算法分析导论	图灵机笔记
CS4820 2013年春季	2013年4月1日，星期一

这些笔记改编自[10]。

1 图灵机和有效计算

在这些笔记中，我们将介绍图灵机（TMs），它们以Alan Turing的名字命名，他于1936年发明了它们。 图灵机可以计算任何通常被认为是可计算的函数；事实上，将可计算定义为图灵机可计算是相当合理的。

TMs在20世纪30年代发明，比真正的计算机出现要早很多。 它们出现在数学家们试图理解“有效计算”的概念时。 他们知道各种有效计算的算法，但他们不太确定如何以一种普遍的方式定义“有效计算”，以区分可计算和不可计算的问题。 为了确定这个概念，出现了几种不同的形式化方法，每种方法都有其自己的特点：

- 图灵机（艾伦・图灵[20]）；
- Post系统（埃米尔・波斯特[12, 13]）；
- μ-递归函数（库尔特・哥德尔[7]，雅克・埃尔布朗）；
- λ-演算（阿隆佐・邱奇[2]，斯蒂芬・克利尼[8]）；以及
- 组合逻辑（莫西斯・舒恩菲克尔[18]，哈斯克尔・柯里[4]）；

所有这些系统都体现了一种或另一种形式的有效计算的思想。 它们处理各种类型的数据；例如，图灵机操作有限字母表上的字符串，μ-递归函数操作自然数，λ-演算操作λ-项，组合逻辑操作由组合符号构建的项。

然而，所有这些不同类型的数据之间存在自然的转换。 例如，让 $\{0,1\}*$表示字母表 $\{0,1\}$上所有有限长度的字符串的集合。 存在一个简单的一一对应关系 $\{0,1\}*$和自然数 $\mathbb{N} = \{0,1,2,\ldots\}$之间，定义如下：

$$x \quad \rightarrow \quad \#(1x) - 1, \tag{1}$$

其中 $\#y$是由二进制字符串 y表示的自然数。 相反地，将几乎任何东西（自然数、 λ-项、$\{0,1,2,\ldots,9\}*$中的字符串、树、图等）编码为$\{0,1\}*$中的字符串是很容易的。 在这些数据的自然编码下，所有上述形式主义都可以模拟彼此，因此尽管它们在表面上有所不同，但它们在计算上是等价的。

如今，我们可以毫不掩饰地利用我们更现代的视角，并将编程语言
如Java或C（或它们的理想化版本）添加到这个列表中-与Church和
G¨odel所必须努力应对的情况相比，这是一种真正的奢侈。

在上面列出的经典系统中，最接近现代计算机的是图灵机。 除了我们将在下面定义的现成模型外，还有许多定制变体（非-确定性、多带、多维带、双向无限带等），它们在计算上等效，即它们都可以相互模拟。

Church的论题

因为这些截然不同的形式主义都是计算上等效的，它们所体现的共同计算概念非常强大，也就是说，在模型中进行相当激进的扰动下，它是不变的。 所有这些数学家都在从不同的角度看同一件事情。 这太引人注目，不可能只是巧合。 他们很快意识到，所有这些系统的共同点必须是他们长期寻求的难以捉摸的*effective computabil-ity*的概念。 可计算性不仅仅是图灵机，也不仅仅是 λ-演算，也不仅仅是μ-递归函数，也不仅仅是Pascal编程语言，而是它们所体现的共同精神。Alonzo Church [3] 表达了这个思想，它后来被称为Church的论题（或者Church-Turing论题）。 这不是一个定理，而是一个声明，它准确地捕捉了我们对于在原则上是有效计算的直觉，不多也不少。

回顾起来，丘奇的论题可能看起来并不像是一件大事，因为现在我们对现代计算机的能力非常熟悉；但请记住，在它首次提出的时候，计算机和编程语言尚未发明。 认识到这一点是一个巨大的智力飞跃。

可能最具说服力的发展导致了对丘奇的论题的接受，那就是图灵机。 这是第一个可以被认为是可编程的模型。 如果有人把其他系统放在你面前，并宣称：“这个系统完全捕捉了我们所说的有效计算”，你可能会怀有一些怀疑。 但是很难对图灵机提出异议。 人们可以正确地对丘奇的论题提出质疑，因为图灵机没有涉及计算的某些方面（例如，随机化或交互式计算），但没有人可以否认，图灵机所捕捉到的有效计算的概念是强大而重要的。

普适性和自指性

这些系统中最引人入胜的一个方面，也是我们研究它们的一个普遍主题，是将程序视为数据的概念。 这些编程系统都足够强大，可以编写能够理解和操作其他以某种合理方式编码为数据的程序。例如，在λ演算中，λ项既是程序又是数据；组合逻辑中的组合符号操作其他组合符号；存在所谓的Gödel编号的μ递归函数，其中每个函数都有一个可以用作其他μ递归函数输入的编号；图灵机可以将其输入字符串解释为其他图灵机的描述。 从这个想法到通用模拟的概念并不遥远，在通用程序或机器U的构造中，它接受另一个程序或机器M的编码描述和一个字符串x作为输入，并对M在输入x上进行逐步模拟。 这种现象的一个现代例子是用Scheme编写的Scheme解释器。

普适性的一个深远推论是自我引用的概念。 正是这种能力导致了对自然不可计算问题的发现。 如果你了解一些集合论，你可以通过基数论证使自己相信不可计算问题必然存在：决策问题有不可数多个，但图灵机只有可数多个。 然而，自我引用使我们能够构造非常简单和自然的不可计算问题的例子。 例如，不存在通用过程可以确定给定的C程序中的代码块是否会被执行，或者给定的C程序是否会停止。 这些是编译器构建者希望解决的重要问题；不幸的是，可以给出一个形式证明它们是不可解的。

自我引用能力最引人注目的例子可能是库尔特·哥德尔的不完全性定理。从20世纪初开始，从怀特海德和罗素的《数理逻辑的原理》[21]开始，有一个运动试图将所有数学归纳为纯符号操作，独立于语义。 这部分是为了理解和避免罗素和其他人发现的集合论悖论。 这一运动由数学家大卫·希尔伯特提倡，并被称为形式主义计划。 它吸引了很多追随者，并推动了我们今天所知的形式逻辑的发展。 它的支持者相信所有数学真理都可以在某个固定的形式系统中推导出来。

仅仅通过从几个公理开始，并以纯机械的方式应用推理规则。这种数学证明的观点高度计算化。当时最流行的形式演绎系统用于推理自然数的是被称为Peano算术（PA）的形式演绎系统，人们认为它足以表达和机械地推导出所有关于数论的真实陈述。 不完全性定理表明这是错误的：在PA中存在着甚至相当简单的数论陈述，它们是真实的但无法被证明的。这不仅适用于PA，也适用于任何合理的扩展。这一揭示对形式主义计划来说是一个重大挫折，并在数学界引起了震动。

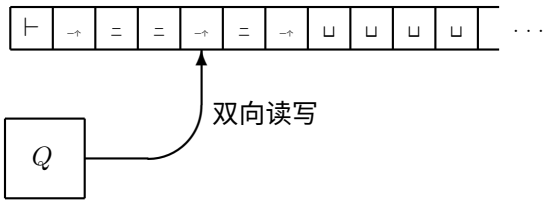
哥德尔使用自指证明了不完全性定理。 这里需要的基本观察是，数论语言足够表达关于自身和关于PA中的证明的内容。例如，可以写下一个数论陈述，它说某个其他的数论陈述在PA中有一个证明，并且可以使用PA本身来推理这样的陈述。 现在，通过涉及替换的巧妙论证，实际上可以构造出关于它们自身是否可证明的陈述。 哥德尔实际上构造了一个句子，它说：“我是不可证明的。”

普适性的后果不仅是哲学上的，也是实际上的。普适性在某种程度上是导致计算机的发展的思想种子，就是我们今天所知道的计算机的存储程序的概念，即可以被硬件读取和执行的软件。 这种可编程性使得计算机非常多才多艺。 尽管几年后才以物理形式实现，但这个概念在图灵在1930年代的理论工作中确实存在。

图灵机的非正式描述

我们在这里描述的是一种确定性的、单带图灵机。 这是标准的现成模型。 有很多变种，表面上可能更强大或更弱，但实际上并非如此。我们将在第3节中考虑其中一些。

图灵机有一个有限的状态集合 Q，一个半无限的带子，左端由一个结束标记$\vdash$限定，右侧无限延伸，还有一个可以在带子上左右移动、读写符号的头部。



输入字符串长度有限，并且最初写在紧邻左边界的连续带子单元上。 输入右侧的无限多单元都包含一个特殊的空白符号 ␣。机器从其开始状态 s开始，其头部扫描左边界。 每一步它都读取头部下的带子上的符号。 根据该符号和当前状态，它在该带子单元上写入一个新符号，将头部向左或向右移动一个单元，并进入一个新状态。 它在每种情况下采取的动作由一个转移函数 δ确定。 通过进入一个特殊的接受状态 t来接受输入，通过进入一个特殊的拒绝状态 r来拒绝输入。 在某些输入上，它可能无限循环而永远不接受或拒绝，这种情况下被称为在该输入上循环。

图灵机的形式定义

严格来说，一个确定性单带图灵机是一个9元组

$$M \quad = \quad (Q,\, \Sigma,\, \Gamma,\, \vdash,\, \sqcup,\, \delta,\, s,\, t,\, r\,),$$

其中

- Q是一个有限集合（状态集）；

- Σ是一个有限集合（输入字母表）；
- Γ是一个有限集合（纸带字母表），其中包含Σ作为子集；
- $\llcorner\lrcorner \in \Gamma - \Sigma$，表示空白符号；
- $\vdash \in \Gamma - \Sigma$，表示左端标记符；
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$，表示转移函数；
- $s \in Q$，表示起始状态；
- $t \in Q$，接受状态；并且
- $r \in Q$，拒绝状态， $r = t$。

直观地说， $\delta(p, a) = (q, b, d)$ 意味着“当处于状态 p扫描符号 a时，在该磁带单元上写入 b，将头部向 d方向移动，并进入状态 q。”符号 L和 R分别代表左和右。

我们限制图灵机，使得左边界标记永远不会被其他符号覆盖，并且机器永远不会移动到左边界标记的左侧；也就是说，对于所有的 $p \in Q$，存在 $q \in Q$使得

$$\delta(p, \vdash) \;=\; (q, \vdash, R)。 \tag{2}$$

我们还要求一旦机器进入接受状态，就永远不会离开它，拒绝状态也是如此；也就是说，对于所有的 $b \in \Gamma$，存在$c, c' \in \Gamma$和$d, d' \in \{L, R\}$使得

$$\delta(t, b) = (t, c, d), \qquad\qquad \delta(r, b) = (r, c', d'). \tag{3}$$

我们有时将状态集和转移函数统称为有限控制器。

例子1。这是一个接受非上下文无关集合 $\{a^n b^n c^n \mid n \geq 0\}$的图灵机。 简单来说，机器从起始状态 s开始，然后从左到右扫描输入字符串，检查其是否符合 $a^*b^*c^*$的形式。
在扫描过程中，它不会写入任何内容（严格来说，它会写入与读取的符号相同的符号）。 当它看到第一个空白符号 $\llcorner\lrcorner$时，它会用右端标记符号 $\dashv$覆盖它。 然后它向左扫描，擦除它遇到的第一个 c，然后是第一个 b，然后是第一个 a，直到遇到 $\vdash$。 然后它向右扫描，擦除一个 a，一个 b和一个 c。 它继续在输入上左右扫描，每次通过擦除每个字母的一个实例。 如果在某次扫描中，它看到至少一个字母的一个实例，而没有看到另一个字母的任何实例，它会拒绝。 否则，它最终会擦除所有字母，并在 $\vdash$和 $\dashv$之间进行一次只看到空白符号的扫描，此时它接受。

从形式上来说，这台机器有

$$Q = \{s, q_1, \ldots, q_{10}, t, r\}, \qquad \Sigma = \{a, b, c\}, \qquad \Gamma = \Sigma \cup \{\vdash, \sqcup, \dashv\}.$$

关于 $\dashv$没有什么特别的；它只是带磁带字母表中的一个额外有用的符号。 转移函数 δ由以下表格指定：

	$\vdash$	一个	$\llcorner\lrcorner$	c	$\sqcup$	一个
s	$(s, \vdash, R)$	(s, a, R)	(q_1, b, R)	(q_2, c, R)	$(q_3, \dashv, L)$	$-$
q_1	$-$	$(r, -, -)$	(q_1, b, R)	(q_2, c, R)	$(q_3, \dashv, L)$	$-$
q_2	$-$	$(r, -, -)$	$(r, -, -)$	(q_2, c, R)	$(q_3, \dashv, L)$	$-$
q_3	$(t, -, -)$	$(r, -, -)$	$(r, -, -)$	$(q_4, \sqcup, L)$	$(q_3, \sqcup, L)$	$-$
q_4	$(r, -, -)$	$(r, -, -)$	$(q_5, \sqcup, L)$	(q_4, c, L)	$(q_4, \sqcup, L)$	$-$
q_5	$(r, -, -)$	$(q_6, \sqcup, L)$	(q_5, b, L)	$-$	$(q_5, \sqcup, L)$	$-$
q_6	$(q_7, \vdash, R)$	(q_6, a, L)	$-$	$-$	$(q_6, \sqcup, L)$	$-$
q_7	$-$	$(q_8, \sqcup, R)$	$(r, -, -)$	$(r, -, -)$	$(q_7, \sqcup, R)$	$(t, -, -)$
q_8	$-$	(q_8, a, R)	$(q_9, \sqcup, R)$	$(r, -, -)$	$(q_8, \sqcup, R)$	$(r, -, -)$
q_9	$-$	$-$	(q_9, b, R)	$(q_{10}, \sqcup, R)$	$(q_9, \sqcup, R)$	$(r, -, -)$
q_{10}	$-$	$-$	$-$	(q_{10}, c, R)	$(q_{10}, \sqcup, R)$	$(q_3, \dashv, L)$

上表中的符号 $-$表示“不关心”。和的转换未包含在表中，只需定义它们满足限制条件（2）和（3）的任何内容。

配置和接受

在图灵机 M的读/写带上的任意时刻，包含一个半无限字符串的形式为 $y\llcorner\lrcorner^\omega$的字符串，其中 $y \in \Gamma^*$ (y是一个有限长度的字符串)， $\llcorner\lrcorner^\omega$表示半无限字符串

$$\sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \cdots.$$

（这里 ω表示最小的无限序数。）尽管字符串是无限的，但它总是有一个有限的表示，因为除了有限多个符号之外，其余的符号都是空白符号 $\llcorner\lrcorner$。

我们将配置定义为 $Q\times\{y\llcorner\lrcorner^\omega \mid y \in \Gamma^*\}\times\mathbb{N}$的元素，其中 $\mathbb{N} = \{0, 1, 2, \ldots\}$。 配置是一个全局状态，在某个时刻给出了有关TM计算的所有相关信息的快照。 配置（p, z, n）指定了当前状态 p的有限控制，当前带内容 z和当前读/写头位置 $n \geq 0$。 我们通常用α, β, γ表示配置。

在输入 $x \in \Sigma^*$的起始配置是配置

$$(s, \vdash x\sqcup^\omega, 0).$$

最后的组成部分0表示机器最初正在扫描左边界 $\vdash$.

可以定义下一个配置关系 $\xrightarrow[M]{1}$. 对于字符串 $z \in \Gamma^\omega$，令 z_n为 z的第n个符号（最左边的符号是 z_0），并且用 $z[n/b]$表示在位置n处用 b替换 z n得到的字符串。例如，

$$\vdash b\ a\ a\ a\ c\ a\ b\ c\ a\ \cdots[4/b] \quad = \quad \vdash b\ a\ a\ b\ c\ a\ b\ c\ a\ \cdots.$$

关系 $\xrightarrow[M]{1}$ 由以下方式定义

$$(p, z, n) \xrightarrow[M]{1} \begin{cases} (q, z[n/b], n-1) \text{ 如果 } \delta(p, z_n) = (q, b, L), \\ (q, z[n/b], n+1) \text{ 如果 } \delta(p, z_n) = (q, b, R). \end{cases}$$

直观地说，如果带子上包含 z，且 M处于状态 p扫描第 n个带子单元，并且 δ指示打印 b，向左移动，并进入状态 q，那么在这一步之后，带子上将包含 $z[n/b]$，头部将扫描第$(n-1)$个带子单元，新状态将为 q。我们定义自反传递闭包 $\xrightarrow[M]{*}$

$$\xrightarrow[M]{} \text{ of } \xrightarrow[M]{1} \quad \text{归纳地说:}$$

- $\alpha \xrightarrow[M]{0} \alpha$,
- $\alpha \xrightarrow[M]{n+1}$ 如果 $\alpha \xrightarrow[M]{n}$, 则$\beta$ $\gamma \xrightarrow[M]{1}$ 对于某个 γ，则β
- $\alpha \xrightarrow[M]{*}$ 如果 $\alpha \xrightarrow[M]{n}$, 则$\beta$ 对于某个 $n \geq 0$，则β

如果机器 M接受输入 $x \in \Sigma^*$，则称其为接受输入x。

$$(s, \vdash x\sqcup^\omega, 0) \quad \xrightarrow[M]{*} \quad (t, y, n)$$

如果机器M拒绝输入 x，则称其为拒绝输入x。

$$(s, \vdash x\sqcup^\omega, 0) \quad \xrightarrow[M]{*} \quad (r, y, n)$$

对于某个 y和 n，则称其为拒绝输入 x。 如果机器在输入 x上要么接受 x要么拒绝 x，则称其为停机。 这只是一个数学定义；机器并不会真的停下来。 有可能既不接受也不拒绝，这种情况下称其为循环。 如果机器在所有输入上都停机，则称其为全停机。 集合 $L(M)$表示被 M接受的字符串集合。我们称这样的集合为

- 递归可枚举$(r.e.)$ 如果它是某个图灵机M的语言 L,
- 协递归 如果它的补集是可枚举的，并且
- 递归如果它是某个全局图灵机M的语言 L.

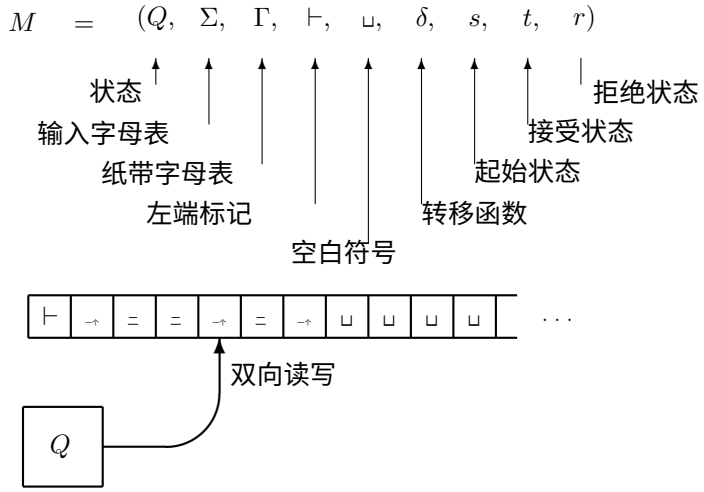
在常用语中，“递归”一词通常指调用自身的算法。 上述定义与此用法无关。 在这里使用，它只是一个总是停机的图灵机接受的集合的名称。

历史注释

虽然阿隆佐・邱奇是第一个明确提出的，但邱奇的论文通常被称为邱奇-图灵论题[3]。 该论题基于邱奇和克利尼的观察，即 λ-演算和G¨odel和Herbrand的 μ-递归函数在计算上是等价的[3]。 邱奇在写[3]时显然不知道图灵的工作，或者如果他知道，他没有提及。而图灵在[20]中明确引用了邱奇的论文[3]，并且显然认为他的机器更具有说服力的可计算性定义。 在[20]的附录中，图灵概述了图灵机和 λ-演算的计算等价性的证明。

图灵机的两个例子

在上一节中，我们定义了确定性单带图灵机：



在每一步中，根据当前读取的纸带符号和当前状态，它在纸带上打印一个新符号，将头部向左或向右移动，并进入一个新状态。 这个动作在转移函数中被形式化地指定

$$\delta : Q \times \Gamma \quad \rightarrow \quad Q \times \Gamma \times \{L, R\}.$$

直观地说， $\delta(p, a) = (q, b, d)$ 的意思是，“当处于状态 p并扫描符号 a时，在该纸带单元上写入 b，将头部向 d方向移动，并进入状态 q。”

我们定义一个配置为一个三元组（q, z, n），其中q是一个状态， z是一个形如 $y\llcorner\lrcorner^{\omega}$的半无限字符串， $y \in \Sigma^*$，描述了带子的内容， n是一个自然数，表示带头位置。

转移函数 δ被用来定义下一个配置关系 $\xrightarrow[M]{1}$ 关于配置及其自反传递闭包 $\xrightarrow[M]{*}$ 。 机器 M接受输入 $x \in \Sigma^*$如果

$$(s, \vdash x\sqcup^{\omega}, 0) \quad \xrightarrow[M]{*} \quad (t, y, n)$$

对于某个 y和 n，并且拒绝输入 x如果

$$(s, \vdash x\sqcup^{\omega}, 0) \quad \xrightarrow[M]{*} \quad (r, y, n)$$

对于某个 y和 n。 上面的左配置是在输入 x上的起始配置。 回想一下，我们限制了图灵机，使得一旦图灵机进入其接受状态，它就永远不会离开它，对于拒绝状态也是如此。 如果 M在输入 x上从未进入其接受或拒绝状态，则称其在输入 x上循环。 如果它接受或拒绝，则称其在输入 x上停机。 对于所有输入都停机的图灵机称为总。定义集合

$$L(M) \quad = \quad \{x \in \Sigma^* \mid M \text{ 接受 } \quad x\}.$$

这被称为机器 M接受的集合. Σ^*的一个子集被称为递归可枚举的（r.e.），如果它是M的 L(M)的形式的集合. 如果一个集合是某个总的 M的$L(M)$的形式，那么它被称为递归的.

目前，术语递归可枚举和递归只是描述图灵机和总图灵机接受的集合的技术术语；它们没有其他意义 . 我们将在第3节讨论这个术语的起源 .

例子**2.** 考虑集合 $\{ww \mid w\in \{a, b\}*\}$，即由两个重复的字符串 w组成的字符串集合 $\{a, b\}$**..** 这是一个递归集合，因为我们可以为它提供一个总的图灵机 M。机器 M的工作方式如下. 对于输入 x，它扫描到第一个空白符号 $\llcorner\lrcorner$，通过对符号数取模来确保 x的长度为偶数，如果不是，则立即拒绝. 它放下一个右端标记符号 $\dashv$，然后反复在输入上来回扫描. 在每次从右到左的扫描中，它用´标记它看到的第一个未标记的 a或 b. 在每次从左到右的扫描中，它用`标记它看到的第一个未标记的 a或 b. 它继续这个过程，直到所有符号都被标记. 例如，对于输入

$$a\ a\ b\ b\ a\ a\ a\ b\ b\ a$$

初始带内容为

$$\vdash a\ a\ b\ b\ a\ a\ a\ b\ b\ a \sqcup \sqcup \sqcup \cdots$$

在前几次迭代后，带的内容如下

$$\begin{aligned}
&\vdash a\ a\ b\ b\ a\ a\ a\ b\ b\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots \\
&\vdash \grave{a}\ a\ b\ b\ a\ a\ a\ b\ b\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots \\
&\vdash \grave{a}\ a\ b\ b\ a\ a\ a\ b\ \acute{b}\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots \\
&\vdash \grave{a}\ \grave{a}\ b\ b\ a\ a\ a\ b\ \acute{b}\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots \\
&\vdash \grave{a}\ \grave{a}\ b\ b\ a\ a\ a\ \acute{b}\ \acute{b}\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots
\end{aligned}$$

用 ` 正式表示标记 a，即写入符号 $\grave{a} \in \Gamma$；因此

$$\Gamma \quad = \quad \{a, b, \vdash, \sqcup, \dashv, \grave{a}, \grave{b}, \acute{a}, \acute{b}\}.$$

当所有符号都被标记时，我们将输入字符串的前一半用`标记，后一半用´标记。

$$\vdash \grave{a}\ \grave{a}\ \grave{b}\ \grave{b}\ \grave{a}\ \acute{a}\ \acute{a}\ \acute{b}\ \acute{b}\ \acute{a} \dashv \sqcup \sqcup \sqcup \cdots$$

我们这样做的原因是为了找到输入字符串的中心。

然后，机器会反复从左到右扫描输入。 在每次扫描中，它会擦除第一个标记为`的符号，但会将该符号记住在有限控制器中（“擦除”实际上是写入空白符号 ␣）。 然后，它会向前扫描，直到看到第一个标记为´的符号，并检查该符号是否相同，并将其擦除。如果两个符号不相同，则拒绝。 否则，当它擦除所有符号时，它接受。 在我们的例子中，每次扫描后，磁带内容如下。

$$\grave{a}\ a\ \grave{b}\ b\ a\ a\ \acute{b}\ b\ a\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$
$$\sqcup\ a\ \grave{b}\ b\ a\ \sqcup\ a\ \acute{b}\ b\ a\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$
$$\sqcup\ \sqcup\ \grave{b}\ b\ a\ \sqcup\ \sqcup\ b\ \acute{b}\ a\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$
$$\sqcup\ \sqcup\ \sqcup\ \grave{b}\ a\ \sqcup\ \sqcup\ \sqcup\ b\ a\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$
$$\sqcup\ \sqcup\ \sqcup\ \sqcup\ \grave{a}\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ a\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$
$$\sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \sqcup\ \dashv \sqcup\ \sqcup\ \sqcup \cdots$$

例子**3.** 我们想要构建一个总的图灵机，如果输入字符串的长度是质数，则接受该字符串。. 我们将给出埃拉托斯特尼筛法的图灵机实现，可以非正式地描述如下。 假设我们想要检查是否 n是质数。 我们按顺序写下从2到 n的所有数字，然后重复以下步骤：找到列表中最小的数字，声明它是质数，然后划掉所有该数字的倍数。 重复此过程，直到列表中的每个数字都被声明为质数或被划掉为较小质数的倍数。

例如，要检查23是否为质数，我们将从2到23的所有数字开始：

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

在第一次遍历中，我们划掉2的倍数：

~~2~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23

剩下的最小数字是3，而且它是质数。 在第二次遍历中，我们划掉3的倍数：

~~2~~ ~~3~~ ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23

然后5是下一个质数，所以我们划掉5的倍数；依此类推。 由于23是质数，它永远不会被任何更小的数的倍数划掉，当所有更小的数都被划掉时，我们最终会发现这个事实。

现在我们展示如何在图灵机上实现这个。假设我们在磁带上写有 a^p。 我们用 $p = 23$来说明算法。

$$\vdash a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \sqcup\ \sqcup\ \sqcup \cdots$$

如果 $p = 0$或 $p = 1$，则拒绝。 我们可以通过查看磁带的前三个单元格来确定这一点。 否则，至少有两个 a。 擦除第一个 a，向右扫描到输入的末尾，并将输入字符串中的最后一个 a替换为符号$。 现在我们在位置 $2, 3, 4$等处有一个 a。 ，p -1和$在位置 p。

$$\vdash \sqcup\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \$\ \sqcup\ \sqcup\ \sqcup \cdots$$

现在我们重复以下循环。 从左边界标记 $\vdash$开始，向右扫描并找到第一个非空白符号，假设出现在位置 m。 那么 m是素数（这是循环的不变量）。 如果这个符号是$，我们完成了： $p = m$是素数，所以我们停止并接受。 否则，这个符号是一个 a。 用一个标记它，并将它和左边界标记之间的所有内容标记为´。

$$\vdash \acute{a}\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \$\ \sqcup\ \sqcup\ \sqcup \cdots$$

现在我们进入一个内部循环，删除所有出现在 m的倍数位置上的符号。
首先，删除.下的 a。 （形式上，只需写下符号 ␣。）

$$\vdash \acute{a}\ \sqcup\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \$\ \sqcup\ \sqcup\ \sqcup \cdots$$

将标记逐个向右移动一个与标记数量相等的距离。 这可以通过来回穿梭、擦除左边的标记并将其写到右边来完成。 我们知道完成时是因为这是最后一个移动的标记。

$$\vdash \sqcup\ \sqcup\ \acute{a}\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \$\ \sqcup\ \sqcup\ \sqcup \cdots$$

完成后，擦除下方的符号。 这是出现在位置2的符号m。

$$\vdash \sqcup \sqcup \acute{a} \sqcup a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ \$ \sqcup \sqcup \sqcup \cdots$$

以这种方式持续移动标记并擦除下方的符号，直到达到末尾。

$$\vdash \llcorner\lrcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner \text{axyaxy}\ \acute{}\$ \llcorner$$

如果我们发现自己在字符串末尾想要擦除$，则拒绝—$p$是 m的倍数但不等于 m。 否则，回到左边并重复。找到第一个非空白符号并标记它及其左边的所有内容。

$$\vdash \acute{\llcorner}\lrcorner \acute{\llcorner}\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner a \llcorner\lrcorner \text{axy}\ \$ \llcorner\lrcorner$$

交替擦除符号并移动标记，直到达到字符串的末尾。

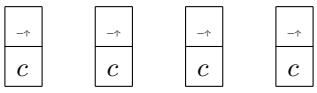
$$\vdash \sqcup \sqcup \sqcup \sqcup a \sqcup a \sqcup \sqcup \sqcup a \sqcup a \sqcup \sqcup \sqcup a \sqcup a \sqcup \sqcup \acute{\sqcup} \acute{\$} \sqcup \sqcup \sqcup \cdots$$

回到左边并重复。

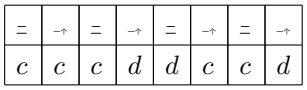
$$\vdash \acute{\sqcup} \acute{\sqcup} \acute{\sqcup} \acute{\sqcup} a \sqcup a \sqcup \sqcup \sqcup a \sqcup a \sqcup \sqcup \sqcup a \sqcup a \sqcup \sqcup \sqcup \$ \sqcup \sqcup \sqcup \cdots$$

如果 M不是总的，这个构造不能给出补集。这是因为“拒绝”和“不接受”对于非总机器来说并不是同义词。 要拒绝，机器必须进入其拒绝状态。 如果M'是通过仅仅交换接受和拒绝状态从 M获得的，那么 M'将接受M拒绝的字符串，并拒绝M接受的字符串；但是M′仍然会在M循环的相同字符串上循环，所以这些字符串既不被任何机器接受也不被拒绝M拒绝的字符串并拒绝 M接受的字符串；但是 M ′ 仍然会在 M循环的相同字符串上循环，所以这些字符串既不被任何机器接受也不被拒绝。

每个递归集合都是可枚举的，但反之不一定。 换句话说，并不是每个图灵机都等价于一个总图灵机。我们将在第4节中证明这一点。 然而，如果 A和 $\sim A$都是可枚举的，则 A是递归的。 为了看到这一点，假设 A和 $\sim A$都是可枚举的。让 M和 M'是这样的图灵机，使得 $L(M) = A$和 $L(M') = \sim A$。 构建一个新的机器 N，对于输入 x同时运行 M和 M ′ 同时在其磁带的两个不同轨道上进行。形式上，N的磁带字母表包含符号



其中 a是M的磁带符号， c是M'的磁带符号。 因此，N的磁带可能包含一个字符串的形式



例如。 额外的标记放置在磁带上，以指示模拟的读/写头的当前位置M和M'。 机器N交替执行M的一步和M'的一步，来回穿梭在M和M'的两个模拟磁带头位置之间，并更新磁带。 M和M的当前状态和转移信息

′ 可以存储在 N的有限控制器中。 如果机器M曾经接受过，则 N立即接受。 如果 M'曾经接受过，则 N立即拒绝。 这两个事件中的一个必定最终发生，取决于 $x \in A$还是 $x \in \sim A$，因为 $L(M) = A$和$L(M') = \sim A$。 然后 N在所有输入上停机，并且 $L(N) = A$。

递归和可枚举集合

回想一下，如果一个集合 A被图灵机接受，则它是可递归可枚举的（r.e.）；如果它被一个总图灵机接受（即对所有输入都停机），则它是可递归的。

可递归集合在补集下是封闭的（我们稍后会看到，可枚举集合则不是）。也就是说，如果 A是可递归的，则 $\sim A = \Sigma^* - A$也是可递归的。为了证明这一点，假设 A是可递归的。 那么存在一个总图灵机 M，使得 $L(M) = A$。 我们可以构建一个新的机器 M'，它与M完全相同，只是M′的接受状态是M的拒绝状态，而M的拒绝状态是M′的接受状 通过交换接受和拒绝状态，我们得到一个总机器 M ′，使得L(M') =□A。 ′ 是 M的接受状态。 通过交换接受和拒绝状态，我们得到一个总机器 M'，使得 $L(M')$ =□A。 ′)

如果 M不是总的，这个构造不能给出补集。这是因为“拒绝”和“不接受”对于非总机器来说并不是同义词。 要拒绝，机器必须进入其拒绝状态。 如果M'是通过仅仅交换接受和拒绝状态从 M获得的，那么 M'将接受M拒绝的字符串，并拒绝M接受的字符串；但是M′仍然会在M循环的相同字符串上循环，所以这些字符串既不被任何机器接受也不被拒绝M拒绝的字符串并拒绝 M接受的字符串；但是 M ′ 仍然会在 M循环的相同字符串上循环，所以这些字符串既不被任何机器接受也不被拒绝。

每个递归集合都是可枚举的，但反之不一定。 换句话说，并不是每个图灵机都等价于一个总图灵机。我们将在第4节中证明这一点。 然而，如果 A和 $\sim A$都是可枚举的，则 A是递归的。 为了看到这一点，假设 A和 $\sim A$都是可枚举的。让 M和 M'是这样的图灵机，使得 $L(M) = A$和 $L(M') = \sim A$。 构建一个新的机器 N，对于输入 x同时运行 M和 M ′ 同时在其磁带的两个不同轨道上进行。形式上，N的磁带字母表包含符号

其中 a是M的磁带符号， c是M'的磁带符号。 因此，N的磁带可能包含一个字符串的形式

例如。 额外的标记放置在磁带上，以指示模拟的读/写头的当前位置M和M'。 机器N交替执行M的一步和M'的一步，来回穿梭在M和M'的两个模拟磁带头位置之间，并更新磁带。 M和M的当前状态和转移信息

′ 可以存储在 N的有限控制器中。 如果机器M曾经接受过，则 N立即接受。 如果 M'曾经接受过，则 N立即拒绝。 这两个事件中的一个必定最终发生，取决于 $x \in A$还是 $x \in \sim A$，因为 $L(M) = A$和$L(M') = \sim A$。 然后 N在所有输入上停机，并且 $L(N) = A$。

可判定性和半可判定性

如果一个字符串的属性 P的集合是递归集合，则称该属性 P是可判定的；换句话说，如果存在一个总图灵机，它接受具有属性 P的输入字符串并拒绝没有该属性的字符串。如果一个字符串的属性 P的集合是可枚举集合，则称该属性 P是半可判定的；换句话说，如果存在一个图灵机，在输入 x后，如果 x具有属性 P，则接受，否则拒绝或循环。例如，给定一个字符串 x是否为 ww的形式是可判定的，因为我们可以构造一个在所有输入上都停机并且仅接受该形式字符串的图灵机。

尽管你经常听到它们被颠倒使用，形容词递归和可枚举最好用于集合，而可判定和半可判定用于属性。 这两个概念是等价的，因为

$$P \text{ 是可判定的} \quad \Leftrightarrow \quad \{x \mid P(x)\} \text{ 是递归的。}$$
$$A \text{ 是递归的} \quad \Leftrightarrow \quad \text{“}x \in A\text{” 是可判定的。}$$

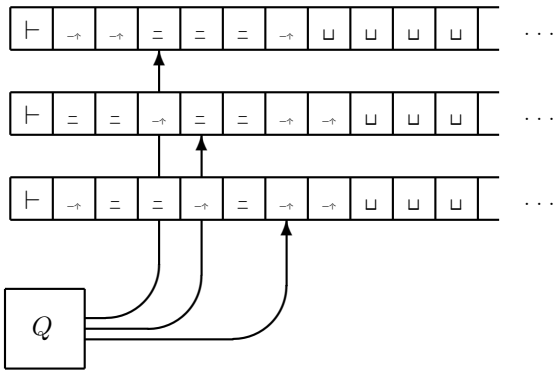
$$P\text{是半可判定的} \quad \Leftrightarrow \quad \{x \mid P(x)\}\text{是可枚举的，}$$
$$A\text{是可枚举的} \quad \Leftrightarrow \quad \text{“}x \in A\text{” 是半可判定的。}$$

3个等价模型

正如提到的，可计算性的概念非常强大。 作为证据，我们将介绍几种看起来比基本模型在第2节中定义的模型更强大或更弱的图灵机，但实际上它们在计算上是等价的。

多带图灵机

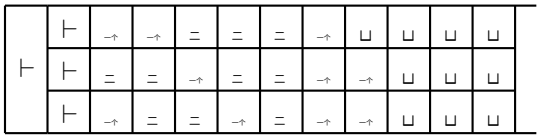
首先，我们展示如何用单带图灵机模拟多带图灵机。 因此，额外的带不会增加任何能力。 三带机器类似于单带图灵机，只是它有三个半无限带和三个独立的读/写头。 初始时，输入占据第一带，其他两个带为空白。 在每一步中，机器读取头下的三个符号，并根据这些信息和当前状态，在每个带上打印一个符号，移动头（它们不必都朝着同一个方向移动），并进入一个新状态。



它的转换函数的类型是

$$\delta : Q \times \Gamma^3 \quad \to \quad Q \times \Gamma^3 \times \{L, R\}^3.$$

假设我们有一台这样的机器 M. 我们构建一台单带机器 N来模拟 M，具体如下。 机器 N将具有扩展的纸带字母表，使我们可以将其纸带划分为三个轨道。 每个轨道将包含 M的一个纸带的内容。 我们还在每个轨道上标记一个符号，以指示该符号当前正在 M的相应纸带上进行扫描。 上面所示的 M的配置可以通过 N的以下配置来模拟。



一个 N的纸带符号可以是 $\vdash$，Σ中的一个元素，或者是一个三元组

其中c, d, e是 M的纸带符号，可以是标记的或未标记的。 形式上，我们可以将 N的纸带字母表定义为

$$\Sigma \cup \{\vdash\} \cup (\Gamma \cup \Gamma')^3,$$

其中

$$\Gamma' \overset{\text{定义}}{=} \{c \mid c \in \Gamma\}.$$

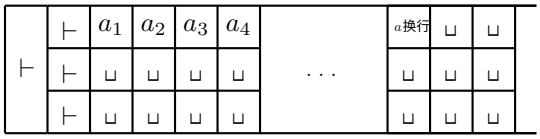
$\Gamma \cup \Gamma'$的三个元素代表 M的三个磁带上对应位置的符号，可以是标记的或未标记的，以及

是 N的空白符号。

在输入 $x = a_1 a_2 \cdots a_n$时， N从磁带内容开始

$$\vdash\ a_1\ a_2\ a_3\ \cdots\ a_n\ \sqcup\ \sqcup\ \sqcup\ \cdots.$$

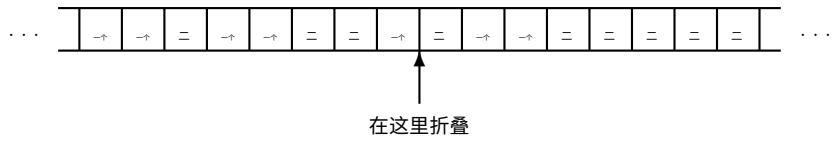
它首先将输入复制到顶部磁带，并用空白符号填充底部两个磁带。 它还将所有内容向右移动一个单元，以便可以在三个磁带的最左侧单元中填充M的模拟左端标记，它用表示 M的起始配置中头的位置。

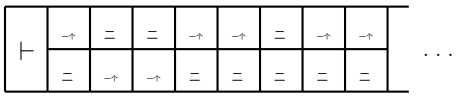


M的每一步都由N的几步模拟。 为了模拟M的一步，N从磁带的左边开始扫描，直到看到所有三个标记，记住有标记的符号。 当它看到所有三个标记时，根据M的转移函数δ决定要做什么，这个函数已经编码在它的有限控制中。 根据这些信息，它回到所有三个标记，重新写入每个轨道上的符号，并适当地移动标记。然后它返回到磁带的左端，模拟M的下一步。

双向无限磁带

双向无限磁带并没有增加任何能力。 只需在某个地方折叠磁带，并在一维无限磁带的两个轨道上模拟它：

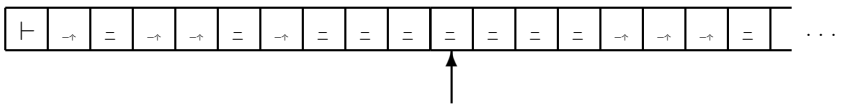




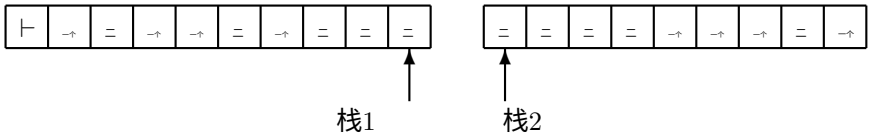
当头部在折叠的右侧时，底部轨道用于模拟原始机器，而顶部轨道用于模拟头部在折叠的左侧时，以相反方向移动的机器。

两个栈

一个具有双向、只读输入头和两个栈的机器与图灵机一样强大。 直观地说，通过将带子内容存储在一个栈的头部左侧，将带子内容存储在另一个栈的头部右侧，可以模拟单带图灵机的计算。 通过从一个栈中弹出一个符号并将其推入另一个栈来模拟头部的移动。例如，



通过模拟实现



计数自动机

k计数自动机是一种配备双向只读输入头和k个整数计数器的机器。
每个计数器可以存储任意非负整数。 在每一步中，自动机可以独立地增加或减少其计数器，并测试它们是否为0，并且可以将其输入头向任意方向移动一个单元。 它不能在带子上写入。

可以通过以下方式模拟栈。 我们可以假设无损失地模拟要模拟的栈的栈字母表只包含两个符号，比如0和1。 这是因为我们可以将有限多个栈符号编码为固定长度的二进制数，比如m；然后通过推入或弹出m个二进制位来模拟推入或弹出一个栈符号。 然后，栈的内容可以被视为一个二进制数，其最低有效位位于栈的顶部。 模拟将此数字保持在两个计数器中的第一个计数器中，并使用第二个计数器执行栈操作。 要模拟将0推入栈中，我们需要将第一个计数器的值加倍。 这可以通过进入一个循环来实现，该循环重复从第一个计数器中减去1并向第二个计数器中加上2，直到第一个计数器为0。

第二计数器中的值是第一个计数器中原始值的两倍。 然后我们可以将该值转移到第一个计数器中，或者只是交换两个计数器的角色。 要推1，操作是相同的，只是在最后将第二计数器的值增加一次。要模拟弹出，我们需要将计数器值除以二；这是通过在每个第二步时递减一个计数器并递增另一个计数器来完成的。 测试原始计数器内容的奇偶性可以判断模拟的1或0是否被弹出。

由于两个堆栈机器可以模拟任意图灵机，并且两个计数器可以模拟一个堆栈，因此可以得出一个四计数器自动机可以模拟任意图灵机。

然而，我们可以做得更好：一个两计数器自动机可以模拟一个四计数器自动机。 当四计数器自动机的计数器中具有值i，j，k，ℓ时，两计数器自动机的第一个计数器中将具有值$2^i3^j5^k7^\ell$。 它使用第二计数器来实现四计数器自动机的计数器操作。 例如，如果四计数器自动机想要将 k（第一个计数器的值）加一

第三计数器），那么双计数器自动机就必须将其第一个计数器的值乘以5。这与上述方式相同，每当我们从第一个计数器中减去1时，就向第二个计数器加上5。为了模拟零的测试，双计数器自动机必须确定其第一个计数器中的值是否可以被2、3、5或7整除，具体取决于四计数器自动机中的哪个计数器正在进行测试。

通过这些模拟，我们可以看到双计数器自动机与任意图灵机具有相同的能力。
然而，正如你可以想象的那样，双计数器自动机模拟一步图灵机需要大量的步骤。

单计数器自动机不如任意图灵机强大。

枚举机

我们将可递归枚举（r.e.）集合定义为那些被图灵机接受的集合。 术语“可递归枚举”来自于一个不同但等价的形式体系，体现了一个想法，即r.e.集合的元素可以以机械方式逐个枚举。

将枚举机器定义为如下。 它有一个有限控制和两个带子，一个读/写工作带和一个只写输出带。 工作带头可以向任何方向移动，并且可以读取和写入Γ的任何元素。 输出带头在写入符号时向右移动一个单元格，并且只能写入Σ中的符号。 没有输入，也没有接受或拒绝状态。 机器从两个带子都为空的起始状态开始。 它根据其转移函数移动，就像图灵机一样，根据转移函数确定时，偶尔在输出带上写入符号。 在某些时候，它可能进入一个特殊的枚举状态，这只是其有限控制的一个特殊状态。 当发生这种情况时，当前写在输出带上的字符串被称为被枚举。 然后，输出带会自动擦除，并将输出头移回到带的开头（工作带保持不变），机器从那一点继续。 机器永远运行。 集合 $L(E)$被定义为Σ^*中所有被枚举机器 E枚举的字符串的集合。 机器可能永远不会进入其枚举状态，在这种情况下 $L(E) = \varnothing$，或者它可能枚举无限多个字符串。 同一个字符串可能被枚举多次。

枚举机和图灵机在计算能力上是等价的：

定理**1**. 被枚举机枚举的集合族正好是可列集族。. 换句话说，一个集合是 $L(E)$对于某个枚举机 E当且仅当它是 $L(M)$对于某个图灵机 M。

证明。我们首先展示，给定一个枚举机 E，我们可以构造一个图灵机 M使得$L(M) = L(E)$。 在输入 x上，M将 x复制到其带子上的三个轨道之一，然后模拟 E，使用另外两个轨道记录 E的工作带和输出带的内容。 对于 E枚举的每个字符串， M将该字符串与 x进行比较，如果它们匹配，则接受。 因此， M接受输入 x当且仅当 x被E枚举，所以 M接受的字符串集合正好是 E枚举的字符串集合。

相反地，给定一个图灵机 M，我们可以构造一个枚举机 E，使得 $L(E) = L(M)$。 我们希望以某种方式模拟 M在Σ^*中的所有可能字符串，并枚举那些被接受的字符串。

这里有一种方法，但并不完全有效。 枚举机 E按某种顺序逐个在其工作带的底部轨道上写下Σ^*中的字符串。 对于每个输入字符串 x，它在其工作带的顶部轨道上模拟 M的输入 x。 如果 M接受 x， E将 x复制到其输出带并进入枚举状态。 然后它继续下一个字符串。

这个过程的问题在于 M可能不会在某些输入 x上停机，然后 E将永远在模拟 M在 x上停机之前不会继续进行列表中的后续字符串（而一般情况下无法确定 M是否会在 x上停机，我们将在第4节中看到）。 因此， E不应该只是按某种顺序列出Σ^*中的字符串并逐个模拟 M在这些输入上，等待每个模拟停机后再继续下一个，因为模拟可能永远不会停机。

这个问题的解决方案是分时。 不是逐个模拟 M在输入字符串上，

枚举机器 E应该同时运行几个模拟，每个模拟工作几步然后转移到下一个。 机器 E的工作带可以被分成由特殊分隔符标记 $\# \in \Gamma$ 分隔的段，每个段中运行着 M的一个不同的输入字符串的模拟。 在每次通过之间，E可以向右移动，创建一个新的段，并在下一个输入字符串上的该段中启动一个新的模拟。 例如，我们可以让 E在第一个输入上模拟 M一步，然后在第一个和第二个输入上各模拟一步，然后在第一个、第二个和第三个输入上各模拟一步，依此类推。如果任何模拟需要比其段中最初分配的空间更多的空间，则可以将其右侧的整个带内容向右移动一个单元。 通过这种方式， M最终在所有输入字符串上被模拟，即使其中一些模拟永远不会停止。

□

历史注释

图灵机是由Alan Turing [20]发明的。 最初，它们以枚举机的形式呈现，因为图灵对可计算实数的十进制展开和实值函数的值进行枚举感兴趣。 图灵在他的原始论文中还引入了非确定性的概念，尽管他没有深入发展这个想法。

r.e.集合的基本属性由Kleene [9]和Post [13, 14]发展出来。

Fischer [5]，Fischer et al. [6]和Minsky [11]研究了计数自动机。

4个通用机器和对角化

通用图灵机

现在我们来观察一下图灵机的能力：存在一些图灵机可以模拟其他图灵机，这些图灵机的描述是作为输入的一部分呈现的。 这并没有什么神秘的，就像在OCaml中编写一个OCaml解释器一样。

首先，我们需要为字母表为$\{0, 1\}$的图灵机确定一个合理的编码方案。 这个编码方案应该足够简单，以至于另一台机器可以通过读取M的编码描述轻松确定与机器M相关的所有数据，包括状态集、转移函数、输入和纸带字母表、结束标记、空白符号以及开始、接受和拒绝状态。 例如，如果字符串以前缀开头

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 0^v 1,$$

这可能表示机器有 n个状态，用数字0到 $n-1$表示；它有 m个纸带符号，用数字0到 $m-1$表示，其中第一个 k表示输入符号；起始、接受和拒绝状态分别是 s、 t和 r；结束标记和空白符号分别是 u和 v。 字符串的其余部分可以由一系列子字符串组成，用于指定转换在 δ中。 例如，子字符串

$$0^p 1 0^a 1 0^q 1 0^b 1 0$$

可能表示 δ包含转换

$$(p, a) \to (q, b, L),$$

头部移动方向由最后一位数字编码。 编码方案的具体细节并不重要。 唯一的要求是它应该易于解释，并能够编码所有的图灵机同构。

一旦我们有了图灵机的适当编码，我们就可以构造一个通用图灵机 U，使得

$$L(U) \stackrel{\text{定义}}{=} \{M \# x \mid x \in L(M)\}^\circ$$

换句话说，给定（一个关于 $\{0,1\}$的编码）的图灵机 M和（一个关于 $\{0,1\}$的编码）的字符串 x，属于 M的输入字母表，机器 U接受 $M\#x$当且仅当 M接受 x。[1] 符号 $\#$ 只是 U的输入字母表中除了0和1之外的一个符号，用于分隔 M和 x。

机器 U首先检查其输入 $M\#x$，以确保 M是一个有效的图灵机编码 和 x是一个有效的属于 M的输入字母表的字符串。 如果不是，它立即拒绝。

如果 M和 x的编码是有效的，机器 U会对 M进行逐步模拟。 这可能会按照以下方式工作。 机器 U的磁带被分成三个轨道。 将 M的描述复制到顶部轨道上，将字符串 x复制到中间轨道上。 中间轨道将用于保存 M 的模拟磁带内容。 底部轨道将用于记住 M的当前状态和当前读/写头的位置。 然后，机器 U逐步模拟 M在输入 x上的运行，来回在顶部轨道上的 M描述和中间轨道上的模拟磁带内容之间穿梭。

在每一步中，它根据 M的转移函数更新 M的状态和模拟磁带内容，这些函数可以从 M的描述中读取。 如果 M停机并接受或停机并拒绝，那么U也会做同样的操作。

正如我们观察到的，字符串 x在 M的输入字母表上和在U的输入字母表上的编码是两个不同的概念，因为这两个机器可能具有不同的输入字母表。 如果 M的输入字母表比 U的输入字母表大，那么 x的每个符号必须被编码为 U的输入字母表上的符号串。 此外， M的纸带字母表可能比 U的纸带字母表大，这种情况下， M 的每个纸带字母必须被编码为 U的纸带字母表上的符号串。 一般来说， M的每一步可能需要模拟 U的多个步骤。

对角线化

我们现在展示如何使用通用图灵机结合一种称为对角线化的技术来证明图灵机的停机问题和成员问题是不可判定的。换句话说，集合

$$\begin{aligned} \mathrm{HP} &\text{ 定义 } \{M\#x \mid M\text{在 } x\text{上停机}\}, \\ \mathrm{MP} &\stackrel{\mathrm{def}}{=} \{M\#x \mid x \in L(M)\} \end{aligned}$$

不是递归的。

对角线化技术首次由康托在19世纪末使用，用来证明自然数 $\mathbb{N}$和其幂集之间不存在一一对应关系。

$$2^{\mathbb{N}} = \{A \mid A \subseteq \mathbb{N}\},$$

是 $\mathbb{N}$的所有子集的集合。 事实上，甚至不存在一个函数

$$f : \mathbb{N} \to 2^{\mathbb{N}}$$

是满射的。 以下是康托的论证过程。

假设（用于推导矛盾）存在这样一个满射函数 f。 考虑一个无限二维矩阵，顶部以自然数$0,1,2,\ldots$为索引。左侧以集合 $f(0), f(1), f(2), \ldots$为索引。

[1] 注意我们同时使用元符号 M表示图灵机及其在 $\{0,1\}$上的编码，以及元符号 x 表示字符串及其在 M的输入字母表 $\{0,1\}$上的编码。 这是为了方便表示。

通过将1放置在位置i, j处，如果 j在集合 $f(i)$中，则填充矩阵，如果 $j \in f(i)$，则填充0。

	0	1	2	3	4	5	6	7	8	9	$\cdots$
$f(0)$	1	0	0	1	1	0	1	0	1	1	
$f(1)$	0	0	1	1	0	1	1	0	0	1	
$f(2)$	0	1	1	0	0	0	1	1	0	1	
$f(3)$	0	1	0	1	1	0	1	1	0	0	
$f(4)$	1	0	1	0	0	1	0	0	1	1	$\cdots$
$f(5)$	1	0	1	1	0	1	1	1	0	1	
$f(6)$	0	0	1	0	1	1	0	0	1	1	
$f(7)$	1	1	1	0	1	1	1	0	1	0	
$f(8)$	0	0	1	0	0	0	0	1	1	0	
$f(9)$	1	1	0	0	1	0	0	1	0	0	
$\vdots$					$\vdots$						$\ddots$

矩阵的第 i行是描述集合 $f(i)$的位串。 例如，在上面的图片中，
$f(0) = \{0, 3, 4, 6, 8, 9, \ldots\}$ 和 $f(1) = \{2, 3$ $, 5, 6, 9, \ldots\}$。根据我们（即将被证明是错误的）假设，每个 $\mathbb{N}$的子集都出现在这个矩阵的一行中。

但是我们可以通过对矩阵的主对角线进行补集操作来构造一个在列表中不存在的新集合（因此称为对角化）。 观察沿着主对角线的无限位串（在这个例子中，为$1011010010\cdots$），并取其布尔补集（在这个例子中，为$0100101101\cdots$）。 这个新的位串表示一个集合 B（在这个例子中， $B = \{1, 4, 6, 7, 9, \ldots\}$）。 但是集合 B在矩阵的左侧列表中没有出现过，因为它与每个 $f(i)$至少在一个元素上不同。 这是一个矛盾，因为根据我们假设 f是满射的，每个 $\mathbb{N}$的子集都应该作为矩阵的一行出现。

这个论证不仅适用于自然数 $\mathbb{N}$，而且适用于任意集合 A。 假设（为了推导出矛盾）存在一个从集合 A到其幂集的满射函数：

$$f : A \rightarrow 2\ A\text{。}$$

令

$$B = \{x \in A \mid x \in f(x)\}$$

（这是对角线补集的形式化方式）。那么 $B \subseteq A$。 由于 f是满射，必然存在一个元素$y \in A$使得 $f(y) = B$。 现在我们询问是否 $y \in f(y)$ 并得到一个矛盾：

$$\begin{aligned} y \in f(y) \quad &\Leftrightarrow \quad y \in B \qquad \text{因为 } B = f(y) \\ &\Leftrightarrow \quad y \in f(y) \text{ 定义 } B\text{的方式。} \end{aligned}$$

因此不存在这样的 f。

停机问题的不可判定性

我们已经讨论了如何将图灵机的描述编码为字符串，使得这些描述可以被一个通用图灵机 U读取和模拟。 机器 U接受一个图灵机 M和一个字符串 x的编码作为输入，并模拟 M在输入 x上的运行。

- 如果 M停机并接受 x，则U停机并接受。
- 如果 M停机并拒绝 x，则U停机并拒绝。
- 如果 M在 x上循环，则U也会循环。

机器 U不对机器 M进行任何复杂的分析，以确定它是否会停机。 它只是盲目地逐步模拟 M。 如果 M在 x上不停机，那么 U将继续快乐地模拟 M。

自然而然地会想，我们是否能做得比盲目模拟更好呢？ 也许有一种方法可以在模拟之前分析M，以确定M在x上是否最终会停机?
如果U能够确定M在x上不会停机，那么它可以跳过模拟并节省很多无用的工作。 另一方面，如果U能够确定M在x上最终会停机，那么它可以继续进行模拟以确定M是接受还是拒绝。 然后，我们可以构建一个名为U'的机器，它接受一个图灵机M和一个字符串x的编码作为输入，并且

- 如果 M停机并接受 x，则U停机并接受。
- 如果 M停机并拒绝 x，则U停机并拒绝。
- 如果M在x上循环，则停机并拒绝。

这将表示$L(U') = L(U) = \mathrm{MP}$是一个递归集合。
不幸的是，一般情况下这是不可能的。肯定有一些机器可以通过某种启发式方法确定停机：例如，起始状态就是接受状态的机器。 然而，没有一种通用的方法可以对所有机器给出正确的答案。

我们可以使用康托对角线技术来证明这一点。 对于 $x\in\{0,1\}*$，让 M_x是输入字母表为 $\{0,1\}$的图灵机，其编码为 x。 （如果 x不是根据我们的编码方案是一个合法的TM描述，我们将 M_x视为某个任意但固定的TM，例如一个立即停止的简单TM。）通过这种方式我们得到一个列表

$$M_\varepsilon,\, M_0,\, M_1,\, M_{00},\, M_{01},\, M_{10},\, M_{11},\, M_{100},\, M_{101},\, \ldots \tag{4}$$

包含了所有可能的图灵机，输入字母表为 $\{0,1\}$，以 $\{0,1\}*$为索引的字符串。 我们确保编码方案足够简单，以便通用机器可以根据输入 M_x从中确定 x以进行模拟。

现在考虑一个无限的二维矩阵，顶部以 $\{0,1\}*$的字符串为索引，左侧以列表(4)中的图灵机为索引。

如果 M_x在输入 y上停机，则矩阵中的位置$x,\, y$上有一个H，如果 M_x在输入 y上循环，则有一个L。

	ε	0	1	00	01	10	11	000	001	010	$\cdots$
M_ε	H	L	L	H	H	L	H	L	H	H	
M_0	L	L	H	H	L	H	H	L	L	H	
M_1	L	H	H	L	L	L	H	H	L	H	
M_{00}	L	H	L	H	H	L	H	H	L	L	
M_{01}	H	L	H	L	L	H	L	L	H	H	$\cdots$
M_{10}	H	L	H	H	L	H	H	H	L	H	
M_{11}	L	L	H	L	H	H	L	L	H	H	
M_{000}	H	H	H	L	H	H	H	L	H	L	
M_{001}	L	L	H	L	L	L	L	H	H	L	
M_{010}	H	H	L	L	H	L	L	H	L	L	
$\vdots$					$\vdots$						$\ddots$

矩阵的第 x行描述了对于每个输入字符串 y， M_x是否停机。 例如，在上面的图片中， M_ε在输入 $\varepsilon, 00, 01, 11, 001, 010$等上停机。而在输入$0, 1, 10, 000$等上不会停机。

假设（用于推导矛盾）存在一个接受集合HP的总机器K，即对于任意给定的x和y，能够在有限时间内确定上述表格的第x，y个条目。因此，在输入$M\#\ x$时，如果机器K接受，则存在一个机器N，它在输入$x\in${0,1}□时

- 如果M在x上停机并接受，则K停机并接受
- 如果M在x上循环，则K停机并拒绝

考虑一个机器N，它在输入$x \in \{0,1\}*$时

(i) 构造机器M_x并将$M\ x\#_x$写入其磁带；

(ii) 运行机器K在输入$M_x\#x$上，如果K拒绝，则接受，如果K接受，则进入一个平凡循环。

注意，N本质上是对上述矩阵的对角线进行补集操作。 那么对于任意的$x \in \{0,1\}*$，

$$\begin{aligned} N \text{ halts on } x &\Leftrightarrow K \text{ rejects } M_x\#x && \text{definition of } N \\ &\Leftrightarrow M_x \text{ loops on } x && \text{关于 } K\text{的假设。} \end{aligned}$$

这意味着 N的行为与每个字符串上的 M_x都不同，即 x。 但是列表 (4) 应该包含所有输入字母表 $\{0,1\}$上的图灵机，包括 N。这是一个矛盾。 □

导致矛盾的错误假设是可以有效地确定矩阵的条目；换句话说，存在一个图灵机 K，给定 M和 x可以在有限时间内确定 Mhalts on x的情况。

我们总是可以在给定的输入上模拟给定的机器。 如果机器最终停止，我们将最终知道这一点，并且我们可以停止模拟并说它停止了；但是如果不是，一般来说没有办法在有限时间内停止并确定它永远不会停止。

成员问题的不可判定性

成员问题也是不可判定的。 我们可以通过将停机问题归约为成员问题来证明这一点。换句话说，我们证明了如果有一种通用的成员判定方法，我们可以将其用作通用停机判定的子程序。但是我们刚刚证明了停机问题是不可判定的，所以成员问题也必须是不可判定的。

下面是我们如何使用一个判定成员的全局图灵机作为一个子程序来判定停机。 给定一个机器 M和输入 x，假设我们想要确定 M在 x上是否停机。 构建一个新的机器 N，它与 M完全相同，只是在 M接受或拒绝时都接受。 机器 N可以通过在 M中添加一个新的接受状态，并使旧的接受和拒绝状态转移到这个新的接受状态来构建。 那么对于所有的 x， N接受 x当且仅当 M在 x上停机。 因此，对于 N和 x的成员问题（询问 x是否属于 L（N））与 M和 x的停机问题（询问 M是否在 x上停机）是相同的。 如果成员问题是可判定的，那么我们可以通过构建 N并询问 x是否属于 L（N）来判定 M是否在 x上停机。 但是我们已经证明了停机问题是不可判定的，因此成员问题也必须是不可判定的。

5个可判定和不可判定的问题

下面是一些涉及图灵机的决策问题的例子。 给定一个图灵机，是否可判定?

(a) 是否至少有481个状态?

(b) 在输入 ε上是否需要超过481步?

(c) 在某些输入上是否需要超过481步?

(d) 在所有输入上是否需要超过481步?

(e) 在输入 ε上是否会将头移动超过481个磁带单元的距离?

(f) 是否接受空字符串 ε?

(g) 是否接受任意字符串?

(h) 是否接受每个字符串?

(i) 是否接受有限集?

(j) 是否接受递归集?

(k) 是否等价于一个描述更短的图灵机?

问题(a)到(e)是可判定的，问题(f)到(k)是不可判定的（证明如下）。 我们将证明问题(f)到(j)是不可判定的，通过展示一个这些问题的决策过程可以用来构建一个判定停机问题的过程，而我们知道这是不可能的。问题(k)稍微困难一些，我们将把它作为一个练习留下。 翻译成现代术语，问题(k)与确定是否存在一个等价于给定Java程序的更短程序是相同的。

展示一个问题是可判定的最好方法是给出一个完全的图灵机，它只接受“是”实例。 因为它必须是完全的，所以它也必须拒绝“否”实例；换句话说，它不能在任何输入上循环。

问题（a）很容易判定，因为可以从 M的编码中读取状态的数量。我们可以构建一个图灵机，给定其输入带上的 M的编码，计算 M的状态数量，并根据数量是否至少为481来接受或拒绝。

问题（b）是可判定的，因为我们可以用一个通用机器模拟 M在输入 ε上的运行，运行481步（在一个单独的轨道上计数），并根据 M是否在那个时间内停机来接受或拒绝。问题（c）是可判定的：我们可以

在481步内模拟 M在长度最多为481的所有输入上的运行。 如果M在某个输入上需要超过481步，则它在长度最多为481的某个输入上也需要超过481步，因为在481步内它最多只能读取输入的前481个符号。

对于问题（d）的论证类似。 如果 M在长度不超过481的所有输入上需要超过481步，则它在所有输入上都需要超过481步。

对于问题（e），如果 M从左边界标记开始从不移动超过481个纸带单元，那么它要么会停机，要么会以一种我们可以在有限时间内检测到循环的方式循环。 这是因为如果 M有 k个状态和 m个纸带符号，并且从左边界标记开始从不移动超过481个纸带单元，那么它可能处于的配置只有$482km^{481}$个，每个选择的头位置、状态和纸带内容都适合于481个纸带单元。 如果它在不移动超过481个纸带单元的情况下运行时间超过这个限制，那么它必定处于一个循环中，因为它必定重复了一个配置。 这可以通过模拟 M的机器来检测到，该机器在一个单独的轨道上计算 M所花费的步骤数，并在超过$482km^{481}$步时将 M声明为处于循环中。

问题（f）到（j）是不可判定的。 为了证明这一点，我们展示了决定任何一个这些问题的能力可以用来决定停机问题。 由于我们知道停机问题是不可判定的，这些问题也必须是不可判定的。这被称为归约。

让我们首先考虑（f）（尽管相同的构造也适用于（g）到（i））。 我们将展示是否给定的机器接受 ε是不可判定的，因为决定这个问题的能力将会给出决定停机问题的能力，而我们知道这是不可能的。

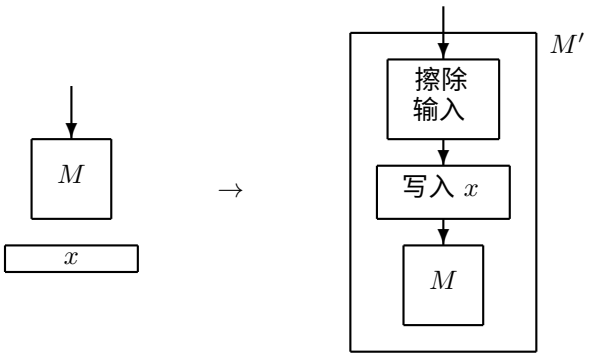
假设我们可以决定给定的机器是否接受 ε。 然后我们可以按照以下方式决定停机问题。 假设我们有一个图灵机 M和字符串 x，我们希望确定 M在x上是否停机。 从 M和 x构造一个新的机器 M'，在输入 y时执行以下操作：

(i) 擦除输入 y;

(ii) 在它的带子上写入 x（M'在其有限控制中已经将 $x$hard-wired）;

(iii) 在输入 x上运行 M（M'也在其有限控制中硬连线了 M的描述）;

(iv) 如果 M在 x上停机，则接受。



注意，对于所有输入 y, M'都会执行相同的操作：如果 M在 x上停机，则 M'接受其输入 y；如果 M在 x上不停机，则 M'在 y上也不会停机，因此不接受 y。 此外，对于每个y都成立。因此

$$L(M') = \begin{cases} \Sigma^* & \text{if } M\text{在 } x\text{上停机，} \\ \varnothing & \text{if } M\text{在 } x\text{上不停机。} \end{cases}$$

现在，如果我们能够决定一个给定的机器是否接受空字符串 ε，我们就可以将这个决策过程应用于刚刚构造的 M'，并且这将告诉我们 M在输入 x上是否停机。 换句话说，我们可以通过以下方式获得一个停机的决策过程：给定 M和 x，构造 M'，然后询问 M'是否接受 ε。 对于后一个问题的答案是“是”，当且仅当 M在输入 x上停机。 由于我们知道停机问题是不可判定的，因此判断一个给定的机器是否接受 ε也是不可判定的。

类似地，如果我们能够决定一个给定的机器是否接受任何字符串，或者是否接受每个字符串，或者它接受的字符串集合是有限的，我们就可以将任何这些决策过程应用于M'，并且这将告诉我们 M在输入 x上是否停机。 由于我们知道停机问题是不可判定的，所有这些问题也必定是不可判定的。

为了证明(j)是不可判定的，选择你最喜欢的可列但非递归集合 A(HP或MP都可以)，并按照以下方式修改上述构造。 给定 M和 x，构建一个新的机器 M''，在输入 y上执行以下操作：

(i) 将 y 保存在磁带的一个独立轨道上;

(ii) 在不同的轨道上写入 x（x在 M''的有限控制中是硬连线的）;

(iii) 在输入 x上运行 M（M也在 M''的有限控制中是硬连线的）;

(iv) 如果 M在输入 x上停机，则 M''在原始输入 y上运行一个接受 A的机器，并在该机器接受时接受。

如果 M不在输入 x上停机，则步骤(iii)中的模拟永远不会停机， M''永远不会接受任何字符串；或者如果 M在输入 x上停机，则 M''接受其输入 y当且仅当 $y \in A$。因此

$$L(M'') = \begin{cases} \text{如果 } M\text{在输入 } x\text{上停机} \\ \text{，则返回 } A\text{，否则返回}\varnothing\text{。} \end{cases}$$

由于 A不是递归的，而 $\varnothing$是递归的，如果能够判断给定的图灵机是否接受一个递归集合，那么就可以将这个决策过程应用于 M''，并且可以判断 M在输入 x上是否停机。

第6节 简化

有两种主要的技术可以证明问题是不可判定的：对角线化和简化。
我们在第4节看到了对角线化的例子，在第5节看到了简化的例子。

一旦我们确定了一个问题如HP是不可判定的，我们可以通过将HP简化为B来证明另一个问题B是不可判定的。 直观地说，这意味着我们可以通过操作HP的实例使它们看起来像问题B的实例，这样HP的“是”实例就变成了B的“是”实例，HP的“否”实例就变成了B的“否”实例。 虽然我们无法有效地判断给定的HP实例是否是“是”实例，但是这种操作保持了“是”和“否”的性质。 如果存在一个问题B的决策过程，那么我们可以将其应用于伪装的HP实例来判断其是否属于HP。换句话说，将问题B的决策过程与操作过程结合起来将得到一个HP的决策过程。由于我们已经证明了HP不存在这样的决策过程，因此我们可以得出结论，问题B不存在决策过程。

我们可以给出一个关于归约的抽象定义，并证明一个通用定理，从现在开始在不可判定性证明中节省了很多工作。

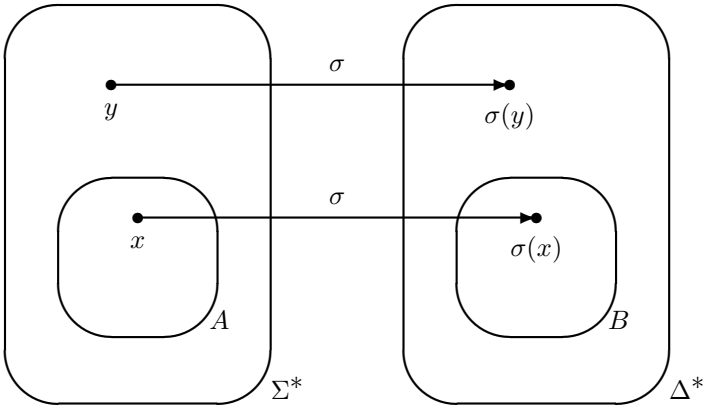
给定集合 $A \subseteq \Sigma^*$和 $B \subseteq \Delta^*$，一个（多对一）归约 A到 B的可计算函数

$$\sigma : \Sigma^* \quad \to \quad \Delta^*$$

使得对于所有的 $x \in \Sigma^*$,

$$x \in A \quad \Leftrightarrow \quad \sigma(x) \in B。 \tag{5}$$

换句话说，属于 A的字符串必须转换为属于 B的字符串，而不属于 A的字符串必须转换为不属于B的字符串。



函数 σ不一定是一对一或映满的。 然而，它必须是总的和有效计算的。 这意味着 σ必须由一个在任何输入 x上停机并在其磁带上写有 $\sigma(x)$的总图灵机计算。 当存在这样的归约时，我们说 A通过映射 σ可归约到 B，并且我们写作$A \leq_{\mathrm{m}} B$。 下标m代表“多对一”，用于区分此关系与其他类型的归约关系。

语言之间的可约性关系 $\leq$是传递的：如果 $A \leq B$且 $B \leq C$，则 $A \leq C$。 这是因为如果 σ将 A约化到 B， τ将 B约化到 C，那么 $\tau \circ \sigma$，即 σ和 τ的组合，是可计算的，并将 A约化到 C。

虽然我们没有明确提到，但在最近的几节中，我们使用了约化来证明各种问题是不可判定的。

例子4。通过展示给定的图灵机是否接受空字符串是不可判定的，我们构造了一个从给定的图灵机 M和字符串 x构建的图灵机 M'，当且仅当 M在 x上停机时，M'接受空字符串。在这个例子中，

$$\begin{aligned} A &= \{M\#x \mid M\text{在 } x\text{上停机}\} = \mathrm{HP}, \\ B &= \{M \mid \varepsilon \in L(M)\}, \end{aligned}$$

而 σ是可计算映射 $M\#x \to M'$。

例子**5**。 在证明给定的图灵机是否接受一个正则集合是不可判定的时候，我们构造了一个给定的图灵机 M 和字符串 x的图灵机 M''，使得 $L(M'')$是一个非正则集合，如果 M在 x上停机，否则是 $\varnothing$。在这个例子中，

$$\begin{aligned} A &= \{M\#x \mid M\text{在 } x\text{上停机}\} = \mathrm{HP}, \\ B &= \{M \mid L(M)\text{是正则的 }\}, \end{aligned}$$

而 σ是可计算映射 $M\#x \to M''$。

这里有一个通用定理可以节省我们一些工作。

定理**2**。

(i) 如果 $A \leq_{\mathrm{m}} B$且 B是可枚举的，那么 A也是可枚举的。 等价地，如果 $A \leq_{\mathrm{m}} B$且 A不可枚举，那么 B也不可枚举。

(ii) 如果 $A \leq_{\mathrm{m}} B$且 B是递归的，那么 A也是递归的。 等价地，如果 $A \leq_{\mathrm{m}} B$且 A不是递归的，那么 B也不是递

证明。(i) 假设通过映射 σ， $A \leq_{\mathrm{m}} B$且 B是可枚举的。让 M是一台图灵机，使得 $B = L(M)$。 构建一台机器 N来处理 A，方法如下：对于输入 x，首先计算 $\sigma(x)$，然后在输入 $\sigma(x)$ 上运行 M，如果 M接受，则N接受。然后

$$\begin{aligned} \text{当且仅当}N\text{接受 } x &\Leftrightarrow M\text{接受 } \sigma(x)\text{，} && N\text{的定义} \\ &\Leftrightarrow \sigma(x) \in B && M\text{的定义} \\ &\Leftrightarrow x \in A && \text{根据 (5)。} \end{aligned}$$

(ii) 从第二讲中回顾，一个集合是递归的当且仅当它和它的补集都是可枚举的。假设 $A \leq$M B通过映射 σ和 B是递归的。 注意，通过相同的映射 σ， $\sim A \leq$M $\sim B$（检查定义！）。 如果 B是递归的，那么 B和 $\sim B$都是可枚举的。根据（i）， A和 $\sim A$都是可枚举的，因此 A是递归的。 □

我们可以使用定理2（i）来证明某些集合不是可枚举的，并使用定理2（ii）来证明某些集合不是递归的。要证明一个集合 B不是可枚举的，我们只需要从我们已经知道不是可枚举的集合 A（例如 $\sim$HP）到 B的归约。 根据定理2（i）， B不能是可枚举的。例子**6**。 让我们通过展示集合FIN $= \{M \mid L(M)$是有限的$\}$来说明既不是可枚举的。

它的补集也不是可枚举的。我们通过将 $\sim$HP归约到它们中的每一个来证明这两个集合都不是可枚举的，其中

$$\sim\mathrm{HP} = \{M\#x \mid M\text{在 } x\text{上不会停机}\} :$$

(a) $\sim\mathrm{HP} \leq_{\mathrm{m}} \mathrm{FIN}$,

(b) $\sim\mathrm{HP} \leq_{\mathrm{m}} \sim\mathrm{FIN}$.

由于我们已经知道 $\sim$HP不是可枚举的，根据定理2(i)，FIN和 $\sim$FIN也不是可枚举的。

对于(a)，我们想要给出一个可计算的映射 σ ，使得

$$M\#x \in \sim\mathrm{HP} \quad \Leftrightarrow \quad \sigma(M\#x) \in \mathrm{FIN}.$$

换句话说，从 $M\#x$我们想要构造一个图灵机 $M' = \sigma(M\#x)$使得

$$M\text{在 } x\text{上不会停机} \;\Leftrightarrow\; L(M')\text{是有限的.} \qquad (6)$$

注意，对于 M'的描述可能取决于 M和 x。 特别地，如果需要， M'可以在其有限控制中硬编码 M和字符串 x的描述。

实际上，我们已经给出了满足（6）的构造。 给定 $M\#x$，构造 M'，使得对于所有输入 y， M'执行以下操作：

(i) 擦除输入 y;

(ii) 在它的带子上写入 x（M'在其有限控制中已经将 $x$hard-wired）;

(iii) 在输入 x上运行 M（M'也在其有限控制中硬连线了 M的描述）;

(iv) 如果 M在 x上停机，则接受。

如果 M在输入 x上不停机，则步骤（iii）中的模拟永远不会停机， M'永远不会达到步骤（iv）。 在这种情况下， M'不接受其输入 y。 对于所有输入 y，都会发生同样的情况，因此在这种情况下， $L(M) = \varnothing$。 另一方面，如果 M在 x上停机，则步骤（iii）中的模拟停机，并且 y在步骤（iv）中被接受。 此外，对于所有 y都成立。 在这种情况下， $L(M) = \Sigma^*$。 因此，如果M在 x上停机，则 $L(M') = \Sigma^* \Rightarrow$ $L(M')$ 是无限的。

$$M\text{不会在 } x \text{ 上停机} \;\Rightarrow\; L(M') = \varnothing \;\Rightarrow\; L(M') \text{ 是有限的。}$$

因此(6)被满足。 请注意，这就是我们需要做的来证明FIN不是可枚举的：我们已经给出了归约(a)，所以根据定理2(i)，我们完成了。

这里有一个常见的陷阱，我们应该小心避免。 重要的是要观察到，可计算映射 σ从 M和 x产生一个 M'的描述，不需要执行程序(i)到(iv)。 它只是产生一个机器 M'的描述，该机器执行(i)到(iv)。计算 σ非常简单——它不涉及模拟任何其他机器或任何复杂的东西。它只是将图灵机 M和字符串 x插入到执行(i)到(iv)的通用描述中。 这可以很容易地通过一个总图灵机完成，所以 σ是总的和有效计算的。

现在 (b). 根据归约的定义，将 $\sim$HP 归约到 $\sim$FIN 的映射也将 HP 归约到 FIN，因此只需要提供一个可计算的映射 τ，使得

$$M\#x \in \text{HP} \quad \Leftrightarrow \quad \tau(M\#x) \in \text{FIN}.$$

换句话说，我们希望从 M和 x构造一个图灵机 $M'' = \tau(M\#x)$，使得

$$M\text{在 } x\text{上停机} \;\Leftrightarrow\; L(M'') \text{ 是有限的.} \tag{7}$$

给定 $M\#x$，构造一个机器 M''，在输入 y时

(i) 将 y保存在一个单独的轨道上;

(ii) 在磁带上写入 x;

(iii) 模拟 M在 x上的 $|y|$步骤（它在模拟 M在 x上的每一步时擦除 y的一个符号）;

(iv) 如果 M在那个时间内没有停机，则接受；否则拒绝。

现在，如果 M在 x上永远不会停机，则在模拟的第(iv)步中， M''在经过 $|y|$步后停机并接受y，对于所有的 y都成立。 在这种情况下， $L(M'') = \Sigma^*$。 另一方面，如果 M在 x上停机，则在某个有限步数后停机，假设为 n。 然后，在(iv)中，如果 $|y| < n$（因为(iii)中的模拟还没有在 $|y|$步完成）， M''接受y；如果 $|y| \geq n$（因为(iii)中的模拟有足够的时间完成）， M''拒绝y。 在这种情况下，M''接受长度小于 n的所有字符串，并拒绝长度为 n或更长的所有字符串，因此 $L(\mathsf{M''})$ 是一个有限集。因此

$$\begin{aligned} M\text{在}x\text{上停机} \;&\Rightarrow\; L(M'') = \{y \mid |y| < M\text{在}x\text{上的运行时间}\} \\ &\Rightarrow\quad L(M'')\text{是有限的,} \end{aligned}$$

$$\begin{aligned} M\text{在}x\text{上不停机} \;&\Rightarrow\; L(M'') = \Sigma^* \\ &\Rightarrow\quad L(M'')\text{是无限的.} \end{aligned}$$

那么(7)成立.

重要的是这两个规约中的函数 σ和 τ可以由总是停机的图灵机计算出来.

历史注释

对角线化技术首先由康托尔[1]用来证明实数代数比实数少.

通用图灵机和康托尔的对角线化技术应用于证明停机问题的不可判定性出现在图灵的原始论文[20]中.

可归约关系由Post [14]讨论；参见[17, 19]。

7 Rice定理

Rice定理说的是不可判定性是规则，而不是例外。 这是一个非常强大的定理，包含了我们之前看到的许多不可判定性结果作为特例。

定理**3**（Rice定理）。 每个非平凡的**r.e.**集合的性质都是不可判定的。

是的，你没听错：那就是r.e.集合的每个非平凡性质。 为了不误解这一点，让我们澄清一些事情。

首先，固定一个有限字母表Σ。r.e.集合的性质是一个映射

$$P : \{\Sigma^* \text{的r.e.子集}\} \to \{\mathbf{1}, \mathbf{0}\},$$

其中 **1**和 **0**分别表示真和假。 例如，空集的性质由映射表示

$$P(A) \quad = \quad \begin{cases} \mathbf{1} & \text{if } A = \varnothing, \\ \mathbf{0} & \text{if } A = \varnothing. \end{cases}$$

要询问这样一个性质 P是否可判定，集合必须以适合输入到图灵机的有限形式呈现。我们假设可列集合是由接受它们的图灵机呈现的。 但请记住，这个性质是关于集合的性质，而不是关于图灵机的性质；因此，它必须独立于所选择的表示集合的特定图灵机而为真或为假。

以下是一些可列集合的性质的其他示例： $L(M)$是有限的； $L(M)$是递归的； M接受101001（即，$101001 \in L(M)$）； $L(M) = \Sigma^*$. 这些性质中的每一个都是由图灵机接受的集合的性质。

下面是一些图灵机的性质示例，这些性质不是可列举集合的性质：M具有至少481个状态；M对所有输入都停机；M拒绝101001；存在一个等价于M的更小的机器。 这些不是集合的性质，因为在每种情况下，可以给出两个接受相同集合的图灵机，其中一个满足该性质，另一个不满足。

对于Rice定理适用，该性质还必须是非平凡的。 这意味着该性质既不普遍成立也不普遍不成立；也就是说，至少存在一个满足该性质的可列举集合和至少一个不满足该性质的可列举集合。只有两个平凡性质，它们都是平凡可判定的。

*Rice*定理的证明。 假设 P是可列举集合的一个非平凡性质。. 不失一般性地假设 $P(\varnothing) = \mathbf{0}$（如果 $P(\varnothing) = \mathbf{1}$，则论证对称）。 由于 P是非平凡的，必然存在一个可列举集合A使得 $P(A) = \mathbf{1}$。 令 K是接受 A的图灵机。

我们将HP归约到集合 $\{M \mid P(L(M)) = \mathbf{1}\}$，从而证明后者是不可判定的（定理2(ii)）。 给定 $M\#x$，构造一台机器 $M' = \sigma(M\#x)$，在输入 y时

（i）将 y 保存在某个独立的轨道上；

（ii）将 x写入它的纸带上（x在 M'的有限控制中是硬连线的）；

（iii）在输入 x（ M的描述也在 M'的有限控制中硬连线）上运行 M；

（iv）如果 M on x停机， M'运行 K on y并接受如果 K接受。

现在，要么 Mon x停机，要么不停机。如果 Mon x不停机，则（iii）中的模拟将永远不会停机，且 M'的输入 y不会被接受。 对于每个 y来说，这种情况都是成立的，因此在这种情况下 $L(M') = \varnothing$。 另一方面，如果 Mon x停机，则 M'总是达到步骤（iv），并且 M'的原始输入 y被接受当且仅当y被 K接受；也就是说，如果 $y \in A$。因此

$$M\text{在}x\text{上停机} \;\Rightarrow\; L(M') = A \;\Rightarrow\; P(L(M')) = P(A)$$
$$M\text{在}x\text{上不停机} \;\Rightarrow\; L(M') = \varnothing \;\Rightarrow\; P(L(M')) = P(\varnothing) = \mathbf{0}.$$

这构成了从HP到集合 $\{M \mid P(L(M)) = \mathbf{1}\}$的归约。 由于HP不是递归的，根据定理2，后者也不是递归的；也就是说，判断 $L(M)$是否满足 P是不可判定的。 □

Rice定理，第二部分

对于可枚举集合的性质 $P : \{\text{可枚举集合}\} \to \{\mathbf{1}, \mathbf{0}\}$，如果对于所有可枚举集合 A和 B，如果 $A \subseteq B$，则 $P(A) \leq P(B)$。 这里的 $\leq$表示在顺序 $\mathbf{0} \leq \mathbf{1}$中的小于等于。 换句话说，如果一个集合具有该性质，则该集合的所有超集也具有该性质。 例如，性质“$L(M)$是无限的”和“$L(M) = \Sigma^*$”是单调的，但性质“$L(M)$是有限的”和“$L(M) = \varnothing$”不是。定理**4**（Rice定理，第二部分）。可枚举集合的非单调性质是半可判定的。 换句话说，如果 P是可枚举集合的非单调性质，则集合 $T\ P = \{M \mid_P (L(M)) = \mathbf{1}\}$不是可枚举的。

证明。 由于 P是非单调的，存在图灵机 M_0和 M_1使得 $L(M_0) \subseteq L(M_1)$， $P(M_0) = \mathbf{1}$，并且 $P(M_1) = \mathbf{0}$。

我们希望将 $\sim$HP归约到 T_P，或者等价地，将HP归约到 $\sim T_P = \{M \mid P(L(M)) = \mathbf{0}\}$。 由于 $\sim$HP不是可枚举 T_P也不会是可枚举的。 给定 $M\#x$，我们想要展示如何构造一台机器 M'，使得 $P(M') = \mathbf{0}$当且仅当 M在输入 x上停机。 令 M'是一台在输入 y上执行以下操作的机器：

(i) 将输入 y写在磁带的顶部和中间轨道上；

(ii) 将输入 x写在底部轨道上（它在有限控制器中已经被硬连线）；

(iii) 以循环方式在顶部轨道上模拟输入 y的 M_0，在中间轨道上模拟输入 y的 M_1，在底部轨道上模拟输入 x的 M；也就是说，它模拟这三台机器的每一步，然后再进行下一步，依此类推（ M_0、 M_1和 M的描述都已经被硬连线在有限控制器中）；

(iv) 如果发生以下两种事件之一，则接受输入 y：

(a) M_0接受 y，或

(b) M_1接受 y并且 M在输入 x上停机。

无论输入 y如何， M是否在输入 x上停机是独立的。 如果 M在输入 x上不停机，则事件（b）在步骤（iv）中永远不会发生，因此在这种情况下， M'将接受 y当且仅当事件（a）发生，因此在这种情况下， L（ M'）$= L$（M_0）。 另一方面，如果M在输入 x上停机，则 y将被接受当且仅当它被 M_0或 M_1接受；也就是说，如果$y \in L$（ M_0）$\cup L$（M_1）。 由于 L（M_0） $\subseteq L$（M_1），这等价于说 $y \in L$（M_1），因此在这种情况下，L（M'）$= L$（M_1）。我们已经证明了

$$M\text{在}x\text{上停机} \;\Rightarrow\; L(M') = L(M$$
$$\Rightarrow\; P(L(M')) = P(L(M_1)) = \mathbf{0},$$

$$M\text{在}x\text{上不停机} \;\Rightarrow\; L(M') = L(M_0)$$
$$\Rightarrow\; P(L(M')) = P(L(M_0)) = \mathbf{1}.$$

从 M和x构造 M'构成了从 $\sim$HP到集合 $T_P = \{M \mid P(L(M)) =$ **1**$\}$的归约。 根据定理2(i)，后一个集合不是可枚举的。 □

历史注释

Rice定理由H. G. Rice [15, 16]证明。

参考文献

[1] G. 康托尔, 关于所有实数代数性质的一个性质, *J.* 纯粹和应用数学, 77 (1874), pp. 258–262. 重印于 *Georg Cantor Gesammelte Abhandlungen*, 柏林, Springer-Verlag, 1932, pp. 115–118.

[2] A. 邱奇, 逻辑基础的一组公设,*Ann.* 数学, 33–34 (1933), pp. 346–366, 839–864.

[3] ——, 一个无解的初等数论问题,*Amer.* 数学, 58 (1936), pp. 345–363.

[4] H.B. 库里, 逻辑替换的分析,*Amer.* 数学, 51 (1929), pp. 363–384.

[5] P.C. Fischer，具有受限内存访问的图灵机，信息与控制，9 (1966)，pp. 364–379.

[6] P.C. Fischer, A.R. Meyer, and A.L. Rosenberg，计数机和计数语言，数学系统理论，2 (1968)，pp. 265–283.

[7] K. G¨odel，关于形式数学系统中不可判定命题，在不可判定性，M. Davis，ed., Raven Press, Hewlitt, NY, 1965, pp. 5–38.

[8] S.C. Kleene，形式逻辑中的正整数理论，美国数学学会杂志，57 (1935)，pp. 153–173，219–244.

[9] ——，递归谓词和量词，美国数学学会交易，53 (1943)，pp. 41–74.

[10] D. Kozen,自动机与可计算性, Springer-Verlag, 1997.

[11] M.L. Minsky, 图灵机的递归不可解性和其他相关问题的研究 *Ann. Math.*, 74 (1961), pp. 437–455.

[12] E. Post, 有限组合过程的表述, I,*J. Symbolic Logic*, 1 (1936), pp. 103–105.

[13] ——, 一般组合决策问题的形式化简化,*Amer. J. Math.*, 65 (1943), pp. 197–215.

[14] ——, 可递归枚举的正自然数集及其决策问题,*Bull. Amer. Math. Soc.*, 50 (1944), pp. 284–316.

[15] H.G. Rice, 可递归枚举集合及其决策问题的分类,*Trans. Amer. Math. Soc.*, 89 (1953), pp. 25–59.

[16] ——，关于完全递归可枚举类及其关键数组，*J.* 符号逻辑，21 (1956)，pp. 304–341.

[17] H. Rogers Jr.，递归函数和有效可计算性理论，MI*T*出版社，1967年.

[18] M. Sch¨onfinkel，关于数理逻辑的基本构件，¨ 数学年鉴，92 (1924)，pp. 305–316.

[19] R.I. Soare，可递归集合和程度，*S*p*ringer*-V*erlag*，柏林，1987年.

[20] A.M. Turing，关于可计算数的一个应用于决策问题的论文，*P*roc. 伦敦数学学会，42 (1936)，pp. 230–265. 勘误：同上，43 (1937)，pp. 544–546.

[21] A.N.怀特海德和B.罗素,数理逻辑学原理, 剑桥大学出版社, 剑桥, 1910–1913. 三卷本.