计算理论

本语境下的”计算“：一切按照明确规则从输入得到输出的过程。问“什么不能被计算”就是问：有没有问题天生就无解——你再聪明、AI 再强也白搭。

计算模型

- **Alphabet and Language**
  - Alphabet: a set of symbols.
    - Types: Roman alphabet(a,b,c,...); Binary alphabet(0,1)
    - 一般记为$\sum$
  - string: 一个从字母表中选出来的符号所组成的、有限长度的序列
    - 空string记为ϵ; 由序列长度定义string的length
    - 对应地，记alphabet$\sum$的set of all strings 为$\sum^*$. 如，$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$。 进一步，记$\sum^n$为the set of string of length n.
    - 操作
      - concatenation: uv表示把u的每个字符依序排列后，再接上v的每个字符
  - Language: a set of strings over an alphabet, $L \subseteq \sum^*$
    - 各种性质：computable/decidable/NP/P/decision problem/computation problem
- **encodings of problem**
  - 含义：计算模型只能处理字符串，因此需要把其他对象（如数字、图、函数、程序）转换为字符串。“编码”就是把数学世界打包进字符串，让计算模型能理解。
  - Any decision problem can be represented as a language
    - 定义：回答是yes or no
    - 例子：判断是否为素数
    - 本课只学这个
  - Any computation problem can be represented as a function
  - Encoding doesn't matters.
    - we can switch between different ways by preprocessing
    - 如，encoding a graph，看其是否存在Hamilton Path. 可用adjacent list vs adj matrix. 但把list和matrix转换就行了，只需要an extra step to switch between encodings
    - 虽然encoding确实影响length，但still polynomial, does not effect the 可计算性，无根本影响
- **Computation model**
  - 含义：计算模型是刻画计算的抽象的形式系统或数学系统。它规定了输入形式、内存结构、操作方式和输出行为，用于模拟和分析实际计算过程
  - **Finite Automaton**

- 直觉：有限自动机就是一种“记不住太多东西”（with no memory）的极简计算机器，它只能根据当前状态和当前输入，自动决定下一步要做什么。只有有限个状态。
- Deterministic finite automaton(DFA)
  - a 5-tuple $(Q, \sum, \delta, q_0, F)$, where Q is a finite set of states, $\sum$ is an alphabet, δ: $Q \times \sum \mapsto Q$ is the transition function（从当前状态和输入字母得出下一个状态）， $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accept states.
  - 行为规则
    - FA accept a string: 当一个字符串逐步输入有限自动机后，最终停在一个“接受状态”（accept state），就说这个自动机“接受了”这个字符串。
    - FA accepts a language: if L is the set of all strings that M accepts. 指的是完全精确地接受所有在语言中的字符串，并拒绝所有不在语言中的字符串。
- Nondeterministic finite automaton(NFA)：对于同一个input，有多种可能。
  - 直觉：对于多个path，同时模拟；只要至少有one path ending in an accept state，就说accept.
  - 定义：5-tuple$Q, \sum, \delta, q_0, F$, 其中
    $\delta : Q \times \sum_\epsilon \mapsto P(Q), where P(Q) \overset{\text{def}}{=} \{S : S \subseteq Q\}$[1]
  - 行为规则
    - NFA accepts a string：对于一个string，存在一组状态（这些状态之间是一个一个往下发展的），使得按照字符串走完，落入接受状态，那么字符串就被 NFA 接受。
- regular language[2]
  - 定义：若一language能被某个DFA accept，则其是正则语言。结构简单、规律清晰、可以靠有限“脑子”读懂的语言。
  - 性质：RL are closed under operations, ∩、∪、补、concatenation, and star[3].即这些操作得到的东西还是RL。

    $$\textbf{(concatenation)}\ A \circ B\ (\textbf{or}\ AB) = \{xy | x \in A\ \ \textbf{and}\ \ y \in B\}$$

    $$\textbf{(star)}\ A^* = \{x_1 x_2 \cdots x_k : k \geq 0\ \ \textbf{and}\ \ x_i \in A\}$$

    - 证明：核心思想是“product construction”，构造同时满足两个自动机条件的自动机[4]。

Thm 3.6 If L_1 and L_2 are regular languages, $L_1 \bigcup L_2$ is also a regular language.

Proof Idea: Simulate M_1 and M_2 simultenously.

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1 \subseteq Q_1)$ accept L_1, $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2 \subseteq Q_2)$ accept L_2.
Construct $M = (Q, \Sigma, \delta, q, F)$ that accepts $L_1 \bigcup L_2$.

1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) : r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ is the same
3. $\delta : Q \times \Sigma \to Q$ is defined as

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

   where $(r_1, r_2) \in Q$, $a \in \Sigma$.

4. $q_0 = (q_1, q_2)$
5. $F = \{(r_1, r_2) | r_1 \in F_1 \text{ and } r_2 \in F_2\}$ $\square$

对每一个操作，都有一个独特的证明方式，其中有些用到NFA。

- 自然地，有个问题是：RL到底是被NFA还是被DFA所接受的语言?
  - 证明：DFA和NFA是等价的。即：若language L被一个NFA接受，则有一个DFA也接受它。
  - 这也就说明，两类FA识别的语言是一样的
  - 证明：A language is regular iff some NFA accepts it.
    - convert a NFA to DFA：用子集构造法。把 NFA 的多个状态的组合 看作一个 DFA 的单一状态。如果 DFA 某个状态（即某个 NFA 状态的集合）包含 任意一个 NFA 的终止状态，那么这个 DFA 状态就是终止状态
- Non-regular language
  - Motivation：学什么是、什么不是RL，都是在问，什么是可计算的?
  - 结论：Almost all languages are not regular.[5]
  - 虽然知道了这个，但sometimes need to prove some specific language is non regular. eg, palindrome
    - Lem（Pumping Lemma）:L为RL，binary. 则有：
      $p \in , \in L, \geq p, x, y, \in {}^*, = xy, xy \leq p, y > 0, \in , xy \in L$
      - 直觉：如果一个语言是正则的，那么它的结构足够“简单”，以至于任何足够长的字符串都可以被“抽水”——即有一段中间的子串可以重复任意次，生成的新字符串依然属于这个语言[6]。
      - 例子：

        💡 **举个例子：**

        考虑语言 $L = \{a^n b^n \mid n \geq 0\}$，它不是正则语言。

        - 如果它是正则的，那就必须满足 pumping lemma。
        - 假设我们选了一个足够长的字符串，比如 $s = a^{100} b^{100}$
        - 抽水引理说中间可以找到一个 “y” 段，只含 a，但如果我们把这个 “y” 多重复几次，得到的是 $a^{100+k} b^{100}$，**就不再是 L 中的元素了**（因为 a 和 b 的数量不相等）。
        - 所以矛盾，说明这个语言不是正则的。

- **Turing Machine**

- Motivation：什么是计算？哪些问题是可以通过机械过程解决的？是否存在一种通用的、机械的方法，可以判断任何给定数学命题是否是可证明的？
- 定义：7-tuple $M = (Q, \sum, \Gamma, \delta, q_0, q_{ccep}, q_{reec})$. Q is a set of states, $\sum$ the input alphabet where $\sum$; Γ the tape alphabet, where, $\in \Gamma, \sum \ \ \Gamma$; $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, S\}$.
  - 直觉：通过一条无限长的纸带、一个读写头和一组有限状态的操作规则，模拟人类进行纸笔计算的过程。move right and left; read and write。转移状态的时候，有三个维度：状态的变化、input的变化和direction的变化
  - 图灵机中对accept state的定义：输入一个accept state后，Turing machine halts。
  - 意义：虽然几千年来都在计算，但这是第一次正式定义什么是计算。图灵机之所以重要，不是因为它能帮你写代码，而是因为它定义了什么是“能被写出来的代码”。
  - 行为规则
    - a TM accepts an input: entering an accept state
    - a language is Turing decidable/recognizable if 有TM accepts it
    - a TM decides a language L in time $T(x)$: 对于L全都接受，outside L全都不接受；对input x，the TM halts in $T(x)$ steps.
    - a TM computes f in time $T(n)$: 能在有定义的输入上输出正确值，并在规定时间内停机，就说它计算了这个函数。
- 例子
  - add 1 to binary. 若q0是0或blank，就变成1，然后S（stay)；若是1，就变成0，然后move right，self-loop
  - decide if palindrome
- Variants
  - 核心思想：这些变种不能多解新问题，但可以更快解问题。
  - 基础：Lemma of Change of alphabet

    Lem 4.7 (Change of alphabet) Let $L \subseteq \{0,1\}^*$. If L is decidable by a TM on alphabet Γ in time $T(n)$, then L is decidable in time $O(\log|\Gamma| T(n))$ on alphabet $\Sigma = \{0,1\}$.

    Proof Encode a symbol in Γ using $k \stackrel{\text{def}}{=} \lceil \log_2 |\Gamma| \rceil$ bits. To simulate one step of M, the new machine M' will

    1. Spend k steps to read the symbol $a \in \Gamma$ and determine the new symbol $b \in \Gamma$ to be written.
    2. Overwrite a by b.
    3. Move the head to the next symbol.
    4. Transit to the next state.

    In total, it takes $k + k + k + 1 = O(k)$ steps. □

    思想：图灵机的判定能力和时间复杂度，在合理的编码下，与所使用的字母表无关。

- Multitape TM

  Def 4.8 (Multitape TM) A k-tape TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\mathtt{accept}}, q_{\mathtt{reject}})$, where

  $$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$$

  Typically, the first tape is the input tape.

  - Lemma：能比标准TM省很多时间

    Lem 4.9 Let $L \subseteq \{0,1\}^*$. If L is decidable by a k-tape TM in time $T(n)$, then it is decidable in time $O(kT^2(n))$ by a single-tape TM.

    Proof Use positions $0, k, 2k, \cdots$ to simulate the first tape.

    Use positions $1, k+1, 2k+1, \cdots$ to simulate the second tape.

    Generally, use position $i-1, k+i-1, \cdots$ to simulate the i^{th} tape, where $i = 1, 2, \cdots, k$.

    For every $a \in \Gamma$, introduce $a, \hat{a} \in \Gamma'$ with $\hat{a}$ marking the location of the head. To simulate one step of M, the machine M' will

    1. Sweep form left to right to read k symbols (marked by $\hat{}$).
    2. Use $M's$ transition function to determine the new state and actions.
    3. Sweep back from right to left to update the symbols and adjust the head positions (i.e., move the hats).

    It takes $O(T(n)) + 1 + O(kT(n))$ to simulate one step. In total, $O(kT^2(n))$. □

- Bidirectional TM

  a normal TM whose tape is infinite in both directions.[7]

  - Lem:

    Lemma 4.10 Let $L \subseteq \{0,1\}^*$. If L is decidable by a bidirectional TM in time $T(n)$, then L is decidable by a single-tape TM in time $O(T(n))$.

    双向TM虽然功能上看起来更“自由”，但从运行时间级别来说，与普通TM没有质的提升。

- **Random access memory TM**
  - 含义：a multi-tape TM with a random access memory. which:
    - M has an infinite memory tape A indexed by 自然数，其中的每个位置都是一个memory cell，可以存储一个整数
    - an address tape，其中每个cell都是一个整数，多个cell组合起来表示一个地址。地址和整数之间的关系与编码方式有关，常见如二进制编码。在进行内存访问操作时，机器首先扫描 address tape 上预定义[8]的一段（或直到遇到终止符号），将其内容解析为一个整数地址
    - Define a set of special state $Q_{cce} \subseteq Q$. 当M进入$q \in Q_{cce}$，执行内存访问操作。即：
      - if the address tape contains R, $A[]$ is written to the cell next to R. 改address tape
      - If contains , then $A[]$ is set to the value . 改memory tape
    - 直观理解：控制器读取 address tape，得知“你要访问第 X 个寄存器”，然后跳到 memory tape 的第 X 个位置，执行读写操作。
  - 运行时间：if $L \subseteq \{0,1\}^*$ is decidable by a RAM TM in time $T(n)$, then it's decidable by a multi-tape TM in time $O(T(n))$. Moreover, if

the address used has bounded length $O(1)$, then L can be decided by a multi-tape TM in time $O(T(n)^2)$.
- 与汇编语言：Every assembly language instruction can be implemented by a RAM TM in $O(1)$ steps. 这就链接了理论与实践，之后的复杂度分析中就可以使用RAM模型了

- 重要相关事物
  - Church–Turing Thesis：图灵机就是“计算”的极限。凡是你能设计算法解决的东西，图灵机都能解决；否则就是真的无法计算。
    - 严肃含义：Every function that can be physically computed can be computed by TM.[9]
    - Strong CT Thesis: ...with polynomial overhead.
  - C++——Assembly Language——RAM TM——multiple TM——single-tape TM
- Universal TM
  - Motivation：有Special purpose TM(calculator)。想找一个TM that can simulate other TMs，这样的话它就通用了，如手机可以运行任意程序
  - 定义：a UTM is a theoretical machine that can simulate any other TMs. 形式上说，记UTM为U, 任意TM为M, 为其的编码，则$U(, x) = M(x)$。核心是，把任意TM的编码也当input输入进去。自然地，问题是怎么把TM编码成string
  - Encoding of TM
    - 也就是把TM的一些要素，初始状态、accept状态等等翻译成编码，$= < M >$
    - Assume our encoding satisfies the following properties
      - every string $\in \{0, 1\}^*$ represents some TM demoted by M_0
      - every TM is represented by infinitely many strings in $\{0, 1\}^*$。因为可以有不同写法表示同一个机器（类似于程序员写的两个程序功能一样但代码不完全一样）
  - UTM存在性定理
    - There is a 3-tape TM such that x, $\in \{0, 1\}^*$, $(x,) = M(x)$. Moreover, if M halts within T steps, then $(x,)$ halts in $O_k(T \operatorname{og} T)$ steps, where k is the number of tapes in M. : the previous TM is encoded as . 就是说，对于任意TM，都能被转化为一个UTM来实现同样操作，同时付出一些并不很多的代价。
    - Tape 1（输入带）：存放输入字符串 x； Tape 2（程序带）：存放图灵机 M 的编码（也就是 ）。；Tape 3（模拟带）：模拟被模拟的图灵机 M 的工作带。
    - for tape 3, core problem is how to combine k tapes into one tape. 直觉是by working over a larger alphabet Γ^k. Assume tapes are two-way infinite. 假设k=3，原alphabet是26个字母，新的alphabet就是一个三维数组，其中每个元素都还是26个字母中的一个。然后把三条

tape“对齐”，基准点是三个head所对应的位置，然后把head的移动变成subtape的“移动”。

- 但是，这是$O_k(T^2)$. 每次原图灵机的 head 移动一格，模拟机的 tape 上就要做一堆复杂的调整，因为你得把很多内容向左或向右移动。如果你每移动一次就搬一次所有内容，那么总成本就很高。
- 为改善效率，引入缓冲区和指数级分区来降低成本[10]。:
  - 环境配置：先split each parallel tapes into zones denoted by $R_0, L_0, R_1, \ldots$, where $R = L = 2^1$. R 是center cell右边的，L是左边的，center cell也就是head所对应的位置。The center cell is not in any zone。引入代表Buffer space的符号，按照以下规则插入Buffer：each zone要么是full or empty or half-full; buffer space的总数要正好是总cell的一半。
  - 然后开始运动——shift op。假设我们关注的是右侧第i个区域R，它的总容量是2，其中一半是正常写入符号的空间，另一半是预留的缓冲区。我们持续往这个区域中添加符号（即模拟写操作），直到这个区域的“非缓冲部分”被写满。这时，触发一个shift：把这个区域最左边的符号拿出来，放到 tape 的“中心”位置（也就是 head 当前停留的地方）；然后将剩下的符号整体向左移动；，其尾部腾出了新的缓冲空间，使得后续的写入操作可以在不搬移整个 zone 的情况下继续进行。虽然一次shift很重，但由于是指数级分区，不会有很多次shift，也就不会频繁触发重新编码或大规模的调整。
  - The idea is to allocate additional buffer space so that each move has an amortized cost of $O_k(\mathrm{og}\, T)$.

Computability

- Halting Problem
  - $L_{hl} = \{(, x) \colon M \text{ hats on input } x\}$
  - Prove this is undecidable
    - 定义$L_{flp} = \{ : M \text{ does not accept } \}$。通过diagonalization argument说明此是undecidable。

      Proof Assume for contradiction that $L_{\mathtt{flip}}$ is decided by some TM M_α. Thus, $L(M_\alpha) = L_{\mathtt{flip}}$.

      1. $\alpha \in L_{\mathtt{flip}}$. By the definition of $L_{\mathtt{flip}}$, M_α does not accept α. So, $\alpha \notin L(M_\alpha) = L_{\mathtt{flip}}$. Contradiction.
      2. $\alpha \notin L_{\mathtt{flip}}$. By definition, M_α accepts α. So, $\alpha \in L(M_\alpha) = L_{\mathtt{flip}}$. Contradiction. □

    - 通过Reduction将halt与flip构建联系：$L_{flp} \leq L_{hl}$，reduce flip to halt，意思是halt至少可flip一样难，或者flip一般更简单。如果halt decidable的话，L_{flp}也是decidable的，而这不可能，故halt undecidable。
    - 假设语言 $L_{\text{hat}} = \{(, x) \quad M \text{ hats on input } x\}$ 是可判定的，即存在图灵机 M_{hat} 可以判定任意图灵机在任意输入上的停机性。我们构造图灵机 M_{fip} 判定语言 $L_{\text{fip}} = \{ \quad M \text{ does not accept } \}$。$M_{\text{fip}}$ 的操作如下：在输入 上，运行 M_{hat} 判断

M 是否在输入 上停机；若 M_{hat} rejects $(,)$，则 M_{fip} 接受(loop forever means machine does not accept the string)；否则模拟 $M()$，若其接受，则 M_{fip} 拒绝，若其拒绝，则 M_{fip} 接受。如此便构造出一个可以判定 L_{fip} 的图灵机，与 L_{fip} 已知不可判定相矛盾，故假设不成立，L_{hat} 不可判定。

- 意义：这是undecidable，意味着没有图灵机能判定所有其他图灵机会不会停。或：不能写一个程序 P，去判断另一个程序 Q 在输入 x 下会不会永远运行
- 例子：用halt的reduction证明 （证明要是这decidable，halt或某个被证明undecidable了的语言也decidable。）
  - $L_{ccep} = \{(,x) : M\, ccep\, x\}$

    Proof Assume for contradiction $L_{\texttt{accept}}$ is decidable, i.e., $\exists$ TM $M_{\texttt{accept}}$ that decides $L_{\texttt{accept}}$. We construct a TM $M_{\texttt{halt}}$ as follows.

    On input (α, x), create a new TM M_β, which always accepts whenever M_α halts. If M_α loops forever, M_β also loops forever. Run $M_{\texttt{accept}}$ on input (β, x), forward the output.

    Clearly, $M_{\texttt{halt}}$ decides $L_{\texttt{halt}}$. Contradiction. □

  - $L_{empy} = \{< M >: L(M) = \}$.什么input都accept不了的TM的encoding。

    Proof Assume for contradiction that $L_{\texttt{empty}}$ is undecidable, i.e., $\exists$ TM $M_{\texttt{empty}}$ that decides $L_{\texttt{empty}}$. We construct a new TM $M_{\texttt{halt}}$.

    On input (α, x), construct a new TM M_β as follows. M_β's input is $y \in \{0,1\}^*$.

    1. Simulate M_α on input x.
    2. If Step (1) halts, M_β always accepts y.

    Clearly, $L(M_\beta) = \varnothing$ if M_α does not halt on x. Otherwise, $L(M_\beta) = \{0,1\}^*$.

    Run $M_{\texttt{empty}}$ on β and flip its output.

    Clearly, $M_{\texttt{halt}}$ decides $L_{\texttt{halt}}$. Contradiction. □

  - $L_{reulr} = \{ : L(M) \text{ is a reguar anguage}\}$

    Proof Assume for contradiction that $L_{\texttt{regular}}$ is undecidable, i.e., $\exists$ TM $M_{\texttt{regular}}$ that decides $L_{\texttt{regular}}$. We will show $L_{\texttt{accept}}$ is decidable. Construct a TM $M_{\texttt{accept}}$, which on input (α, x), as follows:

    1. Construct the following TM M_β, where the input of M_β is y.
       - If y has the form 0^n1^n, accept.
       - If y is not of the form, simulate M_α. On input x, accept y if M_α accepts x.
    2. Run $M_{\texttt{regular}}$ on β. Forward its output.

    Observe

    $$L(M_\beta) = \begin{cases} \{0^n1^n : n \geq 0\}, \texttt{ If } M_\alpha \texttt{ does not accept } x \\ \{0,1\}^*, \texttt{ Otherwise} \end{cases}$$

    If M_α accepts x, $M_{\texttt{accept}}$ accepts (α, x). Otherwise, $M_{\texttt{accept}}$ rejects (α, x).

    Clearly, $M_{\texttt{accept}}$ decides $L_{\texttt{accept}}$. Contradiction. □

- Nontrivial properties of languages
  给定TMs的语言集合 (M)，一个关于语言的性质 P 是nontrivial，当且仅当：
  至少一个图灵机 M_1 使得 (M_1) 满足 P； 至少一个图灵机 M_2 使得 (M_2) 不满足 P。nontrivial 意味着这个性质有辨别力，不是“所有语言都有”或者“所有语言都没有”的性质。P是一类图灵机编码的集合,它表示“哪些图灵机具有某种语言性质.

  所以性质的定义是：

  - $\langle M\rangle \in \mathcal{P} \iff L(M)$ 满足某种语言性质

  这意味着：

  - 虽然我们研究的是语言的性质，但我们要构造的是“**图灵机编码是否属于某类**”的问题

  - Rice's theorem：所有关于图灵可识别语言的非平凡性质都是不可判定的。不管这个性质是“是不是正则语言”、“是不是有限语言”、“是不是包含某个字符串”、“是不是递归语言”等，只要不是“所有语言都满足”或“所有语言都不满足”的性质，它就会落入 Rice 定理的打击范围。
  - 证明

    Proof WLOG $\mathcal{P}$ does not contain the language $\varnothing$. Assume for contradiction that $\mathcal{P}$ is decided by a TM M. Since $\mathcal{P} \neq \varnothing$, pick an arbitrary $\beta \in \mathcal{P}$. Construct a TM $M_{\texttt{accept}}$.

    1. On input (α, x), construct a TM M_γ as follows. Simulate M_α on input x.
       - If M_α does not accept x, M_γ loops forever.
       - If M_α accepts x, simulate M_β.
    2. Run M on input γ. Forward its output.

    We verify $M_{\texttt{accept}}$ decides $L_{\texttt{accept}}$. If M_α accepts x, then $L(M_\gamma) = L(M_\beta)$. Since $\beta \in \mathcal{P}$, $M_{\texttt{accept}}$ will accept (α, x). If M_α does not accept x, then $L(M_\gamma) = \varnothing$. So $\gamma \notin \mathcal{P}$, M will reject. □

- Natural Problems
  - Natural Problems（自然问题），指的是那些看起来自然合理、不是专为构造不可判定性而设计的问题，但它们却也是不可判定的。
  - Post Correspondence Problem
    - Motivation: 有没有某种简单的形式系统，也能表达图灵机不可判定性？构造一个无状态、无变量、只靠字符串拼接的系统.如果给你一堆小卡片（每张卡片有上下两个字符串）， 能不能选出一串卡片，使得拼完上面的字符串和拼完下面的一模一样?

- detailed:

  Domino

  $$\left[\frac{b}{ca}\right]$$

  A collection of dominos

  $$\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}$$

  A match

  $$\left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{a}{ab}\right], \left[\frac{abc}{c}\right]$$

  $\mathrm{P} = \left\{\left[\frac{t_1}{b_1}\right], \cdots, \left[\frac{t_k}{b_k}\right]\right\}$

  $\mathrm{PCP} = \{\langle \mathrm{P} \rangle : \mathrm{P}\ \texttt{is an instance with a match}\}$

  Thm 5.10 (Emil Post 1946) PCP is undecidable.

Complexity

- 核心：How much resources are required to solve a problem? Or what problems can be solved efficiently?
  - Resources: time, space, randomness, paralism...主要关注time and space.
- 基本概念
  - Polynomial
    - 一般认为多项式时间算法 = 高效算法（efficient），多项式时间内可解的问题集合被称为 P类问题（P class）
    - 两者关系：P一定是NP的子集（都能求解出来了，大不了求解后验证；有问题只能被验证而无法被高效求解，如拼图）
    - $T : N \mapsto N, L \subseteq \{0,1\}^*$ is in $DTIME[T(n)] \iff$ there is a TM that decides L in time $O(T(1))$.DTIME means Deterministic time，$DTIME[T(n)]$ is a set of languages which can be decided in time $O(T(n))$.
    - $P = \bigcup_{k \geq 1} DTIME[n^k] = DTIME[n^{O(1)}] = DTIME[poly(n)]$.在输入规模n的某个多项式函数范围内可以被decided的语言。
  - Non-deterministic polynomial
    - NP is the set of languages that can be verified efficiently. 即给出一个解之后，我们可以在多项式时间内检查它是否正确。称我们要验证的那个解为certificate.
    - 例子
      - $GI = \{< G, H >: \text{undirected graphs G and H are isomorphic}\}, GI \in NP$. An isomorphism of G and H is a bijection between $V(G)$ and $V(H)$, i.e., $f : V(G) \to V(H)$such that u and v are adjacent if and only if $f(u)$ and $f(v)$ are adjacent., denoted by $G \cong H$.
      - $CLIQUE = \{< G, k >: \text{G contains a } K_k \text{ subgraph} \subseteq \{0,1\}^*\}$
      - Traveling salesman problem(TSP)
        - Decision problem version. 不需要找最短路，因为有了decision之后，再做一个binary search就是一个优化问题了。总可以把一个优化

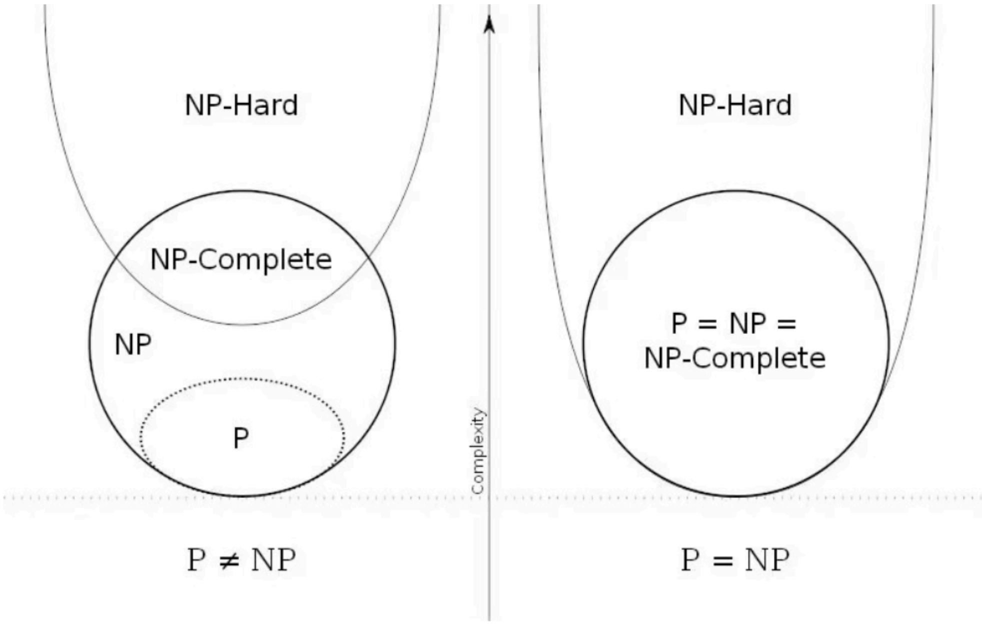
问题转化为decision问题。

- 关系
  - 定理：$P \subseteq NP \subseteq EXP$
    - 后者：EXP是能在exponential时间中被求解的问题。Enumerate all certificates to solve the problem. The number of all certificates is $2^{P(n)} = 2^{n^{O(1)}}$.
  - $NP = \bigcup_{c\geq 1} NTIME[n^c] = NTIME[n^{O(1)}]$， 即NP = 在多项式时间内可由非确定性图灵机接受的语言集合
    - $NTIME[T(n)]$意思是在某个多项式时间内被 NTM 接受的语言的集合
    - 引入Nondeterministic TM:
      - 给定当前状态和符号，机器可以有多个“合法的”转移规则, the power set of $Q \times \Gamma \times \{L, R, S\}$。只要一条路径走通了，就说accept this input. Running time: 所有路径中用时最长的那个
      - NP的每一个certificate就是NTM中一个不同的transition function的集合
    - 证明：分别证两者为对方的subset
    - “在多项式时间内可由非确定性图灵机接受”，意味着 NP 问题可以快速验证某个“猜的解”是否成立，但不保证你能快速找到这个解。
- Reduction
  - Intuition: if $A \leq B$, or A is reducible to B, then if B can be solved efficiently, that algorithm can be used as a subroutine to solve A efficiently.
  - Example
    - Many-one reduction: $L_1 \leq_m L_2$
      - $L_1, L_2 \subseteq \{0,1\}^*$。 there is a computable function把多个（many）不同的输入（来自问题 1）可以都被归约成一个（one）目标问题2的实例。
      - Properties： $L \leq_m L$, cuz $\phi(x) = x$; $L_1 \leq_m L_2 \iff$ 前者的补集m$\leq_m$后者的补集；传递性
    - Polynomial-time many-one reduction(Karp reduction): $L_1 \leq_P L_2$
      - say L_1 is Karp reducible to L_2, $L_1 \leq_P L_2$ if there is a poly-time computable function $\phi(x)$.
    - Turing reduction: $L_1 \leq_T L_2$
      - Def: if there is a oracle TM with an oracle for L_2 that decides L_1. 可以把问题1的求解过程，写成一个调用2的“子程序”来完成的算法，而且可以调用很多次，甚至根据上一次结果来决定下一步
    - Poly-time Turing reduction(Cook reduction): $L_1 \leq$
      - Def: a poly-time oracle TM
    - Log-space reduction
  - Oracle TM
    - A TM with an oracle for language L is a TM that has two additional kinds of states: query states Q_{query} and response states Q_{repone}. There is an

extra tape called the oracle tape. When it enters a query state, the following are performed in one step.[11]

- String z that is written on the oracle tape is erased. If $\in L$, then 1 is written on the leftmost cell. Otherwise, 0 is written. The oracle tape head is moved to the leftmost cell. And the machine enters a response state.

- NP-hard and NP-complete
  - Def: Language L is NP-hard if for any $K \in NP$, $K \leq_p L$. L is NP-complete if $L \in NP$and L is NP-hard. L 至少和所有 NP 问题一样难，甚至可能更难（L本身未必在 NP 中，甚至不可验证）
    - 直观理解：想象 NP 是一个班级，班里的每个同学都是一个问题。NP-complete：是这个班里最难的那几个问题，能代表全班的难度。如果你能解出他们，说明你能解全班。NP-hard：可能是另一个年级的更难问题（比如有些 NP-hard 问题甚至连验证解对不对都做不到），但班上的每个问题都可以转化成它。
    - 关系



  - Cook-Levin theorem
    - SAT[12] is NP-complete. 这是第一个被证明NP的问题
    - 核心：给定一个布尔公式，问：是否存在一种变量赋值，使得整个公式为真?
    - 组件：variable $\in \{True, Fle\}$; literal, a variable or its negation; clause, disjunction(OR) of literals; formula, conjunction(AND) of clauses.

- 证明这个是NP-complete之后，其他就简单了，reduce到这上面。整体思

  🎯 **场景：我们要证明语言 A 是不可判定的**

  1. **选一个已知不可判定的问题**，如：停机问题 $A_{TM} = \{\langle M, w\rangle \mid M \text{ 接受 } w\}$
  2. **构造一个函数 f**，使得：

     $$\forall x, x \in A_{TM} \iff f(x) \in B$$

     换句话说，x 是否属于 A 的信息，被保留在 f(x) 是否属于 B 的信息里。
  3. **说明 f 是可计算的**（即存在图灵机可实现 f）
  4. **结论**：
     如果 B 是判定的，那么 A 就是判定的。
     但 A 不可判定 ⇒ B 也不可判定。

  路是
  - SAT $\le_p$ 3SAT. 3SAT means each clause has $\le$ literals. 证明，思想是把wide clause转化为short. 任何一个 SAT 问题，都可以在多项式时间内转化为一个 3SAT 问题，这说明3SAT 至少和 SAT 一样难
  - 3SAT $\le_P$ INTEGERPROG.
  - 3SAT$\le$ INDSET.
    $INDSET = \{< G, k >$: G has an independent set of sie $\}$. 把formula转化为graph
- Landau Asymptotic Notation
  - **Big O notation**
    - $f, : R \mapsto R, f(x) = O((x))$ if $c > 0, N, x \ge N,$ 有$f(x) \le c(x)$。即，要求在x无限大时，f的增长不超过某个常数倍的g[13]。
      - $(x) = 1$时，也就是有$O(1)$。就像取出第一本这个操作和书架上有多少本书无关，函数最终不会增长超过某个固定上限

- 例子：Think geometrically, Prove algebraically

Ex 4 $1+\frac{1}{2}+\cdots+\frac{1}{n}=\ln n+O(1)$

Proof The result is equivalent to

$$1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n=O(1)$$

So we only need to prove

$$0<1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n\le 1$$

3

As is known $\int \frac{1}{x}dx=\ln x+C$, so $\int_1^n \frac{1}{x}dx=\ln n$.

$$\begin{aligned}1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n&=1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^n\frac{1}{x}dx\\&=1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^2\frac{1}{x}dx-\int_1^2\frac{1}{x}dx-\cdots-\int_1^2\frac{1}{x}dx\\&=1+(\frac{1}{2}-\int_1^2\frac{1}{x}dx)(\le 0)+(\frac{1}{3}-\int_2^3\frac{1}{x}dx)(\le 0)+\cdots+(\frac{1}{n}-\int_{n-1}^n\frac{1}{x}dx)(\le 0)\\&\le 1\end{aligned}$$

$$\begin{aligned}1+\frac{1}{2}+\cdots+\frac{1}{n}-\ln n&=1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^n\frac{1}{x}dx\\&=1+\frac{1}{2}+\cdots+\frac{1}{n}-\int_1^2\frac{1}{x}dx-\int_1^2\frac{1}{x}dx-\cdots-\int_1^2\frac{1}{x}dx\\&=(1-\int_1^2\frac{1}{x}dx)(>0)+(\frac{1}{2}-\int_2^3\frac{1}{x}dx)(>0)+\cdots+(\frac{1}{n-1}-\int_{n-1}^n\frac{1}{x}dx)(>0)+\frac{1}{n}\\&\ge\frac{1}{n}>0\end{aligned}$$

- 性质
  - $f_1=O(_1),\ f_2=O(_2)\ \ f_1f_2=O(_{12})$
  - $f\ \ O()=O(f)$
  - $f_1=O(_1),\ f_2=O(_2)\ \ f_1\ \ f_2=O(\mathrm{ma}(_1,_2))$
  - $f=O(),\ k\in\ \ k\ \ f=O()$
- 一个函数能有很多个O，实际中通常关注最紧的Big-O上界，称渐进最优复杂度
- 衍生
  - $f(x)=O_{x\to}((x))\quad \text{if}\quad (c>0)(\delta>0)(x\in[\ \ \delta,\ \ \delta])(f(x)\le c(x))$
  - $f(x,y)=O_y((x))$ if $(y)(c>0)(N)(x\ge N)(f(x,y)\le C(x))$
    - 例子：证明是，有1 $_{1\delta}=O_\delta(1)$，大概就是这样

      $O(1)$ is an **absolute constant**, $O_\delta(1)$ is a **constant that depends on** δ.

      For example, for $\delta>0$ we have

      $$\frac{1}{1^{1+\delta}}+\frac{1}{2^{1+\delta}}+\cdots+\frac{1}{n^{1+\delta}}=O_\delta(1)$$
  - $f(x)=O((x))$ if $(C>0)(N)(x\ge N)(f(x)\le \mathrm{og}^C x\ \ (x))$
    - 意思是：复杂度大致是(x)，但可能带一些对数项$\mathrm{og}\,n$，我们先忽略它。因为n增大带来的logn增大很缓慢，可忽略。

        - 称为Polylogarithmic time，多重对数。polylog 被认为是“几乎可以忽略不计”的复杂度代价
- **Big Ω notation**
    - $f(x) \in ((x)) \iff C > 0,\ N,\ x \geq N,\ f(x) \geq C(x)$
    - 直觉：当x足够大时，f(x)至少像g(x)一样快地增长（甚至更快）
    - eg: Bubble sort，$O(n^2), (n)$
- **Big notation**
    - $f(x) \in ((x)) \iff C_1, C_2 > 0,\ N,\ x \geq N,\ C_1(x) \leq f(x) \leq C_2(x)$
    - 直觉：跟 g(x)增长速度差不多
    - 也就是说，既要Ω也要O，才有
- **Little o notation**
    - $f(x) \in o((x)) \iff \epsilon > 0,\ N,\ x \geq N,\ f(x) \leq \epsilon(x)$
    - 直觉：f(x)远远小于g(x)，最终连它的影子都追不上；而O只是“没超过”：可能压着跑，也可能完全等于。
- **Little notation**
    - $f(x) \in ((x)) \iff c > 0,\ N,\ x \geq N,\ f(x) \geq c(x)$
    - 直觉：严格下界
- 综合把握
    - Big-O 系列是“工程用的锯子”，Little-o 系列是“数学用的手术刀”。一个粗估趋势，一个精雕极限，各有用途，但前者显然更常见。

---

1. 返回一个状态集合的子集，也就是说，自动机从这个状态，在读入这个字符后，可能跳转到多个状态。↩
2. 机器执行逻辑的最小单位就是“状态”切换，理解正则语言，是机器最简单的一种“语言理解”能力。↩
3. A^*代表从语言A中取任意个串（可以是 0 个），首尾拼接得到的新语言 ↩
4. 对于这个例子：这个并集构造看似只是将状态从一维变成了二维（取笛卡尔积），但本质是用 DFA 的可组合性来并行模拟两个自动机的运行。每一步同时执行两个 DFA 的状态跳转。最终判断是否接受，只需考察它们是否至少有一个进入接受状态。这样的构造充分体现了 DFA 的封闭性与表达能力。让英雄对付英雄，让好汉对付好汉。↩
5. RL的数量等于DFA的数量，which是可数的。而可能的语言的数量是不可数的 ↩
6. 这个 y 部分就像一段“可泵送的液体”，可以被“抽出来再压进去” ↩
7. 标准TM中，不能向左边无限移动 ↩
8. 是固定长度吗、是遇到特殊符号为止吗、还是... ↩
9. Physically computed：指计算由通用物理系统（电子、光学、量子、机械等）承载与实现。Biologically computed：指计算由生物系统（神经系统、DNA、细胞等）承载与实现。但严格说，生物计算也是物理计算的一种 ↩
10. 就像整理书架，如果你在一开始就把书紧紧靠在一起排得满满的，那么每次要插入一本书就得挪动全部后面的书，代价极高。但如果你预留了一些空位（buffer），那么大多数插入操作就可以在局部完成。我们不马上进行移动，而是暂时把内容挪到缓冲区里去。只有当某个缓冲区“快被用满”时，我们才一次性把它“整理一下”——这个过程就叫 shift ↩
11. 这个“预言机”就像一个超级外部函数，你可以向它提问一个问题，它立刻告诉你答案，无需计算。↩

12. SAT means Boolean satisfiability problem ↩
13. 为什么只关注某个常数倍就行了？因为我们关注的是一个函数是以什么速度增长的，关注的是增长量级——常数可以忍，增长级别不能忍； ↩