P vs NP

深圳大学计算机与软件学院
卢亚辉

本章大纲

- P
- NP
- NPC
- 图灵机与停机问题
- NPC的证明

问题的解

- 排序算法复杂度: $O(n^2)$、$O(n\log n)$ 和 $O(n)$。
- 问题：
  - 每个问题都能在多项式时间$O(n^k)$内解决吗？
  - 每个问题都能找到时间复杂度为$O(n^k)$的算法吗？
- 答案是否定的
  - 不可计算（不可判定）的问题：存在一些问题，任何计算机无论耗费多少时间都不能解决。
    - 图灵停机问题
    - 理论上不可行
    - 可计算性理论
  - 超多项式时间的问题：可以解决，但是不能在$O(n^k)$时间内解决。
    - 实际上不可行
    - NP问题：P=NP？
    - 比NP更难的问题

History

- The concept of NP-completeness was introduced in 1971 (see Cook–Levin theorem), though the term NP-complete was introduced later. At the 1971 STOC conference, there was a fierce debate between the computer scientists about whether NP-complete problems could be solved in polynomial time on a deterministic Turing machine.
- In 1972, Richard Karp proved that several other problems were also NP-complete (see Karp's 21 NP-complete problems); thus there is a class of NP-complete problems (besides the Boolean satisfiability problem). Since the original results, thousands of other problems have been shown to be NP-complete by reductions from other problems previously shown to be NP-complete

表 1.2　不同时间复杂性下不同输入规模的运行时间

n	$\log n$	n	$n\log n$	n^2	n^3	2^n	$n!$
1	0n	1n	0n	1n	1n	2n	1n
8	3 n	8 n	24 n	64 n	512 n	256 n	16.78ms
16	4 n	16 n	64 n	256 n	4.096 μ	65.536 μ	5.81h
32	5 n	32 n	160 n	1.024 μ	32.768 μ	4294.967 ms	10^{17} c
64	6 n	64 n	384 n	4.096 μ	262.144 μ	5.85 c	10^{70} c
128	7 n	128 n	896 n	16.384 μ	1997.152 μ	10^{20} c	...
256	8 n	256 n	2.048 μ	65.536 μ	16.777 ms	10^{58} c	
512	9 n	512 n	4.608 μ	262.144 μ	134.218 ms	10^{135} c	
1024	10 n	1.024 μ	10.24 μ	1048.576 μ	1073.742 ms	10^{289} c	
2048	11 n	2.048 μ	22.528 μ	4194.304 μ	8589.935 ms	10^{598} c	
4096	12 n	4.096 μ	49.152 μ	16.777 ms	68.719 s	10^{1214} c	
8192	13 n	8.196 μ	106.548 μ	67.174 ms	549.752 s	10^{2447} c	
16384	14 n	16.384 μ	229.376 μ	268.435 ms	1.222 h	10^{4913} c	
32768	15 n	32.768 μ	491.52 μ	1073.742 ms	9.773 h	10^{9845} c	
65536	16 n	65.536 μ	1048.576 μ	4294.967 ms	78.187 h	10^{19709} c	

n：纳秒　μ：微秒　ms：毫秒　s：秒　h：小时　d：天　y：年　c：世纪

表 1.3　计算机速度提高 10 倍后，不同算法复杂性求解规模的扩大情况

算法	A_1	A_2	A_3	A_4	A_5	A_6
时间复杂性	n	$n\log n$	n^2	n^3	2^n	$n!$
n_2和n_1的关系	$10n_1$	$8.38n_1$	$3.16n_1$	$2.15n_1$	$n_1+3.3$	n_1

P（Polynomial）类

- P类中包含的是在多项式$O(n^k)$时间内可解的问题。
- 如果一个问题存在$O(n^k)$的算法，那么这个问题就属于 P 类问题。
- 这意味着，即使面对大规模数据，人们也能相对容易地得到一个解，比如将一组数排序。
  - 一旦找到一个多项式$O(n^{100})$的算法，那么很快可以找到一些更有效$O(n^{10})$的方法
  - 对很多合理的计算模型，在一个模型（RAM）上用$O(n^k)$可解的问题，在另外一个模型上（图灵机）也可以在$O(n^k)$可解
  - $O(n^k)$可解问题具有很好的封闭性，在加法、乘法和组合运算下，多项式是封闭的

NP问题

- “NP”的全称为“Nondeterministic Polynomial”，而不是“Non-Polynomial”。
- NP 类问题指的是，能在多项式时间内检验verifiable一个解是否正确的问题。至于求解本身所花的时间是否是多项式不管，可能有多项式算法，可能没有，也可能是不知道，这类问题称为NP问题。
- NP is the set of decision problems for which the problem instances, where the answer is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine, or alternatively the set of problems that can be solved in polynomial time by a nondeterministic Turing machine.
- NP概念的奥妙在于，它躲开了求解到底需要多少时间这样的问题，而仅仅只是强调验证需要多少时间

求解与判断的不对称

- 在一个周六的晚上，你参加了一个盛大的晚会。由于感到局促不安，你想知道这一大厅中是否有你已经认识的人。你的主人向你提议说，你一定认识那位正在甜点盘附近角落的女士罗丝。不费一秒钟，你就能向那里扫视，并且发现你的主人是正确的。然而，如果没有这样的暗示，你就必须环顾整个大厅，一个个地审视每一个人，看是否有你认识的人。

- 生成问题的一个解通常比验证一个给定的解时间花费要多得多。这是这种一般现象的一个例子。
- 与此类似的是，如果某人告诉你，数１３７１７４２１可以写成两个较小的数的乘积，你可能不知道是否应该相信他，但是如果他告诉你它可以因式分解为３６０７乘上３８０３，那么你就可以用一个袖珍计算器容易验证这是对的。
- 密码问题：猜一个密码很难，验证一个密码是否对很容易
- 写东西很难，看东西很容易

NP问题

- 比如我的机器上存有一个密码文件，于是就能在多项式时间内验证另一个字符串文件是否等于这个密码，所以“破译密码”是一个 NP 类问题。
- NP 类问题也等价为能在多项式时间内猜出一个解的问题。这里的“猜”指的是如果有解，那每次都能在很多种可能的选择中运气极佳地选择正确的一步。

P 与 NP 问题

- 相似的问题，但是复杂度可能不同

最短与最长简单路径：在第 24 章中，我们看到了即使是在边有负的权值的情况下，也能在一个有向图 $G=(V, E)$中，在 $O(VE)$的时间内，从一个源顶点开始找出最短的路径。然而，寻找两个顶点间最长简单路径问题是 NP 完全的。事实上，即使所有边的权值都是 1，它也是 NP 完全的。

欧拉游程与哈密顿回路：对一个连通的有向图 $G=(V, E)$的欧拉游程是一个回路，它遍历图 G 中每条边一次，但可能不止一次地访问同一个顶点。根据思考题 22-3，可以在 $O(E)$时间内，确定一个图中是否存在着一个欧拉游程，并且，事实上，能够在 $O(E)$时间内找出这一欧拉游程中的各条边。一个有向图 $G=(V, E)$的哈密顿回路是一种简单回路，它包含 V 中的每个顶点。确定一个有向图中是否存在哈密顿回路的问题是 NP 完全的。(在本章的后面，我们还要证明，确定一个无向图中是否存在哈密顿回路的问题也是 NP 完全的。)

2-CNF 可满足性与 3-CNF 可满足性：在一个布尔公式中，可以包含这样一些成分：布尔变量，其取值可以是 0 或者 1；布尔连接词如 $\wedge$(AND)、$\vee$(OR)以及$\neg$(NOT)；括号。对一个布尔公式来说，如果存在着对其变量的某种 0 和 1 赋值，使得它的值为 1 的话，则称它是可满足的。在本章的稍后部分，还将更为形式地定义有关的术语。但是，非形式地，称一个布尔公式为 k 合取范式(conjunctive normal form)或 k-CNF，如果它是用 AND 连接若干个 OR 子句，且每个子句中恰有 k 个布尔变量或其否定形式的话。例如，布尔公式$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$就是一个 2-CNF。(它有一个可满足性赋值 $x_1=1$，$x_2=0$，$x_3=1$。)存在着一个多项式时间的算法来确定一个 2-CNF 公式是否是可满足的，但我们在本章稍后就会看到，要确定一个 3-CNF 公式是否是可满足的这一问题是 NP 完全的。

非NP问题

- 有不是NP问题的问题，如果你不能在多项式的时间里去验证它。
- Hamilton回路——给你一个图，问你能否找到一条经过每个顶点一次且恰好一次（不遗漏也不重复）最后又走回来的路
- NP问题： 存在Hamilton回路
- 非NP问题：不存在Hamilton回路。

P=NP？

- NP 问题能在多项式时间内“解决”，只不过需要好运气。显然，P 类问题肯定属于 NP 类问题。

- 所谓“P=NP”，就是问——是不是所有的 NP 问题，都能找到多项式时间的确定性算法？

- 究竟是否有P=NP？通常所谓的“NP问题”，其实就一句话：证明或推翻P=NP。

判定问题与优化问题

- 判定问题(decision problem, 决策问题)：讨论一个特定的表述是否为真(to be or not not be)
  - 查找：一系列数的序列中是否包含m?
- 优化问题（optimizaiton problem）：寻找一个最好的解答（ to be best）（所谓最好是根据人们的一些标准和规定进行判断的）
  - 最小优化问题：最小生成树、最短路径
  - 最大优化问题：背包问题
- 判定问题可以和优化问题转化
  - 背包问题的决策问题：

题，其对应的决策问题是：给定 n 个物品、物品 i 的价值 v_i、重量 w_i、一个总重量 w 和实数 c，是否存在一个物品组合，其价值大于等于 c，而总重量小于等于 w?

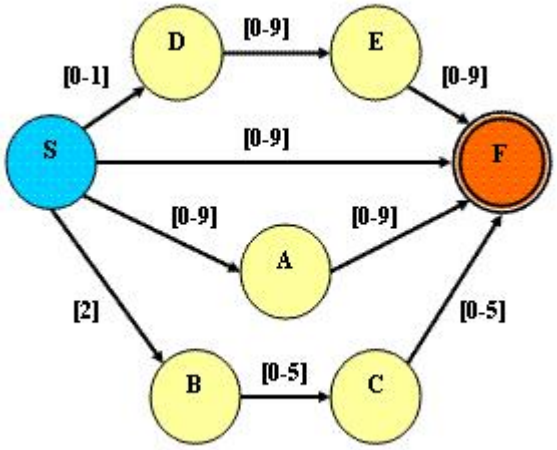
- 一般来说，判定问题比优化问题要容易一些

决策问题（decision Problem）

简单地说，决策问题需要我们回答的就是“是”和“否”两个答案中的一个。但是每个决策问题都必须有一个输入，而我们就是在这个给定输入的环境下来解答问题。

一个特定的输入也常常被称为相关问题的一个实例（Instance）

如果我们对一个决策问题的回答为“是”，则通常需要提供一个“证人”来证明我们的回答。例如，对于汉密尔顿回路问题，节点 v_1，v_2，…，v_n 的任意排列都是一个潜在的证



确定性算法

- **定义12.1** A是问题Π的一个算法。如果在处理问题的实例时，在算法的整个执行过程中，每一步只有一个确定的选择，就说算法是确定性的算法。

- 算法执行的每一个步骤，都有确定的选择。
- 重新用同一输入实例运行该算法，所得到的结果严格一致。

```
RESULT love(boy, girl)

{
    if( boy.有房() AND boy.有车() )
    {
      boy.Set(Nothing);
      return girl.嫁给(boy);
    }
    else if( girl.愿意等() )
    {
      next_year:
      for( day=1; day<=365; day++)
      {
        if( day == 情人节 )
          if( boy.GiveGirl(玫瑰) )
            girl.感情++;
          else
            girl.感情--;
        if( day == girl.生日)
          if( boy.GiveGirl(玫瑰) )
            girl.感情++;
          else
            girl.感情--;
        boy.拼命赚钱();
      }
      年龄++;
      girl.感情--;
      if( boy.有房() AND boy.有车() )
      {
        boy.Set(Nothing);
        return girl.嫁给(boy);
      }
      else if( boy.赚钱 > 100,000 AND girl.感情 > 8 )
        goto next_year;
      else
        return girl.goto( another_boy);
    }
    return girl.goto( another_boy);
}
```

- **定义12.2** 如果对某个判定问题Π，存在着一个非负整数k，对输入规模为n的实例，能够以O(n^k)的时间运行一个确定性的算法，得到yes或no的答案，则该判定问题是一个P类判定问题。

- 特性：P类判定问题是由具有多项式时间的确定性算法来解的判定问题

定义 2 *P* 类问题是一类能够用（确定性的）算法在多项式的时间内求解的判定问题。这种问题类型也称为多项式类型。

P类

- 排序（冒泡、插入、归并、快速、…）
- 查找（顺序、折半、哈希…）
- 最近邻
- 凸包

- O(n^k)

一些难题

- 有一些问题，既没有找到它们的多项式类型算法，也无法证明这样的算法不存在
  - 旅行商问题
  - 背包问题
  - 可满足问题
  - 哈密顿回路
  - 划分问题
  - 装箱问题
  - 图的着色问题
  - 整数线性规划

- 特点：
  - 求解问题很难，但是判断一个特定的解是否解决了该问题很容易

非确定性算法

一、非确定性算法

1、问题Π的非确定性算法的两个阶段：推测阶段和验证阶段。

2、推测阶段：对规模为n的输入实例x，以多项式时间$O(n^i)$产生输出y，而不管y的正确性

3、验证阶段：以多项式时间$O(n^j)$的确定性算法验证两件事情：

1）检查上一阶段的输出y是否具有正确的形式。

如果y不具正确的形式，算法就以答案no结束；

2）如果y具有正确的形式，则继续检查y是否是问题的输入实例x的解。

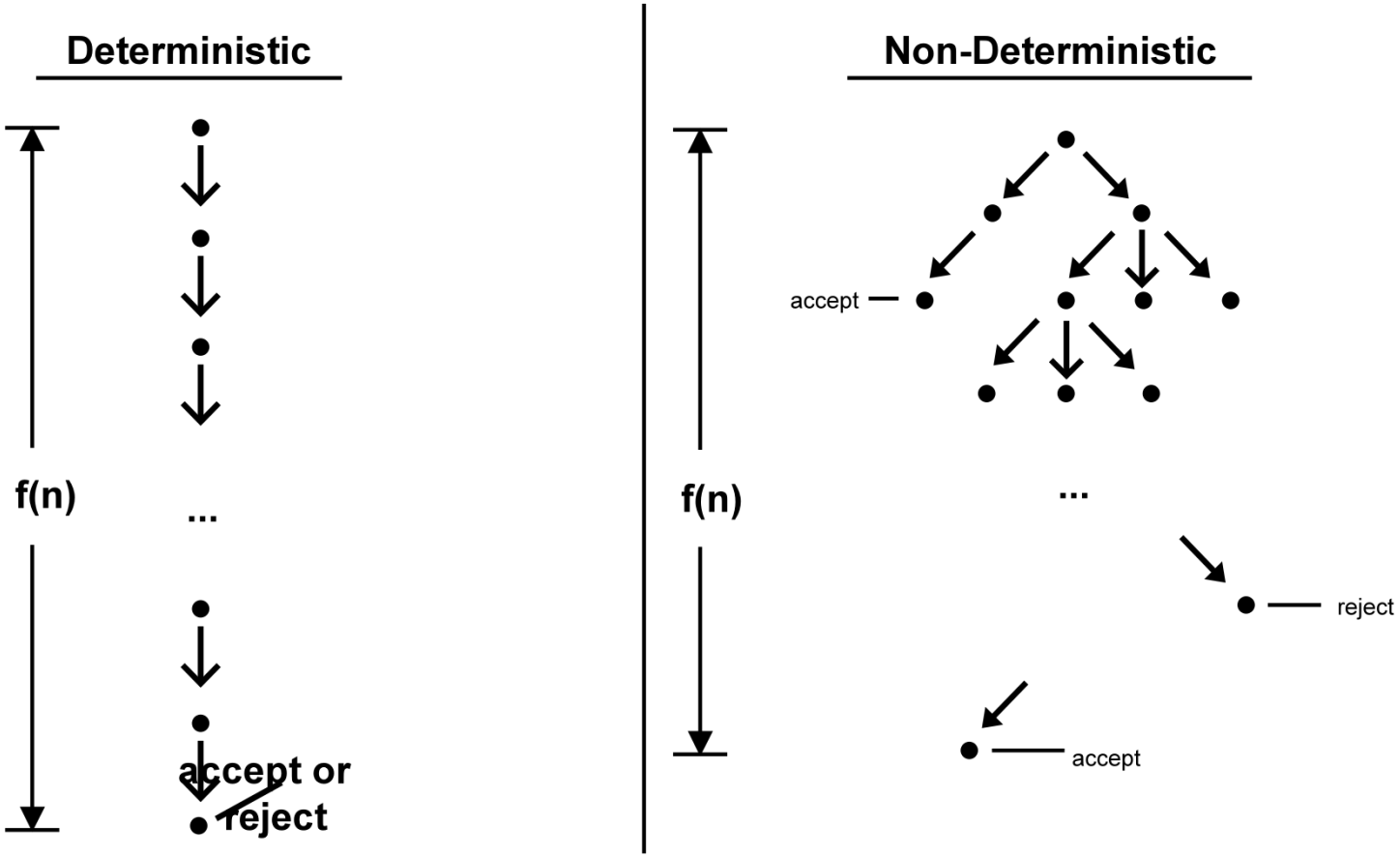
如果它确实是问题实例x的解，则以答案yes结束，否则，以答案no结束。

定义 3 一个**不确定算法**是一个两阶段的过程，它把一个判定问题的实例I作为它的输入，并进行下面的操作。

非确定（“猜测”）阶段：生成一个任意串S，把它当作给定实例I的一个候选解（但也可能是完全不着边际的）。

确定（“验证”）阶段：确定算法把I和S都作为它的输入，如果S的确是I的一个解的话，就输出“是”（如果S不是I的一个解，该算法要么返回“否”，要么根本就不停下来）。

Comparison of deterministic and nondeterministic computation



NP问题

二、 *NP* 类判定问题

1、定义：

定义 12.5 如果对某个判定问题Π，存在着一个非负整数 k，对输入规模为 n 的实例，能够以 $O(n^k)$ 的时间运行一个非确定性的算法，得到 *yes* 或 *no* 的答案，则该判定问题Π是一个 *NP* 类判定问题。

2、特性：

存在确定性的算法，能够以多项式时间，来检查和验证在推测阶段产生的答案。

解决的问题

定义 4 *NP* 类问题是一类可以用不确定多项式算法求解的判定问题。我们把这种问题类型称为**不确定多项式类**。

如果你还是不能理解非确定性图灵机，那我们再打个比喻。当你的左脸突然被人猛击一拳的时候，你会做何反应呢？你会回击？你会逃跑？你会破口大骂？你会打电话叫警察？或者将右脸伸过去让他打？事实上，你会做何反应并不是我们要关心的问题，而且没有人能肯定（你自己也不能肯定）你会做何反应，我们在乎的是上述每种可能都有发生的概率。而这种在输入相同的情况下，结果有可能是不确定的问题就是非确定性图灵机！

当然了，非确定性图灵机要比你高明，它的行为虽然我们不能确定，但它总是会选择最好的反应！（而你却不一定能选择最好的反应！）换一个角度来说，非确定性图灵机是世界上最幸运的猜谜手，它总能在无数可能中猜中最好的选择，即选择最好的状态转换以达到其最终目的（进入接受状态），如果这样一种选择存在！

从另一个角度看，这个问题就是非确定性图灵机能够同时进入所有可能状态，也就是说，能够像孙悟空那样，变出无数个自己，从而同时在多个地方出现！也许你听说过量子理论中的一个假说——所有可能发生的都已经发生，只不过它们发生在不同的宇宙而已！即你既是中国人，也是美国人（这里排除双重国籍），这两种可能都是事实！只不过，在这个宇宙里你只是中国人。（而在另一个宇宙里，你是美国人！）

到现在你也许看出来图灵机的强大了！而且我们也可以看清确定性图灵机和非确定性图灵机之间的区别：DTM 只能跟踪一条计算路径，而 NTM 则拥有一个计算树，即同时跟踪多条计算路径。如果其中一条路径引向终止状态，则我们说 NTM 接受了给定的输入。

P vs NP

一个决策问题 D，如果其满足下列条件，则被认为是多项式时间可求解的：

1）存在一个算法 A，A 的输入是 D 的实例，A 总是正确地输出“是”和“否”的答案。

2）存在一个多项式函数 p，如果 D 的实例大小为 n，则 A 在不超过 $p(n)$个步骤里终结。

如果一个问题是多项式时间可求解的，则我们说这个问题属于 P 类问题！而所有满足上述条件的问题就构成了 P 类问题的集合!

一个决策问题 D，如果满足下列条件，我们就称其为**非确定性多项式时间可解**：

1）存在一个算法 A，A 的输入是 D 的潜在证人，A 总是正确辨认该证人的真假。

2）存在一个多项式函数 p，如果潜在证人对应的 D 的实例大小为 n，则 A 在不超过 $p(n)$个步骤里终结。

如果一个问题是非确定性多项式时间可求解的，则我们说这个问题属于 NP 类问题！

当然不是。它们之间有重大区别！这个区别就在第 1 条上。P 定义的第 1 条是能够给出答案，而 NP 定义的第 1 条是能够指出一个答案是否正确！众所周知，给出答案和判断答案是难度很不相同的两回事情。这就是为什么一般人更愿意做选择题，而不愿意做解答题！

P会不会等于NP？

- 归约(Reducibility)：可以用问题B的解法解决问题A，或者说，问题A可以“变成”问题B。称问题A可以归约为问题B
- 例：求解一个一元一次方程可以规约为求解一个一元二次方程。
- 如果能找到这样一个变化法则，对任意一个程序A的输入，都能按这个法则变换成程序B的输入，使两程序的输出相同，那么我们说，问题A可约化为问题B。
- 存在这样一个NP问题，所有的NP问题都可以约化成它——NPC问题，也就是NP-完全问题。

规约、化简、reduce

- 如何求解最小公倍数？
- Gbs(a,b)
- 输入：两个数a，b
- 输出：两个数的最小公倍数
- 比如：12和8的最小公倍数为24
- {c=Gys(a,b);d=a*b/c;return d;}
- a*b=c*d
- 12和8的最大公约数为4
- 把一个求解gbs的问题转化为求解gys的问题

规约、化简、reduce

生活的秘密在于……用一个烦恼代替另一个烦恼。

——Charles M. Schulz(1922—2000)，美国漫画家，史努比之父

有一个众所周知的关于数学家的笑话，我是这样描述的：X 教授是一个著名的数学家，他注意到，每当他的妻子要烧泡茶的开水时，她会把水壶从厨房的柜子里拿出来，装上水，然后把它放在炉子上。有一次，他的妻子外出了(如果大家一定想知道，她其实是到一家当地书店签名售书去了)，这个教授只能自己烧开水了。他看到水壶已经坐在灶台上了。X 教授是怎么做的呢？他先把水壶放在柜子里，然后再遵循他妻子的烧水程序。

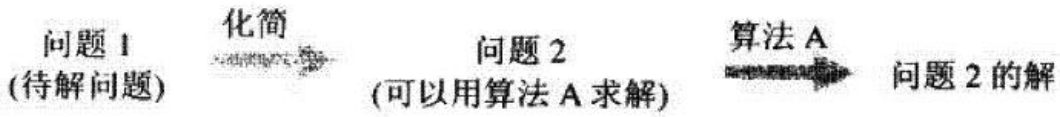


图 6.15　问题化简策略

规约、化简、reduce

定义 5 我们说一个判定问题 D_1 可以多项式地化简为一个判定问题 D_2，条件是存在一个函数 t 能够把 D_1 的实例转化为 D_2 的实例，使得

1. t 把 D_1 的所有真实例映射为 D_2 的真实例，把 D_1 的所有假实例映射为 D_2 的假实例。
2. t 可以用一个多项式算法计算。

这个定义显然意味着如果问题 D_1 可以多项式地化简为某些能够在多项式时间内求解的问题 D_2，那么问题 D_1 就可以在多项式时间内求解（为什么？）。

假如我们要解决问题 R，而我们有一个解决问题 S 的算法，并且有一个转换函数 T，能够将问题 R 转换为问题 S，即如果 R 在输入为 x 时的正确答案为“是”当且仅当 S 在输入 $T(x)$时的正确答案为“是”，则 R 称为可规约到 S（见图 14-2）。

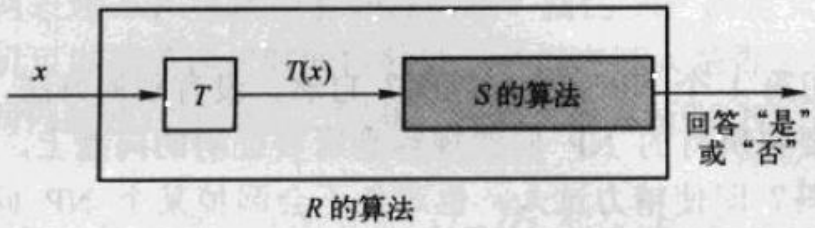


图 14-2　多项式时间规约

如果转换函数 T 的时间复杂性为多项式，则 R 被称为可多项式规约到 S。多项式规约的实际意义是，S 的难度不比 R 小，即如果 S 能够解决，则 R 就能够解决。

- 我们说一个判定问题D是NP难问题，条件是：
  - NP中的任何问题都能够在多项式时间内化简为D
- NPC=NP-hard 并 NP

定理　如果 NP 难里面的任何一个优化问题存在一个多项式时间解，则 P=NP。

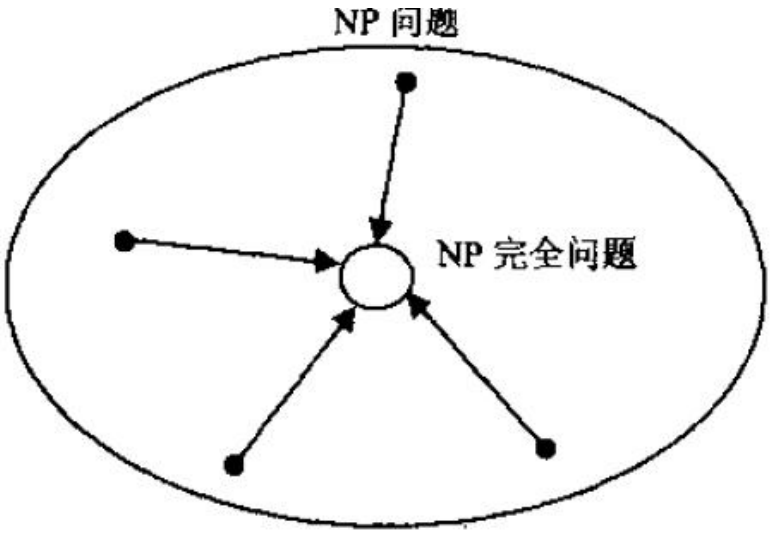


图 10.6　一个 NP 完全问题的概念。箭头表示 NP 问题到一个 NP 完全问题在多项式时间内的化简

定义 6 我们说一个判定问题 *D* 是 *NP* 完全问题，条件是：

1. 它属于 *NP* 类型。
2. *NP* 中的任何问题都能够在多项式时间内化简为 *D*。

直观地讲，NP 完全问题就是 NP 里面最困难的问题！

P会不会等于NP？

- NPC指的是NP问题中最难的一部分问题，所有的NP问题都能在多项式时间内归约到NPC上。
- 目前人们已经发现了成千上万的NPC问题，解决一个，NP=P就得证，可以得千年大奖。
- 图染色、哈密尔顿环都是 NP C问题
- NPC问题目前没有多项式的有效算法，只能用指数级甚至阶乘级复杂度的搜索。
- 正是NPC问题的存在，使人们相信P≠NP

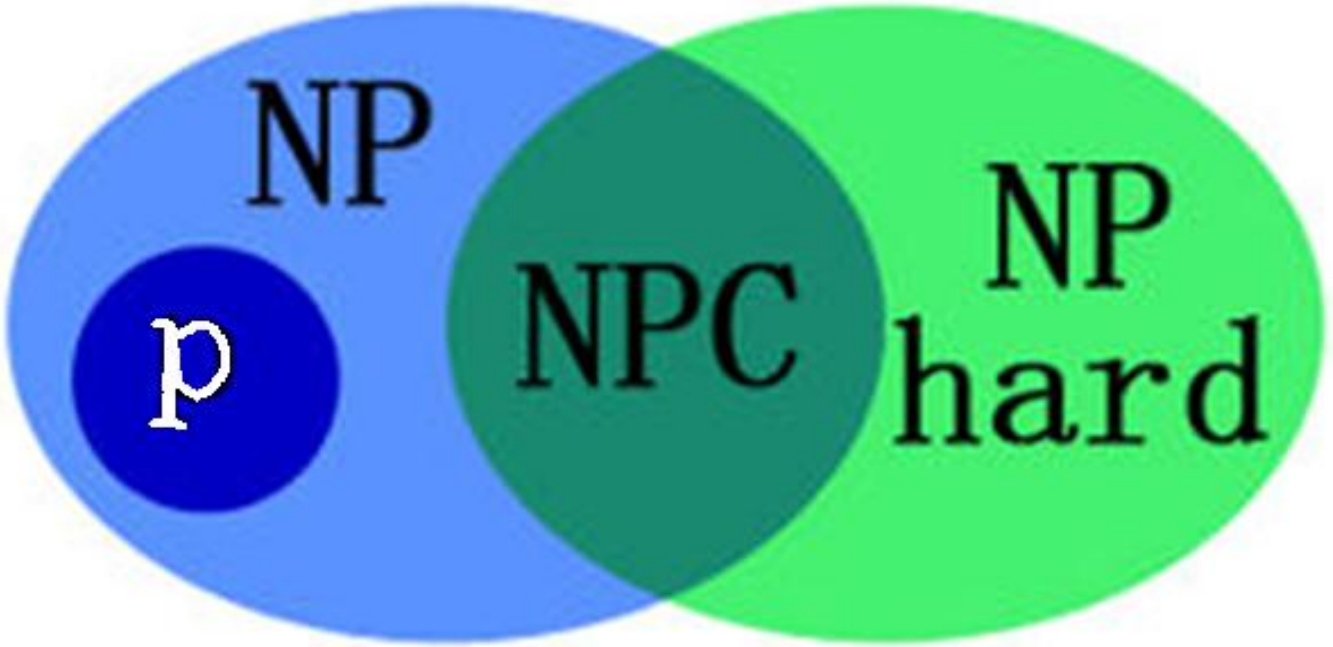
NP-hard问题

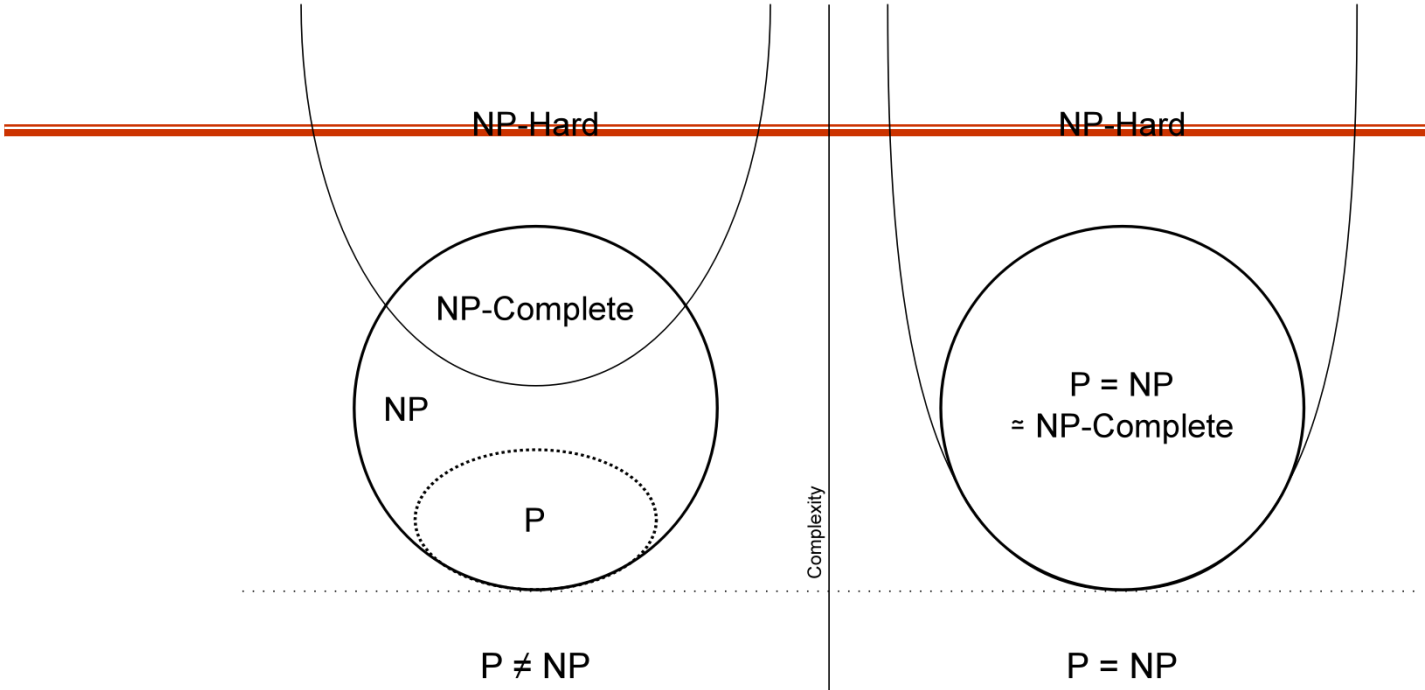
- NP-hard Problem：对于这一类问题，用一句话概括他们的特征就是“at least as hard as the hardest problems in NP Problem”， 就是NP-hard问题至少和NP问题一样难。

- 所有的NP问题都能规约到它，但它不一定是NP问题。

- 存在一些连验证解都不能多项式解决的问题，这些就是NP-hard问题

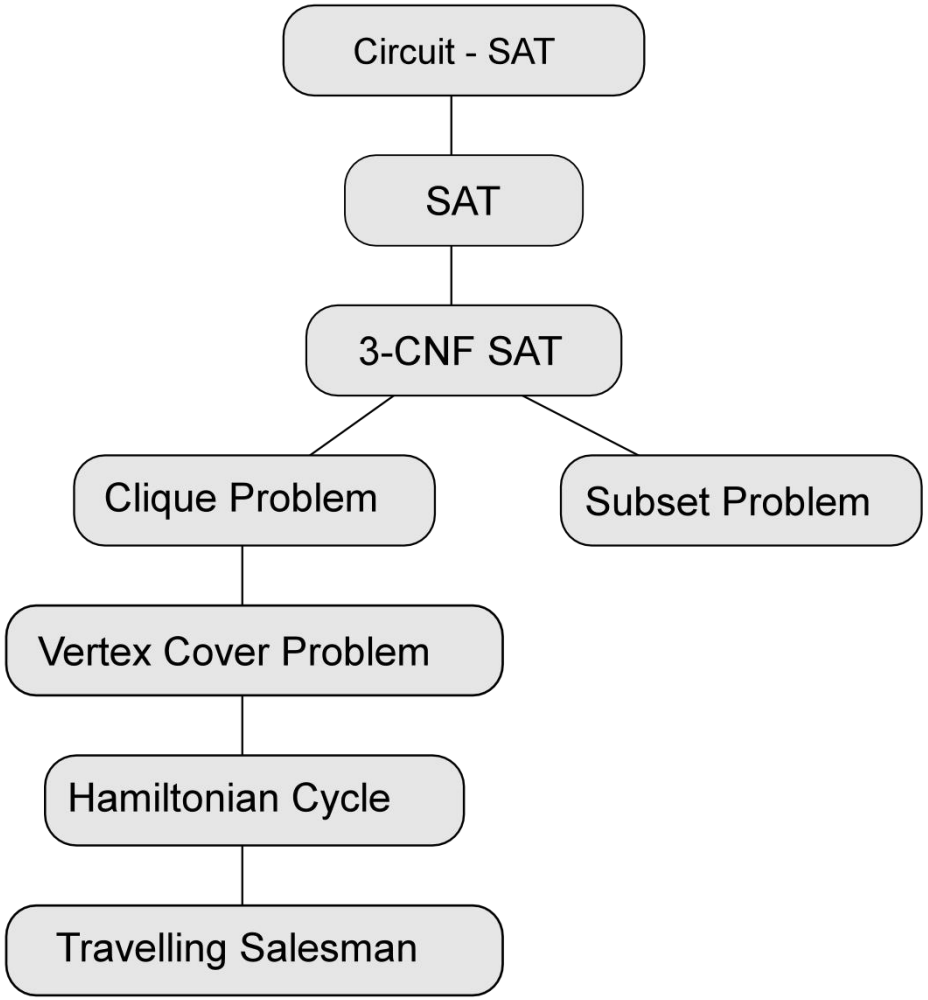
- 从直觉上说，P<=NP<=NP-Complete<=NP-Hard，问题的难度递增。





- Euler diagram for P, NP, NP-complete, and NP-hard set of problems. The left side is valid under the assumption that P≠NP, while the right side is valid under the assumption that P=NP (except that the empty language and its complement are never NP-complete, and in general, not every problem in P or NP is NP-complete).

- Some NP-complete problems, indicating the reductions typically used to prove their NP-completeness



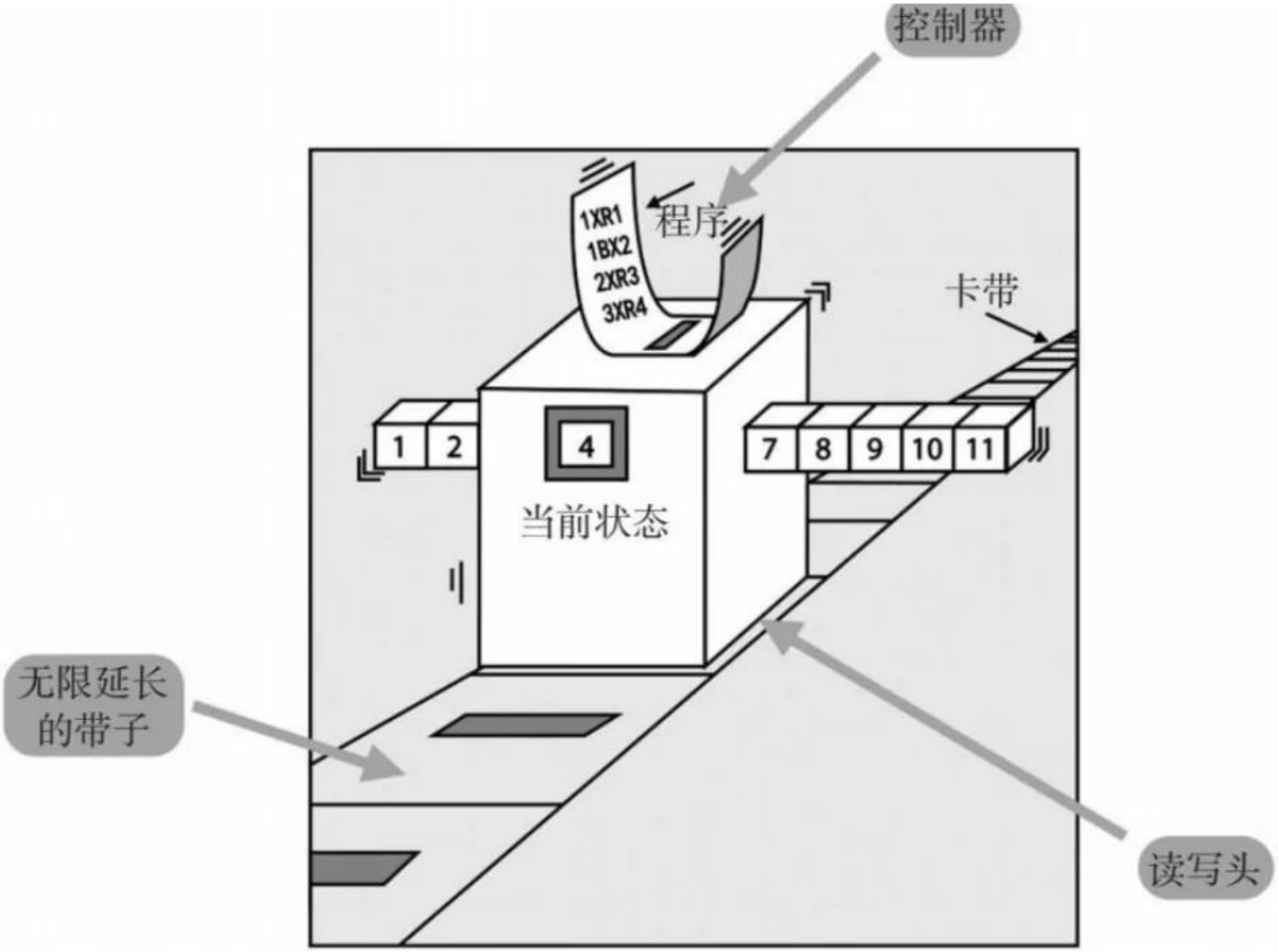
如果P=NP，世界会怎样？

- 假设人类的运气好到 P=NP 是真的，并且找到了复杂度不超过 $O(n^3)$ 的算法。如果到了这一步，我们就会有一个算法，能够很快算出某个帐号的密码。所有的加密系统都会失去效果——应该说，所有会把密码变成数字信息的系统都会失去效果，因为这个数字串很容易被“金钥匙”计算出来。
- 除此之外，我们需要担心或期许的事情还有很多：

1. 一大批耳熟能详的游戏，如扫雷、俄罗斯方块、超级玛丽等，人们将为它们编写出高效的AI，使得电脑玩游戏的水平无人能及。
2. 整数规划、旅行商问题等许多运筹学中的难题会被高效地解决，这个方向的研究将提升到前所未有的高度。
3. 蛋白质的折叠问题也是一个 NPC 问题，新的算法无疑是生物与医学界的一个福音。

Solving NP-complete problems

- Approximation: Instead of searching for an optimal solution, search for a solution that is at most a factor from an optimal one.
- Randomization: Use randomness to get a faster average running time, and allow the algorithm to fail with some small probability. Note: The Monte Carlo method is not an example of an efficient algorithm in this specific sense, although evolutionary approaches like Genetic algorithms may be.
- Restriction: By restricting the structure of the input (e.g., to planar graphs), faster algorithms are usually possible.
- Parameterization: Often there are fast algorithms if certain parameters of the input are fixed.
- Heuristic: An algorithm that works "reasonably well" in many cases, but for which there is no proof that it is both always fast and always produces a good result. Metaheuristic approaches are often used.


控制器
1XR1
1BX2
2XR3
3XR4
程序
卡带
1
2
4
7
8
9
10
11
当前状态
无限延长
的带子
读写头

停机问题简述

- 通俗地说，图灵当年想要证明希尔伯特的可判定性问题：是否存在一种通用的机械过程，能够判定任何数学命题的真假。
- 图灵就设计了一种假想的机器（图灵机）。他首先证明，图灵机就覆盖了所有的“机械过程”，如果存在一个问题，图灵机判定不了，那么就说明，不存在这种“通用的”过程，这样就证明了原问题。
- 然后，图灵就设计了一个问题，确实是图灵机判定不了的，这个问题就是：对于一个输入，让图灵机判定自己是否能够在有限的时间内停下来。
- 图灵证明，这个问题是图灵机回答不了的，所以原问题得以证否。因为这个问题设计得非常巧妙，所以在历史上留下了名字，叫“停机问题”。

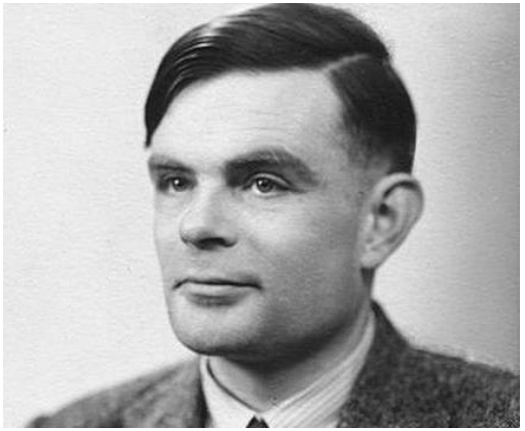
停机问题(1)

- *问题定义：* 给定一个计算机程序和一个输入，判定这个程序是继续工作还是停机

试探解：就用这个给定的输入运行程序呗。如果程序结束，我们就知道程序可以结束，但是如果程序不能在优先时间内而结束，我们也不能下结论说程序不会结束。也可能我们等得不够久？

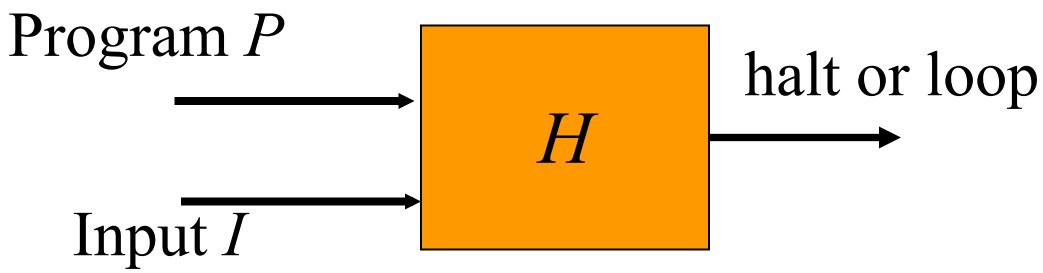
停机问题(2)

- **Alan Turing 证明了停机问题是不可解的**
- **也就是，不存在一个算法能正确地判定一个任意的程序对于一个给定的输入是否会停机**
- **Turing的证明思想是构造了一个反例：如果这样的算法存在，它就会与自己矛盾，所以这种算法不存在**



停机问题(3)

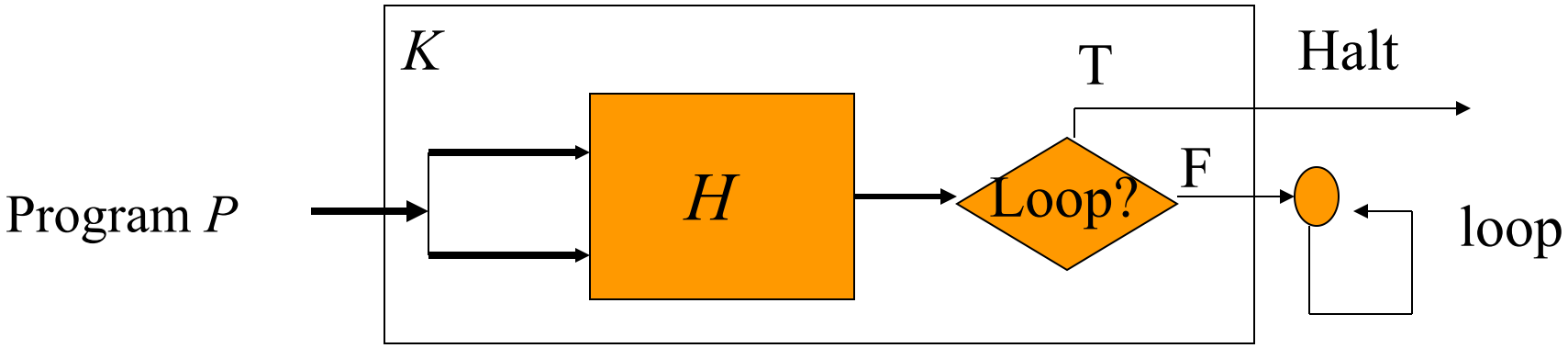
- 假设H是停机问题的解
- *H* 有两个输入：一个程序 *P* ，还有一个输入 *I* 。
- *H* 产生一个输出：停机，当H认为程序P输入I时会停止；否则输出循环



停机问题(4)

- 为了简单，我们把P认为是输入和程序，再设计另一个程序K
  - 如果 *H* 输出循环，则 *K* 停机.
  - 如果*H* 输出停机，则 *K* 永久输出循环
  - 也就是， *K* 总是和 *H*的输出相反.

```
function K( )
    if H( ) == “loop”  return “Halt”;
    else  while(true); // loop forever
```



来看两个代码（1）

```
bool God_algo(char * program, char * input)
{
    if(<program> halts on <input>)
        return true;
    else
        return false;
}
```

这里假设if的判断语句是人类天才思考的结晶，它能像上帝一样洞察所有程序的宿命，

来看两个代码（2）

```
bool Satan_algo(char * program)
{
    if(God_algo(program, program))
    {
        while(true); // loop forever!
        return false; // can never get
    here!
    }
    else
        return true;
}
```

- 和这个程序的名字一样，它太邪恶了。
- 当这个算法运用到自身时：Satan_algo(Satan_algo); 它肯定和所有的程序一样，要么停止，要么永不结束。

来看两个代码（3）

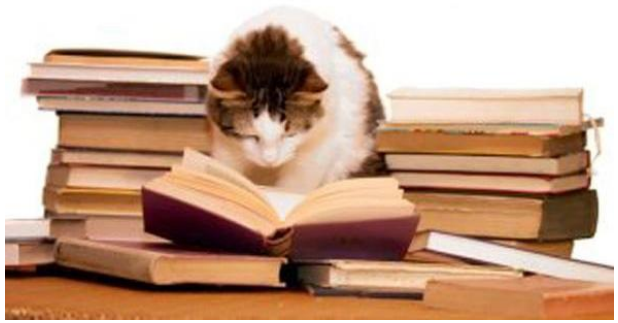
- 那我们先假设这个程序能停机，那上图代码块中的if条件判断肯定为真（因为God_algo(Satan_algo, Satan_algo)这个函数返回true），从而程序进入那个包含while(true);语句的分支，那我们就可以得出这个程序不能停机。
- 我们再假设这个程序不能停机，类似的，我们可以得出这个程序能停机。
- 总之，我们有：
- Satan_algo(Satan_algo)能停机 => 它不能停机
- Satan_algo(Satan_algo)不能停机 => 它能停机
- 那么，我们得出了一个悖论。从而我们可以推翻我们最初的假设：**不存在一个程序（或算法），它能够计算任何程序在给定输入上是否会结束（停机）。**

停机问题是NP-hard

- NP-Hard和NP-Complete有什么不同？简单的回答是根据定义，如果所有NP问题都可以多项式归约到问题A，那么问题A就是 NP-Hard；如果问题A既是NP-Hard又是NP，那么它就是NP-Complete。从定义我们很容易看出，NP-Hard问题类包含了NP- Complete类。但进一步的我们会问，是否有属于NP-Hard但不属于NP-Complete的问题呢？答案是肯定的。例如停机问题，也即给出一个程序和输入，判定它的运行是否会终止。停机问题是不可判的，那它当然也不是NP问题。但对于SAT这样的NP-Complete问题，却可以多项式归约到停机问题。因为我们可以构造程序A，该程序对输入的公式穷举其变量的所有赋值，如果存在赋值使其为真，则停机，否则进入无限循环。这样，判断公式是否可满足便转化为判断以公式为输入的程序A是否停机。所以，停机问题是NP-Hard而不是NP-Complete。

NPC问题：一个故事 (1)

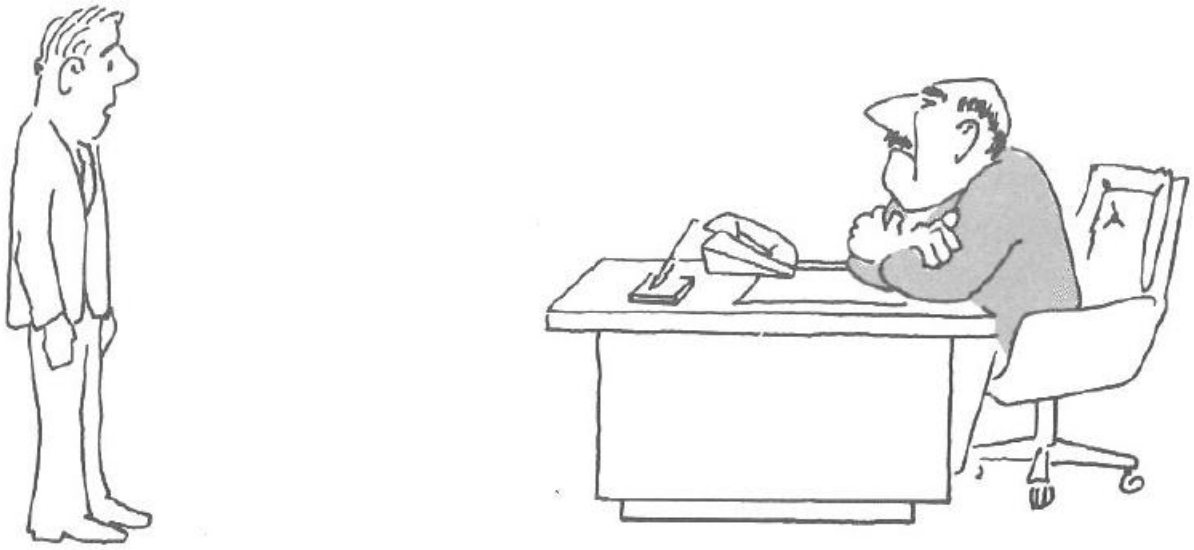
- 你老板给你了一个NPC的问题，让你给出一个有效的解，这对你的公式很重要
- 最初你和你老板都不知道这是NPC问题
- 你花了大量时间，不眠不休，还是没找到答案



- 你怎么办?

NPC问题：一个故事(2)

- 选择1：鼓起勇气告诉老板这事搞不定



"I can't find an efficient algorithm, I guess I'm just too dumb."

- 你完了，老板对你失去了信任...

NPC问题：一个故事(3)

- 选择2：你向老板证明这个问题是不可解的

"I can't find an efficient algorithm, because no such algorithm is possible!"

- 杯具了！证明这个问题不可解就如同解这个问题一样难！

NPC问题：一个故事(4)

- 选择3：你向老板证明这个问题是一个NPC问题，那么多牛人都解不出NPC问题，我解不出来正常啊！

"I can't find an efficient algorithm, but neither can all these famous people."

- 证明一个问题是NPC问题，相对比较容易。

怎么证明一个问题是NPC问题？

- 如果你有足够的兴趣和时间，请参看课本第34章，加油啊！

14.5 如何证明一个问题 S 是 NP 完全

有了上面对规约的讨论，我们可以将 NP 完全的证明方法罗列如下：

1）证明 S 在 NP 里面。

2）选择一个已知的 NP 完全问题 R。

3）证明 R 可以多项式规约到 S。

14.6 第 1 个 NP 完全问题的证明

用什么办法来证明第 1 个 NP 完全问题呢？自然，没有别的办法，只能根据 NP 完全的定义来证明，也就是要将所有的 NP 问题规约到所要证明的问题上。但是 NP 问题数量繁多，这样证明得过来吗？即使精力过人，也难免不会漏掉某个 NP 问题，从而导致证明失败。况且，NP 类问题的数量到底有多少，谁也说不清！

希望似乎在丧失，但不要气馁！显然，我们不可能一个个 NP 问题来规约，但谁说过我们必须这么做呢？还记得 NP 问题的定义吗？该定义使用了图灵机！也就是说，所有 NP 问题都可以用图灵机来表示，因此，我们可以将所有 NP 问题一般化，抽象成一个问题！这样，我只需要证明一次即全部搞定！这就是抽象的能力！

而这正是斯蒂芬·库克用的方法。库克在 1971 年证明了布尔可满足性问题（SAT 问题）是 NP 完全问题（该证明出现在其发表于 1971 年的“The Complexity of Theorem Proving Procedures”文章里），从而开启了 NP 完全理论的风帆，并因此于 1982 年获得图灵奖。Leonid Levin 在其 1972 年发表的论文“Universal Search Problems”里面也独立地证明了该问题为 NP 完全。下面我们就来重温库克的极为精彩的证明！

第一个NP完全问题：Cook1971

一个逻辑公式被认为是 DNF 的，当且仅当它是一个或多个文字的一个或多个合取的析取。

$$(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F)$$

4) The set {DNF tautologies} is the set of strings representing tautologies in disjunctive normal form.

永真式：某个逻辑公式总是为真

矛盾式：某个逻辑公式总是为假

可满足式：若至少存在一个赋值使某个逻辑公式为真，则该逻辑公式为可满足式

$$A \vee \neg A, A \wedge \neg A, A \vee B$$

{DNF永真式}是NP完全问题

Theorem 1: If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

Proof of the theorem: Suppose a nondeterministic Turing machine M accepts a set S of strings within time Q(n), where Q(n) is a polynomial. Given an input w for M, we will construct a proposition formula A(w) in conjunctive normal form such that A(w) is satisfiable iff M accepts w. Thus $\neg A(w)$ is easily put in disjunctive normal form (using De Morgan's laws), and $\neg A(w)$ is a tautology if and only if $w \notin S$. Since the whole construction can be carried out in time bounded by a polynomial in $|w|$ (the length of w), the theorem will be proved.

SATIsfIBILITY is NP-complete

- 可满足性问题是NP完全问题
- 可满足性问题： given the expression, is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true?
- F1={ (x1 ∨x2 ∨x3),(¬x1∨x2),(¬x2∨x3), (¬x3∨x1),(¬x1 ∨¬x2∨ ¬x3) }

3SAT

文字:一个布尔变量或它的非,如x或¬x.

子句:由∨连接的若干文字,如$x_1 \vee (\neg x_2) \vee x_3 \vee x_4$.

合取范式(cnf公式):由∧连接的若干子句,如

$((\neg x_1) \vee x_2 \vee (\neg x_3)) \wedge (x_2 \vee (\neg x_3) \vee x_4 \vee x_5) \wedge ((\neg x_4) \vee x_5)$.

3cnf公式:所有子句为都由三个文字组成,如

$((\neg x_1) \vee x_2 \vee (\neg x_3)) \wedge ((\neg x_3) \vee x_4 \vee x_5) \wedge ((\neg x_4) \vee x_5 \vee x_5)$

3SAT={<ϕ>|ϕ是可满足的3cnf公式}

3SAT是NPC, CLIQUE是NPC.

Deterministic Turing machine

- In a deterministic Turing machine (DTM), the set of rules prescribes at most one action to be performed for any given situation.

- A deterministic Turing machine has a transition function that, for a given state and symbol under the tape head, specifies three things:

- the symbol to be written to the tape (it may be the same as the symbol currently in that position, or not even write at all, resulting in no practical change),
- the direction (left, right or neither) in which the head should move, and
- the subsequent state of the finite control.
- For example, an X on the tape in state 3 might make the DTM write a Y on the tape, move the head one position to the right, and switch to state 5.

- nondeterministic Turing machine (NTM) the set of rules may prescribe more than one action to be performed for any given situation. For example, an X on the tape in state 3 might allow the NTM to:

- Write a Y, move right, and switch to state 5
- or

- Write an X, move left, and stay in state 3.

Following Hopcroft & Ullman (1979, p. 148), a (one-tape) Turing machine can be formally defined as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where

- Γ is a finite, non-empty set of *tape alphabet symbols*;
- $b \in \Gamma$ is the *blank symbol* (the only symbol allowed to occur on the tape infinitely often at any step during the computation);
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of *input symbols*, that is, the set of symbols allowed to appear in the initial tape contents;
- Q is a finite, non-empty set of *states*;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *final states* or *accepting states*. The initial tape contents is said to be *accepted* by M if it eventually halts in a state from F.
- $\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$ is a partial function called the *transition function*, where L is left shift, R is right shift. If δ is not defined on the current state and the current tape symbol, then the machine halts;[19] intuitively, the transition function specifies the next state transited from the current state, which symbol to overwrite the current symbol pointed by the head, and the next head movement.

A nondeterministic Turing machine can be formally defined as a six-tuple $M = (Q, \Sigma, \iota, \sqcup, A, \delta)$, where

- Q is a finite set of states
- Σ is a finite set of symbols (the tape alphabet)
- $\iota \in Q$ is the initial state
- $\sqcup \in \Sigma$ is the blank symbol
- $A \subseteq Q$ is the set of accepting (final) states
- $\delta \subseteq (Q \backslash A \times \Sigma) \times (Q \times \Sigma \times \{L, S, R\})$ is a relation on states and symbols called the *transition relation*. L is the movement to the left, S is no movement, and R is the movement to the right.

Definitions

- A decision problem is in NP if it can be solved by a non-deterministic algorithm in polynomial time.
- An instance of the Boolean satisfiability problem is a Boolean expression that combines Boolean variables using Boolean operators.
- An expression is satisfiable if there is some assignment of truth values to the variables that makes the entire expression true.

Idea

- Given any decision problem in NP, construct a non-deterministic machine that solves it in polynomial time. Then for each input to that machine, build a Boolean expression which computes whether that specific input is passed to the machine, the machine runs correctly, and the machine halts and answers "yes". Then the expression can be satisfied if and only if there is a way for the machine to run correctly and answer "yes", so the satisfiability of the constructed expression is equivalent to asking whether or not the machine will answer "yes".

Proof

- his proof is based on the one given by Garey and Johnson.[6]

- There are two parts to proving that the Boolean satisfiability problem (SAT) is NP-complete. One is to show that SAT is an NP problem. The other is to show that every NP problem can be reduced to an instance of a SAT problem by a polynomial-time many-one reduction.

- SAT is in NP because any assignment of Boolean values to Boolean variables that is claimed to satisfy the given expression can be verified in polynomial time by a deterministic Turing machine. (The statements verifiable in polynomial time by a deterministic Turing machine and solvable in polynomial time by a non-deterministic Turing machine are totally equivalent

Now suppose that a given problem in NP can be solved by the nondeterministic Turing machine $M = (Q, \Sigma, s, F, \delta)$, where Q is the set of states, Σ is the alphabet of tape symbols, $s \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta \subseteq ((Q \setminus F) \times \Sigma) \times (Q \times \Sigma \times \{-1, +1\})$ is the transition relation. Suppose further that M accepts or rejects an instance of the problem in time $p(n)$ where n is the size of the instance and p is a polynomial function.

For each input, I, we specify a Boolean expression which is satisfiable if and only if the machine M accepts I.

The Boolean expression uses the variables set out in the following table. Here, $q \in Q$, $-p(n) \leq i \leq p(n)$, $j \in \Sigma$, and $0 \leq k \leq p(n)$.

Variables	Intended interpretation	How many?
$T_{i,j,k}$	True if tape cell i contains symbol j at step k of the computation.	$O(p(n)^2)$
$H_{i,k}$	True if the M's read/write head is at tape cell i at step k of the computation.	$O(p(n)^2)$
$Q_{q,k}$	True if M is in state q at step k of the computation.	$O(p(n))$

Define the Boolean expression B to be the conjunction of the sub-expressions in the following table, for all $-p(n) \le i \le p(n)$ and $0 \le k \le p(n)$:

Expression	Conditions	Interpretation	How many?
$T_{i,j,0}$	Tape cell i initially contains symbol j	Initial contents of the tape. For $i > n$-1 and $i < 0$, outside of the actual input I, the initial symbol is the special default/blank symbol.	O($p(n)$)
$Q_{s,0}$		Initial state of M.	1
$H_{0,0}$		Initial position of read/write head.	1
$\neg T_{i,j,k} \vee \neg T_{i,j',k}$	$j \neq j'$	At most one symbol per tape cell.	O($p(n)^2$)
$\bigvee_{j \in \Sigma} T_{i,j,k}$		At least one symbol per tape cell.	O($p(n)^2$)
$T_{i,j,k} \wedge T_{i,j',k+1} \rightarrow H_{i,k}$	$j \neq j'$	Tape remains unchanged unless written.	O($p(n)^2$)
$\neg Q_{q,k} \vee \neg Q_{q',k}$	$q \neq q'$	Only one state at a time.	O($p(n)$)
$\neg H_{i,k} \vee \neg H_{i',k}$	$i \neq i'$	Only one head position at a time.	O($p(n)^3$)
$(H_{i,k} \wedge Q_{q,k} \wedge T_{i,\sigma,k}) \rightarrow \bigvee_{((q,\sigma),(q',\sigma',d)) \in \delta} (H_{i+d,\ k+1} \wedge Q_{q',\ k+1} \wedge T_{i,\ \sigma',\ k+1})$	$k<p(n)$	Possible transitions at computation step k when head is at position i.	O($p(n)^2$)
$\bigvee_{0 \le k \le p(n)} \bigvee_{f \in F} Q_{f,k}$		Must finish in an accepting state, not later than in step $p(n)$.	1

If there is an accepting computation for M on input I, then B is satisfiable by assigning $T_{i,j,k}$, $H_{i,k}$ and $Q_{i,k}$ their intended interpretations. On the other hand, if B is satisfiable, then there is an accepting computation for M on input I that follows the steps indicated by the assignments to the variables.

There are $O(p(n)^2)$ Boolean variables, each encodeable in space $O(\log p(n))$. The number of clauses is $O(p(n)^3)$ so the size of B is $O(\log(p(n))p(n)^3)$. Thus the transformation is certainly a polynomial-time many-one reduction, as required.

Complexity [edit]

While the above method encodes a non-deterministic Turing machine in complexity $O(\log(p(n))p(n)^3)$, the literature describes more sophisticated approaches in complexity $O(p(n)\log(p(n)))$.[7][8][9][10][11] The quasilinear result first appeared seven years after Cook's original publication.

Generalized versions of boolean satisfiability have encodings with stronger bounds still: quantified boolean formulas (QBF's) encode non-deterministic Turing machines in polynomial complexity to the machine's space bound (as opposed to time bound), and dependency quantified boolean formulas (DQBF's) encode non-deterministic Turing machines in an ideal logarithmic complexity to the machine's space bound.[12][13]

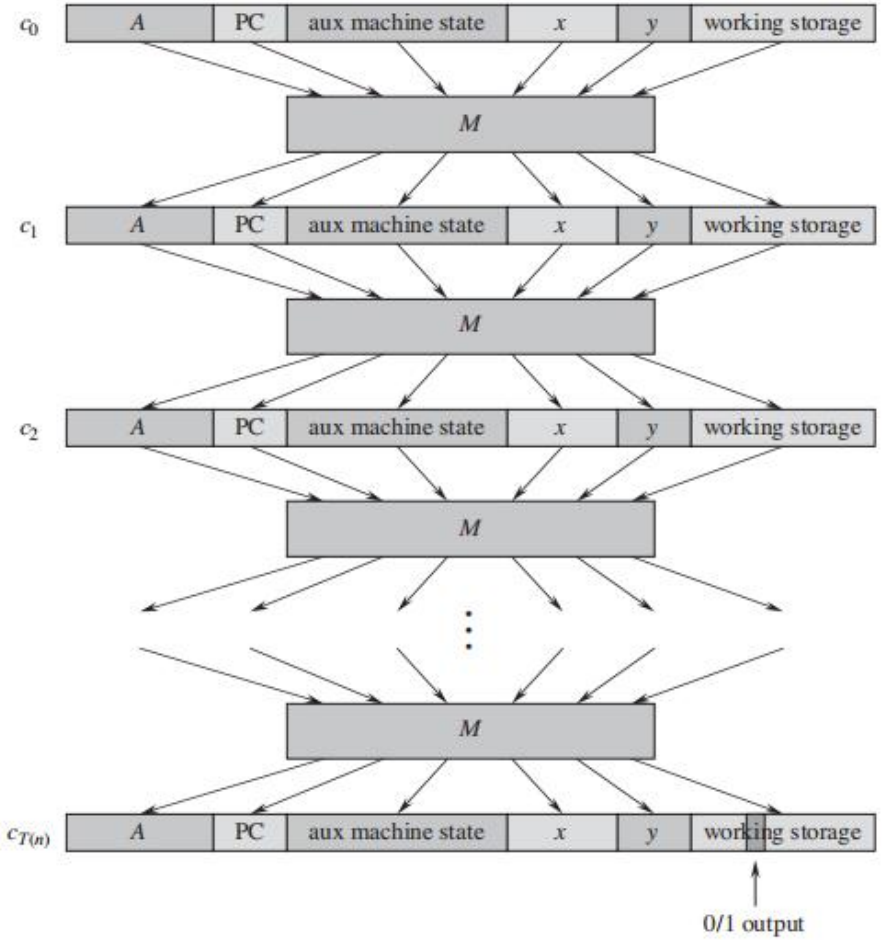
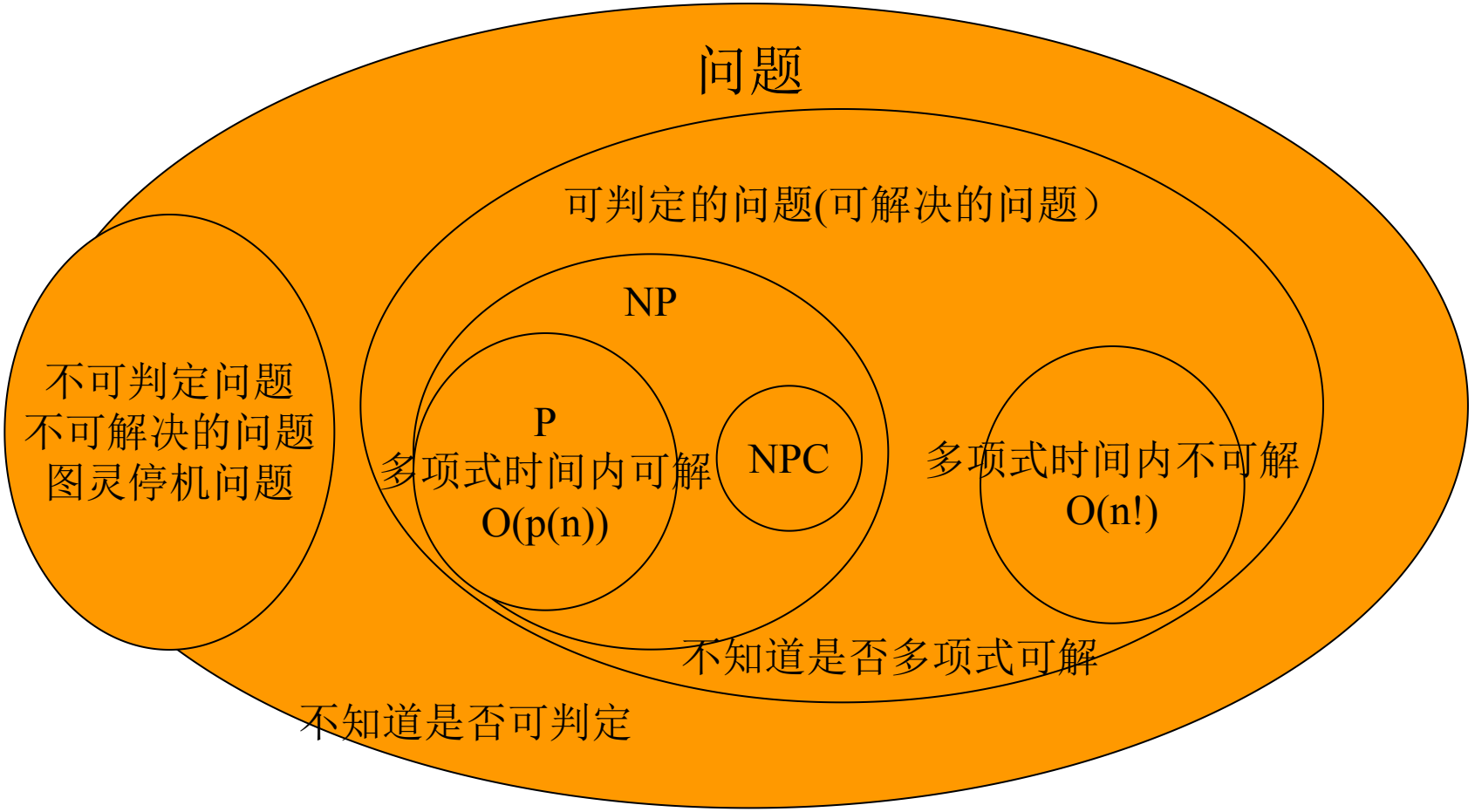


Figure 34.9 The sequence of configurations produced by an algorithm A running on an input x and certificate y. Each configuration represents the state of the computer for one step of the computation and, besides A, x, and y, includes the program counter (PC), auxiliary machine state, and working storage. Except for the certificate y, the initial configuration c_0 is constant. A boolean combinational circuit M maps each configuration to the next configuration. The output is a distinguished bit in the working storage.


问题
可判定的问题(可解决的问题）
NP
不可判定问题
不可解决的问题
图灵停机问题
P
多项式时间内可解
O(p(n))
NPC
多项式时间内不可解
O(n!)
不知道是否多项式可解
不知道是否可判定