CHAPTER 5

在上一章，我们已证明 A_{TM} 不可判定。E_{TM} 与 EQ_{TM} 看起来难于 A_{TM}，应当亦不可判定；只是我们终究未给出证明。本章将介绍一种证明不可判定性的有力工具：归约。借其之力，我们得以证明包括 E_{TM} 与 EQ_{TM} 在内的诸多语言皆不可判定。

归约的思想很简单：

$\boxed{语言A可判定 \iff \exists 判定器M: \mathcal{L}(M)=A}$

?

$\boxed{语言B可判定 \iff \exists 判定器N: \mathcal{L}(N)=B}$

如果我们能借助 N 的力量构造出某个 $M: \mathcal{L}(M)=A$，那么便有 B可判定 $\Rightarrow$ A可判定，且 A不可判定 $\Rightarrow$ B不可判定，记作 $A \leq B$。不难验证这里定义的"$\leq$"是一种二元序关系。数学化地表达即

$$\leq \;:= \{(A,B) \in 全体语言类^2 \mid 能够通过 B 的判定器 N 来构造 A 的判定器 M\}$$

注意 B 未必真的有判定器。在"$\leq$"的定义中，我们在假设 N 已知的前提下判断能否构造 M。

Proposition 1 设 $A \leq B$。若 B 可判定，则 A 可判定。若 A 不可判定，则 B 不可判定。

对我们来说，因为已知 A_{TM} 不可判定，所以若想证 B 不可判定，只须证 $A_{TM} \leq B$ 就够了。

Theorem 2 设 $HALT := \{\langle M, w\rangle \mid M 是 TM 且 M 在输入 w 时能停下(即接纳或拒绝)\}$。HALT 不可判定。

proof. 只需证 $A_{TM} \leq HALT$。

设判定器 $N: \mathcal{L}(N) = HALT$。我们

构造 D："在输入 $\langle M, w\rangle$ 时，

1° 先调用 $N(\langle M, w\rangle)$，判定 M 是否会在输入 w 时停下。若是，则继续；否则，拒绝。

2° 模拟 M 在输入 w 时的行为。若 M 接纳，则接纳；若 M 拒绝，则拒绝。"

不难看出，D 先借助 N 的超能力「消除」了 M 在 w 上迷途的情形，从而在第 2° 步时便能高枕无忧地等待 M 停下，因此 D 总不会迷途，$\mathcal{L}(D) = A$，从而 $A_{TM} \leq HALT$. ■

Theorem3. E_{TM} 不可判定。

proof. 只需证 $A_{TM} \leq E_{TM}$.

设判定器 N: $\mathcal{L}(N) = E_{TM}$。我们尝试构造 A_{TM} 的判定器 D："在输入 $\langle M, w\rangle$ 时，

1° 构造 TM M': $\mathcal{L}(M') = \mathcal{L}(M) \cap \{w\}$.

2° 调用 $N(\langle M'\rangle)$。若 N 接纳，则拒绝；否则接纳。" ■

remark. 第 1° 步总是能够做到的，因为第三章 Theorem1 给出了方法。第 2° 步的理由在于

$$N \text{ 接纳} \Leftrightarrow \mathcal{L}(M') = \emptyset \Leftrightarrow \mathcal{L}(M) \cap \{w\} = \emptyset \Leftrightarrow w \notin \mathcal{L}(M).$$

另外，由于 $\overline{E_{TM}}$ 是可识别的，故结合上一章 Theorem5 可推论：E_{TM} 不仅不可判定，而且不可识别。

Theorem4 EQ_{TM} 不可判定。

proof. 只需证 $E_{TM} \leq EQ_{TM}$.

设判定器 N: $\mathcal{L}(N) = EQ_{TM}$.

构造 E_{TM} 的判定器 D："在输入 $\langle M\rangle$ 时，调用 $N(\langle M, M'\rangle)$，其中 M' 是随便一个语言为空的 TM。若 N 接纳，则接纳，否则拒绝。" ■

remark. 证明中可见 E_{TM} 实是 EQ_{TM} 的特例——把两个参数中的一个取定为 M': $\mathcal{L}(M') = \emptyset$，则 EQ_{TM} 退化为 E_{TM}。

在归约的道路上我们简直势如破竹。可用一张图来小结我们刚才的工作：

$$
\begin{tikzcd}
\boxed{A_{TM}} \arrow[r, "\leq"] \arrow[d, "\leq"'] & \boxed{HALT} \\
\boxed{E_{TM}} \arrow[r, "\leq"] & \boxed{EQ_{TM}}
\end{tikzcd}
$$

而一切的不可判定性起点都源自 A_{TM} 之不可判定性。

在此再次澄清："$\leq$"是全体语言类上的二元关系，**无论 A_{TM} 是否真的不可判定**，"$A_{TM} \leq E_{TM} \leq EQ_{TM}$" 是确实成立的。这种关系链**辅以** A_{TM} 之不可判定性，方得链上语言之不可判定性。这一要点值得通过上面的证明再三揣摩。

这也便意味着，即便我们未知语言 A 是否可判定，去证明关系链 $A \leq B \leq C \leq \cdots$ 也是有意义的。它有如下两个好处：

1° 若在将来，有人证明了 A 的确不可判定，那么链上的其它语言便随之不可判定了。

2° 若在将来，有人证明了链的末尾可判定，那么链上的其它语言便随之可判定了。

作为巩固，我们再证一个定理。

Theorem 5 设 $REG_{TM} := \{ \langle M \rangle \mid M$ 是 TM 且 $\mathcal{L}(M)$ 是正则语言$\}$. REG_{TM} 不可判定。

proof. 只需证 $A_{TM} \leq REG_{TM}$.

设 N 为 REG_{TM} 的判定器。现构造 A_{TM} 的判定器 D："在输入 $\langle M, w \rangle$ 时，

1° 构造 TM M'：

$$\mathcal{L}(M') := \begin{cases} \Sigma^*, & 若 M 接纳 w \\ \emptyset, & 否则 \end{cases}$$

例如，M' 可以这么做：不论输入是什么，先二话不说地调用 $M(w)$。若 M 在 w 上迷途则 M' 自然迷途了；若 M 在 w 上拒绝，则 M' 拒绝一切输入；若 M 在 w 上接纳，则 M' 接纳一切输入。易知 $\mathcal{L}(M')$ 符合需求。

2° 构造M''：$\mathcal{L}(M'')=\{0^n1^n \mid n\in\mathbb{N}\}\cup\mathcal{L}(M')$.

3° 调用$N(\langle M''\rangle)$。若N接纳，则意味着$\mathcal{L}(M'')$是正则语言。但因$\{0^n1^n\}$不是正则语言，故唯有$\mathcal{L}(M')=\Sigma^*$，也就是说M必然接纳w，故令D接纳。

反之，若N拒绝，则意味着$\mathcal{L}(M'')$非正则，从而$\mathcal{L}(M')=\emptyset$，亦即$M'$不接纳$w$，故令$D$拒绝。” ■

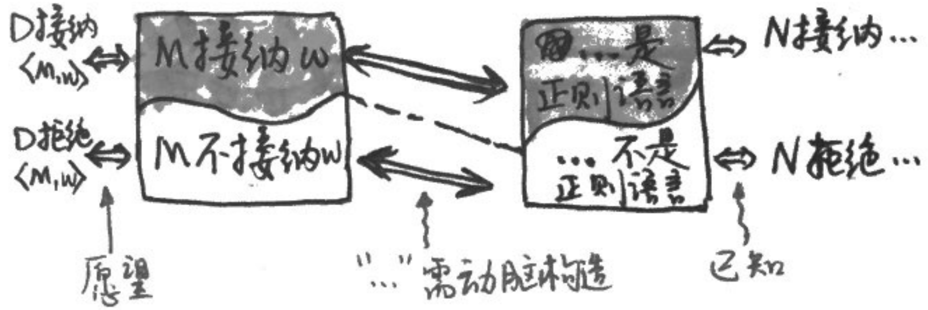
remark. 这个证明似乎有些绕，但一旦经梳理，就显得自然而然了：

$$
\begin{aligned}
D\text{接纳}\langle M,w\rangle &\iff N\text{接纳}\langle M''\rangle\\
&\iff \mathcal{L}(M'')\text{是正则语言}\\
&\iff \mathcal{L}(M')=\Sigma^*\\
&\iff M\text{接纳}w.
\end{aligned}
$$

在证明伊始，我们的目标是构造出D（判定器），使

D接纳$\langle M,w\rangle \iff M$接纳$w$

怎么做呢？唯有利用N。我们必须通过「M是否接纳w」的分野，去制造出「N是否接纳$f(M,w)$」的分野，亦即「$f(M,w)$是否为正则语言」的分野。我们构造的M'与M''，做的无非是划分出上述界限。

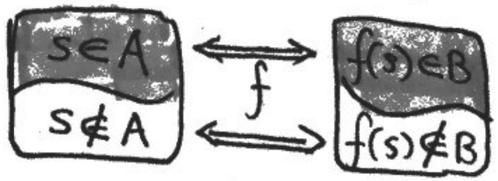


把这一思路规范下来，我们便有了如下引理。

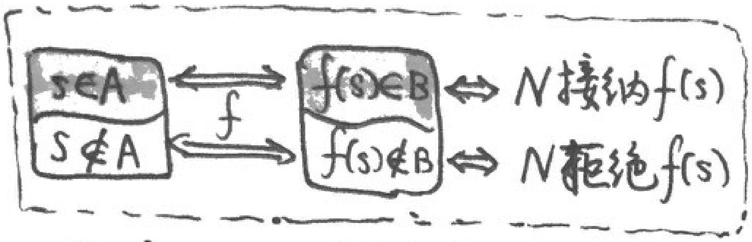
Lemma 6. 设A与B是两门语言。若存在一种TM可实现的变换$f:\Sigma^*\to\Sigma^*$，使得

$$s\in A \iff f(s)\in B$$

那么$A\le B$.（注意f未必是单射或满射）



proof.



设N是B的判定器，那么

$s\in A \Leftrightarrow f(s)\in B \Leftrightarrow N$接纳$f(s)$

从而可构造A的判定器D："输入s时，

1° 将s变换为$f(s)$

2° 调用$N(f(s))$，若N接纳$f(s)$，则接纳；若N拒绝$f(s)$，则拒绝。"

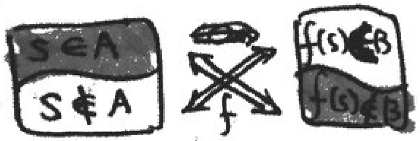
显然$\mathcal{L}(D)=A$且D永远不迷途。■

类似地，有兄弟引理

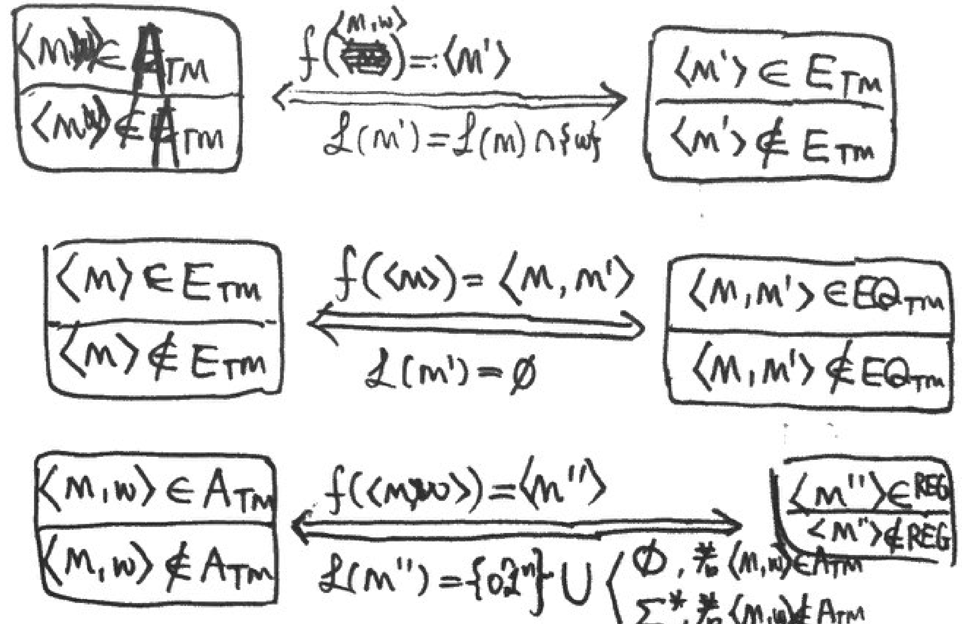
Lemma7 设A与B是两门语言。若存在一种TM可实现的变换$f: A\to B$，使得

$$s\in A \Leftrightarrow f(s)\notin B$$

那么$A\leq B$.



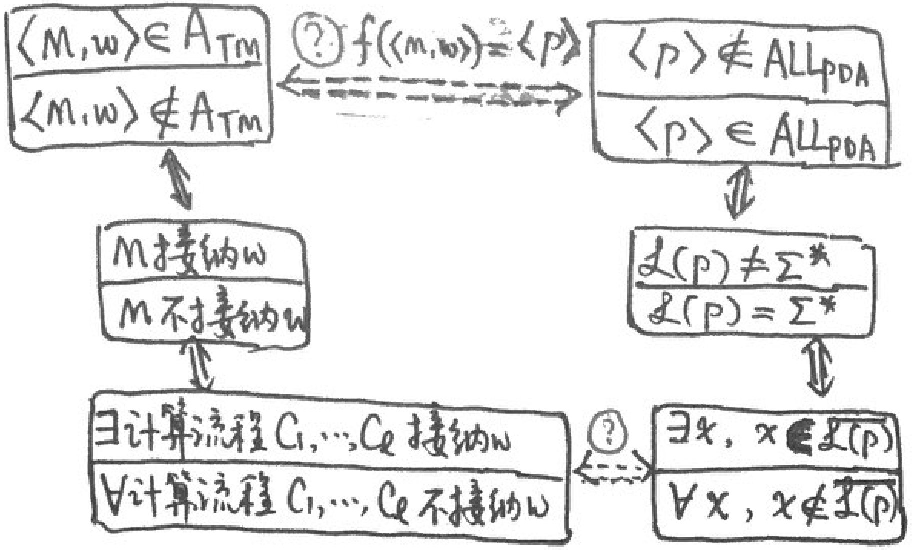
用Lemma 6,7的视角去考察Theorem 3-5的证明，会发现它们脱胎于同一个套路



认识到这个层面，以下的证明便不难理解了。

Theorem 8 设$ALL_{PDA} := \{\langle P\rangle \mid P$是PDA且$\mathcal{L}(P)=\Sigma^*\}$; ALL_{PDA}不可判定。

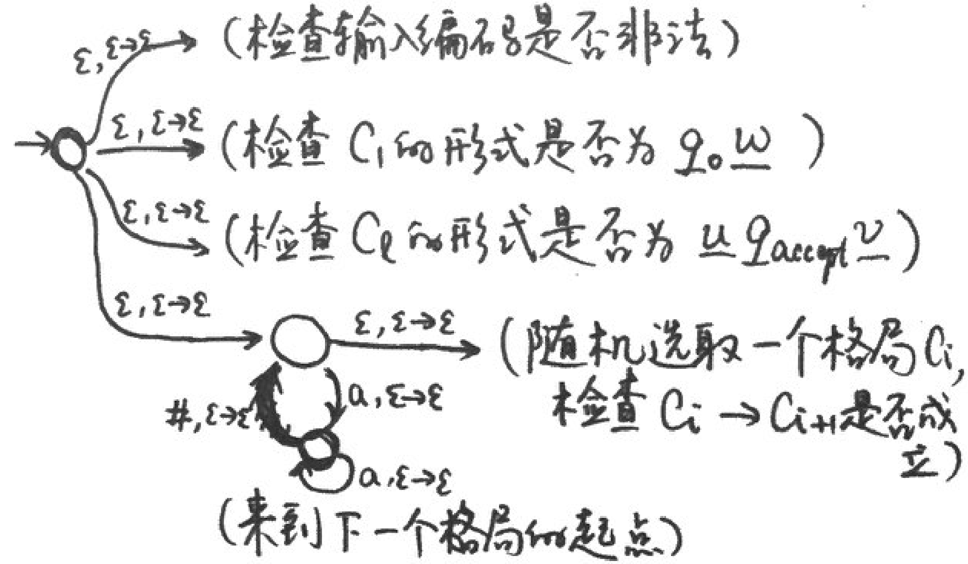
proof 只需证明$A_{TM}\leq ALL_{PDA}$。为此，我们先根据Lemma 6,7来分析思路。



这样一来，构造几乎顺理成章了。如果我们构造的 p 满足 $\overline{\mathcal{L}(p)} := \{\langle C_1,C_2,\cdots,C_\ell\rangle \mid C_1,\cdots,C_\ell$ 是 M 在 w 上的计算流程且 C_ℓ 状态为接纳$\}$，那么②处的双箭头就得以建立起来。下面的任务，便是证明上述 p 真的能被 TM「变换」出来。

欲使 $\overline{\mathcal{L}(p)} := \{\langle C_1,C_2,\cdots,C_\ell\rangle \mid C_1,\cdots,C_\ell$ 接纳 $w\}$，
也就是 $\mathcal{L}(p) := \{\langle C_1,C_2,\cdots,C_\ell\rangle \mid$ 编码非法，或 C_1 不是 M 的起始格局，或 $C_i \not\to C_{i+1}$，或 C_ℓ 不在 M 的接纳状态$\}$

P 可构造如下：



具体的实现是简单的，留作习题。唯一值得提示的一点是：为了能顺利比对 C_i 与 C_{i+1}，编码格局时应当采取

“C_1 # C_2^R # C_3 # …”

的格式，以便适应栈操作。■

remark. P 能被成功构造的原因在于

1°. P 的不确定性可用于「一票通过」.

2°. C_i 与 C_{i+1} 若满足 $C_i \to C_{i+1}$，则至多仅有两个位置的差别，故比较仅限局部。

problem

我们已知道 E_{PDA} 是可判定的。ALL_{PDA} 与 E_{PDA} 如此相似，为何难度有很大差别？上面证明的方法套在 E_{PDA} 上，会在哪里卡住？

Theorem 9 EQ_{PDA} 不可判定.

proof. 与 Theorem 4 类似，$ALL_{PDA} \le EQ_{PDA}$.

因为 $\overline{EQ_{PDA}}$ 可识别（通过枚举所有输入并比对输出），故根据上一章 Theorem 5 推论，EQ_{PDA} 不可识别。

总结前面一系列的归约过程，我们得到了如下的树形结构：

$$\begin{tikzcd}
 & \boxed{HALT} & \\
 & \boxed{E_{TM}} \arrow[r, "\le"] & \boxed{EQ_{TM}} \\
\boxed{A_{TM}} \arrow[uur, bend left, "\le"] \arrow[ur, "\le"] \arrow[r, "\le"] \arrow[dr, bend right, "\le"'] & \boxed{REG_{TM}} & \\
 & \boxed{ALL_{PDA}} \arrow[r, "\le"] & \boxed{EQ_{PDA}}
\end{tikzcd}$$

关于上一章末尾所列表格的讨论，目前尚有一丝缺憾：虽已证明 EQ_{TM} 不可判定，但未能说明其不可识别。这是因为上一章 Theorem 5 在此派不上用场 —— $\overline{EQ_{TM}}$ 事实上也是不可识别的。怎么办呢？办法仍是归约。

~~Proposition 10 若 A ≤ B~~

def. $\preceq := \{(A,B) \in$ 全体语言类$^2 \mid$ 能够通过识别B的TM来构造识别A的TM$\}$

Proposition 10 设 $A \preceq B$。若B可识别，则A可识别。若A不可识别，则B不可识别。

Lemma 11 设A与B是两门语言。若存在一种TM可实现的变换 $f: \Sigma^* \to \Sigma^*$，使得 $s \in A \Leftrightarrow f(s) \in B$，那么 $A \preceq B$.

于是，欲证B不可识别，可以考虑以 $\overline{A_{TM}}$ 为起点，证明 $\overline{A_{TM}} \preceq B$.

Theorem12 EQ_{TM} 与 $\overline{EQ_{TM}}$ 均不可识别.

proof. 我们这里只证前者。后者留作习题。

欲证 EQ_{TM} 不可识别，只需证 $\overline{A_{TM}} \le EQ_{TM}$

用 Lemma11 的方法来分析。

$\langle M,w\rangle \in \overline{A_{TM}}$	$\Longleftarrow \overset{?}{=} \Longrightarrow$ $f(\langle M,w\rangle)=\langle M_1,M_2\rangle$	$\langle M_1,M_2\rangle \in EQ_{TM}$
$\langle M,w\rangle \notin \overline{A_{TM}}$		$\langle M_1,M_2\rangle \notin EQ_{TM}$

$\Updownarrow$

$w \notin \mathcal{L}(M)$
$w \in \mathcal{L}(M)$

借鉴以前的经验，很容易想到令 $M_2: \mathcal{L}(M_2)=\emptyset$,

$$M_1: \mathcal{L}(M_1)=\begin{cases}\emptyset, & 若\ w\notin \mathcal{L}(M)\\ \{w\}, & 若\ w\in \mathcal{L}(M)\end{cases}，那么$$

?处的等价关系必能成立。■

至此，关于A、E、EQ问题的讨论完满结束。作为本章收尾，我们来证明一个非常优美的定理。它的证明高度集中了Lemma 6,7,11的精髓。

Rice's Theorem. 对于任何状如

$\{\langle M\rangle \mid M是TM 且 \mathcal{L}(M)满足性质P\}$

的语言A（P是任意性质），且 $\exists M_1, M_2$ $\langle M_1\rangle \in A$，$\langle M_2\rangle \notin A$，那么A不可判定。

注意这里的P是抽象的，~~……~~任意的。不难想见，该定理具有相当的普适性。然而事实上，其证明方法只不过是Theorem5证明之延伸。

proof. 我们来证 $A_{TM} \le A$.

$\langle M,w\rangle \in A_{TM}$	$\Longleftarrow \overset{?}{=} \Longrightarrow$ $f(\langle M,w\rangle)=\langle M'\rangle$	$\langle M'\rangle \in A$
$\langle M,w\rangle \notin A_{TM}$		$\langle M'\rangle \notin A$
$\Updownarrow$		$\Updownarrow$
$w \in \mathcal{L}(M)$		$\mathcal{L}(M')$满足性质P
$w \notin \mathcal{L}(M)$		$\mathcal{L}(M')$不满足性质P

设N为A的判定器。又设 M_1 与 M_2: $\mathcal{L}(M_1)$满足性质P，$\mathcal{L}(M_2)$不满足性质P。我们分两种情况来构造 A_{TM} 的判定器 D.

case 1° $\mathcal{L}(M_1)\cup\mathcal{L}(M_2)$ 满足性质 P.

$\mathcal{D}$:= "输入 $\langle M, w\rangle$ 时,

(1) 构造 M': $\mathcal{L}(M') = \begin{cases} \mathcal{L}(M_1), & \text{若}M\text{接纳}w \\ \emptyset, & \text{否则} \end{cases}$

(2) 构造 M'': $\mathcal{L}(M'') = \mathcal{L}(M')\cup\mathcal{L}(M_2)$.

(3) 调用 $N(\langle M''\rangle)$。若 N 接纳, 则接纳; 若 N 拒绝, 则拒绝."

不难看出, $\mathcal{D}$ 接纳 $\langle M, w\rangle \Leftrightarrow$ N 接纳 $\langle M''\rangle$ $\Leftrightarrow \mathcal{L}(M'')$ 满足性质 P $\Leftrightarrow \mathcal{L}(M')\cup\mathcal{L}(M_2)$ 满足性质 P $\Leftrightarrow \mathcal{L}(M') = \mathcal{L}(M_1) \Leftrightarrow$ M 接纳 w。

Case 2° $\mathcal{L}(M_1)\cup\mathcal{L}(M_2)$ 不满足性质 P.

$\mathcal{D}$:= "输入 $\langle M, w\rangle$ 时,

(1) 构造 M': $\mathcal{L}(M') = \begin{cases} \mathcal{L}(M_2), & \text{若}M\text{接纳}w \\ \emptyset, & \text{否则} \end{cases}$

(2) 构造 M'': $\mathcal{L}(M'') = \mathcal{L}(M')\cup\mathcal{L}(M_1)$

(3) 调用 $N(\langle M''\rangle)$。若 N 接纳, 则拒绝; 若 N 拒绝, 则接纳."

论证同 Case 1°. ∎

我们甚至能更进一步, 将定理推广至多 TM 的情形:

Rice's Theorem (Generalization)

设 P 是任意关于 k 台 TM 语言的性质。

$A := \{\langle M_1, M_2, \cdots, M_k\rangle \mid \forall i, M_i \text{是TM}, \text{且} (\mathcal{L}(M_1), \mathcal{L}(M_2), \ldots, \mathcal{L}(M_k)) \in P\}$.

若 A 非空、非全, 那么 A 不可判定。

注意这里的 $P \subseteq (\Sigma^*)^k$, 它用集合的形式表达了 k 台 TM 语言的联合性质.

proof. 原则上可与前面一样, 分 2^k 种情形讨论, 但为了避免繁琐, 可不失一般性地假设 $(\emptyset, \emptyset, \cdots, \emptyset) \notin P$ (否则令 $P = \overline{P}$). 如此一来, 仅用一种情形即可完成证明。 ∎

APPENDIX.
Gödel Numbers

我们此前大量使用了诸如 $\langle M\rangle$ 的写法来表示 TM M 的编码。那么，这样写真的没有问题吗？任意TM真的都具有编码吗？本附录将探讨这个话题。

因为TM描述起来太过原始，不便于编码，所以我们先引入一个与TM等价的「高层次」模型，再来研究那个模型的编码。又由于二者是相互等价的，且可以方便地互转，故编码了后者也就编码了前者。

def 无限制寄存器模型(URM)：一台具有无限多个存储单元的机器，每个存储单元可存放一个自然数。每台URM还带有一段「程序」$P=(I_1, I_2, \cdots, I_n)$，其中$I_i$只能取以下四种指令之一：

1°. Zero(j)　[含义：将存储单元 j 清零]
2°. Inc(j)　[含义：将存储单元 j 自增 1]
3°. Copy(s, t)　[含义：将存储单元 s 的值赋给存储单元 t]
4°. Jump(j, k, t)　[含义：比较存储单元 j 与 k，若相等则跳转至 I_t]

(指令可类比TM中的 δ)。

def URM的格局：URM当前所有存储单元的内容及当前的指令编号。若存储单元内容为 $r_1, r_2, r_3, \cdots$ 且指令编号为 i，则表示为 $C = [i]\, r_1 r_2 r_3 \cdots$ (末尾的0可省略)。

仿照TM，我们也可给出URM计算流程的严格定义。这里我们就不再自寻烦恼，而只是用文字表达：

URM初始时格局为 $C = [1]\, x$ ← 输入

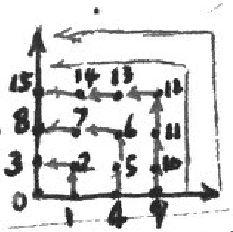
它依次读指令并依指令的含义来计算。若遇Jump(j, k, t)且 $r_j = r_k$，则下一步跳转至指令 I_t，而不是顺序执行。

不难看出，给定URM U，我们必能构造出一台TM M来模拟U的行为。反过来，给定一台TM M，我们也能找出与之等价的URM U。这说明TM与URM在表达能力上等价。（互相转换的细节留作思考）

紧接着我们能推论：若寻找到了一种编码URM的方案，我们也就变相地寻找到了一种编码TM的方案。现在，我们就来编码URM。

Lemma 1 存在双向可计算的双射 $f:\mathbb{N}_0^2\to\mathbb{N}_0$.

proof.



从(0,0)出发，依红色路径行进，给途中的点编号0, 1, 2, …

显然，如此构成了双射 $f:\mathbb{N}_0^2\to\mathbb{N}_0$.

下面说明f为何双向可计算。给定 $(i,j)\in\mathbb{N}_0$，我们有

$$f(i,j)=\begin{cases}(j+1)^2-i-1, & i<j\\ i^2+j, & i\geq j\end{cases}$$

反过来，给定 $n\in\mathbb{N}_0$，可由小及大枚举i，使得 $n=i^2+k$ $(k\leq 2i)$，那么

$$f^{-1}(n)=\begin{cases}(i,k), & k\leq i\\ (2i-k,\ i), & k>i\end{cases}$$ ■

Lemma 2 存在双向可计算的双射 $g:\mathbb{N}_0^*\to\mathbb{N}_0$.

proof. $\forall(i_1,i_2,\cdots,i_k)\in\mathbb{N}_0^*$，先反复调用$f$（共计$k-1$次）将其编码为$i$，再令 $g(i_1,i_2,\cdots,i_k):=f(k,i)$。■

Theorem 3 存在双向可计算的双射h，将任意URM映为一个自然数。

proof. h描述如下：

对任意URM U，设它的程序为$\{I_1,\cdots,I_n\}$，先将每条指令预编码。若I_i为

$$\left.\begin{array}{ll}1^\circ\ \text{Zero}(j)\text{，} & \text{则编成 } f(1,j)\\ 2^\circ\ \text{Inc}(j)\text{，} & \text{则编成 } f(2,j)\\ 3^\circ\ \text{Copy}(s,t)\text{，} & \text{则编成 } f(3,f(s,t))\\ 4^\circ\ \text{Jump}(j,k,t)\text{，} & \text{则编成 } f(4,f(j,f(k,t)))\end{array}\right\}=:I_i'$$

最后，再输出 $h(U):=g(I_1',I_2',\cdots,I_n')$. ■

至此，我们建立了URM（TM）与自然数间的一一对应。

APPENDIX.

Recursion Theorem & Logic Theories

本附录分两个部分。第一部分探讨TM「自我复制」的可能性；第二部分简述TM与逻辑推理、证明之间交融的可能性。它们均具有深邃的哲学意味。

「自我复制」在生物界屡见不鲜。细胞的复制（分裂）是在DNA/RNA指导下进行的，步骤大致归结如下：

1° DNA/RNA自身复制。（例如DNA，在酶的作用下解螺旋，然后A、G、C、T分别与外界的T、C、G、A配对，形成两条DNA螺旋）

2° DNA/RNA指导必要物质的合成。（例如蛋白质是由经DNA转录成的RNA和特定酶、氨基酸合成的）

3° DNA/RNA指导细胞的二分。

上述过程看似没有问题，但若追问「指导」是什么意思、DNA/RNA内部「烧录」的程序是否能独力支持指导工作，那么我们恐怕得陷入迷思。究竟有没有一种程序，或者说信息，能够在运行之后完整地复制自身？还是说，仅有程序（如DNA/RNA）是不够的，复制与繁殖必须仰赖超理性的、不可被规则描述的作用？

这样深刻的问题，在计算理论的框架下，能够得到简洁的解答：TM可以自我复制。为证明此，先作一点铺垫。

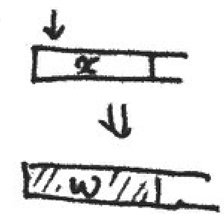
def $F: \Sigma^* \to \{M \mid M\text{是TM}\}$

$w \mapsto F_w$

其中 F_w :=“输入x时，

1° 抹去输入

2° 输出w”



例如 F_{110} = → ○ → 抹去输入 → 读写头返回最左侧 → 写"110"

注意"110"是嵌入在F_{110}内的。另外还请注意：虽然F_{110}的功能可以有多种实现方法，但是我们在此定义F_{110}——及其它F_w——就是上的那种实现。这样一来，当我们给了w，F_w就是一台唯一确定的TM。

下面我们就来构造一台能够自我复制的TM，取名为SELF。这里所说的「自我复制」，指的是SELF运行时无需任何输入，且运行结束后存储单元上的内容恰为$\langle SELF\rangle$.

SELF由两个子程序构成，分别名之为$SELF_1$与$SELF_2$。也就是：SELF :="

1° 运行$SELF_1$

2° 运行$SELF_2$"

其中，$SELF_1 := F_{\langle SELF_2\rangle}$（当然，只有写出了$SELF_2$才能写出$SELF_1$）

$SELF_2$:="输入x时，

(1) 在x后面生成$\langle F_x\rangle$

(2) 交换x与$\langle F_x\rangle$并重编码为$\langle F_x, x\rangle$。"

因为$SELF_2$的定义根本没有涉及$SELF_1$，所以它没有循环定义。在有了$SELF_2$以后，$SELF_1$的空缺也就补上了。

现在，把$SELF_1$、$SELF_2$联合起来看，

SELF = "

1° 运行$SELF_1 = F_{\langle SELF_2\rangle}$。结束时，存储器上的内容是$\langle SELF_2\rangle =: x$。

2° 运行$SELF_2$，也就是说

(1) 在$\langle SELF_2\rangle$后面生成$\langle F_{\langle SELF_2\rangle}\rangle = \langle SELF_1\rangle$

(2) 交换$\langle SELF_2\rangle$与$\langle SELF_1\rangle$，并重编码"

从而SELF结束时存储器上内容恰为$\langle SELF_1, SELF_2\rangle = \langle SELF\rangle$，也就是说SELF成功完成了自我复制。

简而言之，SELF1 把 SELF2 的编码输出到存储器上，而SELF2 则把能够生成这串编码的 TM（即SELF1）的编码输出到存储器上。关键在于：SELF2 并不直接引用SELF1（否则就循环定义了），而是借助于存储器上的信息间接推测出SELF1。

我们可以用任何编程语言来验证这一想法。例如C语言版本：

```
#include <stdio.h>
int main(){                                SELF1
    char x[500];
    sprintf(x, "......");  填入下半部分
------------------------------------------------
    printf("#include <stdio.h> \n int main(){ \n
     char x[500]; \n sprintf(x,\"");
    printf(x);  ← 关键（避免了循环引用）
                                           SELF2
    printf("\"); \n")
    printf(x);
    return 0;
}
```

但上面的程序有一个问题，那就是诸如\"、\n 等的转义字符之处理。事实上，要避免此类问题，唯有不使用转义字符，细节留作思考。无论如何，由转义字符引发的琐碎细节并非SELF构造的问题，而是程序设计语言语法的问题，因而可通过巧妙方法规避。

SELF的构造毋庸置疑地表明纯机械化的自我复制并非天方夜谭。只要规则设置得当，辅以一定量的资源协助，就能够一生万物。

然而，SELF终究是一个特殊的例子。对于其它有实际用途的TM，我们能否将其包装，使其具备同样的自我复制能力？答案是肯定的。事实上，你可能已经发现：若在SELF2末尾加入一些别的操作并不会影响SELF的本质，因此我们总可以将

实用的TM嵌于 $SELF_2$ 末尾，以实现其自我复制。
更一般地，我们有如下定理。

Recursion Theorem 设M是任意工具TM（即目的在于通过输入计算输出的TM）。那么必存在工具TM N，满足
$\forall w\in\Sigma^*$，$N(w)$的计算输出 $= M(\langle N\rangle, w)$的计算输出。

证明前，先给一个定义。（F的改版）

def. $G:\Sigma^*\to\{M\mid M\text{是TM}\}$
$x\mapsto G_x$
G_x :=“在输入后面附加 x”.
与 F_x 一样，G_x 是确定的.

接下来我们证明定理。

proof. 给定M，我们来构造合条件的N。
构造很大程度上参照SELF。
N:=“在输入w时，
1° 运行 N_1，
2° 运行 N_2.
3° 运行 M.”

其中 $N_1 := G_{\langle N_2, M\rangle}$
N_2 :=“输入w, y时，
(1) 生成 $\langle G_y\rangle$
(2) 输出 $\langle G_y, y\rangle$, w.”

那么，N=“输入w时，
1° 运行 $N_1 = G_{\langle N_2, M\rangle}$. 结束时存储上的值为 w, $\langle N_2, M\rangle$.
2° 运行 N_2，也就是
(1) 生成 $\langle G_{\langle N_2,M\rangle}\rangle = \langle N_1\rangle$
(2) 输出 $\langle N_1, N_2, M\rangle$, w
$= \langle N\rangle$, w
3° 运行 M. 输出当然是 $M(\langle N\rangle, w)$ 的计算输出。” ■

这个看似奇怪的定理该如何应用呢？以下是一些实例：

e.g. 我们希望构造一个能自我复制的TM，那么可令 $M(\langle T\rangle, w) := \langle T\rangle$.
由Recursion Theorem，$\exists N: N(w) = M(\langle N\rangle, w)$

$= \langle N\rangle$，于是N就是我们所找的TM。

e.g. 我们希望构造一个能将自身 q_0 相关的转移函数 $\delta(q_0,\cdot)$ 输出的TM，那么可令

$M(\langle T\rangle, w) :=$ "输出T中与 q_0 相关的转移函数 $\delta(q_0,\cdot)$"。

由 Recursion Theorem，$\exists N$: $N(w) = M(\langle N\rangle, w)$ = N中与 q_0 相关的转移函数 $\delta(q_0,\cdot)$.

e.g. 我们希望构造一个「病毒」TM，它不仅能复制自身，而且还能记录复制了多少次。

可令 $M(\langle T\rangle, w) :=$ "$\langle T\rangle, w+1$"。

由 Recursion Theorem，$\exists N$:

$N(w) =$ "$\langle N\rangle, w+1$"。N就是理想的病毒。

此外，Recursion Theorem还能用于证明 A_{TM} 等语言不可判定。示范如下：假设 A_{TM} 可判定，那么存在TM M："在输入 $(\langle T\rangle, w)$ 时，若 $\langle T\rangle$ 接纳w则拒绝，若 $\langle T\rangle$ 不接纳w则接纳"。由 Recursion Theorem，$\exists N$:

$N(w) = M(\langle N\rangle, w)$，产生矛盾 ↯

remark. 定理证明中，N的执行被放在最后，这与我们前面说的"在 $SELF_2$ 后面附加内容"实是一样的。

下面我们进入第二部分，探讨逻辑推理机械化的可能性。我们这里所说的逻辑，仅限于一阶谓词逻辑。详细的定义请参见数理逻辑教材；我们仅作简单提要：

- 论域D：逻辑公式中出现的变元(如 x, y, z …)的取值范围。
- 谓词（n元）：$D^n \to \{T, F\}$ 的映射
- 量词：$\exists$ 和 $\forall$.
- 约束变元：受量词约束的变元
  自由变元：未受量词约束的变元
- 陈述：仅含谓词常量、约束变元和常量的合式公式。

当我们写出诸如 $\forall x\, P(x,y)$ 的公式时，由于 y 不受约束，且 P 没有取定，所以真值是不可判断的。仅有将 y 约束住或取成常量，并将 P 取成特定的谓词，方形成一条陈述，可断真假。比如

$\exists y \forall x$ ~~[illegible]~~ $Equal(x,y)$,

$\forall x \forall y\ Greater(x,y)$,

$\forall x\ Divides(x,2)$

均为陈述。

现在我们把目光聚焦在论域 $D:=\mathbb{N}$ 上，并且定义两个3元谓词

$$+ : \mathbb{N}^3 \to \{T,F\}$$

$$+(a,b,c) = \begin{cases} T &, 若\ a+b=c \\ F &, 若\ a+b\neq c \end{cases}$$

$$\times : \mathbb{N}^3 \to \{T,F\}$$

$$\times(a,b,c) = \begin{cases} T &, 若\ a\times b=c \\ F &, 若\ a\times b\neq c. \end{cases}$$

我们记 $(\mathbb{N},+,\times)$ 为所有在论域 $\mathbb{N}$ 上、仅含谓词 $+$ 与 $\times$ 的陈述之集合。

比如下面两条陈述均在 $(\mathbb{N},+,\times)$ 中：

1° $\forall x \forall y \exists z\, [+(x,y,z) \wedge \times(z,y,z)]$

2° $\forall n \exists p \exists k \forall x \forall y \exists i\, [+(n,k,p) \wedge +(x,i,p) \wedge \neg\times(x,y,p)]$

(即 $\forall n$，$\exists p \geq n$，p 是素数)

这样写起来有些繁琐，所以我们可简写成

$\forall n \exists p \forall x \forall y\, [p \geq n \wedge x<p \wedge p = x\bar{y}]$

其中的"$>$"、"$<$"、"$=$"均是通过谓词 $+$、$\times$ 实现的。

类似地，也可把诸如哥德巴赫猜想、费马大定理等重要的数论命题形式化为 $(\mathbb{N},+,\times)$ 中的陈述。如此一来，我们当然有动力去研究明白是否有一种纯机械的过程能判定 $(\mathbb{N},+,\times)$ 中的陈述是真还是假。若能，那么哥德巴赫猜想就不会再困扰数学家了。

def $Th(\mathbb{N},+,\times) := \{\phi \in (\mathbb{N},+,\times) \mid \phi 是真的\}$

我们希望知道 $Th(\mathbb{N},+,\times)$ 可否判定。(注意：即便 $Th(\mathbb{N},+,\times)$ 不可判定，也不代表说其中的陈述不可证明。只是说，不存在通用算法来决断 $(\mathbb{N},+,\times)$ 中陈述之真假)

注意：

Theorem 1 (Church)
$Th(\mathbb{N},+,\times)$ 不可判定。

proof (sketch).
我们来证明 $A_{TM} \le Th(\mathbb{N},+,\times)$

$$\boxed{\begin{array}{c}\langle M,w\rangle \in A_{TM} \\ \hline \langle M,w\rangle \notin A_{TM}\end{array}} \xLeftrightarrow{\Phi(\langle M,w\rangle)=\phi} \boxed{\begin{array}{c}\phi \in Th(\mathbb{N},+,\times) \\ \hline \phi \notin Th(\mathbb{N},+,\times)\end{array}}$$

$$\phi = \Phi(\langle M,w\rangle) := \exists x\,[\cdots]$$

一个编码了计算流程的数

检查 x 是否代表 M 在 w 上的计算流程，且终态是否为 q_{accept}.

编码/解码的细节不多作讨论，这里只重点说明其思想。计算流程是一段有穷序列 $C_1, C_2, \cdots, C_n$，而其中每个 C_i 都可视作三元组 $C_i = \underline{u}\,q\,\underline{v}$，从而我们可以用上一附录中所讲方法对每个 C_i 各自编码，再统一合成一个大数 $x \in \mathbb{N}$。反过来，给定 x，希望解读出计算流程，那么无非是需要反复调用 f^{-1}（定义见上一附录 Lemma 1）说到底，便是去把 x 表成 $x = i^2 + k$ $(k \le 2i)$ 的形式，再去还原原象。这可利用 $\exists$、$\forall$ 来约束，故必能用 $(\mathbb{N},+,\times)$ 中的陈述来模拟解码流程。■

这一定理具有重大启示意义：数学家不要指望用机械化的方法解决一切 $(\mathbb{N},+,\times)$ 中陈述之真假问题。有些陈述天然就庇存着「神秘」。

当然，对于 Th(N, +, ×) 加以限定后所得子集是有机会能被判定的。这里我们介绍一个最为有趣的例子：Th(N, +)。尽管仅仅把「×」谓词去除了，但这足以使之成为了一门可判定的语言。

为了证明，先作一番准备。

Lemma 2 ~~[illegible]~~

令 $\Sigma=\left\{\begin{pmatrix}0\\0\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\end{pmatrix},\begin{pmatrix}0\\1\\0\end{pmatrix},\cdots,\begin{pmatrix}1\\1\\1\end{pmatrix}\right\}$，那么，存在一个 DFA D，在面对任意输入 $w\in\Sigma^*$ 时，D 接纳 w 当且仅当 w 的第一行 + w 的第二行 = w 的第三行。 ■

例如，D 接纳 $w=\begin{pmatrix}0\\0\\1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}$，但拒绝 $w'=\begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\1\\0\end{pmatrix}$。可以认为，$D$ 是一台「加法判定器」。

Lemma 2 证明起来很简单，留作习题。将其略加推广，令 $\Sigma=\left\{\begin{pmatrix}0\\ \vdots\\0\end{pmatrix},\cdots,\begin{pmatrix}1\\ \vdots\\1\end{pmatrix}\right\}$（每个元素均为 n 维向量），那么亦存在 DFA 能验证第 i 行 + 第 j 行 = 第 k 行。（i,j,k 事先给定）

读到这儿，我们眼前的道路已基本明晰：任何陈述 $\phi\in$ (N, +) 无非是由有限个变元、若干个「+」谓词、若干个「与或非」及量词组成。不考虑量词，那么依据 Lemma 2 的构造，我们已经有能力造出一台 DFA，去接纳所有使 ϕ 满足的变元组合。余下任务便是把量词考虑进来。下面定理证明的主线就在此。

Theorem 3 Th(N, +) 是可判定的。

proof. 任给 $\phi\in$ (N, +)，我们希望判定 ϕ 是否在 Th(N, +) 中。

设 $\phi=Q_1x_1Q_2x_2\cdots Q_\ell x_\ell\,\psi$

其中 $x_1,x_2,\cdots,x_\ell$ 是变元，

$Q_1,Q_2,\cdots,Q_\ell$ 是 $\exists$ 或 $\forall$ 量词，

ψ 是一个仅含有自由变元的公式。

令 $\phi_i := Q_i x_i Q_{i+1} x_{i+1} \cdots Q_l x_l \psi$

($i=1,2,\ldots,l+1$。显然 $\phi_{l+1}=\psi$，$\phi_1=\phi$)

下面我们逐步构造一系列的 ~~DFA~~ FA

$D_{l+1}, D_l, \ldots, D_1$，使得：

D_i 识别的语言 = 使 ϕ_i 为真的自由变元组合

例如 $\mathcal{L}(D_{l+1}) = \{\langle x_1,\ldots,x_l\rangle \mid \phi_{l+1}(x_1,\ldots,x_l)=T\}$

$\mathcal{L}(D_l) = \{\langle x_1,\ldots,x_{l-1}\rangle \mid \phi_l(x_1,\ldots,x_{l-1})=T\}$

注意 $\langle\cdots\rangle$ 表示 Lemma 2 中所使用的编码方案。

1° 基础情形（D_{l+1} 的构造）

这并不难。由 Lemma 2 的推广可构造出诸多 DFA，分别处理 $\phi_{l+1}=\psi$ 中的每个谓词子句，比如 $+(x_1,x_3,x_4)$ 与 $+(x_2,x_2,x_1)$ 等。然后，根据 ψ 中的逻辑联结词 $\neg$、$\wedge$、$\vee$，对应地用补、$\cap$、$\cup$ 去合成一个大 ~~D~~FA —— 不难看出它正好符合 D_{l+1} 的要求。

2° 归纳步骤（已有 D_{i+1}，构造 D_i）

已知 $\mathcal{L}(D_{i+1}) = \{\langle x_1,\ldots,x_i\rangle \mid \phi_{i+1}(x_1,\ldots,x_i)=T\}$

欲求 $\mathcal{L}(D_i) = \{\langle x_1,\ldots,x_{i-1}\rangle \mid \phi_i(x_1,\ldots,x_{i-1})=T\}$。

留意 D_i 与 D_{i+1} 的输入维度并不相同，

因为 $\Sigma(D_{i+1}) = \left\{ \begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}, \ldots, \begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix} \right\}$ （i 维向量集）

而 $\Sigma(D_i) = \left\{ \begin{pmatrix}0\\\vdots\\0\end{pmatrix}, \ldots, \begin{pmatrix}1\\\vdots\\1\end{pmatrix} \right\}$ （$i-1$ 维向量集）。

若想借用 D_{i+1} 来构造 D_i，必须想法弥补输入格式之差异。然而 ~~D~~FA 又不会栈，不可能边读输入边修改，故唯一的办法是改动 D_{i+1} 的转移函数。

(2.1) $Q_i = $ "$\exists$"

那么希望 $\mathcal{L}(D_i) = \{\langle x_1,\ldots,x_{i-1}\rangle \mid \exists x_i, \phi_{i+1}(x_1,\ldots,x_i) = T\}$

怎么把输入中没有的 x_i 给「补上」呢？靠「猜」。

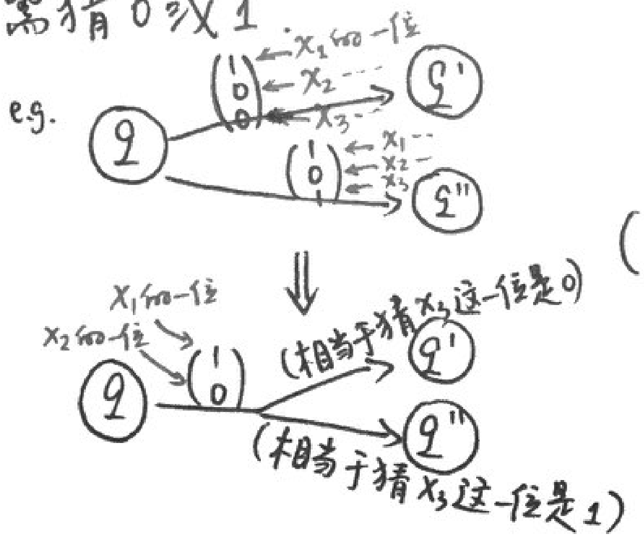
设 δ_{i+1} 为 D_{i+1} 的转移函数，δ_i 为 D_i 的转移函数。

设 $\delta_{i+1}(q, (b_1,b_2,\ldots,b_{i-1},0)^T) = q'$

$\delta_{i+1}(q, (b_1,b_2,\ldots,b_{i-1},1)^T) = q''$

则令 $\delta_i(q, (b_1,b_2,\ldots,b_{i-1})^T) := \{q', q''\}$

这么做的意义是：既然只要求 x_i 存在以使 ϕ_{i+1} 为真，那么不妨胡乱地去猜 x_i，利用不确定性去试这猜想是否令 ϕ_{i+1} 为真。又因输入是一位一位给出的，故在每一位上都需猜0或1



(i=3的例子)

不过这还没完 —— x_i 未必与输入同长度，所以应当在一开始「向输入补0」：

设 $\delta_{i+1}(q, (0,0,\dots,0,b_i)^T) = r$

则复制一份状态 q、r，名之为 $\hat{q}$，$\hat{r}$，

并令 $\delta_i(\hat{q}, \varepsilon) := \hat{r}$，

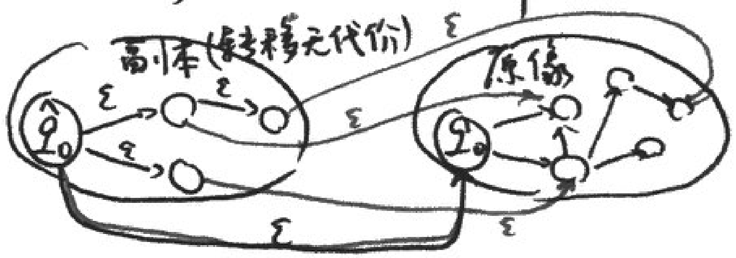
$\delta_i(\hat{q}, \varepsilon) := q$，

$\delta_i(\hat{r}, \varepsilon) := r$。

令 D_i 的初状态为 q_0，D_{i+1} 的初状态为 q_0

令 $\delta_i(q_0, \varepsilon) := \hat{q}_0$。

上面的描述也许有点迷惑。其意义是：把那些在 D_{i+1} 中能通过 $(0,0,\dots,0,b_i)^T$ 到达的状态拎出来，做成副本。D_i 首先在副本中遨游，不读输入的前提下仿佛给输入补了0，补完0再转移至原像中正常处理。



(2.2) Q_i = "$\forall$"

因为 $\phi_i = \forall x_i \phi_{i+1} = \neg\exists x_i \neg\phi_{i+1}$，所以可类似(2.1)处理。

3° 归纳结果

最终我们得以构造 D_1，使得

$\mathcal{L}(D_1) = \{\langle\,\rangle \mid \phi_1 = T\}$

也就是说，D_1 无须任何输入即可自动运行，若结果为接纳，则意味着 $\phi_1 = T$；否则 $\phi_1 = F$. 从而我们可由 D_1 的结果直接断言 ϕ 之真假。又因为前述构造均为机械化的，故存在一个能终止的 TM 实现这流程，从而断定 ϕ 之真假

$\Rightarrow$ $Th(\mathbb{N}, +)$ 可判定。 ■

Remark 我们之所以要在证明中用 FA，是因为它支持 ∩、∪、补操作。PDA 及 TM 对补操作不封闭，故在证明至 (2.2) 时会出差错。

对比 $Th(\mathbb{N}, +, \times)$ 与 $Th(\mathbb{N}, +)$，我们立即推论：×谓词是不能用+谓词表达的。