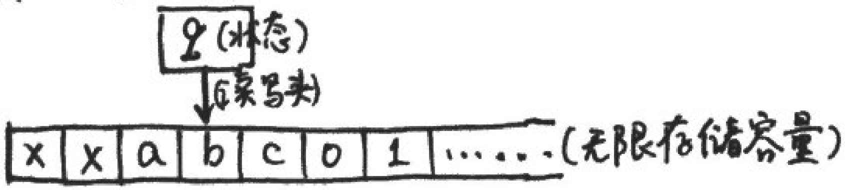
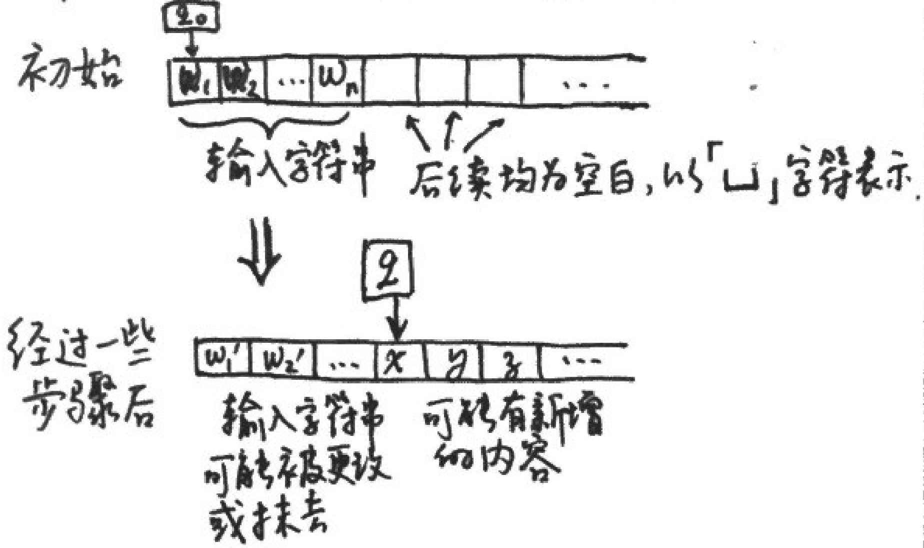
CHAPTER 3

DFA/NFA的局限，在于存储容量的缺乏；PDA（及DPDA）的局限，在于使用容量的手段受限。欲突破二者局限，势必要进一步放开约束，提供更自由的存储访问。图灵机(TM)正是沿着这一路径而提出的一种计算模型。

与PDA一样，TM有着不限量的存储空间。但是，它对存储的访问是任意的，既不限位置，也不限次数。机器上有一只读写头，指向当前操作的存储单元。它根据读到的信息以及当前的状态决定：(1)转移到什么状态；(2)把该单元改写成什么内容；(3)读写头往左还是往右移动一格。

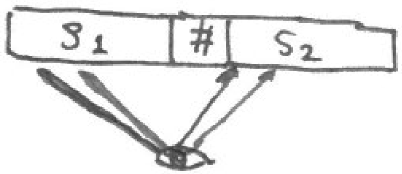


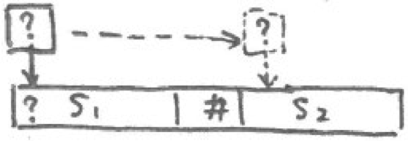
为了贯彻「访问不受限」的原则，TM的输入并不以字符流的形式给出，而是「固化」在存储器上，供机器无限次地、乱序地阅读和处理。事实上，这反倒简化了模型的表述——我们不用再像PDA那样区分输入流和栈内容。从这层意义上来说，TM是简洁优美的。



在精确定义图灵机以前，不妨先主观感受一下它的计算流程与人的计算流程何等相似。例如，在我们比较字符串S_1与S_2是否相同时(假定以"$S_1\#S_2$"的格式输入)，

人眼会折返于 S_1 与 S_2 之间，逐一比较其中字符是否匹配。TM也可以模拟此举，让读写头往返 S_1 与 S_2 之间，逐一判断字符是否匹配。





由此不难理解图灵机模型敏锐地捕捉到了人类作运算的步骤，是一种与现实高度一致的抽象。是故，在学者们各执一辞、流派纷杂的1936年，图灵机一经提出，便教诸如λ-演算、post System、递归函数等抽象数学模型自愧不如，以致很快便一统天下。

def 图灵机(TM)：

一个七元组 $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

1° Q是一个有限集合。它指定了该TM所有可能的状态。

2° Σ是一个有限集合。它指定了该TM的输入字符集。

3° $\Gamma \supseteq \Sigma \cup \{\sqcup\}$ 是一个有限集合，指定了该TM在存储单元内允许存放的字符集。⊔是空白字符。

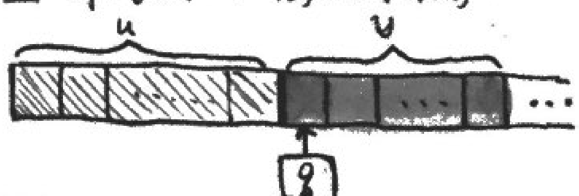
4° $\delta: Q\times\Gamma \to Q\times\Gamma\times\{L, R\}$. 它指定了该TM在各种状态下、读到各种字符时，应当转移至什么状态、把当前存储单元的内容改写成什么、将读写头朝左还是朝右移动一个单元。我们要求 $\delta(q_{accept}, \cdot) = (q_{accept}, \cdot, \cdot)$ 及 $\delta(q_{reject}, \cdot) = (q_{reject}, \cdot, \cdot)$.

5° $q_0 \in Q$ 是该TM的起始状态。

6°, 7° $q_{accept}, q_{reject} \in Q$ 分别是该TM的接纳状态和拒绝状态。$q_{accept} \neq q_{reject}$.

remark. 第4°条中的要求实质上是说：一旦进入接纳状态，则再也跳不出去；亦即该TM接纳后不得反悔。拒绝的情况同理。

def TM的格局：TM当前的状态、读写头位置及存储单元的全部内容。常用“uqv”来表示下面的格局：



def 格局的推导关系.

设TM $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$.
对于任意格局 $C=\underline{ua}\,q\,\underline{bv}$ $(u,v\in\Gamma^*,\ a,b\in\Gamma)$，
若 $\delta(q,b)=(q',b',L)$，则称格局 C 可推得格局 $C'=\underline{u}\,q'\,\underline{ab'v}$；若 $\delta(q,b)=(q',b',R)$，则称格局 C 可推得格局 $C'=\underline{uab'}\,q'\,\underline{v}$. 记作 $C\Rightarrow C'$.

remark. 上述定义并未涵盖 $C=q\,\underline{bv}$ 的特殊情况（即读写头处在最左侧的情况）。这时，我们将左移（L）视作无效。也就是说，若 $\delta(q,b)=(q',b',L)$，则称格局 C 可推得格局 $C'=q\,\underline{b'v}$。右移的情况无须特别处理。

remark. 我们之所以要额外定义所谓「格局」及其推导关系，是为了在后面定义计算流程时简洁明了。这里的「推导关系」或「$\Rightarrow$」，实质上规定了TM每一步究竟是如何运转的。

在前两章，我们没有介绍这样的概念，无非是因为DFA/NFA、PDA的「推导关系」能利用 $=$、$\in$ 之类的符号表达而已。

def. TM的计算流程.

设TM $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$.
对其输入字符串 $w=w_1w_2\cdots w_n\in\Sigma^*$.
若格局序列 $C_1,C_2,\cdots,C_m$ 满足

1° $C_1=q_0\,\underline{w}$.

2° $C_i\Rightarrow C_{i+1}$ $(\forall i\in\{1,2,\cdots,m-1\})$

3° $state(C_m)\in\{q_{accept},q_{reject}\}$

则称序列 $C_1,\cdots,C_m$ 是M对输入 w 的计算流程.

remark. 与DFA/NFA、PDA不同，TM并非对于任意输入串都存在（按上述定义的）计算流程。这是因为条件3°未必总可满足。对于一些TM和特殊的 w，TM有可能陷入死循环，从而无法在有限步内到达 q_{accept} 与 q_{reject} 之中的一个。这是一个有趣的现象，后面会详细考察之。

对于那些的确存在计算流程的串 w 来说，若忽略在 q_{accept} 或 q_{reject} 打转的冗杂，那么计算流程是唯一的。换言之，TM

的计算是确定的。

def TM的接纳、拒绝与迷途.

设M是一台TM，对其输入w，若存在计算流程 $C_1, C_2, \cdots, C_m$ 且 C_m 所处状态为 q_{accept}，则称M接纳w；若 C_m 所处状态为 q_{reject}，则称M拒绝w。否则（即不存在计算流程），称M在w下迷途。

乍一看，「迷途」这种状态与「拒绝」相似，为何不把它归于「拒绝」呢？其实，它们是有本质区别的两种状态，有必要差别对待。试考虑下面的例子。

e.g.1 假设我们有一台TM M，$L(M) = \{w \in \{0,1\}^* \mid w$ 中0与1个数相同$\}$。也就是说，凡是满足条件的w输入M中，总能够被M接纳；凡是不满足条件的w输入M中，总不会被M接纳。现在，我们给M输入w字符串，并等待一小时。假定M的每步推导均耗费0.1ms。一小时以后，我们观察M的状态，发现它既不处于 q_{accept} 也不处于 q_{reject}。此刻我们陷入了两难境地：已经算了这么久，弃之可惜，说不定再算一会儿便能接纳/拒绝；然则终究说不准它是否已迷途其中，若真的是，那么算下去也是白搭。

正是因为「迷途」的存在，我们对TM的进展不能完全自信。

e.g.2 假设我们有另一台TM M′，$L(M') = L(M)$，并且还已知M′无论对何种输入均不会迷途。也就是说，M′总能算出个结果，非接纳即拒绝。现在，给M′输入字符串w，等待一小时。当一小时后发现它既不处于 q_{accept} 也不处于 q_{reject} 时，我们满可以拍胸脯说：M′还在算着呢，再等等一定有结果！

由以上两例可作结论：「迷途」的存在对于我们估测TM之进展有着不容小觑的影响。纵然它还有种种比这更为重要的

影响，我们先就此打住，留待后文陆续揭露。

既然「迷途」看起来是个不好的东西，那么有没有方法将其剔除？比如，能不能修改TM的定义，使之不可能出现？

为此，先回顾DFA/NFA的定义。定义中，F是接纳状态集；凡是使DFA/NFA读完输入字符串后到达F的，即为接纳；到达Q−F的，即为拒绝。

依葫芦画瓢，把TM的 q_{accept} 升格成F，q_{reject} 升格成Q−F，那么TM中止时不就必取其一了吗？

然而，TM什么时候中止呢？

在状态机中，输入读完了便中止，没有二话可说。但TM则不然。输入本就是存储在存储单元内供TM任意取读、更改的，那么又何有「读完」之说？是故，TM的中止，必须要靠人为规定 $\delta(q_{accept},\cdot)=(q_{accept},\cdot,\cdot)$ 及 $\delta(q_{reject},\cdot)=(q_{reject},\cdot,\cdot)$；如若把 q_{accept} 和 q_{reject} 升格为F与Q−F，那便势必出现「所有状态下均打转」的结果，这显然是不足取的。

事实上，「迷途」这一问题似乎很难通过修改定义来规避，于是人们干脆直接地给了如下定义：

def 判定器：一台对任何输入都不迷途（即要么接纳要么拒绝）的TM。

def 图灵可识别的语言，图灵可判定的语言。

设A是一个字符串集合。若存在TM M：$\mathcal{L}(M)=A$，则称A是图灵可识别语言。

更进一步，若存在判定器 M'：$\mathcal{L}(M')=A$，则称A是图灵可判定语言。

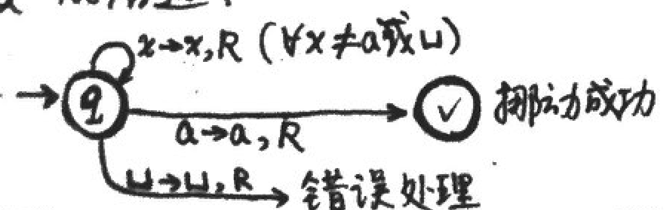
自然，图灵可判定语言必为图灵可识别语言。那么逆命题是否为真？若是，则意味着我们能够把任何可能迷途的TM转换成从不迷途的TM（即判定器），绝对是一大福音。

只可惜事与愿违，人们已经发现了许多图灵可识别但不可判定的语言，从而说明有些问题天生就比别的问题困难。关于可判定性的讨论，我们留待第4章完成。

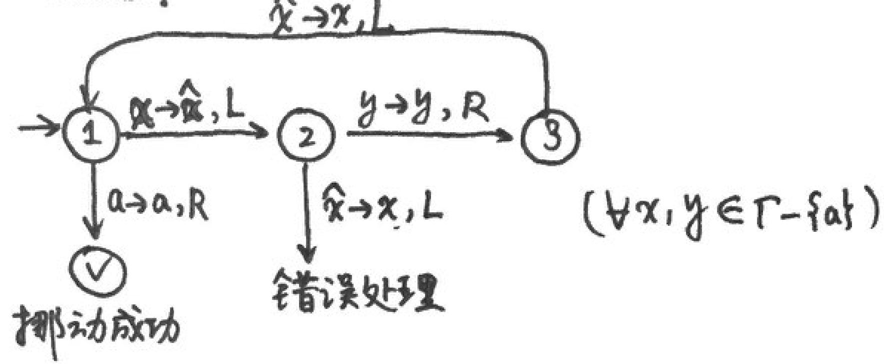
按照惯例，每逢定义完一门语言，紧接着就该研究其性质。然而，鉴于TM描述起来不太方便，而我们大约不太希望每个证明都沦为冗长的细节讨论，是故，我们暂时偏离主航道，先来开发一些实用的「工具机器」，以后用到时「调用」即可。开发的过程也有助于我们习惯TM的运作模式及能力范围。

1° 读写头定位

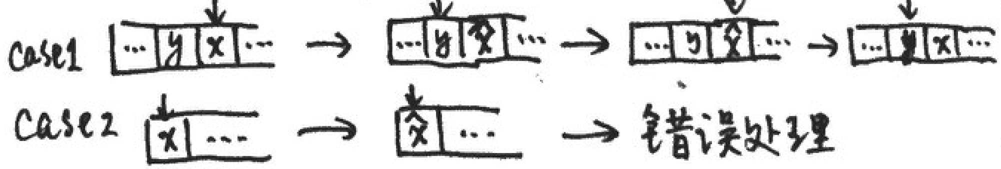
将读写头挪动到当前位置右侧(含当前位置)的首个"a"的右边：



将读写头挪动到当前位置左侧(含当前位置)的首个"a"的右边：



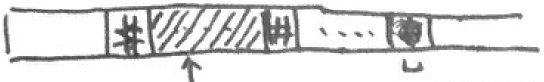
其中第1步尝试左移，并给原位置打上标记"^"。如果读写头不处在最左边，则左移成功，读写头所处的新位置不会标记。此种情形下，通过2→3→1清除标记"^"。相反，如果读写头处在最左边，则左移失败，读写头所处的新位置就是含有标记的老位置。此种情形下，转入错误处理。

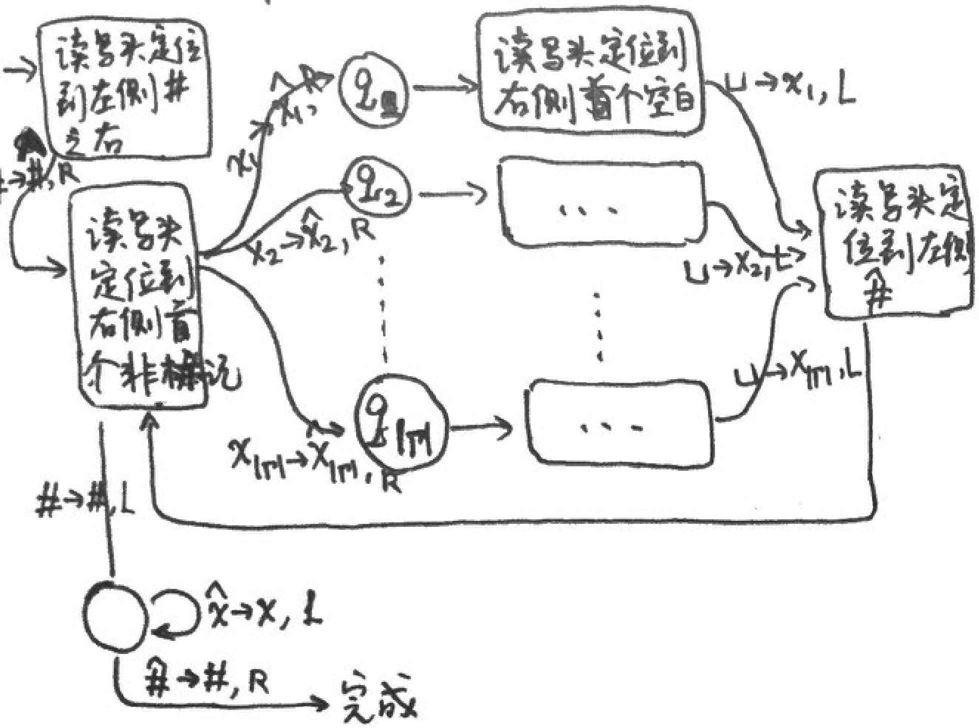


(注：如果把上两个机器中关于a的部分去掉，那么便可实现把读写头挪到最右或最左边的功能)

2°. 复制字符串

把处在当前位置左右两"#"之间的串复制到右侧首个空白处



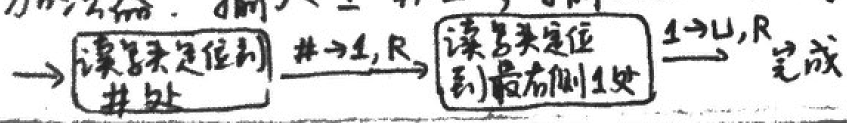


3°. 比较字符串

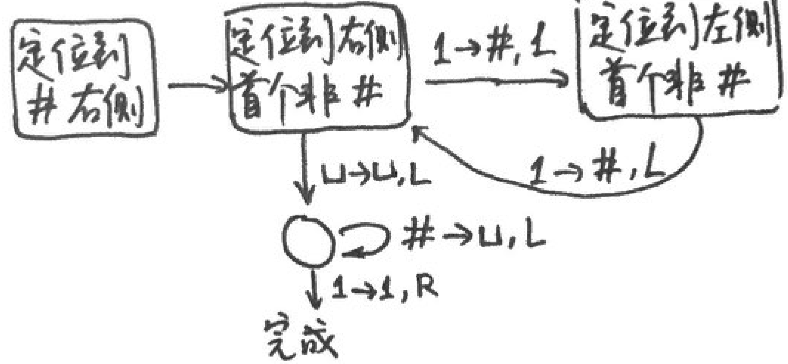
与2°类似。能比较字符串，也就能比较一进制数是否相等

4°. 一进制算术

加法器. 输入 $1^i\#1^j$，输出 1^{i+j}. $(i,j>0)$



减法器. 输入 $1^i\#1^j$，输出 1^{i-j} $(i\geq j)$.



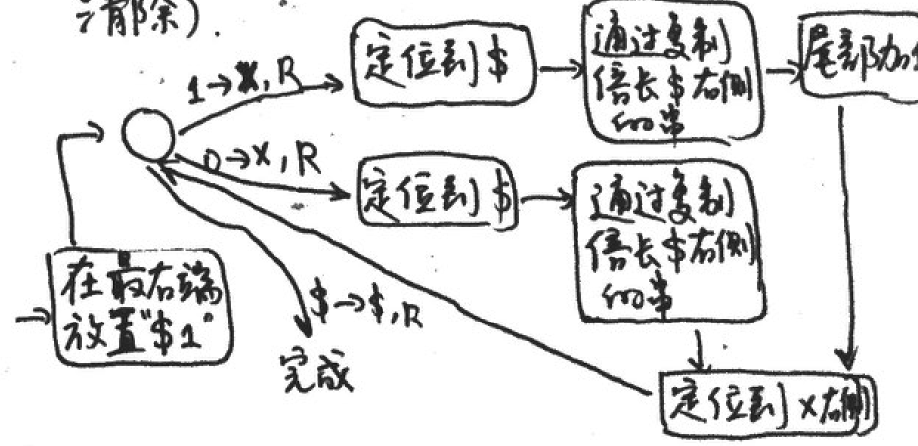
乘法器.（类似，略.）

2^n 比较器. 判定输入 1^i 中 i 是否为2的幂.

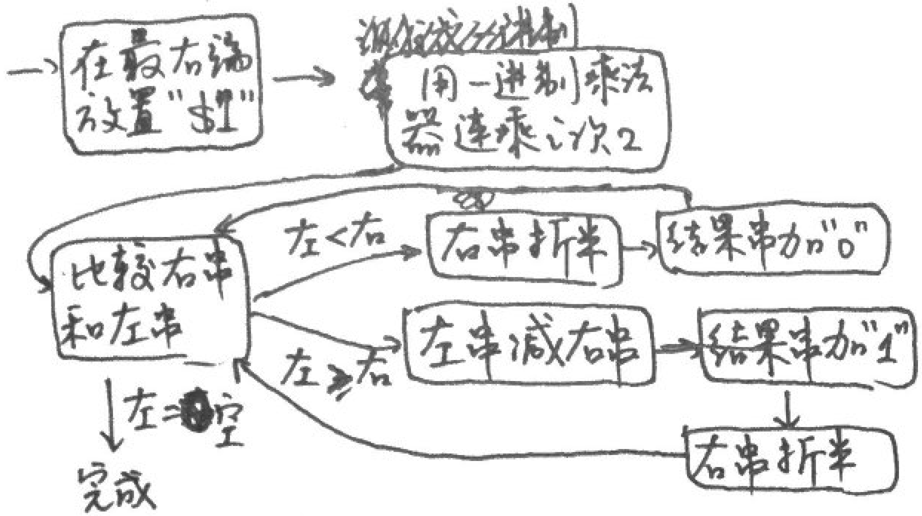
（思路：每轮折半. 细节略）

5°. 二进制与一进制互转.

二进制转一进制。给定二进制串，计算一进制串（存放位置不重要，因为可以复制和清除）.



一进制转二进制，给定一进制串 1^c，计算二进制串



（至此，我们也便间接实现了二进制串的加、减、乘法及比较运算。）

有了以上工具，我们已经具备用自然语言来表述图灵机的能力。仔细想来，我们日常的编程归根结底不过是一系列的赋值、判断、转移和运算，这些操作均可用TM的复制、比较、定位与状态转移及算术工具完成，因此，TM的能力直观上至少不亚于编程语言，更不用说正则语言和CFL了。

Theorem1 图灵可识别语言类对 $\cup$、$\cap$、$\circ$、$*$ 操作封闭。

proof. 这里只证明对 $\cup$ 封闭。其余运算类似。

初拿到这个问题，必定能形成这样的思路：

1° 把输入复制一份

2° 调用 M_1，让它在副本上运行。若 M_1 接纳，则接纳；否则继续。

3° 清空原始输入以外的区域，调用 M_2，让它在原始输入上运行。若 M_2 接纳，则接纳；否则拒绝。

问题在于，M_1 有可能迷途。一旦如此，M_2 将永无执行之日。若输入 $w \in L(M_2)$，那么机器本该进入接纳状态，如今却无法实现。

解决这一问题的通用思路是人为限定运行步数。在第一轮，限定 M_1 与 M_2 运行不得超过1步；在第二轮，限定 M_1 与 M_2 运行不得超过2步；等等。

下面给出技术细节。

设A与B均为图灵可识别语言，那么，存在TM M_1与M_2：$\mathcal{L}(M_1)=A$，$\mathcal{L}(M_2)=B$。我们希望构造M：$\mathcal{L}(M)=A\cup B$。

M对内存进行适当的划分：

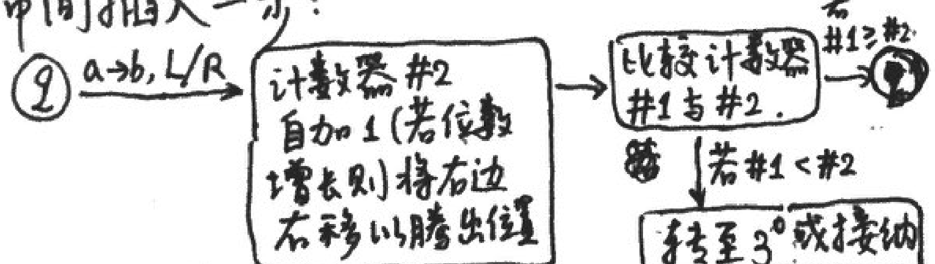
输入	$	计数器#1	$	计数器#2	$	运行区域…

其中后三部分是可变长的。M的运行规则为：

1°. 适当地添加"$"隔断。令计数器#1与计数器#2的值均为1。把输入复制到运行区域。

2°「调用」M_1。所谓「调用」，指的是把读写头挪到运行区域，并将状态切换为M_1的初始状态。

为了限定M_1运行步数，必须事先对M_1做些手脚。针对M_1中δ规定的每一种转移 $s \xrightarrow{a\to b,\ L/R} q$，我们都强行在中间插入一步：



3° 清空运行区域，并重新将输入复制过去。令计数器#2 = 1。

4° 调用M_2。与M_1一样，M_2也被做了手脚。

5° 清空运行区域，并重新复制输入。令计数器#1加一，计数器#2 = 1，重复2°–5°。

显然，若$w\in A\cup B$，则它必然能够在有限步内被M_1或M_2接纳，故M也能接纳w。反之，若$w\notin A\cup B$，则结论亦否。故$\mathcal{L}(M)=A\cup B$. ■

remark. 实际上，我们可以在证明中找到操作系统调度器的雏形。通过限定计算步数，M_1与M_2得以分时运行。另外，调度器是通过「中断」来调用的。

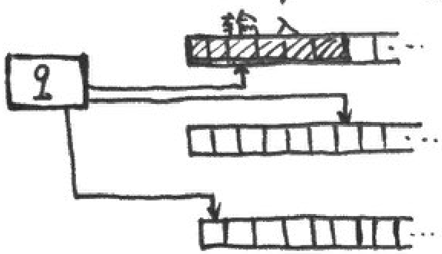
虽然图灵可识别语言对这许多运算封闭，但它对补操作不封闭，理由如下：假若它对补操作封闭，那么对任意图灵可识别语言A，存在M与M'：$\mathcal{L}(M)=A$

及 $\mathcal{L}(M')=\overline{A}$。那么，仿照上面定理的方法，我们必能用 M 与 M' 构造出 M''，使 (1) $\mathcal{L}(M'')=A$；(2) M'' 是判定器，从而说明任何图灵可识别语言均可判定，与前面介绍的结论矛盾。

接下来，我们介绍一些TM的变种，并证明它们与TM等价。

1° 多带图灵机（注：「带」指「纸带」，即存储器）

含有 k 个存储器及 k 个读写头，状态转移函数 $\delta: Q\times\Gamma^k \to Q\times\Gamma^k\times\{L,R\}^k$



(k=3)

2° 非确定图灵机

每次转移都具有一种或多种可能。状态转移函数 $\delta: Q\times\Gamma \to 2^{Q\times\Gamma\times\{L,R\}}$

只要有一种转移路径（计算流程）到达接纳状态，非确定TM就接纳。

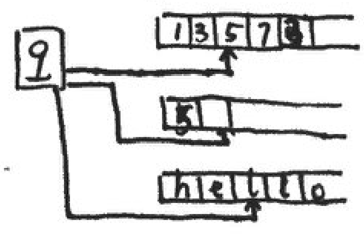
Theorem 2 多带图灵机与图灵机等价，即：二者识别的语言类无差别。

proof. 首先，给定任意一台TM，它本身就是多带TM的特例 (k=1)，故定理的"TM ⇒ 多带TM"是显然的。

然后我们考察反方向。给定一台多带TM M，我们希望找到TM M'，使 $\mathcal{L}(M')=\mathcal{L}(M)$。

思路仍与Theorem 1相似。用"\$"将存储器隔断为 k 份，每份大小可变，

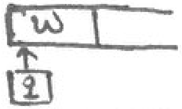
分别存放M运行时 k 个存储器上的内容。为了记录M的读写头位置，将M的字符集扩张一倍，对每个字符 a 引入标记符 $\dot{a}$。例如，M的格局为

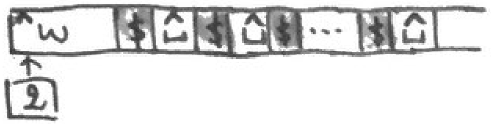


(k=3)

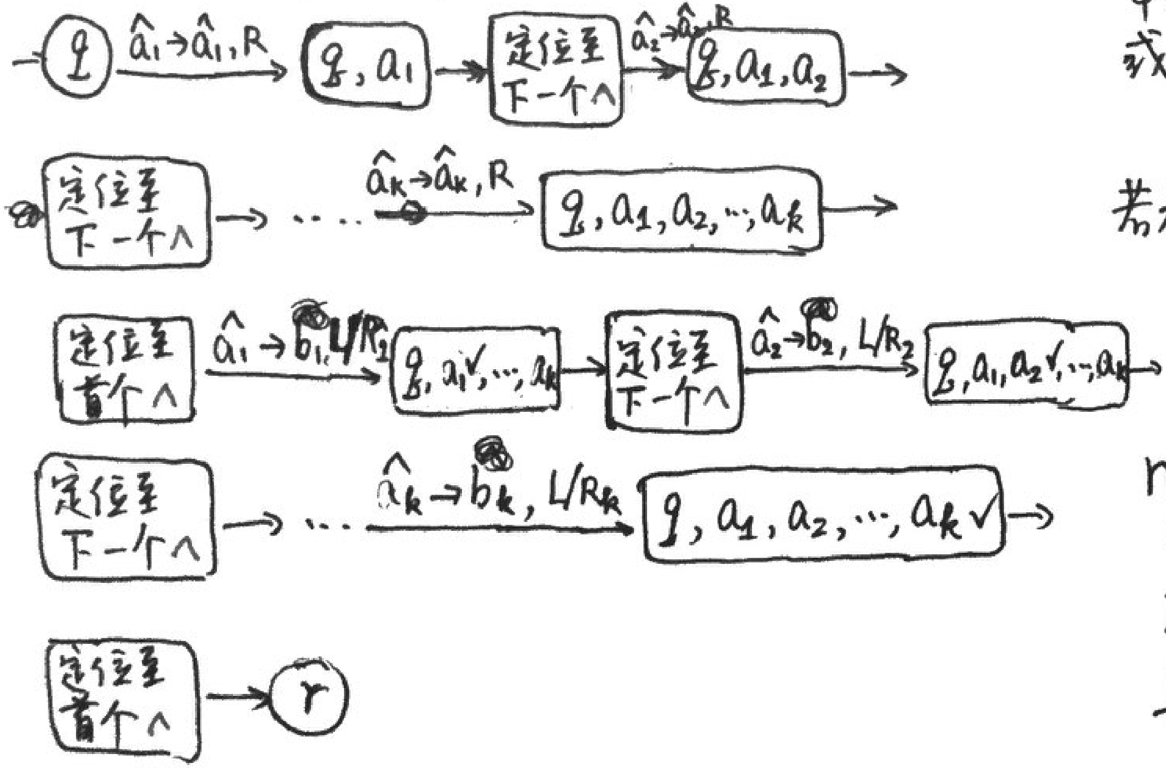
则 M' 对应的格局为

当然，这种对应不是天然产生的，而是需要我们去实现和维护的。M' 描述如下：

1° 将初始格局  调整为



2° 模拟M的行为。具体说来，针对每一条M中的规则 $(q, a_1,\cdots,a_k)\to(r, b_1,\cdots,b_k, L/R_1, L/R_2,\cdots,L/R_k)$ 我们在M'中用一串规则去实现：



简而言之，就是从左至右读遍带∧标记的元素，逐步确定出M的每个读写头所读到的信息，并将这信息记忆在M'的状态中。确定完以后，M进入了状态 $\boxed{q, a_1,\cdots,a_k}$。接下来，根据这一信息，依次更新 $a_1,\cdots,a_k$ 为 $b_1,\cdots,b_k$ 并移动∧标记（图中未画这一步）。此外，还有一个细节需要处理：若移动∧时遇到\$，要相应地作异常处理（禁止左移或腾挪右侧空间）。

可以想见，M'的状态数必以千万计，若是详细写出必使人头昏脑胀。

不难看出，上述构造等价地模拟了M，故 $L(M')=L(M)$. ■

remark. Theorem2的结论很有用处。当我们需要设计复杂TM的时候，不妨采用多带模型，因为它天然就具备「内存隔离」能力。

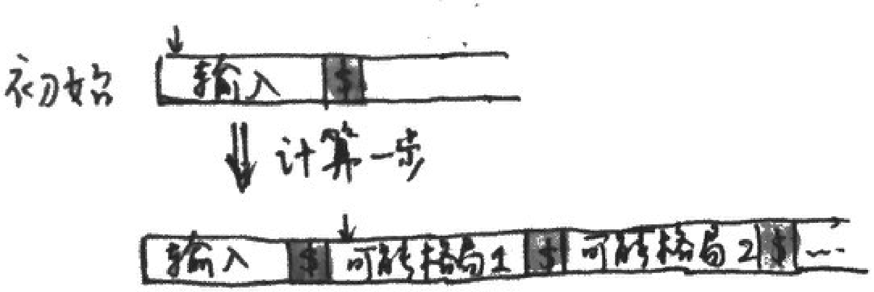
利用该定理可以容易地证明：双侧无限存储的TM与TM等价。(留作习题)

Theorem3 非确定图灵机与图灵机等价。

proof 非确定性给我们造成的麻烦在于有许多并行的线程。在处理NFA时，我们将当前所有可能出现的状态包裹成DFA中一个大状态，从而证明了NFA ⇒ DFA。

在这里，由于潜在的格局是无限多的，故不能用状态来包裹格局，而应用格局来包裹格局。

说白了，就是想办法用一个TM的格局来囊括当前非确定TM所能到达的全部格局。答案呼之欲出：队列。



给定非确定TM N，我们构造TM M：它从输入格局开始，把N下一步所有能推得的格局均附加在最右端，以"$"分隔。若有任一格局为接纳，则接纳；否则，进入「可能格局#1」，将N下一步所有能推得的格局均附加在最右端，以"$"分隔，如此等等。

显然 $\mathcal{L}(M)=\mathcal{L}(N)$. ■

由Theorem2,3易证：k-PDA(即有k个栈的PDA)与TM等价。

前面已说过，TM是一种与人类计算模式很相似的模型，因而人们相信它能刻画出计算能力的极限。这并不是说图灵可识别语言类囊括了任何语言，而是说在此类语言以外的语言不能被人类/物理机器计算。

固然有很多「超计算模型」如谕言模型(Oracle Model)等，其表达能力超越TM。

但是它们仅停留在概念层面，无法被真正地制造成物理实体以实现它们所规定的功能。无法被制造的原因是多种多样的，多数是因为它们违背物理定律（如热力学的定律、相对论等），与「永动机」如出一辙。

于是，我们可以这样断言：在现有的物理框架下，不应指望制造出超越TM表达能力的计算机器；TM模型是目前而言人类对于计算的最佳抽象。此即Church-Turing论题。

这一论断意义重大，因为它给我们以讨论通用计算的基础。如果有什么东西是不可被TM计算的，我们就认为它真的是不可计算的；如果有什么语言是超越TM的识别/判定范围的，我们就认为它真的是不可识别/判定的。换言之，我们将关于TM的可计算性/可判定性上升为通用的可计算性/可判定性。