式。

输入：φ

目标：找到一个赋值T使得$\mathrm{T}|=\varphi$，即$\varphi(\mathrm{T}(x_1),\mathrm{K},\mathrm{T}(x_n))=true$。

于是有相应的关系$\boldsymbol{R}_{\mathbf{SAT}}=\{(\varphi,\mathrm{T}):\mathrm{T}|=\varphi\}$。显然，$\boldsymbol{R}_{\mathbf{SAT}}$为多项式时间可判定的。

断言： $\boldsymbol{R}_{\mathbf{SAT}}$是自归约的。

事实上，要构造一个算法，利用判定$\mathbf{SAT}=L(\boldsymbol{R}_{\mathbf{SAT}})$的Oracle A，求解$\boldsymbol{R}_{\mathbf{SAT}}$上的求解问题。该算法是通过建立部分赋值逐步构造出一个解，具体如下：

- 询问是否$\varphi\in\mathbf{SAT}$，若回答"*no*"，则放弃。
- 对每个i，$1\le i\le n$，令$\varphi_i(x_{i+1},\Lambda,x_n):=\varphi(\sigma_1,\mathrm{K},\sigma_{i-1},1,x_{i+1},\Lambda,x_n)$，询问Oracle，检验是否$\varphi_i\in\mathbf{SAT}$。若回答"*yes*"，则取$\varphi_i=1$；否则，取$\varphi_i=0$。

显然，部分赋值$\mathrm{T}(x_1)=\sigma_1$，Λ，$\mathrm{T}(x_i)=\sigma_i$，可以扩充成一个可满足的赋值。因此，算法或者放弃或者输出一个赋值T使得$\mathrm{T}|=\varphi$。

由此可见，若**SAT**是多项式时间可判定的，则存在有效算法求解$\boldsymbol{R}_{\mathbf{SAT}}$的搜索问

题，由于已经证明**SAT**为***NP***-完备问题，因此***NP***-完备问题是自归约的。

例5.7　图同构问题（GRAGH　ISOMORPHISM）

输入：两个简单图$G_1=(V,E_1)$，$G_2=(V,E_2)$，不妨设没有孤立点。

目标：找到G_1与G_2之间的一个同构，即有一个置换$\varphi:V\to V$，使得$(u,v)\in E_1$当且仅当$(\varphi(u),\varphi(v))\in E_2$。

于是，有相应关系$R_{GI}=\{(G_1,G_2,\phi):\phi$为$G_1$与$G_2$的同构$\}$

断言： $\boldsymbol{R}_{GI}$是自归约的。

事实上，设 A 为给定(G_1,G_2)判定G_1与G_2是否同构的 Oracle。构造询问 A 的算法**M**：

在每一步，算法**M**固定G_1的一个点u，并找到G_2的一个顶点v，定义映射φ，使得$\varphi(u)=v$。进而将φ补充为图的同构。具体如下：

为了检查是否u是否可以被映射到v，利用下面方式标注这两点：

1） 在u,v两点处添加$|V|$个叶片形成星，得到新的图$G_1^{'}$，$G_2^{'}$。

2） 用$(G_1^{'},G_2^{'})$询问 Oracle A，得到关于$G_1^{'}$与$G_2^{'}$是否同构的回答。若回答为no，则放弃；

若回答为yes，则G_1与G_2之间有一个同构φ，使得$\varphi(u)=v$（由于u,v的度数严格大于$G_1^{'}$与$G_2^{'}$中其它顶点的度数，因此，$G_1^{'}$，$G_2^{'}$之间存在同构$\varphi^{'}$当且仅当G_1与G_2之间有一个同构φ使得$\varphi(u)=v$）。

3） u与v被确定后，以添加$2|V|$个叶片，$3|V|$个叶片等等分别构成星来标注V中的顶点，

并不断询问 Oracle，确定对应的顶点，直到φ被完全确定。

一个***NP***语言也许同时有多个搜索问题，亦即有多个关系，但是一个给定的关系$\boldsymbol{R}$的自归约不能直接蕴含另一个关系$\boldsymbol{R}^{'}$的自归约。另一方面，人们认为并非所有的***NP***语言都有自归约的关系。

例5.8　对于合数语言$L_{comp}=\{N:N=n_1n_2;n_1,n_2>1\}$，有关系

$$\boldsymbol{R}_{comp}=\{(N;(n_1,n_2)):N=n_1n_2;n_1,n_2>1\}。$$

显然，$|(n_1,n_2)|=poly(|N|)$，而且可在多项式时间内判定$\boldsymbol{R}_{comp}$，故$L_{comp}\in\boldsymbol{NP}$。

已知存在一个随机算法可在多项式时间内判定L_{comp}，$\boldsymbol{R}_{comp}$上的搜索问题是寻找(n_1n_2)，$n_1,n_2>1$，使得$N=n_1n_2$。显然，该计算问题等价于分解问题（FACTORING）。但目前我们没有概率多项式算法求解 FACTORING。故$\boldsymbol{R}_{comp}$不是自归约的。

另外，对于语言$L_{QR}=\{(N,x):x$为模N的二次剩余$\}$，有关系

$$\mathrm{R}_{QR}=\{((N,x),y):y^2\equiv x \mod N\}。$$

$\boldsymbol{R}_{QR}$上的搜索问题在随机归约(即，归约算法是概率多项式时间的，这比 Cook 归

约更一般。)下等价于分解问题（FACTORING）。若假定分解问题比判定 L_{QR} 更难，则 $\boldsymbol{R}_{QR}$ 不是自归约的。

习题

(1) 令 $L \in NTIME(2^{n^c})$，其中c > 0为某常数。定义语言 $L_{pad} = \{(x,1^z) \mid z = 2^{|x|^c}\}$。证明 L_{pad} ∈NP。进而，如果P=NP，则NEXP=EXP。

(2) 证明：对于任意语言 L， $L \in DTIME(n^k)$ 当且仅当 $L' = \{(x,1^z) \mid x \in L,\ z = |x|^k\} \in \mathrm{DTIME}(n)$。进而，P≠PSPACE。

(3) 一个强非确定图灵机(sNDTM)是一个非确定图灵机，其有三个可能输出1，0和？。称sNDTM M判定一个语言L，如果

i）对于 $x \in L$，M关于 x 的每个计算得到1或？，并且至少存在M关于 x 的一个计算得到1。

ii）对于 $x \notin L$，M关于 x 的每个计算得到0或？，并且至少存在M关于 x 的一个计算得到？。

证明，L由一个sNDTM M在多项式时间内判定当且仅当L∈NP∩coNP。

(4) 考虑语言 $D = \{p(\cdot) \mid p$为多变量的整系数且有整数根的多项式$\}$。如，

$p_1(x,y) = 2x^2 + y - 6xy + 3$ 有整数根x = 0，y =-3，于是 $p_1 \in D$ 但

$p_2(x,y,z) = (2x+1)^2 + (z^2-2)^4 + 5y^6$ 没有整数解，因此 $p_2 \in D$。

(a) 事实上D 是不可判定语言，即，不存在图灵机，输入一个多项式表达式p，可判定是否p有整数根。试找出下列关于 $D \in NP$ 的证明的错误：

对于D，我们可以用与SAT相同的验证者。给定一个变量为$(x,\mathrm{L}\ ,z)$ 的多项式p，验证者V的证书是对这些变量的一个整数赋值的候选者，并且V只是简单计算$p(c)$。若$p(c)$ = 0 ，则接受；若$p(c)$≠0，则拒绝。显然，p有整数根当且仅当对某c 有V(p; c)接受。因此 $D \in NP$。

(b) 证明D是NP-困难的，即，NP中的任意问题都可以多项式归约到D。

(提示：如果Boolean变量有比特值0和1，则验证 $x \wedge y = xy$ (比特值的积)并且对于 $x \vee y$ 和 $\overline{x}$ 类似地可找到合适的表达式。因此，任意Boolean公式都可以变换为多项式。最后，找到一个办法使得，这样的多项式只要有整数根，则有仅以0，1 为输入值的整数根。)。