

Lecture 11 NP 完全问题

绳伟光

上海交通大学微纳电子学系

2021-11-18



- 1 可计算性
 - 通用计算模型 —— 图灵机
 - 可判定性
- 2 计算复杂性
 - 复杂性类
 - NP 完全性与可归约性
 - NP 完全性的证明

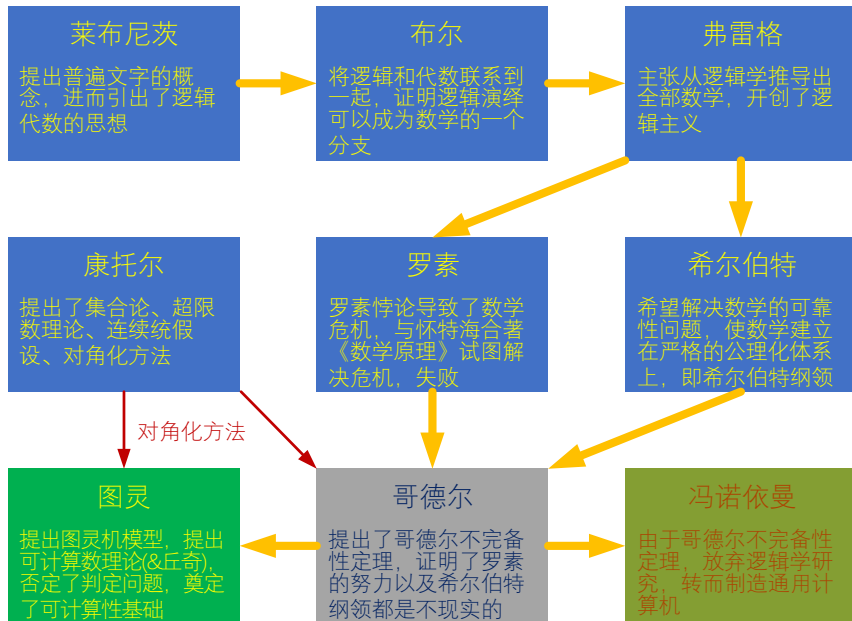


- 1 可计算性
 - 通用计算模型 —— 图灵机
 - 可判定性

- 2 计算复杂性
 - 复杂性类
 - NP 完全性与可归约性
 - NP 完全性的证明



通用计算模型的发展简史



莱布尼茨/布尔/弗雷格/希尔伯特

- 莱布尼茨的理想：寻求一个符号系统，每个元素是一个概念，发展一种语言，仅凭符号演算，根据它们之间存在的关系，就可以确定用这种语言写成的句子哪些为真：“严肃的具有善良意志的人们围坐在桌子旁解决某个棘手问题，用普遍文字写出这个问题后，人们就可以说：“让我们算一下。”于是人们拿出笔得到一个解答，其对错必然可以为所有人接受。”
- 布尔：将逻辑变为代数，使古典逻辑原地踏步 2000 年后走向了数理逻辑
- 弗雷格：布尔把普通代数作为出发点，用代数符号表示逻辑关系；弗雷格反其道而用之，主张从逻辑学推导出全部数学，开创了逻辑主义
- 1900 年国际数学家大会上，法国数学家庞加莱兴高采烈地宣称：“借助集合论的概念，我们可以建造起整个数学大厦……今天，我们可以说，绝对的严格性已经达到了。”
- 数学领袖希尔伯特谨慎乐观，他在会上提出 23 个问题，2 号问题就是算术公理系统的无矛盾性（即一致性）：“在这些无数个问题之上，我倾向于确定下面这个问题才是最重要的：这些公理经过有限步骤推演后不会导致相互矛盾的结论……也就是说，我们需要一个关于算术公理一致性的证明。”

康托尔的集合概念

- **可数集**：又称可列集、可数无穷集合，是可以与自然数（正整数）集合 $1, 2, 3, \dots$ 建立一一对应的无穷集合，所以其基数（势）与自然数相同。就是说，存在双射函数，可以将一个集合的所有元素一一对应地映射到自然数集，故而可以将集合 S 的元素排队，从第一个数起，必有唯一后继者可以数，每个都可以数到而不会遗漏。当然，永远也数不完
- 所有偶数的集合、所有整数的集合、所有有理数的集合都是可数集，都与自然数集等势（阿列夫 0）
- 实数集的势（阿列夫 1）大于自然数集的势
- **连续统假设**：不存在一个基数绝对大于可列集而绝对小于实数集的集合



康托尔的对角化方法

试证：实数集不可列。

证 假设实数集可列，我们将实数的小数部分展开，将其与自然数集的对应关系列举如下：

1	$z_{10}.$	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	...
2	$z_{20}.$	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}	...
3	$z_{30}.$	d_{31}	d_{32}	d_{33}	d_{34}	d_{35}	...
4	$z_{40}.$	d_{41}	d_{42}	d_{43}	d_{44}	d_{45}	...
5	$z_{50}.$	d_{51}	d_{52}	d_{53}	d_{54}	d_{55}	...
6	$z_{60}.$	d_{61}	d_{62}	d_{63}	d_{64}	d_{65}	...
⋮							

现在，构造一个新的实数，它的第 i 位小数与上述的 d_{ii} 不同，则该实数与上述列出的任何一个实数至少都有一位不同，它不等于我们列出的任何实数，这与我们已列出所有实数的假设是矛盾的，因而实数集是不可列的，不能与自然数集一一对应。



罗素

- 弗雷格的算术使用了集合的集合，罗素从中发现了悖论，也称为理发师悖论，即如果一个集合是它自身的一个元素（自指）时所陷入的困境
- 弗雷格：“正当工作就要完成之时，发现那大厦的基础已经动摇。对于一个科学工作者来说，没有什么比这更为不幸的了。伯特兰·罗素的一封信使我置身于这样的境地。”
- 罗素与怀特海合著《数学原理》，试图为数学大厦重新奠基，三卷相继于 1910 年、1912 年和 1913 年出版。这部 2000 多页的巨著引言只陈述了一个目标，那就是“完整地列出数学推理的所有方法和步骤”，这就是逻辑主义的纲领，但被哥德尔证明不可行
- 罗素：“我以人们寻找宗教信仰的热忱寻找确定性。我以为在数学中最可能找到它。然而，我找到越来越多的不可靠。多年劳累的结论是，我和任何人都不能使数学成为无可怀疑的知识。”



罗素悖论

- **理发师悖论**：小城里的理发师放出豪言：他只为，而且一定要为，城里所有不为自己刮胡子的人刮胡子。那么，理发师该为自己刮胡子吗？
- **罗素悖论**：设命题函数 $P(x)$ 表示 $x \notin A$ ，设由性质 P 确定了一个类 A ，即 $A = \{x | x \notin A\}$ 。那么 $A \in A$ 是否成立？若 $A \in A$ ，则 A 是 A 的元素，那么 A 具有性质 P ，由命题函数 P 知 $A \notin A$ ；若 $A \notin A$ ，也就是说 A 具有性质 P ，而 A 是由所有具有性质 P 的类组成的，所以 $A \in A$
- 哥德尔不完备性定理同样借鉴了这种自指的方法



希尔伯特

- 为了给数学奠定牢固的基础，希尔伯特提出了希尔伯特纲领，他提出：为了消除对数学可靠性的怀疑，避免出现悖论，就要设法绝对地证明数学的一致性，使数学奠定在严格的公理化基础上
- 希尔伯特希望彻底抛弃公理体系中的含义，构造一个纯粹形式化的公理体系，这个体系内的各种表达式仅仅具有符号意义。如果能证明这种公理体系的一致性，那么把任何含义赋予这个公理体系时，都必然是无矛盾的、一致的
- 提出了元数学或证明论的概念，希尔伯特希望，一致性证明将在元数学内部完成，数学和逻辑则将以一种纯形式的符号语言被发展出来
- 1930 年，希尔伯特发表退休演讲，再次重申了完备性证明和判定问题证明的梦想，并喊出了“我们必须知道，我们必将知道”的口号。但就在前一天，同样在柯尼斯堡，24 岁的哥德尔对完备性问题给出了否定回答。6 年后，24 岁的图灵对判定问题给出了否定回答

希尔伯特第十问题：丢番图方程的可解性

- 希尔伯特在 1900 年的世界数学家大会上提出了 23 个未解决的数学问题
- 希尔伯特第十问题：对于任意多个未知数的整系数不定方程 (丢番图方程)，要求给出一个可行的方法 (verfahren)，使得借助于它，通过有限次运算，可以判定该方程有无整数解
- 上述的 verfahren 为德文，对应的英文为 algorithm
- 丢番图方程在编译器的循环自动并行化方面有重要应用，但 1970 年马季亚谢维奇证明希尔伯特第十问题不可判定



希尔伯特第二问题：算术公理之相容性

- 希尔伯特第二问题：关于一个公理系统的相容性问题，即如何采用严谨的形式化的方法，基于最基础的几条公理，便可证明系统内所有命题的相容性
- 希尔伯特本想以其第二问题为基础，为现代数学提供完全形式化的严谨的基础 (即希尔伯特纲领)
- 歌德尔不完备性定理给出了否定的答案



歌德尔的不完备性定理

- **歌德尔不完备性定理：**任何自治 (相容) 的形式系统，只要蕴涵皮亚诺算术公理，则：1) 就可以在其中构造在体系中既不能证明也不能否证的命题 (即体系是不完备的)；2) 该系统不能用于证明它本身的自治性
- 不自洽的形式系统中不存在不可判定的命题。也就是说，只要容忍自相矛盾，就可以证明一切；而一个不包含算术的自治的形式系统，是有可能不存在不可判定的命题的
- **歌德尔不完备性定理的启示：**可用纯粹逻辑的方法，证明数学本身的力量是有界限的。在数学的领地上，有些东西我们不知道，也不可能知道



皮亚诺算术公理

皮亚诺的这五条公理用非形式化的方法叙述如下：

- 1) 0 是自然数
- 2) 每一个确定的自然数 a ，都有一个确定的后继数 a' ， a' 也是自然数
- 3) 对于每个自然数 b 、 c ， $b = c$ 当且仅当 b 的后继数等于 c 的后继数
- 4) 0 不是任何自然数的后继数
- 5) 任意关于自然数的命题，如果证明：它对自然数 0 是真的，且假定它对自然数 a 为真时，可以证明对 a' 也真。那么，命题对所有自然数都真 (保证了数学归纳法的正确性)



歌德尔不完备定理的讨论

- 歌德尔不完备性定理完全粉碎了希尔伯特纲领
- 不完备性定理的证明借鉴了康托尔的对角线方法
- 不完备性定理的证明过程：
 - 1) 建立一阶算术公理系统 N
 - 2) 构造自指命题 U
 - 3) 证明自指命题 U 在 N 中不可判定
- 歌德尔不完备定理的证明技巧被用于了后来图灵关于不可判定性的证明中
- 另一个副作用是使得冯·诺依曼放弃了对逻辑学的研究，而将精力转向了通用数字计算机的研究



图灵简介

- 阿兰·图灵 (1912 年 6 月 23 日 – 1954 年 6 月 7 日), 英国数学家、逻辑学家, 计算机科学之父
- 1936 年发表划时代的论文《论可计算数及其在判定问题上的应用 (On Computable Numbers, with an Application to the Entscheidungsproblem)》, 提出了图灵机这一通用计算模型, 建立了可计算性理论
- 二战中领导破解德国的 Enigma 密码
- 提出图灵测试, 作为判断机器是否有智能的标准
- 在生物数学方面做出重要贡献
- 图灵其人



字母表、串、语言与问题

- **字母表**：字母表是符号的有穷非空集合，一般用符号 Σ 表示字母表，常见字母表有：1) $\Sigma = \{0, 1\}$ ，二进制字母表；2) $\Sigma = \{a, b, \dots, z\}$ ，小写字母表；3) 所有 ASCII 字符集合
- **串**：也称为单词，是从某个字母表中选择的符号的有穷序列
- **语言**： Σ 是某个具体的字母表，从 Σ^* 中选出的串的一个集合称为语言。如果 Σ 是字母表，且 $L \subseteq \Sigma^*$ ，则 L 是 Σ 上的语言
- **问题**：自动机理论中，一个“问题”就是判定一个给定的串是否属于某个具体语言。如果 Σ 是字母表， L 是 Σ 上的语言，则问题 L 就是：给定 Σ^* 中的一个串 w ，判断 w 是否属于 L

上面 Σ^* 中的符号 $*$ 代表闭包运算， Σ^* 表示由字母表 Σ 中字母所组成的任意长度的串

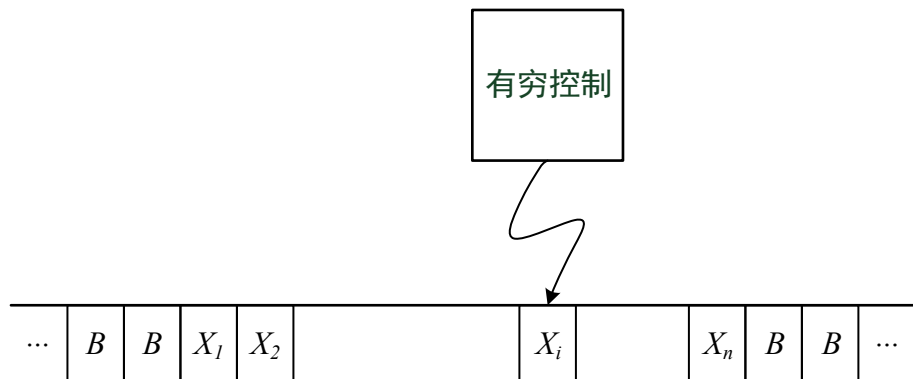
将千差万别的问题统一为字符串的处理简化了问题的讨论！

判定问题与优化问题

- 虽然各种问题千差万别，但基本可分为判定问题与优化问题
- 判定问题 (decision problem): 要求判断一个特定的表述是否为真，答案只有 Yes 和 No 两种可能
- 优化问题 (optimization problem): 要求从一系列可能的解中选择一个最好的或者得分最高的解
- 计算理论研究中，通常需要将优化问题变换为判定问题
 - 旅行商问题: 给定一个带权重的完全图和实数 c ，则转换为判定问题为：是否存在一个权重不超过 c 的哈密顿回路？
 - 背包问题的判定形式: 给定 n 个物品，物品 i 的价值 v_i ，体积 w_i ，背包容量 W 和实数 c ，是否存在一个物品组合，其价值大于等于 c ，总体积小于等于 W ？

算法研究中最常碰到优化问题；但计算理论研究中，判定问题更容易处理，使得所有问题均简化为判断是否接受某个串/语言！

图灵机模型



图灵机 (TM) 可用一个七元组来表示：

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

图灵机七元组模型的解释

- 图灵机 (TM) 可用一个七元组来表示:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

- Q : 有穷控制器的状态的有穷集合
- Σ : 输入符号的有穷集合
- Γ : 带符号的完整集合, Σ 总是 Γ 的子集
- δ : 转移函数 —— $\delta(q, X) \rightarrow (p, Y, D)$, 表示当前状态是 q 且读写头当前位置内容为 X 时, 图灵机将当前位置内容 X 改写为 Y , 将状态跳转到 p , 同时按照方向 D 的规定左移 (L) 或右移 (R) 1 格
- q_0 : 初始状态, 属于 Q , 开始时有穷控制器就处于这个状态
- B : 空格符号, 此符号属于 Γ 但不属于 Σ , 即不是输入符号, 开始时, 除输入符号外的纸带上所有位置都是 B
- F : 终结状态或接受状态的集合, 是 Q 的子集



图灵机的移动

- 对给定图灵机 M , 其当前格局为 $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$, 表示当前读写头位置在 X_i , 当前有穷控制器的状态是 q
- 设当前转移函数为 $\delta(q, X_i) = (p, Y, L)$, 则图灵机的移动可如下表示: $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \vdash_M X_1X_2\cdots X_{i-2}pX_{i-1}YX_{i+1}\cdots X_n$, 但包含两种意外情况:
 - 如果 $i = 1$, M 移动到 X_1 左边的空格, 因此 $qX_1X_2\cdots X_n \vdash_M pBYX_2\cdots X_n$
 - 如果 $i = n$ 且 $Y = B$, 则在 X_n 上写下的符号 B 将被吸收, 不出现在下一个格局中, 从而 $X_1X_2\cdots X_{n-1}qX_n \vdash_M X_1X_2\cdots X_{n-2}pX_{n-1}$
- 设当前转移函数为 $\delta(q, X_i) = (p, Y, R)$, 则有: $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \vdash_M X_1X_2\cdots X_{i-1}YpX_{i+1}\cdots X_n$, 同样包含两个意外情况:
 - 如果 $i = n$, 则第 $i + 1$ 个单元包含空格, 这个空格不可被吸收, 因此 $X_1X_2\cdots X_{n-1}qX_n \vdash_M X_1X_2\cdots X_{n-1}YpB$
 - 如果 $i = 1$ 且 $Y = B$, 则在 X_1 上写下的符号 B 被前面的空格吸收吸收, 不出现在下一个格局中, 从而 $qX_1X_2\cdots X_n \vdash_M pX_2\cdots X_n$



接受语言 $L = \{0^n 1^n | n \geq 1\}$ 的图灵机

接受 L 的图灵机的非形式化描述

开始时，在带上给定 0 和 1 的有穷序列，前后都是无穷的空格。交替地，该 TM 将把一个 0 改成 X，然后把一个 1 改成 Y，直到所有的 0 和 1 都匹配为止。

接受 L 的图灵机的形式化描述

该图灵机的形式化说明 $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$ ，其中 δ 由下表给出：

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

接受语言 $L = \{w\#w | w \in \{0, 1\}^*\}$ 的图灵机

- 语言 $L = \{w\#w | w \in \{0, 1\}^*\}$ 代表由 # 隔开的两个任意长度的相同子串 w 的字符串, L 无法由有限自动机识别
- L 可由图灵机识别, 图灵机每次在 # 左侧划掉一个字符, 然后向右移动读写头, 再划掉一个与左侧刚划掉的相同字符, 如果最后纸带为空, 则接受此语言, 如果在处理过程中在右侧找不到匹配字符, 则拒绝 L

0 1 1 0 0 0 # 0 1 1 0 0 0 □ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 □ ...
x 1 1 0 0 0 # x 1 1 0 0 0 □ ...
x 1 1 0 0 0 # x 1 1 0 0 0 □ ...
x x 1 0 0 0 # x 1 1 0 0 0 □ ...



图灵机的相关概念

- 如果经过系列移动后，图灵机 TM 最终进入了接受状态，则称 TM 接受输入语言
- 另一种“接受”：以停机方式接受，即如果 TM 进入状态 q ，扫描带符号 X ，并且在这种情况下没有移动 (即 $\delta(q, X)$ 无定义)，则说 TM 停机
- 无论是否接受都总是停机的图灵机是“算法”的好模型
- 如果解答给定问题的算法存在，就说该问题是“可判定的”
- 其它图灵机的扩展，比如多带图灵机、带状态存储图灵机、非确定型图灵机，其能力和单带图灵机一致，即能互相模拟



图灵机与可判定性

图灵可识别

如果一个语言能被某一图灵机识别，则称该语言是图灵可识别的，或者称该语言为递归可枚举语言 (RE)。识别过程中可能出现接受、拒绝或循环 (死机，不停机) 状态

图灵可判定

如果一个语言能被某一图灵机判定，则称该语言是图灵可判定的，或者称该语言为递归语言 (R)。判定指图灵机肯定会停机，不会循环

一个问题有没有解和一个问题是否是可判定的是不同的，因为某个问题可能无解，但却是可判定的，因为可以判定该问题不可解

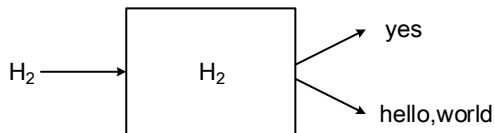
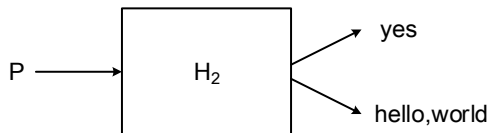
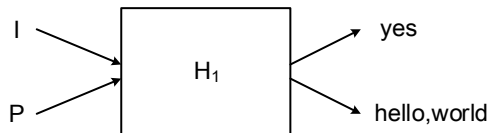
任意程序是否输出"hello, world" 不可判定

```
main() { printf("hello, world\n"); }
```

```
main() {  
    int n, total, x, y, z;  
    scanf("%d",&n);  
    total = 3;  
    while(1){  
        for(x=1; x<=total-2; x++)  
            for(y=1; y<=total-x-1; y++){  
                z = total - x - y;  
                if( exp(x,n) + exp(y,n) == exp(z,n))  
                    printf("hello, world\n");  
            }  
        total++;  
    }  
}
```



检验"hello, world" 问题不可判定性的证明



- 证 1) 假设存在检验"hello, world" 的程序 H , 根据检测成功与否分别输出"yes" 或"no"
- 2) 构造程序 H_1 , 其它不动, 只是"no" 替换为"hello, world"
- 3) 继续构造 H_2 , 将 P 既作为程序又作为数据
- 4) 将 H_2 作为它自身的输入, 制造矛盾, 导致 H_2 不可能存在, 于是 H_1 不存在, 从而 H 不存在

由此例推广, 编写能检测所有 bug 或者病毒的万能检测程序是不可能的!



存在计算机不能求解的问题的直观解释

- 首先明确，图灵机的计算能力强于所有已有的计算机
- 任何一个图灵机皆可编码为某一固定长度的字符串，世上存在的所有图灵机的个数与自然数集等势
- 假设所有图灵机能处理的所有问题也可以编码为串，则根据对角化方法总能构造出未出现过的串，该串对应的问题不能由任何已有图灵机解决
- 虽然可以为上述新串构造一个新的图灵机解决该问题，但问题是根本没有办法列出所有的图灵机，也就没办法开发出通用的万能图灵机

问题总比办法多!

图灵停机问题是不可判定的

图灵停机问题不可判定

A_{TM} 是不可判定的, 即 $A_{TM} = \{\langle M, w \rangle | M \text{ 是一个 TM 且 } M \text{ 接受 } w\}$ 具有不可判定性

证 1) 假设 A_{TM} 是可判定的, H 是 A_{TM} 的判定器, 构造如下:

$$H(\langle M, w \rangle) = \begin{cases} \text{ACCEPT,} & M \text{ accept } w \\ \text{REJECT,} & M \text{ reject } w \end{cases}$$

2) 从 H 构造图灵机 D , 以图灵机 M 的串描述 $\langle M, w \rangle$ 为输入:

$$D(\langle M \rangle) = \begin{cases} \text{ACCEPT,} & M \text{ reject } \langle M \rangle \\ \text{REJECT,} & M \text{ accept } \langle M \rangle \end{cases}$$

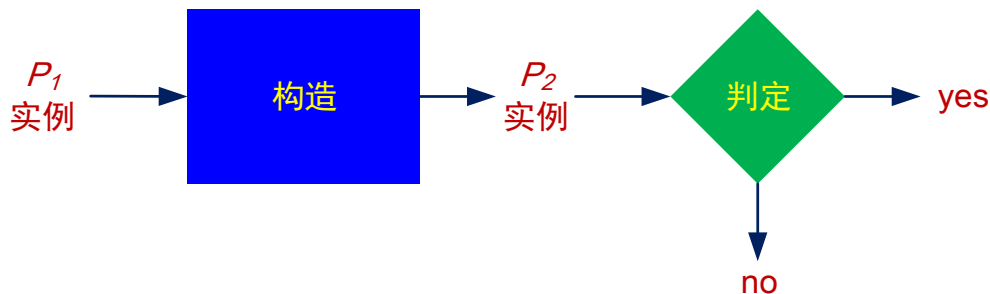
3) 制造自指, 从而制造矛盾:

$$D(\langle D \rangle) = \begin{cases} \text{ACCEPT,} & D \text{ reject } \langle D \rangle \\ \text{REJECT,} & D \text{ accept } \langle D \rangle \end{cases}$$

4) 由于矛盾, TM D 不存在, 从而 TM H 不存在。



不可判定性的通用证明思路



- 通用证明思路：将已有不可判定问题归约为欲证明的问题
- 上述归约方向不可反过来，否则无法证明
- 已知 P_1 不可判定，将 P_1 实例构造为 P_2 实例，如果 P_2 是可判定的，则 P_1 也是可判定的，矛盾，借此证明 P_2 不可判定

不可判定性为 CS 的研究划定了明确的界限，如果要求解的问题是无可判定的，则没有必要继续付出努力去求解！

不可判定性证明中的规约方向

不可判定证明中，应该从已知不可判定问题归约到待证的不可判定问题，原因如下：

- 1) 假设已知不可判定问题为 A ，待证不可判定问题为 B
- 2) 因为待证的是 B 不可判定，如果将 B 归约为 A ，即使 B 可判定，我们也可以放弃努力使得 B 可以规约为不可判定问题 A 。即此时通过 A 不可判定并不能说明什么问题
- 3) 如果归约方向反过来，将 A 规约为 B ，待证的是 B 不可判定，其否为 B 可判定，所以逆否命题为 B 可判定可归约为 A 可判定，这是不可能的，从而得证



提纲

- 1 可计算性
 - 通用计算模型 —— 图灵机
 - 可判定性

- 2 计算复杂性
 - 复杂性类
 - NP 完全性与可归约性
 - NP 完全性的证明



易解与难解

- “可计算性” 给出了哪些问题是无法用计算机求解的
- “计算复杂性” 研究理论上可解的问题的**实际求解难度**
- 不是所有问题都有高效的算法
- 一般认为具有多项式复杂度算法的问题为易解问题，需要指数或更高复杂度的时间/空间求解的问题为难解问题



描述复杂类的基础 —— 问题及其编码

抽象问题

定义抽象问题 Q 为在问题实例集合 I 和问题解集合 S 上的一个二元关系。

编码

要用一个计算机程序来求解一个抽象问题，应该用一种程序能理解的方式来表示问题实例，即将问题映射为二进制的串，编码过程由 e 来表示



P 复杂类的形式化描述

形式语言可用来表示判定问题与求解算法间的关系：

- 如果对给定输入 x ，算法输出 $A(x) = 1$ ，则算法 A “接受” 串 $x \in \{0, 1\}^*$
- 被算法 A 接受的语言是串的集合 $L = \{x \in \{0, 1\}^* : A(x) = 1\}$
- 如果 $A(x) = 0$ ，则算法 A “拒绝” 串 x
- 算法 A 不一定会拒绝一个输入串 $x \notin L$

$P = \{L \subseteq \{0, 1\}^* : \text{存在算法 } A \text{ 可在多项式时间内判定 } L\}$

$P = \{L : L \text{ 能被一个多项式时间算法所接受}\}$



P 复杂类的非形式化描述

多项式时间可解

一个判定问题 D ，如果其满足下述条件，则认为多项式时间可解的：

- 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 Yes 或 No 的答案
- 2) 存在一个多项式函数 p ，如果 D 的实例的大小为 n ，则 A 在不超过 $p(n)$ 个步骤内结束

如果一个问题多项式 (Polynomial) 时间可解的，就说该问题属于 P 类问题

对 P 类问题复杂度的解读

- 复杂度 $O(n^{100})$ 的问题虽然是一个难解问题，但在计算复杂性领域仍然是易解的 P 类问题，且实际中很难找到如此复杂的问题
- 对很多合理的计算模型来说，在一个模型上用多项式时间可解的问题，在另一个模型上也可以在多项式时间内解决
- 由于在加法、乘法和复合运算下多项式是封闭的，所以多项式时间可解问题具有很好的封闭性



NP 复杂类的形式化描述

验证算法

验证算法是含两个自变量的算法 A ，一个自变量是普通输入串 x ，另一个是称为“证书”的串 y 。如果存在 y 满足 $A(x, y) = 1$ ，则称算法 A 验证了输入串 x 。 A 所验证的语言是：

$$L = \{x \in \{0, 1\}^* : \exists y \in \{0, 1\}^*, A(x, y) = 1\}$$

NP 复杂类

NP 复杂类是能被一个多项式时间算法验证的语言类。一个语言 L 属于 NP，当且仅当存在 2 输入的多项式时间算法 A 和常数 c ，满足：

$$L = \{x \in \{0, 1\}^* : \exists y \text{ (证书)}, |y| = O(|x|^c), A(x, y) = 1\}$$


则算法 A 在多项式时间内验证了语言 L

NP 类问题

- NP 是与 P 对应的概念，但并不代表：No Problem, Not Polynomial, Not Possible...
- NP 的真正含义：non-deterministically polynomial-time solvable，即非确定性多项式时间可解
- P 类问题多项式时间可解实际指的是确定性多项式时间可解

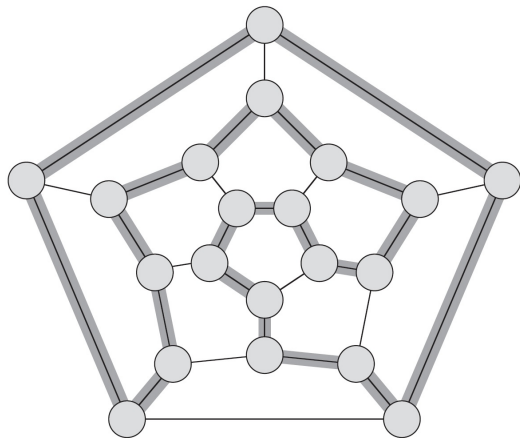
非确定性多项式时间可解

一个判定问题 D ，如果其满足下述条件，则认为是非确定性多项式时间可解的：

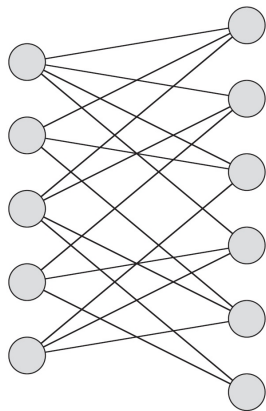
- 1) 存在一个算法 A ， A 的输入是 D 的一个可能解答， A 总是正确地判断该解答是否正确
- 2) 存在一个多项式函数 p ，如果  解答的大小为 n ，则 A 在不超过 $p(n)$ 个步骤内结束判断过程

如果一个问题是非确定性多项式时间可解的，就说该问题属于 NP 类问题

验证实例



(a)



(b)

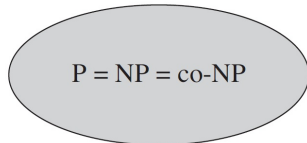
对于哈密顿环问题，图 (a) 的十二面体存在哈密顿环，图 (b) 具有奇数个顶点的二部图不存在哈密顿环

P vs NP

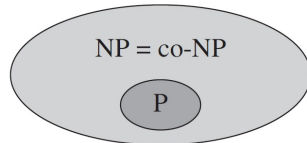
- 注意 P 与 NP 定义的差别，主要集中在第 1 条
- P 定义要求第 1 条能够给出问题的答案； NP 的第 1 条要求能够判断一个给定的答案是否正确
- 另一角度： NP 代表的是可以被非确定性图灵机在多项式时间内求解的问题，或者说 NP 代表的是其解答可以被一个确定性图灵机在多项式时间内验证的问题
- 一般认为，求解一个问题的难度要大于验证一个解的难度，所以一般有 $P \subseteq NP$
- 但是否 P 是 NP 的真子集，或者 $P = NP$ ？，都是既没有证实也没有证伪的问题
- NP 不但不意味着不存在多项式时间解，而且人们为很多 NP 问题找到了多项式时间解，比如素数测试问题



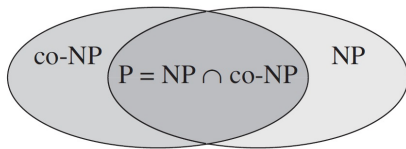
复杂类之间的可能关系



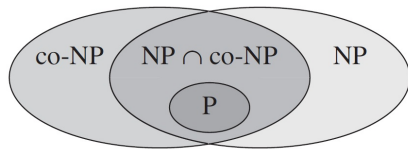
(a)



(b)



(c)



(d)

上述 co-NP 问题是 NP 问题的补问题，对于子集和问题，原问题 (比如) 为“是否存在某子集的和为 0?”，则补问题为“是否任何子集的和皆非 0?”

可归约性

归约函数

基于判定问题的形式语言体系，说语言 L_1 在多项式时间内可以归约为语言 L_2 ($L_1 \leq_P L_2$)，如果存在一个多项式时间可计算函数 $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ ，满足对所有的 $x \in \{0, 1\}^*$ ：

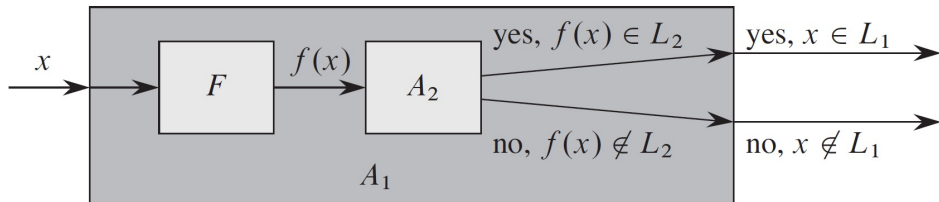
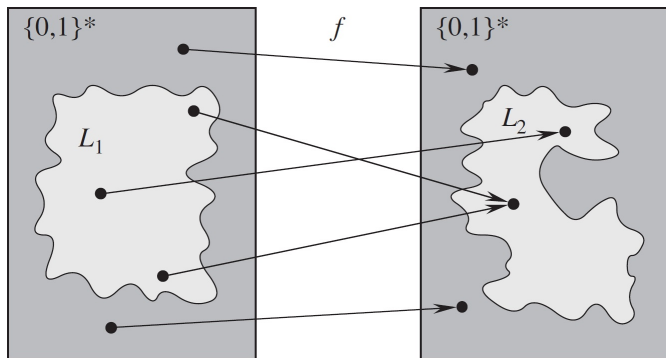
$$x \in L_1 \iff f(x) \in L_2$$

则称函数 f 为归约函数，计算 f 的多项式时间算法 F 称为归约算法

引理

如果 $L_1, L_2 \subseteq \{0, 1\}^*$ 是满足 $L_1 \leq_P L_2$ 的语言，则 $L_2 \in P$ 蕴含 $L_1 \in P$

归约函数/过程示意



哈密顿回路问题 — 旅行商问题的归约

- 哈密顿回路问题：设无向图 $G = (V, E)$ ，其中 V 是点集， E 是边集，寻找经过图 G 中每个节点一次且仅一次的回 (环) 路，称为 Hamilton 回路问题
- 旅行商问题：当上述图 G 是加权完全图时，求该图的最短哈密顿回路问题成为旅行商问题
- 哈密顿回路问题可以多项式归约到旅行商问题：给定图 G ，结点为 v_1, v_2, \dots, v_n ，在图 G 上构造带权重的完全图 H 如下：如果一条边 $\{v_i, v_j\}$ 在原来的图 G 里，则该边的权重为 1；否则，该边的权重为 2。显然，图 G 的哈密顿环问题可以通过解决图 H 里的旅行商的判定问题来解决



NP 难

NP 难

语言 $L \subseteq \{0, 1\}^*$ 是 NP 难的 (NP -hard), 如果对每一个 $L' \in NP$, 有 $L' \leq_p L$

- NP 难不代表某问题在 NP 里而且很难
- 问题 S 是 NP 难要求 NP 里的每一个问题 Q 都可以多项式时间规约到问题 S
- 多项式规约指这个转换可以在多项式时间内完成, 因此解决了 S 就解决了 Q , 并且它们的解决方案的效率之差不会超过一个多项式复杂度。如果 S 能够在多项式时间内解决, Q 也能够 在多项式时间内解决
- 本质上 S 是 NP 难表示的是 S 不比 NP 里面的任何问题容易

NP 完全

NP 完全性

语言 $L \subseteq \{0, 1\}^*$ 是 NP 完全的 (NPC), 如果:

- 1) $L \in NP$
- 2) 对每一个 $L' \in NP$, 有 $L' \leq_P L$

NPC 的含义有两层:

- NP: 非确定性图灵机多项式时间可解
- 完全: 解决一个则解决了所有问题

引理

如果任何 NPC 问题是多项式时间可解的, 则 $P = NP$ 。反之, 如果存在某一 NPC 问题不是多项式时间可解的, 则所有 NP 完全问题都不是多项式时间可解的

$NP-hard$ 与 NPC 的关系

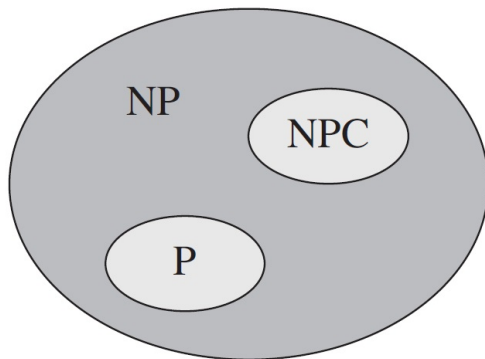
- 比较 $NP-hard$ 与 NPC 的定义, NPC 除要求对每一个 $L' \in NP$ 都可以 $L' \leq_p L$ 外, 还要求 $L \in NP$, 所以 NPC 是 $NP-hard$ 的子集
- $NP-hard$ 说明某些问题连多项式时间的验证都做不到, 比如停机问题
- 在计算理论以外的领域中, 有时 $NP-hard$ 与 NPC 可以混用, 都表示待求解的问题“很难”



$P/NP/NPC$ 的关系

一般认为 NP 问题是 NP 里最困难的问题， NP 是一个复杂性类，其中包含上千个问题，分布在各种领域，但它们中的任意两个问题都是可以相互规约的

$P/NP/NPC$ 三者之间，一般认为最可能满足如下关系：



NP 完全性的证明

如果可以证明某问题为 NP 完全问题，则无需再浪费资源寻求本不存在的高效确定型算法！

NP 完全性的证明思路

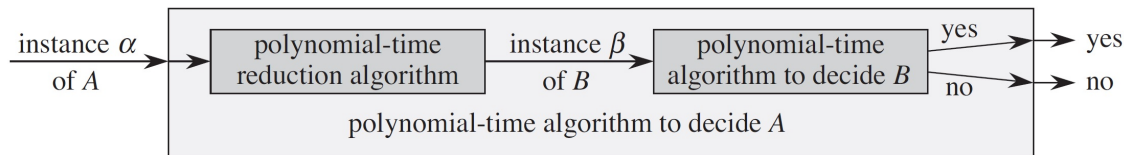
- 可以严格根据 NP 完全性的定义来证明，即对于问题 S ，必须证明 S 在 NP 里且 NP 中任何一个问题皆可以多项式规约为 S
- 用归约法来证明
- 上述两种方法后者的难度明显更低



归约法证明 NP 完全性

归约法的步骤

- 1) 对于给定判定问题 B ，首先证明 B 在 NP 里面
- 2) 选择一个已知的 NP 完全问题 A
- 3) 证明 A 可以多项式归约到 B



由于 A 是已知的 NP 完全问题，所有 NP 里的问题都可以多项式时间归约到 A ，而由于第 3 步证明了 A 可以多项式归约到 B ，因此 NP 里的问题都可以多项式归约到 B 。由于第 1 步证明了 B 在 NP 里，因此 B 是 NP 完全问题

最大的问题是：上哪里找到第一个 NP 完全问题?!

第一个 NP 完全问题的证明

- 第一个 NP 完全问题只能严格根据定义来证明，即将所有 NP 问题归约到所要证明的问题上
- 上述方法的难点在于，所有 NP 问题的数目是未知的
- 解决办法是用图灵机将所有 NP 问题进行抽象，抽象为一个问题，从而仅需证明一个问题
- 斯蒂芬·库克 1971 年证明了布尔可满足 SAT 问题是 NP 完全的 (The Complexity of Theorem Proving Procedures)，并获得了 1982 年的图灵奖。利奥尼德·莱文 (Leonid Levin) 在 1972 年的论文 "Universal Search Problems" 里面也独立的证明了该问题为 NP 完全。

库克定理

布尔组合电路可满足性属于 NP 完全问题。证明从略！



布尔组合电路可满足性

电路可满足性

给定一个单输出由 AND、OR、NOT 门构成的布尔组合电路，将此电路 C 编码为一个二进制串 $\langle C \rangle$ ，该串的长度与电路本身规模呈多项式关系，则我们可定义：

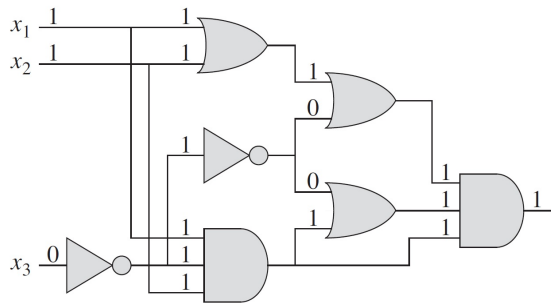
$$CIRCUIT - SAT = \{ \langle C \rangle : C \text{ 是一个可满足的布尔组合电路} \}$$

电路可满足性问题在 EDA 领域很重要，如果一个子电路始终输出 0，则可以将其所有扇出接于零电位端并将此子电路优化掉。

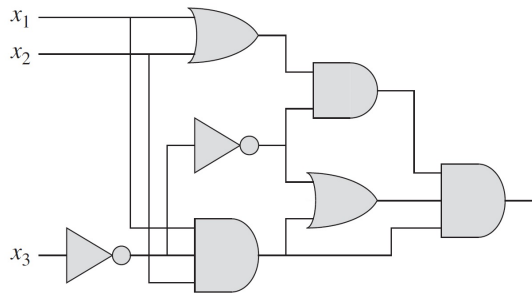
库克定理的证明思路

首先证明电路可满足性问题属于 NP 类；接着证明 NP 中的每一种语言可在多项式时间内归约为 CIRCUIT-SAT

CIRCUIT-SAT 问题示意



(a)



(b)



示例：证明公式可满足性 (SAT) 问题属于 *NPC* 问题

公式可满足性

公式可满足性问题 SAT 的实例是由下列成分组成的布尔公式 ϕ ：

1. n 个布尔变量 x_1, x_2, \dots, x_n
2. m 个布尔连接词 $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
3. 括号

从而 SAT 问题可采用形式语言描述为：

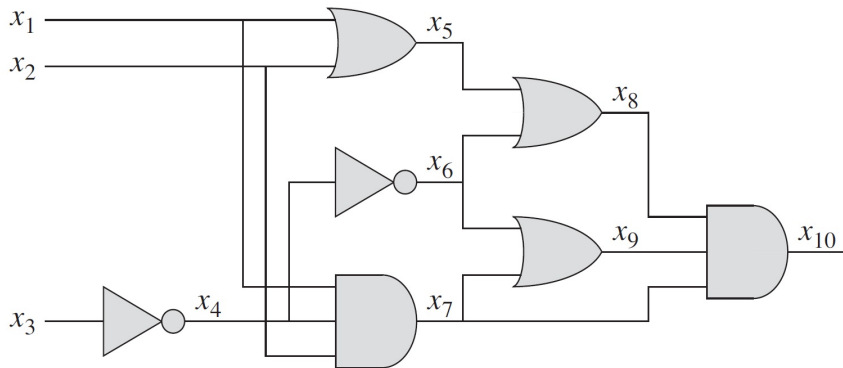
$$SAT = \{ \langle \phi \rangle : \phi \text{ 是一个可满足的布尔公式} \}$$

SAT 公式例子

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4) \wedge \neg x_2)$$

定理：布尔公式的可满足性问题是 *NP* 完全的

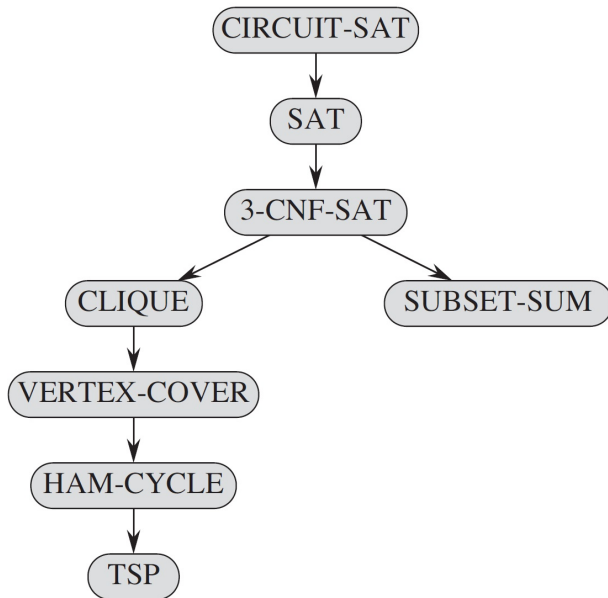
证明思路：将 CIRCUIT-SAT 归约为 SAT 问题



$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7))\end{aligned}$$



NPC 问题的归约路线图



其它常见 *NPC* 问题

1. 子图同构问题：两个无向图 G_1 、 G_2 ，问 G_1 是否和 G_2 某个子图同构
2. 整数线性规划问题 ILP
3. 一般非线性规划问题
4. 最大独立集问题 MIS：补图的最大团
5. 图着色问题
6. 作业调度问题 JSP
7.



证明 NP 完全性时的归约技巧

需要选择合适的 NP 完全问题来进行归约：

- 3-SAT：当其它问题难以归约到所要证明的问题时可以考虑采用
- 整数划分：要证明的问题涉及很大的数时使用
- 顶点覆盖：证明任何需要进行选择的、与图相关的问题时使用
- 哈密顿环：证明任何依赖排序的问题时使用
- 使用限制：证明问题的一个特例或者一部分是 NP 完全问题
- 局部替换：将局部的东西进行替换从而达到归约，比如 3-SAT 的证明



小结

本章仅需了解相关的基本概念，考试也仅考查基本概念！

