第 7 章　递归论的基本知识

递归论是递归函数论的简称。它数理逻辑的一个重要分支，由于递归函数是直观上的可计算函数的精确化，递归论也被称为可计算性理论。

递归论创立于 1930 年代，最初是为了精确定义可计算性（即可判定性），有了可计算性（可判定性）的精确定义，人们才可以证明什么是不 可计算的或不 可判定的。很多经典的不可判定性结果，如停机问题和一阶逻辑的普遍有效性都是不可判定的，都是在 1930 年代建立的。之后人们的注意力转向了各种相对可计算性和由此产生的（不可解）度的研究。递归论的现代口号是研究可定义性，因为可定义性是可计算性的一种自然推广。依照对可定义性要求苛刻程度的大小，递归论的研究范围包括：（部分）理论计算机科学，古典递归论（一阶算术），（部分）描述集合论和（部分）集合论。还有各类高等递归论等等。

我们介绍递归论的动机除了了解可计算性概念之外，在系统内“表示”递归关系也是证明哥德尔 不完全性定理的重要组成部分。

第 1 节　原始递归函数

7.1.1　原始递归函数的定义

定义 7.1. 以下三类函数称为*初始函数*：零函数 $Z(x) = 0$，后继函数 $S(x)$，和投射函数 $\pi_i^n(x_1, \cdots, x_n) = x_i$，其中 n 为正整数且 $1 \leq i \leq n$。

令 n 为自然数，且 $g : \mathbb{N}^n \to \mathbb{N}$ 和 $h : \mathbb{N}^{n+2} \to \mathbb{N}$ 分别为 n- 元和 $(n+2)$-元函数。我们称 $(n+1)$-元函数 $f : \mathbb{N}^{n+1} \to \mathbb{N}$ 为从 g 和 h 经*原始递归* 得到的，如果

$$\begin{aligned} f(x_1, \cdots, x_n, 0) &= g(x_1, \cdots, x_n), \quad \text{且} \\ f(x_1, \cdots, x_n, y+1) &= h(x_1, \cdots, x_n, y, f(x_1, \cdots, x_n, y))。 \end{aligned}$$

（注意：这里的 $y+1$ 实际上是 y 的后继 $S(y)$，严格地说，并不是加法。）

为方便起见，我们规定一个 0-元函数 g 就是一个固定常数 c，这样一元函数 f 也可以像上面一样从 g 和 h 经原始递归得到：

$$f(0) = c, \quad \text{且} \quad f(y+1) = h(y, f(y))。 \tag{7.1}$$

定义 7.2. 全体*原始递归函数* 的集合 C 是最小的满足下列条件的集合:

(1) 所有的初始函数都在 C 中。

(2) C 对函数复合封闭，即，如果 $f(x_1,\cdots,x_n)=g(h_1(x_1,\cdots,x_n),\cdots,h_r(x_1,\cdots,x_n))$，并且 $g(y_1,\cdots,y_r)$ 和 $h_i(x_1,\cdots,x_n)$ $(1\leq i\leq r)$ 都在 C 中，则 f 也在 C 中。

(3) C 对原始递归封闭。

我们称 C 中的元素为*原始递归函数*。

在第 **??** 章第 **??** 节中我们讨论过关于闭包的“自上而下”和“自下而上”的定义方式，并且论证了它们是等价的。定义 **??** 是“自上而下”的，换成“自下而上”的等价形式，我们有：每个原始递归函数 f 都有一个有穷的*生成序列*：$\langle f_1,f_2,\cdots,f_n\rangle$，其中 $f_n=f$ 并且对任意 $1\leq i\leq n$，f_i 或者是初始函数，或者是由前面的函数通过复合或原始递归得到的。注意生成序列是不唯一的。我们可以沿着生成序列做归纳。例如，我们可以证明所有的原始递归函数都是*全* 函数，即，如果 $f:\mathbb{N}^n\to\mathbb{N}$ 是原始递归的，则 $\mathrm{dom}f=\mathbb{N}^n$（练习）。此外，也请大家思考一下所有的原始递归函数的确都是直观上可计算的。

例 7.1. 自然数的加法是原始递归的。通常是利用后继函数由下列递归方程定义的:

$$\begin{aligned} x+0 &= x,\\ x+(y+1) &= S(x+y)。\end{aligned}$$

作为例子，我们给出它的一个生成序列:(今后我们将只给递归方程，而将生成序列留给对方程有疑义的读者。)

$S(x_1)$（后继函数），$\pi^1_1(x_1)=x_1$（一元投射函数），$\pi^3_3(x_1,x_2,x_3)=x_3$（三元投射函数），$g(x_1,x_2,x_3)=S\circ\pi^3_3(x_1,x_2,x_3)=S(x_3)$（第一项与第三项的复合），$f(x_1,x_2)$（由第二项和第四项经原始递归得到）

$$\begin{aligned} f(x_1,0) &= \pi^1_1(x_1);\\ f(x_1,y+1) &= g(x_1,y,f(x_1,y))。\end{aligned}$$

显然，$f(x,y)=x+y$。

引理 7.1. 下列函数都是原始递归的:

1. 常数函数 $C^n_k(x_1,\cdots,x_n)=k$，其中 k 是一个固定的自然数。

2. 乘法函数 $x\cdot y$、指数函数 x^y 和阶乘函数 $x!$。

3. 如下定义的**非零检测** 函数 σ 和**零检测** 函数 δ*:*

$$\sigma(x) = \begin{cases} 0, & \text{如果 } x = 0; \\ 1, & \text{如果 } x \neq 0。 \end{cases} \qquad \delta(x) = \begin{cases} 1, & \text{如果 } x = 0; \\ 0, & \text{如果 } x \neq 0。 \end{cases}$$

4. 前驱函数 $\text{pred}(x)$。

5. 如下定义的**截断减法** （或减法）

$$x \dot{-} y = \begin{cases} 0, & \text{如果 } x < y; \\ x - y, & \text{否则。} \end{cases}$$

证明*:* 练习。 □

利用投射函数，我们可以将一个原始递归函数的自变量任意重排，所得到的仍是原始递归函数。具体地说，我们有下面的引理:

引理 7.2. 令 $f : \mathbb{N}^k \to \mathbb{N}$ 为一个原始递归函数。定义一个新的函数 $g : \mathbb{N}^r \to \mathbb{N}$ 为 $g(x_1, \cdots, x_r) = f(y_1, \cdots, y_k)$，其中每个 y_j 或者是 x_i （$1 \leq i \leq r$）或者是一个常数。则 g 也是原始递归的。

证明*:* g 可以通过复合从 f 和投影或常数函数得到。 □

例如，如果 $f(x, y)$ 是原始递归的，则 $g(x) = f(x, x)$ 和 $h(x, y, z) = f(z, y)$ 也是。

7.1.2　原始递归集合和谓词

在数学中，人们常常利用特征函数把集合“转化”成函数。对一个 k-元谓词 P ，我们也可以定义它的特征函数为:

$$\chi_P(\vec{x}) = \begin{cases} 1, & \text{如果 } P(\vec{x}) \text{ 成立;} \\ 0, & \text{否则。} \end{cases}$$

利用特征函数，我们很自然地把原始递归的概念从函数扩展到集合和谓词。

我们称 $\mathbb{N}^k$ 的一个子集 A 或一个 k-元谓词 P 为**原始递归的**，如果它的特征函数是一个原始递归函数。一个 k-元谓词 P 为原始递归的也可等价地定义为集合 $\{\vec{x} : P(\vec{x})\}$ 是一个原始递归的集合。

例如，二元谓词 $=$ 和 $\leq$ 都是原始递归的 （练习）。

引理 7.3.

1. 如果 $A, B \subseteq \mathbb{N}^k$ 都是原始递归的集合，则 $\mathbb{N}^k - A, A \cup B$ 和 $A \cap B$ 也是。

2. 如果 P 和 Q 都是原始递归的谓词，则 $\neg P$，$P \vee Q$ 和 $P \wedge Q$ 也是。

引理 7.4 (分情形定义)**.** 如果 f_1 和 f_2 都是原始递归函数，并且 P 是原始递归谓词，则函数

$$f(x) = \begin{cases} f_1(x), & \text{如果 } P(x) \text{ 成立;} \\ f_2(x), & \text{否则} \end{cases}$$

也是原始递归的。

用程序语言来说，我们可以执行条件判断的指令。

证明: $f(x) = f_1(x)\chi_P(x) + f_2(x)(1\dot{-}\chi_P(x))$。 □

现在我们可以处理四则运算的最后一则运算—除法了。首先用符号 $\mathrm{quo}(x, y)$ 和 $\mathrm{rem}(x, y)$ 分别表示用 x 去除 y 而得到的商和余数。为了使它们成为全函数，我们规定 $\mathrm{quo}(0, y) = 0$ 和 $\mathrm{rem}(0, y) = 0$。

引理 7.5. 函数 $\mathrm{quo}(x, y)$ 和 $\mathrm{rem}(x, y)$ 都是原始递归的。

证明: 基本思路是利用下面的递归定义，我们把对定义的仔细分析留给读者。

$$\mathrm{rem}(x, y+1) = \begin{cases} \mathrm{rem}(x, y) + 1, & \text{如果 } \mathrm{rem}(x, y) + 1 \neq x; \\ 0, & \text{否则。} \end{cases}$$

和

$$\mathrm{quo}(x, y+1) = \begin{cases} \mathrm{quo}(x, y) + 1, & \text{如果 } \mathrm{rem}(x, y) + 1 = x; \\ \mathrm{quo}(x, y), & \text{否则。} \end{cases}$$

□

回忆一下关于**有界量词**的定义。我们定义 $(\exists x < a)\varphi(x)$ 为 $\exists x(x < a \wedge \varphi(x))$ 和定义 $(\forall x < a)\varphi(x)$ 为 $\forall x(x < a \to \varphi(x))$。

接下来我们引进**有界极小算子**。下文中的希腊字母 μ 可以读作“最小的”。

定义 7.3. 令 $P(\vec{x}, z)$ 为一个 $(k+1)$-元的性质。定义

$$(\mu z \leq y)P(\vec{x}, z) = \begin{cases} \text{最小的满足 } P(\vec{x}, z) \text{ 且} \leq y \text{ 的 } z, & \text{如果它存在;} \\ y + 1, & \text{否则。} \end{cases}$$

引理 7.6. 如果 $f(\vec{x}, y)$ 是原始递归的，则有界和 $\sum_{y \leq z} f(\vec{x}, y)$ 和有界积 $\prod_{y \leq z} f(\vec{x}, y)$ 都是原始递归的。

证明: 练习。 □

我们用符号 $X := Y$ 表示 X 是按照 Y 来定义的，或按照 Y 定义的那个 X。

引理 7.7. 假定 $P(\vec{x}, z)$ 是一个原始递归的谓词。则

(a) 谓词 $E(\vec{x}, y) := (\exists z \leq y)P(\vec{x}, z)$ 和 $A(\vec{x}, y) := (\forall z \leq y)P(\vec{x}, z)$ 都是原始递归的。

(b) 函数 $f(\vec{x}, y) := (\mu z \leq y)P(\vec{x}, z)$ 也是原始递归的。

证明: (a) 根据定义，我们有：谓词 $(\forall z \leq y)P(\vec{x}, z)$ 为真当且仅当 $\prod_{z \leq y} \chi_P(\vec{x}, z) = 1$；并且谓词 $(\exists z \leq y)P(\vec{x}, z)$ 等价于 $\neg(\forall z \leq y)\neg P(\vec{x}, z)$。由此可以得到 (a)。

(b) 只须注意 $(\mu z \leq y)P(\vec{x}, z) = \sum_{z=0}^{y} \prod_{r=0}^{z} \chi_{\neg P}(\vec{x}, r)$ 即可。特别注意当满足条件的 z 不存在时，等式右边恰恰等于 $y + 1$。 □

7.1.3 编码

我们下面讨论一些有关素数的可计算性，目的是利用素数分解定理来编码。

引理 7.8. *(1)* 谓词“x 整除 y”是原始递归的。

(2) 谓词“x 是合数”和“x 是素数”都是原始递归的。

(3) 函数 $p(n) :=$ 第 n 个素数 是原始递归的。这个函数我们今后经常会用到。$p(n)$ 也常被写作 p_n，例如，$p_0 = 2, p_1 = 3, p_2 = 5, \cdots$ 等等。

证明: 练习。 □

我们现在可以利用素数分解定理来能行地编码了。

定义 7.4. *(1)* 我们用尖括号 $\langle a_0, \cdots, a_n \rangle$ 来表示乘积 $p_0^{a_0+1} \cdot \cdots \cdot p_n^{a_n+1}$，并把它称为有穷序列 $(a_0, \cdots, a_n)$ 的**哥德尔数**。定义空序列 $\langle\,\rangle$ 的哥德尔数为 1。

(2) 定义函数 $\mathrm{lh} : \mathbb{N} \to \mathbb{N}$ 为 $\mathrm{lh}(a) = \mu k \leq a\ (p_k \nmid a)$。我们称 $\mathrm{lh}(a)$ 为**长度**函数，因为 $\mathrm{lh}(1) = 0$ 且对于任意的哥德尔数 $a = \langle a_0, \cdots, a_n \rangle$，都有 $\mathrm{lh}(a) = n + 1$。

(3) 定义关于 a 和 i 的二元函数 $(a)_i = \mu k \leq a\ [p_i^{k+2} \nmid a]$。我们称 $(a)_i$ 为**分量**函数，因为它刚好从编码为 a 的有穷序列中挑出第 i 项：对任意的哥德尔数 $a = \langle a_0, \cdots, a_n \rangle$，$(a)_i = a_i\ (0 \leq i \leq n)$。

(4) 对自然数 a, b 定义**串接**函数 $a\hat{}b$ 如下：

$$a\hat{}b = a \cdot \prod_{i < \mathrm{lh}(b)} p_{\mathrm{lh}(a)+i}^{(b)_i+1}。$$

注意：函数 $\mathrm{lh}(a)$ 和 $(a)_i$ 都是全函数。特别地，函数 lh 对不是哥德尔数的自然数 a 也有定义，只不过我们不关心它的值罢了。对函数 $(a)_i$ 当 $i \geq \mathrm{lh}(a)$ 时，情形也一样。

引理 7.9. *(1)* 全体有穷序列的哥德尔数构成的集合是原始递归的。

(2) 函数 $\mathrm{lh}(a)$ 和 $(a)_i$ 都是原始递归的。

(3) 函数 $a\hat{}b$ 是原始递归的且 $\langle a_0, \cdots, a_n\rangle\hat{}\langle b_0, \cdots, b_m\rangle = \langle a_0, \cdots, a_n, b_0, \cdots, b_m\rangle$。还有，如果 a 和 b 都是某个有穷序列的哥德尔数，则 $a\hat{}b$ 也是。

证明: 习题。 □

回忆一下原始递归的定义，当我们定义 $f(y+1)$ 的时候，我们仅仅用到了 f 在前一点的值 $f(y)$。大家很自然地会猜测，我们并不非得要用前一点的值，只要我们以前已经算过的值我们应该都可以用，即，在定义 $f(y+1)$ 时，我们可以利用任何 $f(z)$ 的值，只要 $z \leq y$ 即可。下面的引理说的就是这件事情。首先引入两个自然的术语。对任何一个全函数 $f : \mathbb{N}^{k+1} \to \mathbb{N}$，令 $F(\vec{x}, n) = p_0^{f(\vec{x},0)+1} p_1^{f(\vec{x},1)+1} \cdots p_n^{f(\vec{x},n)+1}$。我们称它为 f 的*历史函数*。我们称函数 f 为从 g 和 h 经*强递归* 得到的，如果

$$\begin{aligned} f(\vec{x}, 0) &= g(\vec{x}); \\ f(\vec{x}, y+1) &= h(\vec{x}, y, F(\vec{x}, y))。 \end{aligned}$$

引理 7.10. 如果函数 f 为从 g 和 h 经强递归得到的，且 g 和 h 都是原始递归的，则 f 也是原始递归的。

证明: 很容易看出 f 的历史函数 $F(\vec{x}, y)$ 是原始递归的：

$$\begin{aligned} F(\vec{x}, 0) &= 2^{g(\vec{x})+1}; \\ F(\vec{x}, y+1) &= F(\vec{x}, y) \cdot p_{y+1}^{h(\vec{x},y,F(\vec{x},y))+1}。 \end{aligned}$$

所以，$f(\vec{x}, y) = (F(\vec{x}, y))_y$ 也是原始递归的。 □

引理 **??** 在后面非常有用，尤其是在讨论语法的算术化的时候。

第 2 节　递归函数

7.2.1　非原始递归函数

是不是所有的直观上可计算的函数都是原始递归的呢？答案是否定的。

我们可以用“对角线”方法来论证。首先注意：我们直观上有一个程序，它可以系统地把所有原始递归函数枚举出来。比如，我们可以把所有可能的生成序列一一枚举出

来。令 $g_0, g_1, \ldots$ 表示这样枚举出来的原始递归函数序列。定义函数 $F : \mathbb{N} \to \mathbb{N}$ 为 $F(n) = g_n(n) + 1$。这个函数 F 是直观上可计算的，但它不出现在原始递归函数的序列中，因而不是原始递归的。

一个更为具体的例子是所谓的阿克曼函数 $A(x, y)$，定义如下：

$$\begin{aligned} A(0, y) &= y + 1, \qquad A(x + 1, 0) = A(x, 1), \\ A(x + 1, y + 1) &= A(x, A(x + 1, y))。 \end{aligned}$$

例如，$A(1, y) = y + 2$、$A(2, y) = 2y + 3$ 还有 $A(3, y) = 2^{y+3} - 3$。粗略地说，$A(x + 1, y)$ 是通过 y 次叠代运算 $A(x, y)$ 而得到的。

利用双重归纳，我们不难证明 $A(x, y)$ 是一个全函数，即，对所有的自然数 x 和 y，$A(x, y)$ 都是有定义的。这个归纳的过程同时告诉我们直观上怎样计算阿克曼函数。但它不是原始递归的，原因是它增长得太快了，任何一个原始递归函数最终都会被它优超。具体的断言和证明我们留作习题。

7.2.2　递归函数

从程序语言的角度看，原始递归函数可以处理所有的算术运算、条件判断、以及形如“从 $i = 1$ 到 n 执行……”这样的循环。我们想一想，同一般的程序语言相比，我们还缺什么呢？答案是，我们还缺少“无（事先给定的）界的循环”，即处理“重复……直到……”这样的指令。下面的定义试图弥补这一缺陷。

定义 7.5. 令 $f : \mathbb{N}^{n+1} \to \mathbb{N}$ 为一个全函数。我们称函数 $g(\vec{x})$ 是从 f 通过*正则极小化*或*由正则 μ- 算子*得到的，如果 $\forall \vec{x} \exists y f(\vec{x}, y) = 0$ （称为正则性条件）并且 $g(\vec{x})$ 是使得 $f(\vec{x}, y) = 0$ 的最小的 y。沿用记号 μ，我们把它写成 $g(\vec{x}) = \mu y[f(\vec{x}, y) = 0]$。

定义 7.6. 全体*递归函数*的集合为最小的包含所有初始函数，并且对复合，原始递归和正则极小化封闭的函数集合。

同前一样，如果一个集合的特征函数是一个递归函数，我们则称该集合为一个*递归集*。递归集就是*可判定集合*的精确说法。

定义中的正则性条件 $\forall \vec{x} \exists y g(\vec{x}, y) = 0$ 从可计算的角度看是非常复杂的。我们无法能行地判断正则性条件是否成立。很难想象我们在设计一个算法或在写一个程序的时候需要时而不时地检查正则性条件。我们希望能够把它删掉。删掉它的后果是我们对 y 的搜寻可能永远不停止，因此必须面对所谓的部分函数，即在某些点没有定义的函数。从现在起，当我们考虑函数 $f : A \to B$ 时，f 的定义域可以是 A 的一个真子集。对 A 中的一个点 x，我们用 $f(x) \downarrow$ 表示“函数 f 在 x 点有定义”（或称为“$f(x)$ 是收敛的”）；而 $f(x) \uparrow$ 表示“函数 f 在 x 点没有定义的”（或称为“$f(x)$ 是发散的”）。

7.2.3　部分递归函数

定义 7.7. 令 f 为一个部分函数。我们称函数 g 是从 f 通过*极小化* 或*由 μ-算子* 得到的，如果

$$g(\vec{x}) = \mu y\,[\forall z \le y(f(\vec{x}, z)\downarrow) \wedge f(\vec{x}, y) = 0]。$$

我们解释一下条件 $\forall z \le y(f(\vec{x}, z)\downarrow)$。由于函数 $f(\vec{x}, y)$ 可以是部分函数，很有可能在它的最小的根（比如说 y_0）出现之前，它在某些点（比如说 z_0）上是没有定义的。这时候我们应该怎样处理呢？让我们参考一下编程时的做法。我们只能耐心地一个点一个点地检验。假如我们跳过 z_0，即使看到了 y_0 这个根，我们也不敢断定它是最小的，因为我们不可能能行地知道 $f(\vec{x}, z_0)$ 是发散的（见后面停机问题的讨论)。因此，我们宁可达不到真正的根，也不能跳过发散点，这就是条件 $\forall z \le y(f(\vec{x}, z)\downarrow)$ 的目的。后面在习题 **??** 中，我们会看到如果允许跳过发散点，则定义出来的函数类比可计算函数类要大。最后再说明一点，后面的克林尼正规型定理告诉我们，对部分函数使用极小算子是不那么重要的，每个部分递归函数本质上都可以通过对某个原始递归谓词使用一次极小算子产生出来。

定义 7.8. 全体*部分递归函数* 的集合为最小的包含所有初始函数，并且对复合，原始递归和极小化封闭的函数集合。

对初学者来说，部分函数的概念可能不容易接受，尤其是我们关心的对象似乎主要是递归（全）函数。后面会提到一些部分函数所具有的一些优点。但我们研究部分函数最重要的原因是由可计算性的本质决定的：每一个算法（或程序）都天然地对应一个部分函数，而不是全函数。

在本章中的部分递归函数自然可以是部分函数。但我们使用“递归函数”这个词时，指的是递归全函数。有时我们为了强调，也会用“递归全函数”。

引理 7.11. 阿克曼函数是部分递归的全函数。

我们下面给出证明的梗概。基本思路是搜索一个“包含所有计算所需信息的编码”。这一方面说明极小算子带来的好处，另一方面，确认一个编码包含所有中间步骤的完整信息比确认最终答案要容易，这件事情本身也有意思，后面证明克林尼正规型定理时也会用到这一想法。当然，这也不难理解，确认一个定理的证明是对是错比确认一个猜想是一个定理要容易多了。

证明：比照阿克曼函数的定义，我们称一个三元组的有穷集合 S 为*好的*，如果它满足下列条件：

(a) 如果 $(0, y, z) \in S$ 则 $z = y + 1$；

(b) 如果 $(x + 1, 0, z) \in S$ 则 $(x, 1, z) \in S$；

(c) 如果 $(x+1, y+1, z) \in S$ 则存在一个自然数 u 使得 $(x+1, y, u) \in S$ 且 $(x, u, z) \in S$。

注意：一个好的三元组集 S 具有如下性质：如果 $(x, y, z) \in S$，则

(1) $z = A(x, y)$。

(2) S 包含计算 $A(x, y)$ 过程中所需的所有的“先前的”三元组 $(x', y', A(x', y'))$。

接下来我们把三元组 (x, y, z) 编码成 $\langle x, y, z\rangle = 2^{x+1}3^{y+1}5^{z+1}$；并把一个三元组的编码的有穷集 $\{u_1, \cdots, u_k\}$ 编码成 $\langle u_1, u_2, \cdots, u_k\rangle$。所以一个三元组的有穷集可以被编码成一个自然数 v。

令 S_v 表示编码为 v 的三元组集。则谓词“$(x, y, z) \in S_v$”是原始递归的，因为它等价于“$\exists i < v((v)_i = \langle x, y, z\rangle)$”。更进一步，“$S_v$ 是一个好的三元组的集合”也是一个原始递归谓词（练习）。所以谓词

$$R(x, y, v) := \text{“}v \text{ 是一个好的三元组集的编码并且 } \exists z < v\ ((x, y, z) \in S_v)\text{”}$$

也是原始递归的。

于是函数 $f(x, y) = \mu v R(x, y, v)$ 是部分递归的。所以 $A(x, y) = \mu z((x, y, z) \in S_{f(x,y)})$ 也是部分递归的；并且 $f(x, y)$ 和 $A(x, y)$ 都是全函数。 □

第 3 节　图灵机

英国数学家图灵是二十世纪最伟大的数理逻辑学家之一，也被人称为计算机科学和人工智能之父。在他 1936 年的文章中 [1]，他提出了图灵机的概念，并且证明了停机问题是不可判定的，从而解决了希尔伯特提出的判定性问题。虽然哥德尔和丘奇等人也在更早或差不多同时也给出了判定问题的否定答案，但图灵提出的被称为图灵机的计算模型可以被视为纯机械的物理机器，因而比形式系统更能代表任何的机械装置，从而在不可判定问题的解答上更为直观也更有说服力。

7.3.1　图灵机的定义

我们先看图灵机的物理描述。图灵分析了一个“计算员（computor）”进行计算的过程，得出了一台图灵机应该包括以下要素：

[1] *On Computable Numbers, with an application to the Entscheidungsproblem*, Proc. Lond. Math. Soc.

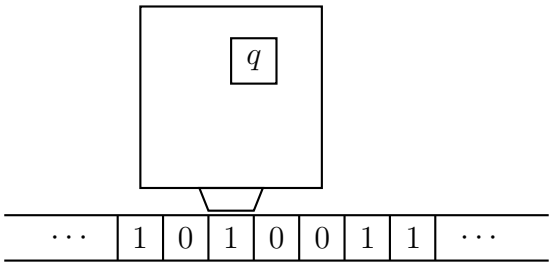


图 7.1: 图灵机示意图

1. 一条两个方向都可**无限延长的纸带**，被分成一个个小的格子。这些格子中或者是空白的，用 0 表示，或者可以写上一个字符，这些字符来自一个事先给定的有穷的字母表 $A = \{a_1, a_2, \cdots, a_n, 0\}$。

2. 一个**读写头**，每次可以扫描纸带上的一个格子。读写头可以辨别格子是空白的还是有字的，它还能在空白格子中写上符号，也能将已有的字符抹去，使其重新成为空白的；读写头可以左右移动，但每次只能移动一格。

3. 一个**有穷的状态集** $Q = \{q_1, \cdots, q_n\}$，在任何一个给定时刻，图灵机都处在当中的某个状态 q_i。

图灵机的数学定义则是根据它的“软件”而来的。

定义 7.9. 一台*图灵机*是由下面几个部分组成的：一个有穷的字母表 A、两个特殊的方向符号 L（左）和 R（右）、一个有穷的状态集 Q、和一个有穷的指令集 δ，其中每个指令是一个具有下列形式的四元组：

(a) $qaa'q'$ 其中 $q, q' \in Q$ 并且 $a, a' \in A$。

(b) $qaLq'$ 其中 $q, q' \in Q$ 并且 $a \in A$。

(c) $qaRq'$ 其中 $q, q' \in Q$ 并且 $a \in A$。

此外，我们还假定对任意的状态 q 和 字符 a，至多只有一个四元组以 qa 起头。

这个四元组集对应着一个从 $Q \times A$ 到 $(A \cup \{R, L\}) \times Q$ 的一个部分函数，我们称它为一个转换函数。

指令 $qaa'q'$ 的解读如下：如果图灵机的当前状态为 q 并且当前读写头在格子中看到的符号为 a，则将格子中的 a 改成 a'，并把状态改成 q'。指令 $qaLq'$ 和 $qaRq'$ 的解读类似，只不过是把读写头分别地向左和向右移动一格，而不改动格子内的符号。

注意在不同的教科书中，对图灵机的描述和设计会有不同。但就计算能力来说，各种模型都是一样的，即如果一个函数可以被一种模型的图灵机计算，那它也可以被另一种计算。当然计算的效率（如所花费的时间）会很不一样。我们列出几种常见的变形，它们的等价性有些我们会证明，有些留作练习，有些超出我们的讨论范围。

- 纸带的设计：单向无穷还是双向无穷的。
- 字母表的选取：仅用 $\{0,1\}$ 还是允许其它有穷多个字符。
- 图灵机的程序是用四元组来表达，还是用五元组，如 $qaa'q'D$ 其中 $D = L$ 或者 R (甚至还可以是 S - 原地不动)。
- 允许不允许多个读写头，以及多个纸带。
- 图灵机是确定性的还是非确定性的：对每一个由状态 q 和字符 a 形成的对，我们定义的图灵机只允许至多一个四元组以 qa 开头，因此是**确定性的**；非确定性的图灵机则允许多个以 qa 开头的四元组。
- 经典的图灵机还是量子的图灵机。

我们下面描述图灵机是怎样进行计算的，也顺带引入一些常见的与计算有关的概念和约定。

- **格局**：直观上说，在任何一个时刻，如果我们对着图灵机拍个快照，照片上的信息就是在那一刻的格局。精确地说，在某个时刻的图灵机格局包括以下三部分：(1) 当前纸带上所有有字符的格子的信息（注意：在任何时刻，纸带上至多有有穷个非空的格子。)；(2) 读写头的位置；和 (3) 目前图灵机所处的状态 q。通常我们把一个格局简记成 $C = uqav$，其中 q 是目前的状态，u 和 v 是由字母表 A 中符号所组成的字符串，a 是 A 中的字符，纸带上的所有有字的格子自左向右依次为：字符串 u 位于读写头的左边，字符 a 位于读写头的下边和字符串 v 位于读写头的右边。例如，图 **??** 中的格局就是：$uq1v$，其中 $u = 10$ 和 $v = 0011$。
- 格局的转换（单步计算)：给定一个格局 $C = uqav$，如果在 δ 中不存在以 qa 开头的四元组，则称 C 是一个**终止格局**；否则，我们根据不同的指令，定义新的格局 C' 如下，并称格局 C **产生** C'；
  (a) 如果 $qaa'q' \in \delta$ 则 $C' = uq'a'v$；
  (b) 如果 $qaRq' \in \delta$ 则 $C' = uaq'bv'$ 其中 $v = \langle b\rangle\hat{}v'$；
  (c) 如果 $qaLq' \in \delta$ 则 $C' = u'q'bav$ 其中 $u'\hat{}\langle b\rangle = u$。
- 图灵机的一个计算就是一个格局的序列（可以是有穷的，也可以是无穷的）$\langle C_i\rangle$ 使得对每一个 i，如果 C_i 不是终止格局，则 C_i 产生 C_{i+1}。

- 为了方便起见，我们假定每个图灵机都包含两个特别的状态：一个是**初始状态** 记为 q_s，所有的计算都是以该状态开始的；另一个是**停机状态** q_h，每当遇到终止格局是，如果它不是处于停机状态，我们都要把它转换到停机状态 (必要时可以增加一些四元组)。

- 为了确定地描述计算过程，让我们固定一个输入和输出的表示法。令 1^x 表示由连续 x 个 1 组成的串。我们规定输入向量 $\vec{x} = (x_1, x_2, \cdots, x_k)$ 的格局为 $q_s 1^{x_1+1} 0 1^{x_2+1} 0 \cdots 0 1^{x_k+1}$。我们规定输出时的格局为 $q_h 1^y$，$(y \geq 0)$，表示输出为 y。当然输入输出的表示法不是唯一的。我们在输入时不用 1^x 表示 x 是因为当 $x = 0$ 时我们无法知道输入是几元的。注意，我们这里对输出的格式要求较严，一般教科书往往不要求纸带上的 1 形成一个连续串，而只要有 y 个 1 在纸带上就可以了。

定义 7.10. 我们称一个部分函数 $f : \mathbb{N}^k \to \mathbb{N}$ 是 *被图灵机 M 所计算的*，或者说图灵机 M 计算函数 f，如果

$$f(\vec{x}) = \begin{cases} y, & \text{如果 } M \text{ 对输入 } \vec{x} \text{ 最终输出 } y; \\ \text{没有定义}, & \text{如果计算过程是无限的。} \end{cases}$$

我们称一个部分函数为*图灵机可计算的*，如果存在一个图灵机 M 计算它。

例 7.2. 设计一个计算函数 $f(x) = 2x$ 的图灵机程序。

解答. 纸带上最初有 $x + 1$ 个 1，我们可以先把它复制一遍，这样纸带上就有 $2x + 2$ 个 1。最后再擦去两个 1 即可。（当然我们也可以先擦去一个 1 再复制，没太大区别。）复制的时候，最重要的是区别已经复制过的、还没有复制的和新写上的 1 的区别，幸运的是我们可以利用不同的字符，比如，已经复制的就换成 a，还没复制的自然仍是 1，新写上的我们可以用 b。下面是具体的四元组：

$$q_s 1 a q_1, q_1 a R q_1, q_1 1 R q_1, q_1 b R q_1, q_1 0 b q_2,$$

(将 1 换成 a，右移至空白写一个 b。)

$$q_2 b b q_3, q_3 b L q_3, q_3 1 L q_3, q_3 a R q_4, q_4 1 1 q_s$$

(左移到最右的 a，再右移一格，如果看到 1 则重复。)

$$q_4 b R q_4, q_4 0 L q_5, q_5 b 1 q_5, q_5 1 L q_5, q_5 a 1 q_5, q_5 0 R q_6$$

(如果看到 b，则表示复制完毕，下一步把 a 和 b 还原成 1。)

$$q_6 1 0 q_6, q_6 0 R q_7, q_7 1 0 q_7, q_7 0 R q_h$$

(擦掉两个 1 后停机。)

7.3.2　用有向转移图来表示图灵机

图灵机最令人惊奇的地方就是它的简单。只靠左移，右移，写上，擦掉这样的动作和有穷多个四元组，图灵机理论上能够完成一切人类所能进行的复杂计算，这似乎是难以置信的。但也正是因为它的简单，它的效率太低了。因此，写图灵机的程序没有任何实用价值。我们所有的例子和练习，目的都是帮助大家理解图灵机的运作，而不是训练大家写图灵程序。

今后我们会越来越少地直接使用四元组，而更多地使用“高级”语言来描述对读写头和纸带内容的控制。中间路线是使用“有向转移图”，其节点为少数事先指定好的子程序，图的连接则告诉我们子程序之间的串联或并联关系。

我们先指定几个简单的子程序，我们后面会把它们当作部件组装起来。

例 7.3. 假定字母表为 $\{0,1\}$。

(1) 图灵机 $R=\{q_s0Rq_h, q_s1Rq_h\}$ 和 $L=\{q_s0Lq_h, q_s1Lq_h\}$ 的动作（无论纸带上是什么内容）分别是把读写头向右和向左移动一格。

(2) 图灵机 $P_0=\{q_s00q_h, q_s10q_h\}$ 和 $P_1=\{q_s01q_h, q_s11q_h\}$ 的动作（无论纸带上是什么内容）分别是在当前扫描的格子中写一个 0（即擦去字符）和写一个 1。P_0 有时也直接写成 0。一般地，对任何字母表中的元素 a，我们有图灵机 P_a 或 a 做类似的事情。

例 7.4. 令字母表 $A=\{0,1,a,b,c,d\}$。写一个图灵程序 R_a 把读写头移到严格在右边（即当前格子不算）的第一个写着 a 的格子（如果没有这样的格子，则 R_a 永不停机)。

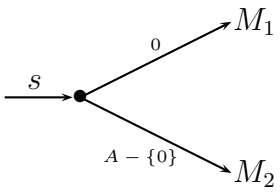
解答. 容易看出，R_a 可以选为 $\{q_s0Rq_1, q_s1Rq_1, q_saRq_1, q_sbRq_1, q_scRq_1, q_sdRq_1\}$（无论纸带上写什么，先向右移一格）$\cup\{q_10Rq_1, q_11Rq_1, q_1aaq_h, q_1bRq_1, q_1cRq_1, q_1dRq_1\}$（除了见到 a 停机之外，见到其它符号均继续向右。)

下面我们描述两种常见的使用子程序的方法:

例 7.5. 给定图灵机 M_1 和 M_2，设计一个新的图灵机 M 使得它先执行程序 M_1 之后继续执行程序 M_2。我们用 M_1M_2 来表示 M。

解答. 重新命名 M_2 的状态集，使得它的新的初始状态为 M_1 的停机状态，其它的状态都不在 M_1 的状态集中出现；并相应地修正 M_2 的四元组集即可。

例 7.6. 给定图灵机 M_1 和 M_2，设计一个新的图灵机 M 使得它程序“如果目前扫描的符号是 0 则执行 M_1，否则执行 M_2”。我们用下面的有向图来表示 M:



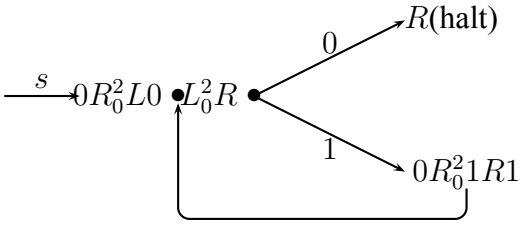
解答. 不妨假定 M_1 和 M_2 的状态集交集为空；且它们的起始和停机状态分别为 $q_s^1, q_h^1, q_s^2, q_h^2$。还不妨假定 M_1 和 M_2 共享一个字母表 A。挑选两个不在 M_1 和 M_2 中出现的新的状态 q_s, q_h。定义 M 的四元组集为

$$\delta_1 \cup \delta_2 \cup \{q_s 00 q_s^1\} \cup \{q_s a a q_s^2 : a \in A \setminus \{0\}\} \cup \{q_h^1 a a q_h, q_h^2 a a q_h : a \in A\}$$

即可，其中 δ_1 和 δ_2 分别是 M_1 和 M_2 的四元组集。

例 7.7. 以有向转移图的方式设计一个计算函数 $f(x,y) = 2x + y$ 的图灵机。

解答. 首先，我们的初始格局是 $q_s 1^{x+1} 0 1^{y+1}$。我们用 $0R_0^2 L0$ 把最左和最右的 1 消成 0，执行完之后，纸带上的字符串为 $1^x 0 1^y$，并且读写头停在 y-串的最后一个 1 上（如果有的话，不然 $y = 0$ 我们停在 0 上）。



接下来，我们用 $L_0^2 R$ 把读写头移到 x-串的开头。这时有两种情况：如果看到的是 1 的话，我们把它消去，并在输出串的末尾再添两个 1，这是 $0R_0^2 1R1$ 的作用，做完后再循环；如果看到 0 的话，说明 x-串已经没有了，因此把读写头移到输出串的第一位，停机。

第 4 节　图灵可计算函数与部分递归函数

我们这一节的中心内容是证明下面的定理：

定理 7.1. 一个函数是图灵可计算的当且仅当它是部分递归的。

我们分两部分来证。

7.4.1　从部分递归函数到图灵可计算函数

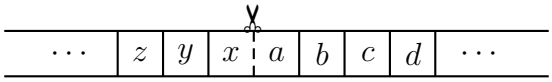
引理 7.12. 每个初始函数都是图灵可计算的。

证明: 见上一节的习题（习题 **??**）　□

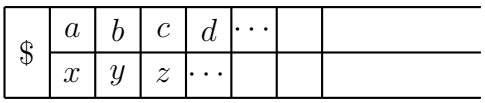
接下来我们验证图灵可计算函数具有我们想要的封闭性。从一般计算机程序的角度看这是很自然的。但由于图灵机的特殊性（特别是纸带的限制），仍有一些细节需要讨论。例如，当我们在计算复合函数 $g(h_1(x_1,x_2),h_2(x_1,x_2))$ 的时候，就需要调用输入 x_1 和 x_2 两次，因此我们需要保存一个备份（例如，存在某个固定的寄存器里），使得图灵机在计算 $h_1(x_1,x_2)$ 时，不会动到我们的备份。有些教科书以无穷存储机器作为计算的基本模型，最大的优越性就在这里。我们下面采取另一种方法。我们证明图灵机的纸带可以是单向无穷的，因而我们可以用“一半”纸带来保存必要的信息。当然这个命题本身也有它自己的意义。

引理 7.13. 任何一台（标准的）图灵机 M_1 都可以被一台纸带是单向无穷的图灵机 M_2 所模拟。

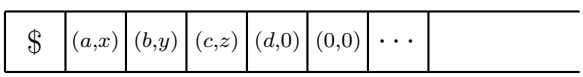
证明: 我们只给出证明概要。基本想法是先把双向无穷的纸带从中间剪开：



把左边的纸带反转，放到右边纸带的下方，并挑选一个不在 M_1 的字母表 A 中出现的新字符 \$，放置在最左端表示新纸带的左边界。



把这两条平行的纸带想象成一条纸带的两轨，再把同一位置上下轨道中的符号（例如，a 和 x）改写成有序对（例如，(a,x)）。并且扩充旧的字母表，添上 $A\times A$ 以包含所有这样的有序对。图灵机 M_2 的单向无穷纸带的看上去就是这样：



有了这些直观图像之后，我们就不难描述 M_2 应该怎样模拟 M_1 的计算过程了：

首先，把输入向右平移一格后，重新抄写到上半轨（例如，输入 1^{x+1} 就变成了 $(1,0)^{x+1}$），并在空出来的第一格写上左边界符号 \$。细节我们留给读者。

接下来我们在双轨上模拟 M_1 的（“双向”）计算。细节如下：对每一个 M_1 的状态 q，M_2 都有一对状态 $(q,1)$ 和 $(q,2)$ 与之相应。状态 $(q,1)$ 用来模拟上轨的计算，$(q,2)$ 模拟下轨。假定 M_1 的四元组集为 δ_1。我们定义 M_2 的（与之相应的）四元组集 δ_2 如下（注意：这只是 M_2 四元组的一部分。）：

(1) （在上轨上模拟 M_1）如果 $qaa'q' \in \delta_1$，则对每一个 $b \in A$，我们都把四元组 $(q,1)(a,b)(a',b)(q',1)$ 放入 δ_2；如果 $qaLq' \in \delta_1$（或分别地 $qaRq' \in \delta_1$），则对每一个 $b \in A$，我们都把四元组 $(q,1)(a,b)L(q',1)$（或分别地 $(q,1)(a,b)R(q',1)$）放入 δ_2。

(2) （在下轨上模拟 M_1）如果 $qaa'q' \in \delta_1$，则对每一个 $b \in A$，我们都把四元组 $(q,2)(b,a)(b,a')(q',2)$ 放入 δ_2；如果 $qaLq' \in \delta_1$（或分别地 $qaRq' \in \delta_1$），则对每一个 $b \in A$，我们都把四元组 $(q,2)(b,a)R(q',2)$（或分别地 $(q,2)(b,a)L(q',2)$）放入 δ_2。注意我们这里调转了读写头的运动方向。

(3) （轨道转换）对 M_1 的每一个状态 q，我们都把四元组 $(q,1)\$R(q,2)$ 和 $(q,2)\$R(q,1)$ 放入 δ_2 中。

我们不难看出：M_2 可以一步一步地模拟 M_1 的计算，即如果 M_1 可以从格局 C_1 中产生格局 C_2，则 M_2 也可以从与 C_1 相应的格局 D_1 中产生与 C_2 相应的格局 D_2。证明细节我们略去。

最后，一旦 M_1 的计算停机了，M_2 则转入收尾状态：将纸带从双轨变回为单轨，使得纸带上 1 的个数等于上下轨上原有的 1 个数的总和，删掉 \$，将读写头移到第一个 1 之上（如果输出不是 0），停机。　□

推论 7.1. 任何一个图灵可计算函数 h 都可以被一台加了如下限制的图灵机计算：在初始格局中，纸带上有一个不在字母表中的新字符 \$，它可以在事先给定的任何位置，只要不混在输入字符串中间。在计算完成后，\$ 左边的纸带内容不变；而且输出字符串的位置起始于 \$ 右边第一格。

引理 7.14. 图灵可计算函数构成的类对复合运算封闭，即，令

$$f(x_1,x_2,\cdots,x_n)=g(h_1(x_1,x_2,\cdots,x_n),\cdots,h_r(x_1,x_2,\cdots,x_n))。$$

如果 g 和 $h_1,\cdots,h_r$ 都是图灵可计算的，则 f 也是。

证明概要：利用推论 **??**，并引入 $r+1$ 个新字符，分别用来标记纸带上储存输入 $x_1,x_2,\cdots,x_n$，和中间过渡的输出 $y_i=h_i(x_1,x_2,\cdots,x_n)$（$i=1,\cdots,r$）的区域。这

些过渡输出是通过已给的计算 h_i 的子程序而产生的。再调用已给的计算 g 的程序来计算 $g(y_1, y_2, \cdots, y_r)$ 并在计算完成后清理纸带，把它变成规定的输出格局即可。

类似地，我们也可以证明：图灵可计算函数构成的类对原始递归和极小算子都是封闭的（习题)。因而我们就证明了：

定理 7.2. 任何部分递归函数都是图灵可计算的。

7.4.2　从图灵可计算函数到部分递归函数

我们现在考虑另外一个方向，即，图灵可计算的函数都是部分递归的。

第一步：我们把一个图灵机 M 的所有“硬件和软件”(例如，格局和程序等等）都通过编码的方式转换成自然数。我们用符号 $\ulcorner O \urcorner$ 表示对象 O 的编码。

首先，我们假定图灵机的字母表为 $A = \{0, 1\}$ 其中 0 是空白。对表示两个方向符号，我们定义 $\ulcorner L \urcorner = 2$ 和 $\ulcorner R \urcorner = 3$。规定 M 的状态集 Q 为自然数的子集 $\{4, 5, \cdots, n\}$；其中状态 4 代表初始状态 q_s 且状态集中最大的自然数 n 代表停机状态 q_h。

接下来我们把每个四元组 $qaa'q'$ 编码成 $\langle q, a, a', q' \rangle = 2^{q+1}3^{a+1}5^{a'+1}7^{q'+1}$。如果 M 的四元组集为 $\delta = \{s_1, s_2, \cdots, s_n\}$，我们定义它的编码 $\ulcorner \delta \urcorner = \langle s_1, s_2, \cdots, s_n \rangle$。事实上，$e = \ulcorner \delta \urcorner$ 包含了 M 中的所有计算所需的信息，我们把它规定为 M 的编码，即，$e = \ulcorner M \urcorner$。

当然，具体的编码方式并不重要，重要的是编码和解码可以能行地进行，也就是说，我们可以能行地从编码 $\ulcorner M \urcorner$ 得到关于图灵机 M 全部程序。例如，我们有：

引理 7.15. 下列关于图灵机编码的谓词都是原始递归的：“e 是一个图灵机（程序）的编码”、“图灵机 e 中包含四元组 s”和“状态 q 是图灵机 e 的停机状态”。

证明: 练习。 □

我们接着给格局编码。给定一个格局 $C = \cdots b_1 b_0 q a c_0 c_1 \cdots$。注意靠近读写头的脚标较小，并且字母表只有 $\{0, 1\}$。定义读写头左右两边纸带内容的编码分别为：$x = \sum b_i 2^i$ 和 $y = \sum c_i 2^i$，注意这里的和实际上是有穷和。定义格局 C 的编码 $\ulcorner C \urcorner$ 为 $\langle x, q, a, y \rangle = 2^{x+1}3^{q+1}5^{a+1}7^{y+1}$。

引理 7.16. 谓词“c 是一个格局的编码”是原始递归的。

证明: 练习。 □

第二步: 有了格局编码之后，图灵机的计算过程就成为格局编码之间的转换过程。我们定义一些函数来描述这些转换，并且论证它们都是原始递归的。

给定一个图灵机 M 的编码 $e = \ulcorner M \urcorner$。我们引进下列函数帮助我们描述整个计算过程。

- **输入函数** $IN(x_1, x_2, \cdots, x_n) = \ulcorner C_0 \urcorner$，其中 C_0 是初始格局 $q_s\, 1^{x_1+1}01^{x_2+1}0\cdots01^{x_k+1}$。
- **转换函数** $NEXT$ 来描述一步计算：$NEXT(e, c) = d$ 当且仅当 c 和 d 分别是格局 C 和 D 的编码，并且 C 产生 D。注意：这里 C 产生 D 是与图灵机的码 e 有关的。而且，我们这里的描述是相容的，即可以适用于任何 e。
- 谓词 $TERM(e, c)$ 表示 c 是某个终止格局的编码。注意：这也与图灵机的码 e 有关。
- **输出函数** $OUT(c)$ 用来从终止格局的编码 c 中读出输出的值。实际上，我们可以定义得更宽一点，即，如果 $c = \ulcorner C \urcorner$ 且 $C = q1^y$（其中 q 是任意状态），则 $OUT(c) = y$。

引理 7.17. 函数 $IN, OUT, NEXT$ 和谓词 $TERM$ 都是原始递归的。

证明: 让我们验证 $TERM$ 是原始递归的，其余部分留作习题。$TERM(e, c)$ 成立当且仅当 c 是某个格局的编码，且 $(c)_1$（即，c 的第二个分量 q）是图灵机 e 的停机状态。根据引理 **??** 和 **??**，$TERM$ 是原始递归的。 □

定义谓词 $T(e, x, z)$ 为“z 是程序 e 对输入 x 的计算过程的编码”，称为**克林尼 T 谓词**。

引理 7.18. 克林尼 T 谓词 $T(e, x, z)$ 是原始递归的。

证明: 只需注意 $T(e, x, z)$ 成立当且仅当“z 是一个格局的有穷序列 $\langle \ulcorner C_0 \urcorner, \cdots, \ulcorner C_m \urcorner \rangle$ 的编码，并且 $\ulcorner C_0 \urcorner = IN(x)$ 和 $(\forall i < m) NEXT(e, \ulcorner C_i \urcorner) = \ulcorner C_{i+1} \urcorner$ 和 $TERM(e, \ulcorner C_m \urcorner)$”。 □

第 3 步：利用 μ-算子来寻找计算过程的编码，即克林尼 T 谓词 $T(e, \vec{x}, z)$ 中的 z。

引理 7.19. 如果一个函数 f 是图灵可计算的，则它是部分递归的。

证明: 假设 f 可以被图灵机 e 计算。

对于任意的 $\vec{x}$，我们首先用 μ-算子来寻找计算过程的编码 z。一旦我们找到了 z，则取出它的最后一项 $\ulcorner C_m \urcorner$；$f(\vec{x})$ 的值就是 $OUT(\ulcorner C_m \urcorner)$。

由于 $T(e, \vec{x}, z)$ 和 OUT 都是原始递归的，$f(\vec{x})$ 是部分递归的。 □

7.4.3　丘奇论题

小结一下：图灵可计算函数构成的类与部分递归函数构成的类相等。

在 1930 年代，人们从不同的角度来研究可计算性，图灵可计算函数和部分递归函数是两种不同的刻画，动机和方法都截然不同，但它们刻画的函数类却是一样的。此外，人们出于别的动机还定义了其它的函数类，例如，丘奇定义了 λ-演算和 λ-可计算性，但刻

画的也是同一个函数类。人们自然地猜测这并不是巧合，而一定有更本质的原因在后面。于是丘奇提出了下列论题：

丘奇论题：直观上的可计算函数类就是部分递归函数构成的类。因而也就是图灵可计算函数构成的类。

我们称它为论题，因为它不同于数学定理，它不是一个严格的命题。但在递归论中却是经常用到的，我们常常用“高级语言”写一个计算某个函数 f 算法，然后“根据丘奇论题”断言，f 是部分递归的。这样做的优点是避免了繁琐的对 f 的生成过程的讨论，直观上又非常清楚，而且又有一定的理论根据。

7.4.4　克林尼正规型定理

定理 7.3 (克林尼)**.** 存在原始递归函数 $U : \mathbb{N} \to \mathbb{N}$ 和原始递归谓词 $T(e, x, z)$ 使得对任意的部分递归函数 $f : \mathbb{N} \to \mathbb{N}$ 都存在一个自然数 e，满足 $f(x) = U(\mu z\, T(e, x, z))$。

推论 7.2. 一个函数是递归的当且仅当它是部分递归的全函数。

Proof. （$\Rightarrow$）同原始递归函数类似，每个递归函数也有一个生成序列。注意到正则性条件保证了对全函数使用正则极小算子仍得到全函数，通过对生成序列归纳我们很容易得到每个递归函数都是全函数。而根据定义，显然递归函数都是部分递归的。

（$\Leftarrow$）假定 f 是一个部分递归的全函数。根据克林尼正规型定理，对某个自然数 e，我们有 $f(x) = U(\mu z\, T(e, x, z))$。所以我们只是用 μ-算子一次，而且是对一个原始递归谓词 $T(e, x, z)$ 使用的。由于 f 是全函数，所以对任意的 x，满足 $T(e, x, z)$ 的 z 总是存在的，即这个极小算子是正则的。因此，f 是递归的。 □

从克林尼正规型定理，我们有以下重要推论。

定理 7.4 (通用函数定理)**.** 存在一个通用的部分递归函数；即，存在一个二元的部分递归函数 $\Phi : \mathbb{N}^2 \to \mathbb{N}$ 满足：对任何的一元递归函数 $f : \mathbb{N} \to \mathbb{N}$ 都存在一个自然数 e 使得对所有的 x 都有 $f(x) = \Phi(e, x)$。

证明: 只要取 $\Phi(e, x) = U(\mu z\, T(e, x, z))$ 即可。 □

通用函数 $\Phi(e, x)$ 的存在使得我们可以能行地把所有的部分递归函数枚举出来：

$$\varphi_0, \varphi_1, \cdots$$

其中 $\varphi_e(x) = \Phi(e, x)$。这也是我们考虑部分递归函数的原因之一。与之形成对照的是下面关于递归（全）函数的定理：

定理 7.5. 对递归函数来说不存在通用函数，即，不存在递归函数 $T : \mathbb{N}^2 \to \mathbb{N}$ 满足：对任何的一元递归函数 $f : \mathbb{N} \to \mathbb{N}$ 都存在一个自然数 e 使得对所有的 x 都有 $f(x) = T(e, x)$。

证明: 练习。 □

最后我们再给一个例子，它说明有些部分递归函数是不能通过把递归函数限制到它的定义域上而得到。因此，部分递归函数类实质地扩张了递归函数类。

例 7.8. 存在一个部分递归函数 $f(x)$ 使得对任何递归（全）函数 $g(x)$ 都存在自然数 $n \in \mathrm{dom}(f)$ 使得 $f(n) \neq g(n)$.

证明: 令 $f(x) = \Phi(x,x)+1$，其中 $\Phi(x,y)$ 为通用函数。考察任何一个递归全函数 g。固定 m 使得 $g(x) = \Phi(m,x)$。由于 $g(x)$ 是全函数，所以 $\Phi(m,m)$ 是有定义的，因而 $m \in \mathrm{dom}(f)$。但我们有 $f(m) = \Phi(m,m)+1 \neq \Phi(m,m) = g(m)$。 □

第 5 节　递归可枚举集

定义 7.11. 我们称一个集合 A 为*递归可枚举* 的，简称为 r.e. 的，[2]如果 $A = \emptyset$ 或者 A 是某个递归（全）函数 $f : \mathbb{N} \to \mathbb{N}$ 的值域，即，$A = \{y : \exists x\ f(x) = y\}$。

我们马上会看到递归可枚举集还有很多等价的刻画。我们选择了递归函数的值域作为基本定义，原因是它更能说明“枚举”这个词。例如 $f(0) = 7 \square f(1) = 2, f(2) = 7, f(3) = 4, \cdots$，我们很自然地会联想到一个枚举过程，第一个元素为 7, 第二个为 2，第三个仍是 7 (我们允许重复枚举), ……等等。$A = \mathrm{ran}(f) = \{2, 4, 7, \cdots\}$ 就是一个递归可枚举集。此外，递归枚举也让我们自然地会联想“能行地或系统地产生”。例如，所有普遍有效的闭语句可以能行地从逻辑公理中“产生”出来（通过列出它们的证明序列），因此是递归可枚举的。

引理 7.20. 令 A 为自然数 $\mathbb{N}$ 的一个子集。则下列命题等价:

(a) A 是递归可枚举的。

(b) A 是空集或 A 是某个原始递归函数的值域。

(c) A 是某个部分递归函数的值域。

(d) A 的部分特征函数是部分递归的，其中 A 的部分特征函数 $\chi_{A_P}(x)$ 定义如下:

$$\chi_{A_P}(x) = \begin{cases} 1, & \text{如果 } x \in A; \\ \text{没有定义}, & \text{否则。} \end{cases}$$

(e) A 是某个部分递归函数的定义域。

[2]r.e. 为英文 recursively enumerable 的缩写。

(f) 存在一个二元递归谓词 $R(x,y)$（事实上可取为原始递归谓词）使得 $A = \{x : \exists y\ R(x,y)\}$。

在证明之前，我们先解释一下每条的意义。前三条是一组，说明对一个非空的 r.e. 集来说，我们可以放松或收紧枚举它的函数的条件。(d) 说明用特征函数来刻画 r.e. 集是不方便的（参见习题）。顺便提一句，“递归可枚举”只能用来形容集合，称一个函数为“递归可枚举”是没有意义的。(e) 说明，从可计算的角度看，一个函数的定义域和值域区别不大。(f) 实际上是从可定义性（或说从定义的语法形式）上来刻画递归可枚举集的。

证明: 我们证明“(c) ⇒ (b) ⇒ (a) ⇒ (f) ⇒ (e) ⇒ (d) ⇒ (c)”。

“(c) ⇒ (b)”: 假设集合 A 满足 (c) 的条件，即，对某个部分递归函数 $f(x)$，$A = f[\mathbb{N}]$。根据克林尼正规型定理，存在原始递归函数 $U : \mathbb{N} \to \mathbb{N}$、原始递归谓词 $T(e,x,z)$ 和自然数 $e_0 \in \mathbb{N}$，使得 $f(x) = U(\mu z\, T(e_0,x,z))$。如果 $A = \emptyset$，则 (b) 显然成立。如果 $A \neq \emptyset$，则固定任何一个 $a_0 \in A$，定义 $F : \mathbb{N}^2 \to \mathbb{N}$ 如下:

$$F(x,n) = \begin{cases} U(\mu z \le n\, T(e_0,x,z)), & \text{如果 } \exists z \le n\, T(e_0,x,z); \\ a_0, & \text{否则。} \end{cases}$$

F 是原始递归的并且 $F[\mathbb{N}^2] = A$（练习）。最后令 $g(z) = F((z)_0,(z)_1)$，其中 $(z)_0$ 和 $(z)_1$ 是标准的原始递归的解码函数（例如，见习题 **??**），我们就得到了 (b) 所需要的一元原始递归函数 g。

“(b) ⇒ (a)”显然，任何原始递归函数都是递归函数。

“(a) ⇒ (f)”如果 $A = \emptyset$ 则取 $R = \emptyset$。现假定 $A \neq \emptyset$ 且 $A = f[\mathbb{N}]$ 是某个递归函数 f 的值域。令 $R(x,y)$ 为谓词 $|f(y) - x| = 0$。它是递归的，因为 $f(x)$ 是递归的。并且 $x \in A$ 当且仅当 $\exists y\ R(x,y)$。

“(f) ⇒ (e)”假定对某个递归谓词 R，我们有 $A = \{x : \exists y\ R(x,y)\}$。令 $g(x) = \mu y R(x,y)$。则 g 是部分递归的，且 $\mathrm{dom}(g) = A$。

“(e) ⇒ (d)”假定 $A = \mathrm{dom}(g)$ 是某个部分递归函数 g 的定义域。令 C_1 为恒等于 1 的常函数 $C_1(x) = 1$。则 $\chi_{A_P}(x) = C_1 \circ g$。所以它是部分递归的。

“(d) ⇒ (c)”假定 A 的部分特征函数 χ_{A_P} 是部分递归的。定义 $f : \mathbb{N} \to \mathbb{N}$ 为 $f(x) = x \cdot \chi_{A_P}(x)$。则 f 是部分递归的且 $\mathrm{dom}(f) = \mathrm{dom}(\chi_{A_P}) = A$。此外，对所有 $a \in A$，我们有 $f(a) = a \cdot 1 = a$。所以 (c) 成立。 □

定理 7.6. 一个自然数的集合 A 是递归的当且仅当集合 A 和它的补集 $\mathbb{N} \setminus A$ 都是递归可枚举的。

证明: 练习。 □

虽然我们在定义 **??** 中只对 $\mathbb{N}$ 的子集定义了递归可枚举这个概念，我们可以很自然地把它推广到 $\mathbb{N}^k$ 的子集上。例如，我们可以利用任何的原始递归的双射 $\phi : \mathbb{N}^k \to \mathbb{N}$，定义 $A \subseteq \mathbb{N}^k$ 是 r.e. 的，如果 A 在 ϕ 下的像 $\phi[A]$ 在 $\mathbb{N}$ 中是 r.e. 的。

定理 7.7. 假定 $\mathbb{N}^k$ 的子集 A 和 B 都是递归可枚举的。则

(a) 集合 $A \cup B$ 和 $A \cap B$ 都是递归可枚举的。

(b) 集合 $C = \{\vec{x} \in \mathbb{N}^{k-1} : \exists y\ (\vec{x}, y) \in A\}$ 也是递归可枚举的，即，递归可枚举关系对存在量词封闭。

证明: 练习。 □

定理 7.8. 集合 $K = \{e : \varphi_e(e) \text{ 有定义 }\}$ 是递归可枚举的，但不是递归的。

由于 $\varphi_e(e) \downarrow$ 说的是第 e 台图灵机对输入 e 停机，集合 K 也被称为停机问题。定理 **??** 也被叙述成“停机问题是不可判定的”。

证明: 首先 K 是 r.e. 的，因为它是部分递归函数 $\Phi(x, x)$ 的定义域。

假定 K 是递归的，则它的补集 $\mathbb{N} \setminus K$ 也是递归的。因此 $x \in K$ 和 $x \notin K$ 都是递归谓词。根据分情形定义定理，函数

$$f(x) = \begin{cases} \varphi_x(x) + 1, & \text{如果 } x \in K; \\ 0, & \text{如果 } x \notin K。 \end{cases}$$

也是递归的。因此，存在自然数 e，使得对所有的 x 都有 $f(x) = \varphi_e(x)$。考察 $x = e$：如果 $e \in K$，则 $f(e) = \varphi_e(e) + 1 \neq \varphi_e(e)$，矛盾；如果 $x \notin K$，则 $f(e) = 0$，而 $\varphi_e(e) \uparrow$，也矛盾。因此，K 不是递归的。 □

推论 7.3. 递归可枚举集不对取补运算封闭。

证明: 考察定理 **??** 中的递归可枚举集 K，如果 $\mathbb{N} \setminus K$ 是递归可枚举的，则根据定理 **??** K 就是递归的。与定理 **??** 矛盾。 □