CHAPTER 8

在研究复杂性的道路上，我们已取得丰硕的成果。一系列的语言类一个套一个，构成了层次关系：

$$L \subseteq NL = coNL \subseteq P \subseteq NP \subseteq NPSPACE = PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

然而，每一个 $\subseteq$ 是否是真包含关系，人们目前还不清楚。在这种意义下，所谓的层次不过是伪层次罢了。——万一「$\subseteq$」都是「$=$」，就无所谓层次了。

人们的直觉是：给定更多资源，就能解决更多问题。该直觉未必总是准确，例如，在多项式空间下，引入非确定性于事无补。但也许，它在多数情况下的确准确？我们在本章的首个议题，就是用数学证明这一直觉。

def 空间上易行的函数。

设 $f: \mathbb{N} \to \mathbb{N}$。如果存在一台空间复杂度为 $O(f(n))$ 的 TM M，使得对于 $\forall n \in \mathbb{N}$，只要给 M 输入长度为 n 的串，M 就能~~在 $O(f(n))$ 的空间内~~算出 $f(n)$ 的二进制表示，那么我们称 f 是空间上易行的。

该定义只是为了避免一些丑陋的 f。常见的函数诸如 $f(n) = n^k$ $(k \in \mathbb{N})$、$f(n) = n \log n$、$f(n) = 2^n$ 等等都是空间上易行的。作为习题，试证 $f(n) = \lfloor n^q \rfloor$ $(q \in \mathbb{Q}^+)$ 是空间上易行的。

类似地，不难验证上述函数也满足下面的定义。

def 时间上易行的函数。

设 $f: \mathbb{N} \to \mathbb{N}$。如果存在一台时间复杂度为 $O(f(n))$ 的 TM M，使得 $\forall n \in \mathbb{N}$，只要给 M 输入长为 n 的串，M 就能~~在 $O(f(n))$ 的时间内~~算出 $f(n)$ 的二进制表示，那么我们称 f 是时间上易行的。

下面便可以介绍著名的「层次定理」。

Theorem 1 (Space hierarchy)

设 f 是任何一个空间上易行的函数，那么对任何 $g=o(f)$，均有 $\mathrm{SPACE}(g(n)) \subsetneq \mathrm{SPACE}(f(n))$.

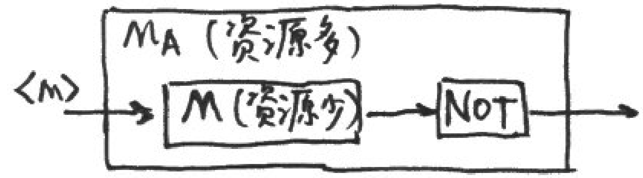
proof. 等价于证明：$\exists$ 语言 A，$A \in \mathrm{SPACE}(f(n))$ 但 $A \notin \mathrm{SPACE}(g(n))$.

亦即：$\exists$ 语言 A，A 可以被某个空间复杂度为 $S_{M_A}(n) = O(f(n))$ 的TM判定，但却无法被任何空间复杂度为 $O(g(n)) = o(f(n))$ 的TM判定。

我们下面就构造这么一个语言 A。与以前不同，A 很难用集合的符号写出，而要通过一台TM M_A 来定义。即 $A := \mathcal{L}(M_A)$

如何能够设计 M_A，使得那些凡是空间复杂度为 $o(f(n))$ 的TM M 都有 $\mathcal{L}(M) \neq \mathcal{L}(M_A) = A$ 呢？大体思路是这样的：既然 M_A 可以使用 $O(f(n))$ 的空间复杂度，而 M 则只有 $o(f(n))$ 的空间复杂度，所以一定能够用前者去模拟后者直至后者结束。如此一来，我们对结果取反即可。

至于那些空间复杂度为 $\Omega(f(n))$ 的TM，我们根本不关心。



因而我们设计出第一版的 M_A = "在输入 w 时，

1° 将 w 解读成TM的编码 $\langle M \rangle$.

2° 计算 $f(|w|) =: limit$.

3° 逐步模拟 M 在 w 上的运行，并监测 M 消耗的空间。如果空间超出了 limit，则直接拒绝（接纳亦可，因为这个 M 根本不是 $o(f(n))$ 的，$\mathcal{L}(M) = \mathcal{L}_A(M)$ 或 $\mathcal{L}(M) \neq \mathcal{L}_A(M)$ 都无所谓）

4° 若 M 接纳 w，则拒绝；否则，接纳。"

其中2°所花的空间是 $O(f(n))$，3°所花的空间也是 $O(f(n))$，因此 M_A 的空间复杂度 $S_{M_A}(n)=O(f(n))$。

可惜的是这个 M_A 的设计有瑕疵。首先，它有可能会迷途（因为输入的 $\langle M\rangle$ 未必是一台判定器，它极可能在 limit 空间内打转）；其次，步骤3°中说「空间超出 $limit=f(n)$ 则直接拒绝」，但万一 M 的空间复杂度为 $10^9\log f(n)$ 呢？若 $n=|\langle M\rangle|$ 很小，那么 $10^9\log f(n)\gg limit$，故 M_A 会错把 M 当作 $\Omega(f(n))$ 的 TM，从而错失了执行步骤4°的机会。要是真的那么巧，$L(M)=L(M_A)$，那么便存在空间复杂度为 $o(f(n))$ 的 TM M 使 $L(M)=L(M_A)$，构造就失败了。

为了修复这两个问题，我们把 M_A 修改为 M_A :=“输入 w 时，

1° 找出 w 右侧的首个1，并将其左侧解读成某台TM的编码 $\langle M\rangle$。也就是 $w=\langle M\rangle 10\cdots0$

2° $n:=|w|$，$limit:=f(n)$

3° 逐步模拟 M 在 w 上的运行。如果下列情况之一出现，则立即拒绝（接纳亦可）。

(1) M 运行了超过 2^{limit} 步。

(2) M 使用了超出 limit 的空间。

4° 若 M 接纳 w，则拒绝；否则，接纳。”

显然3°(1)是为了解决问题一所添加的。那么问题二是如何解决的呢？留意到我们处理输入的方式有所改良——对于一台给定的 M，无论向 M_A 输入 $\langle M\rangle 1$、$\langle M\rangle 10$ 还是 $\langle M\rangle 10^{100}$，$M_A$ 均会模拟 M 的运行，无形中增加了「与 M 相撞」的次数。只要 M 的空间复杂度 $S_M(n)=o(f(n))$，则必存在 $n_0\in\mathbb{N}$：$S_M(n_0)<f(n_0)$，于是，考虑 $w_0:=\langle M\rangle 10^{n_0-|\langle M\rangle|-1}$，

M_A 在输入 w_0 时必能运行到步骤 4°，从而 $w_0 \in \mathcal{L}(M_A)$ 与 $w_0 \in \mathcal{L}(M)$ 有且仅有一个成立，因此 $\mathcal{L}(M_A) \neq \mathcal{L}(M)$。这也就说明，凡是空间复杂度为 $o(f(n))$ 的 M，均不可能判定 $A := \mathcal{L}(M_A)$. ■

remark. 这证明与 A_{TM} 不可判定的证明异曲同工。本质上，都是利用了「超能力」去俯视不具备超能力的「芸芸众生」。打个比方，这就像猜拳游戏中，看见别人出了剪刀，自己才出石头一样。

Theorem 2 (Time hierarchy)
设 f 是任何一个时间上易行的函数，那么对任何 $g(n) = o(f(n)/\log n)$，均有
$TIME(g(n)) \subset TIME(f(n))$.

proof 等价于证明：∃ 语言 A，A 可由某个时间复杂度为 $T_{M_A}(n) = O(f(n))$ 的 TM 判定，但却无法由任何时间复杂度为 $O(g(n)) = o(f(n)/\log n)$ 的 TM 判定。

证明思路与 Theorem 1 极其相似。构造 $A := \mathcal{L}(M_A)$，而 $M_A :=$ "输入 w 时，

1° 找出 w 右侧的首个 1，并将左侧解读成某台 TM 的编码 $\langle M \rangle$。也就是 $w = \langle M \rangle 10\cdots0$.

2° $n := |w|$， $limit := f(n)/\log n$

3° 逐步模拟 M 在 w 上的运行。若 M 运行时间超过 limit，则拒绝（接纳亦可）

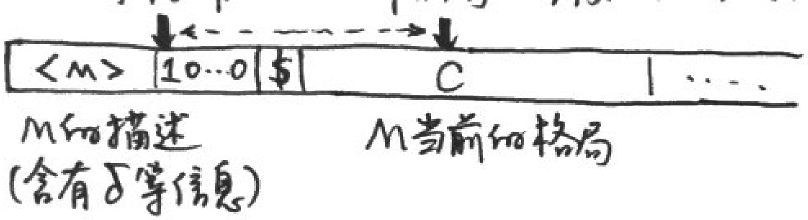
4° 若 M 接纳 w，则拒绝；否则接纳。"

乍一看，「$f(n)/\log n$」令人摸不着头脑。其实，$\log n$ 的引入是因为 3° 的模拟有 $\log n$ 因子的开销。

M 每运行一步，M_A 需要耗费 $\log n$ 步来模拟（原因待会说明）。与之相对，M 每用一单元空间，M_A 只需耗费常数单元的空间即可模拟，是故空间层次定理不似时间层次定理这样，有 $\log n$ 的一道鸿沟。

现在我们说明 $\log n$ 因子的开销从何而来。回忆我们说的「模拟」是什么意思——模拟器在自己的存储器上记录下 M 当前的格局（包含状态、读写头位置，以及存储器内容），移动自己的读写头以找出 M 即将读取的内容 a，以及 M 的当前状态 q。然后，寻找 $\langle M\rangle$ 中描述的 $\delta(q,a)$ 等于什么，再据此更新 M 的下一步格局。

这样一来，M_A 为了模拟 M 的行为，必须穿梭在 M 的格局以及 $\langle M\rangle$ 之间。



如果不加优化，那么在最坏情况下，M 每转移一步，M_A 都要移动至少 $\Theta(|c|) = \Theta(T_M(n))$ 步，这是相当低效的。

不过，如果我们动动脑筋，很容易想到优化方案：让 $\langle M\rangle$ 在存储器上「滑动」就好了。为此，我们把字符集 Γ 扩张，令

$$\hat{\Gamma} := \left\{ \binom{a}{b} \,\middle|\, a,b\in\Gamma \right\}$$

这相当于给存储器扩展了一条滑槽。我们把 $\langle M\rangle$ 放置于滑槽之中，让它跟随 M 的读写头滑动。优化过后，M 每转移一步，M_A 只需花费 $O(\log|\langle M\rangle|) = O(\log n)$ 步。于是，3° 中花费的时间为 $O(f(n)/\log n)\cdot O(\log n) = O(f(n))$。因此，$M_A$ 的时间复杂度就是 $O(f(n))$。

另一方面，$\forall$TM $M: T_M(n) = o(f(n)/\log n)$ 均 $\exists n_0: T_M(n_0) < f(n_0)/\log n_0$。考虑 $w_0 := \langle M\rangle 1 0^{n_0-|\langle M\rangle|-1}$，显然 $w_0\in\mathcal{L}(M_A)$ 与 $w_0\in\mathcal{L}(M)$ 之中有且仅有一个成立，故 $\mathcal{L}(M) \neq \mathcal{L}(M_A) = A$。 ■

由 Theorem 1,2，可以轻松导出如下的重要推论：

Corollary 3 $\forall r_1 < r_2 \in \mathbb{R}^+$，均有 $\mathrm{SPACE}(n^{r_1}) \subset \mathrm{SPACE}(n^{r_2})$，$\mathrm{TIME}(n^{r_1}) \subset \mathrm{TIME}(n^{r_2})$

Corollary 4 $P \subset EXPTIME$,
$PSPACE \subset EXPSPACE$.

你也许会说，层次定理证明中的构造太人为了，万一处在鸿沟之中的语言都是如此呢？那么层次定理岂不是缺乏实用价值？为了让我们心安，下面定义一门「自然的」语言，并简要说明它属于 EXPSPACE−PSPACE.

回忆正则表达式的组成：只允许 $\cup$、$\circ$、$*$ 三种运算符。现引入一个新运算符 $\uparrow$。$R\uparrow k$ 的含义等同于 $\underbrace{R\circ R\circ\cdots\circ R}_{k\text{次}}$. 我们称扩展后的正则表达式为「扩展正则式」。

定义语言 $ALL_{REX\uparrow} := \{ \langle R\rangle \mid R$ 是扩展正则式且 $\mathcal{L}(R)=\Sigma^* \}$。可以说，这是一门相当「自然的」、有实用价值的语言。

接下来，我们说明 $ALL_{REX\uparrow} \in EXPSPACE$。这不困难，以下的算法即可做到判定 $ALL_{REX\uparrow}$:

输入扩展正则式 $\langle R\rangle$.

1° 将 R 转成等价的正则表达式 R'

2° 由 R' 生成等价的 NFA N

3° 由 Chap.7 所讲的 ALL_{NFA} 判定器去判定 $N \in ALL_{NFA}$.

显然该算法只消耗指数空间。(习题)

最后，我们说明 $ALL_{REX\uparrow} \notin PSPACE$. 只须证明 $\forall A \in EXPSPACE$ 均有 $A \leq_p ALL_{REX\uparrow}$ 即可。(为什么？)

证明方法其实与 Cook-Levin Theorem 大同小异。$\forall A \in EXPSPACE$，均有一台 TM M 满足 $\mathcal{L}(M)=A$ 且 $S_M(n) = 2^{n^k}$ (k 是常数)。因此 M 的格局可被编码为长为 2^{n^k} 的串。

$w \in A \iff \exists$ 计算流程 $C_1,\cdots,C_\ell$，其中 C_1 是初始格局，C_ℓ 是接纳格局.

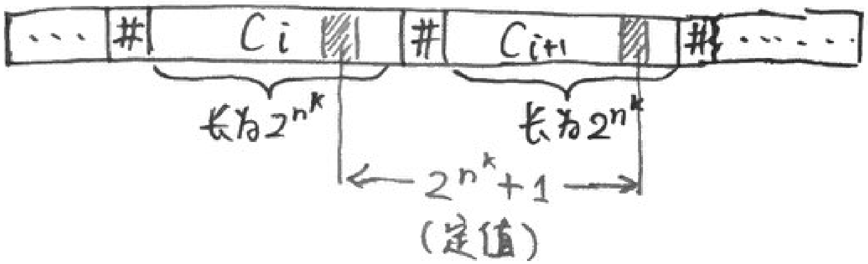
$\iff \{(C_1,\cdots,C_k) \mid k\in\mathbb{N}$, $C_1,\cdots,C_k$ 不构成合法的接纳流程$\} \neq \Sigma^*$

这启发我们构造扩展正则式 R，使得 $\mathcal{L}(R)$ 正是所有非法的接纳流程。

$R := R_1 \cup R_2 \cup R_3$

其中$L(R_1)$是所有初始格局不正确的串，
$L(R_2)$是所有推导关系不正确的串，
$L(R_3)$是所有不含q_{accept}的串。

R_1与R_3的构造很简单。至于R_2，则以下图作为提示。具体构造留作习题。



既然层次定理能够证明时间、空间类上严格的包含关系，那么可否遵循同样的思路，攻克P≟NP问题？

首先得弄清楚「同样的思路」指的是什么。在证明层次定理时，我们用「大机器」去模拟「小机器」，小机器每走一步，大机器跟着动几步，全过程是机械的、没有「动脑筋」的。待模拟结束，大机器再将小机器的结果取反，造成不对等。用一方去模拟另一方，是这思路的核心要素。我们即将揭示~~模拟~~它的局限性。

def 先知图灵机(OTM)及其计算.

设A是一门给定的语言。
一台关于A的OTM是一个七元组$(Q,\Sigma,\Gamma,\delta,q_0,q_a,q_r)$，其含义与通常的双带TM一样，除了$\delta: Q\times\Gamma^2 \to Q\times\Gamma^2\times\{L,R,O\}$比原来新增了一种可能O (oracle)。在计算过程中，若$\delta(q,a,b)=(q',c,d,O)$，则在下一格局中，二号存储器的读写头位置会被改写成0/1，具体是0还是1将由二号存储器上的串$\in A$来决定。

remark.

1° A是任意选取的；它甚至不必是TM可识别的语言

2° 形象地说，所谓的「O」操作可理解成OTM向某位先知求助，请他解答「二号存储器上的串是否属于A」这个问题。先知慷慨地在指针头位置写下了答案。

至于这位先知是如何知晓答案的，则并不为OTM所关心。换言之，OTM把它认为困难的问题丢给了一位万能的朋友，并理所当然地取回了正确答案。

自然，语言A越是难判定，关于A的OTM所获实惠就越多，能力就越强。例如，$A=SAT$，那么关于A的OTM就能够在线性时间内判定一切给定的NP中的语言。但若A本身很容易判定，例如$A=\{0^n \mid n\in\mathbb{N}\}$，那么关于A的OTM则基本上没讨到什么好处，能力与普通TM差不多。

def P^A与NP^A语言类

$P^A :=$ 能在多项式时间内被某关于A的OTM判定的语言类

$NP^A :=$ 能在多项式时间内被某关于A的NOTM判定的语言类.

Lemma 5 对任意语言A，

(1) 若用模拟的方法证明了$P=NP$，则必有$P^A=NP^A$

(2) 若用模拟的方法证明了$P\subset NP$，则必有$P^A\subset NP^A$

proof.

(1) 若用模拟的方法证明了$P=NP$，也就是说，$\forall B\in NP$，$\exists$ NTM N 及 TM M，均在多项式时间内判定B，且M是通过高效地模拟N来判定B的。

现在考虑$\forall C\in NP^A$。据定义，$\exists$ NOTM N'在多项式时间内判定C。在运行期间，N'可能寻求了若干次先知的帮助。无论如何，我们总可以构造OTM M'，它依照M模拟N的方式去模拟N'，只是在N'

寻求先知帮助时，M' 也对应地寻求先知帮助。于是，M' 亦可在多项式时间内判定 C，从而 $C\in P^A$。是故，$P^A=NP^A$.

(2) 与 (1) 类似。■

下面的定理直接否定了以纯模拟方法研究 $P \overset{?}{=} NP$ 的可行性。

Theorem 6

(1) 存在一门语言 A_1 : $P^{A_1}\neq NP^{A_1}$

(2) 存在一门语言 A_2 : $P^{A_2} = NP^{A_2}$.

结合 Theorem 6, Lemma 5 可知，若用模拟方法证明了 $P=NP$ 或 $P\subset NP$，均会导致矛盾，于是模拟方法一定无助于解决 $P \overset{?}{=} NP$.

下面我们证明 Theorem 6。

proof. 先证 (2)，再证 (1)。

(2) 取 $A_2 = TQBF$. 我们有

$$NP^{TQBF}\subseteq NPSPACE = PSPACE \subseteq P^{TQBF}.$$

其中，第一个包含关系是因为：$\forall B\in NP^{TQBF}$，存在某关于 TQBF 的 NOTM 判定 B，而我们总可以把「先知」嵌入在 N 之中，把 N 化为 NTM。这个先知每次回答问题时仅需花费多项式空间（因为 $TQBF\in PSPACE$），所以修改后的 N 也仅需花费多项式空间，故 $B\in PSPACE$。

第二个包含关系是因为 $TQBF\in P^{TQBF}$，而 TQBF 本身又是 PSPACE 完备的，故 $\forall B\in PSPACE$ 均有 $B\in P^{TQBF}$。

(1) A_1 的构造相较之下便比较奇特。我们无法具体写出 A_1 的形式，甚至也无法找出一台判定/识别 A_1 的 TM，但我们却能确证满足定理的 A_1 是存在的。

在构造 A_1 以前，先作一个定义。设 A 是任何给定的语言，定义

$$\hat{A} := \{w\in\Sigma^* \mid \exists w'\in A : |w|=|w'|\}.$$

例如 $A=\{0, 1, 100, 010\}$，那么 $\hat{A}$ 就是全体长度为 1、3、4 的串。

显然无论 A 取什么，总有 $\hat{A} \in NP^A$（留作思考）。这给了我们极大的自由。从现在起，可以为所欲为地构造 A_1 并想方设法令 $\hat{A}_1 \notin P^{A_1}$。若成功了，则直接推得 $P^{A_1} \neq NP^{A_1}$。

构造伊始，我们先固定一个全体多项式时间 OTM 的枚举：$M_1, M_2, \cdots$。值得注意的是，无论「先知」回答的是关于 L_1 的问题，还是~~关~~关于别的什么 L_2 的问题，OTM 的描述并不理会之 —— 机器只是在执行「O 命令」而已。「先知」仅在运行时介入，以决定 OTM 的计算流程，但他与机器本身的描述没有任何关系。是故，尽管我们尚未指出 $M_1, M_2, \cdots$ 是关于什么语言的 OTM，这个枚举总归是存在的，且与具体什么语言没有关系。

不失一般性，假定 $\Gamma=\{0,1\}$，且 M_i 的运行时间为 $T_i(n) \leq n^i$。作为上述准备以后，我们归纳地构造 A_1。每一步中，我们往 A_1 中添加有限个元素。

初始值 $A_1=\emptyset$， $m_0 := -1$

归纳假设 假定我们已完成了前 $i-1$ 步归纳，且 $\forall w: |w| \leq m_{i-1}$，$w \in A_1$ 的问题已经得到解决，而其余串在 A_1 中的归属问题亟待解决。

归纳步骤 现在我们开始第 i 步归纳。选取 $n_i \in \mathbb{N}$ 且 $n_i > m_{i-1}$ 且 $n_i^i < 2^{n_i}$。然后，给 M_i 输入字符串 0^{n_i} 并观察其行为（注意：我们是在数学上、思想上模拟其行为，是故无需花费任何时间）。每当 M_i 询问 $w \in A$ 时，无非两种情形：

① $|w| \leq m_{i-1}$。那么 $w \in A_1$ 是前 $i-1$ 步归纳中已经指定好的。因此先知依原样回答。

② $|w| > m_{i-1}$。那么 $w \in A_1$ 尚未解决。这时，我们令 $w \notin A_1$ 并依此报告 M_i。换句话说，我们按照 M_i 的行为来动态地

分配了那些被其询问的串之归属。

待到M_i计算结束，我们可以得到其计算结果。注意$T_i(n) \le n_i^i < 2^{n_i}$，因此，即便$M_i$每一步都在寻求先知协助，它也无法询问遍所有长度为$n_i$的串，其中必定有漏网之鱼。正是这些漏网之鱼给了我们可乘之机 —— 如果M_i接纳0^{n_i}，我们便令全体漏网之鱼$\notin A_1$；否则，便令全体漏网之鱼$\in A_1$。最后，令$m_i = n_i^i$，则可确保M_i不可能询问过长度$> m_i$的串。如果目前仍有$w \in \Sigma^*: |w| \le m_i$未定归属，则一律令$w \notin A_1$。至此，构造结束。

我们指出：$\forall M_i$，$\mathcal{L}(M_i) \neq \hat{A}_1$。这是因为构造$A_1$的时候保证 M_i接纳$0^{n_i} \Leftrightarrow \forall w \in \Sigma^*: |w| = n_i$ 均有$w \notin A_1 \Rightarrow 0^{n_i} \notin \hat{A}_1 \Rightarrow \mathcal{L}(M_i) \neq \hat{A}_1$。

于是，不存在多项式时间的OTM能判定$\hat{A}_1$，即$\hat{A}_1 \notin P^{A_1}$，证明完成。■

remark. 这个证明比层次定理的证明更「耍赖」，在不违反规则的情形下跟别人对着干。一定要理解清楚 OTM 描述与语言A_1的无关性，否则$M_1, M_2, \cdots$这个枚举就没有意义了。可举一现实类比：OTM的描述就好比库函数接口，函数名是固定不变的，但函数的功能与实现却是可以替换的。无论函数内部如何改变，调用它的程序代码无需改变。