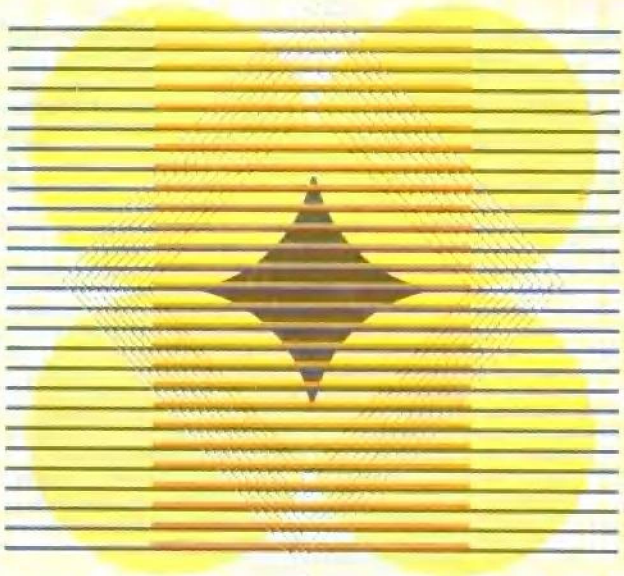
可计算性理论

杨东屏 李昂生 著



科学出版社

中国科学院科学出版基金资助出版

可计算性理论

杨东屏　李昂生 著

科 学 出 版 社

1999

内 容 简 介

本书全面系统地介绍了50年代至今在可计算性理论方面的主要方法与成果. 主要内容包括:可计算性理论基础知识,可计算枚举集,有穷和无穷延伸方法,有穷损害优先方法,无穷损害优先方法,计算复杂性理论,及时单纯集和间段、余间段方法,n-可计算枚举集和可计算逼近函数的图灵度,树构造和O'''方法,圆界极小度定理.

本书可供大学数学系和计算机科学系的教师和研究生、科研人员阅读.

图书在版编目(CIP)数据

可计算性理论/杨东屏,李昂生著. -北京:科学出版社,1999

ISBN 7-03-006378-3

Ⅰ. 可… Ⅱ. ①杨… ②李… Ⅲ. 可计算性-理论 Ⅳ. O141.3

中国版本图书馆CIP数据核字(97)第24188号

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

新蕾印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1999年4月第 一 版 开本:850×1168 1/32

1999年4月第一次印刷 印张:11 3/4

印数:1—2 300 字数:310 000

定价:28.00元

(如有印装质量问题,我社负责调换(新欣))

前　言

可计算性理论产生于对算法概念的数学研究，它是研究可计算对象的计算复杂性和不可计算对象的结构的学科，是数理逻辑的主要领域之一，也是计算机科学的理论基础．它的目的是研究计算和相对计算的本质．

在本世纪 30 年代产生了几种等价的刻划算法本质的精确定义，其中主要有 λ －可定义函数，递归函数，图灵可计算函数以及由波斯特产生式系统定义的算法可产生集．随着人们寻求谓词演算系统判定过程的失败，特别是著名的哥德尔不完全性定理的出现，人们转向了对不可计算对象的研究，找到了许多不可计算的例子．在这基础上也开始了相对可计算性的研究．

图灵归约及其它归约形式使人们可以对不可计算对象加以分类，以研究不可计算对象的数学结构．在这过程中也产生了研究算法和相对可计算性的工具，如对角线方法和损害优先方法．

可计算性理论的出现给其它学科也带来了好处，例如 A. M. Turing 的通用图灵机为现代通用计算机体系设计思想的产生提供了理论准备。

我们大体上沿历史发展的线索，介绍可计算性理论在半个多世纪以来的研究中所产生的一些基本概念、基本理论和基本方法．在第十、第十一章，我们介绍了一些最近的结果和方法．书中也介绍了中国学者的一些工作．书末丰富的参考文献提供了一个极大的资料源泉．

杨东屏写了本书的前八章，李昂生写了其余章节．眭跃飞为本书的部分章节做了细致的审校工作．吴国华、张平也为本书做了大量的工作．

在写作过程中，我们参考了国际上有关的主要书籍，例如，M. Lerman 的 Degrees of Unsolvability, R. I. Soare 的 Recursively

Enumerable Sets and Degrees, P. Odifreddi 的 Classical Recursion Theory 以及 R. Shore 在我国讲学时用的讲义 Constructing Recursively Enumerable Sets. 除上述书籍外，我们还参考了大量的国内外文献. 在此我们感谢关心和支持本书的一切国内外同行. 我们还特别感谢科学出版社第一编辑室的全体同志，特别是毕颖同志为本书做了大量的编辑和修改工作.

由于我们水平有限，书中错误和不妥之处在所难免，欢迎读者批评指正.

杨东屏、李昂生

1999 年 1 月

目　录

第一章 可计算性理论基础知识

§1 关于可计算性的基本概念

可计算性理论是由递归函数的产生而开始发展的. 递归函数是算法这个直观概念的精确定义. 大约在公元前 3 世纪, 古希腊和中国的数学家就有了算法的概念, 人们就开始寻找解决各种数学问题的算法, 最著名的是被称为欧几里德算法的辗转相除法. 古代的印度人也注意寻求算法. 在印度人的影响下, 阿拉伯人也寻求各种数学问题的算法. 英文字 Algorithm 的意思是算法, 但它是外来语, 是由阿拉伯著名数学家穆罕莫得 · 以布 · 穆萨 · 阿尔哥里兹米的名字演变来的. 阿尔哥里兹米是生活在大约公元 780 年到 850 年之间的人, 他给出了大家熟悉的关于自然数运算的规则. 在公元 7 世纪时印度的布拉马古布塔给出了对任意整数 a, b, c, 丢方图方程 $ax+by+c=0$ 是否有整数解 x,y 的算法. 这在当时是相当难的, 因为它包含了检验 a,b 的最大公因数是否可被 c 整除的算法.

在阿拉伯人的影响下, 西班牙的莱芒德 · 卢利 (1235—1315) 要找发现真理的算法, 他还提出要制造机器来实现推理过程. 也许我们可以认为他是最早设想制造推理机的人. 他的想法虽然不能为他同时代人接受, 但对后世是有影响的.

在卢利影响下, 卡丹诺 (1501—1576) 和法拉里 (1522—1565) 分别给出了三次和四次方程根式解的算法. 后来寻求五次和更高次方程根式解的算法产生了伽罗瓦理论.

法兰西斯、维也特系统地使用了符号, 这对日后算法的发展有重要作用.

由于在代数里找到了大量的算法, 笛卡尔 (1598—1650) 寻求几何证明的算法. 为了要利用代数里已有的算法, 他引进坐标系, 用代数式来刻画几何对象, 从而产生了解析几何.

1623 年，施卡特制造了第一架计算器，第一次有了协助人进行思维活动的机械装置. 稍后在 1641 年巴士噶 (1623—1662) 也造出了计算器.

莱布尼兹 (1646—1716) 在卢利的影响下要找出一通用算法以发现一切真理，至少可以发现一切数学定理. 为此他要建立一个可以陈述一切数学命题的语言，以及用此语言去解决一切数学问题的算法. 由此他创造出了逻辑演算，后来发展成为今天的谓词演算. 而他要找的算法即谓词演算的判定算法，现在已经知道这样的算法是不存在的.

数学家们在寻求算法的过程中，发现越来越多的问题找不到解决它们的算法，于是人们怀疑是否有的数学问题不可能存在解决它们的算法，这样就提出了要数学地处理算法这个概念的要求. 由于可以通过编码的办法用自然数来描写算法问题，所以定义算法可计算函数的问题可以归结为寻找用数论函数定义的算法可计算函数. 而当时人们所知道的算法可计算函数都是原始递归函数，所以不少人认为原始递归函数可作为算法可计算函数的数学定义.

一切原始递归函数的类是什么样的函数类呢？它们的定义如下：

原始递归函数类 是由初始函数

零函数， $0(x_1,\cdots,x_n)=0$;

后继函数， $s(x)=x+1$;

投影函数， $U_i^n(x_1,\cdots,x_n)=x_i,\ 1\leq i\leq n$,

经过 **代入算子**,

$$h(x_1,\cdots,x_n)=f(g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n))$$

和 **原始递归算子**,

$$\begin{cases} h(x_1,\cdots,x_n,0)=f(x_1,\cdots,x_n); \\ h(x_1,\cdots,x_n,y+1)=g(x_1,\cdots,x_n,y,h(x_1,\cdots,x_n,y)) \end{cases}$$

而产生的最小函数类.

可知如下的一些函数是原始递归函数：

$x+y$. 它可以定义为

$$\begin{cases} x+0=x; \\ x+(y+1)=(x+y)+1=s(x+y). \end{cases}$$

$x\cdot y$. 它可以定义为

$$\begin{cases} x\cdot 0=0; \\ x\cdot(y+1)=x\cdot y+x. \end{cases}$$

x^y. 它可以定义为

$$\begin{cases} x^0=1; \\ x^{y+1}=x^y\cdot x. \end{cases}$$

$x\dot{-}1$. 它可以定义为

$$\begin{cases} 0\dot{-}1=0; \\ (x+1)\dot{-}1=x. \end{cases}$$

$x\dot{-}y$. 算术差，它的含义是

$$x\dot{-}y=\begin{cases} x-y & \text{若}x\geq y; \\ 0 & \text{否则}. \end{cases}$$

它可定义为

$$\begin{cases} x\dot{-}0=x; \\ x\dot{-}(y+1)=(x\dot{-}y)\dot{-}1. \end{cases}$$

$\mathrm{sg}(x)$. 它的含义是

$$\mathrm{sg}(x)=\begin{cases} 0 & \text{若}x=0; \\ 1 & \text{若}x\neq 0. \end{cases}$$

它可定义为

$$\begin{cases} \mathrm{sg}(0)=0; \\ \mathrm{sg}(x+1)=1. \end{cases}$$

$\overline{\mathrm{sg}}(x)$. 它的含义是

$$\overline{\mathrm{sg}}(x)=\begin{cases}1 & \text{若}x=0;\\ 0 & \text{若}x\neq 0.\end{cases}$$

它可定义为 $\overline{\mathrm{sg}}(x)=1\dot{-}\mathrm{sg}(x)$, 它可由 $x\dot{-}y$ 及 $\mathrm{sg}(x)$ 两函数经代入而得.

$|\,x-y\,|$. $x-y$ 的绝对值, 它可定义为

$$|\,x-y\,|=(x\dot{-}y)+(y\dot{-}x).$$

$x!$. 它的含义是 x 的阶乘: $x!=1\cdot 2\cdot\cdots\cdot x$. 它可定义为

$$\begin{cases}0!=1;\\ (x+1)!=x!\cdot(x+1)=x!\cdot s(x).\end{cases}$$

$\min(x,y)$. 它的含义是 x 和 y 的极小值. 它可定义为

$$\min(x,y)=x\dot{-}(x\dot{-}y).$$

$\max(x,y)$. 它的含义是 x 和 y 的极大值. 它可定义为

$$\max(x,y)=x+(y\dot{-}x).$$

$\mathrm{rm}(x,y)$. 它的含义是 x 被 y 除后的余数, 我们约定 $\mathrm{rm}(x,0)=x$. 它的定义是

$$\begin{cases}\mathrm{rm}(0,y) &=0;\\ \mathrm{rm}(x+1,y) &=(\mathrm{rm}(x,y)+1)\cdot\mathrm{sg}(|\,y-(\mathrm{rm}(x,y)+1)\,|)\\ &=(\mathrm{rm}(x,y)+1)\cdot\mathrm{sg}(|\,y-s(\mathrm{rm}(x,y))\,|).\end{cases}$$

$\mathrm{qt}(x,y)$. 它的含义是 x 被 y 除后的商, 为了使 $\mathrm{qt}(x,y)$ 是一全函数, 定义 $\mathrm{qt}(x,0)=0$. 它可以定义为

$$\begin{cases}\mathrm{qt}(0,y)=0;\\ \mathrm{qt}(x+1,y)=\mathrm{qt}(x,y)+\overline{\mathrm{sg}}(|\,y-(\mathrm{rm}(x,y)+1)\,|).\end{cases}$$

这是因为

$$\mathrm{qt}(x+1,y)=\begin{cases}\mathrm{qt}(x,y)+1 & \\ \quad (=s(\mathrm{qt}(x,y))) & \text{若}\mathrm{rm}(x,y)+1=y;\\ \mathrm{qt}(x,y) & \text{若}\mathrm{rm}(x,y)+1\neq y.\end{cases}$$

$\mathrm{div}(x,y)$. 它的含义是 x 被 y 除的特征函数，即

$$\mathrm{div}(x,y)=\begin{cases}1 & \text{若}y\text{除尽}x;\\ 0 & \text{否则.}\end{cases}$$

我们约定，当 $y=0$ 时， 0 除尽 y, 当 $y\neq 0$ 时， 0 不能除尽 y. $\mathrm{div}(x,y)$ 可定义为 $\mathrm{div}(x,y)=\overline{\mathrm{sg}}(\mathrm{rm}(x,y))$.

我们称一谓词 $P(x)$ 是**原始递归谓词**, 若有原始递归函数 $p(x)$ 使得： $P(x)$ 真当且仅当 $p(x)=1$; $P(x)$ 假当且仅当 $p(x)=0$. 可知：若 $P(x)$ 为原始递归谓词，则 $\neg P(x)$ 亦为原始递归谓词. 若 $P(x)$ 和 $Q(x)$ 皆为原始递归谓词，则 $P(x)\wedge Q(x)$ 与 $P(x)\vee Q(x)$ 亦为原始递归谓词. 这是因为若 $p(x)$ 与 $q(x)$ 分别是 P 与 Q 的特征函数，则 $\neg P(x)$ 的特征函数是 $1\dot{-}p(x)$, $P(x)\wedge Q(x)$ 的特征函数是 $p(x)\cdot q(x)$, $P(x)\vee Q(x)$ 的特征函数是 $\max(p(x),q(x))$.

对受囿和算子 $\sum\limits_{z<y}f(x,z)$ 和受囿积算子 $\prod\limits_{z<y}(f(x,z))$, 原始递归函数类也封闭. 这是因为

$$\begin{cases}\sum\limits_{z<0}f(x,z)=0;\\ \sum\limits_{z<y+1}f(x,z)=\sum\limits_{z<y}f(x,z)+f(x,y).\end{cases}$$

相仿可知， $\prod\limits_{z<y}f(x,z)$ 算子对原始递归函数类封闭. 我们以 $\mu z<yP(z)$ 表示最小的 $z<y$ 使得 $P(z)$ 成立. 我们称

$$\begin{aligned}g(x,y) &=\mu z<y[f(x,z)=0]\\ &=\begin{cases}\text{使}f(x,z)=0\text{的最小}z<y & \text{若如此}z\text{存在;}\\ y & \text{否则.}\end{cases}\end{aligned}$$

为 **受囿极小算子**, 可以证明原始递归函数类对受囿极小算子封闭, 这可以如下证明: 对给定 x, y, v, 设

$$h(x,v) = \prod_{u \le v} \mathrm{sg}(f(x,u)),$$

对给定 x, y, 设 $z_0 = \mu z < y[f(x,z) = 0]$, 可知当 $v < z_0$ 时 $h(x,v) = 1$; 当 $z_0 \le v < y$ 时 $h(x,v) = 0$. 所以

$$z_0 = \text{小于}y\text{满足}h(x,v)\text{等于1的那些}v\text{的个数} = \sum_{v<y} h(x,v).$$

于是 $\mu z < y[f(x,z) = 0] = \sum_{v<y} [\prod_{u \le v} \mathrm{sg}(f(x,y))]$.

若 $f(x,w)$ 和 $k(x,w)$ 是原始递归函数, 同样可以证明, 函数 $\mu z < k(x,w)[f(x,z) = 0]$ 也是原始递归函数.

可知上面的 x 改为 $x_1, x_2, \cdots, x_n$ 时, 各结果仍然成立. 因此可知以下函数是原始递归函数:

$\mathrm{D}(x)$. 它的含义是可以除尽 x 的除数的个数, 这里约定 $\mathrm{D}(0) = 1$. 它可以定义为

$$\mathrm{D}(x) = \sum_{y \le x} \mathrm{div}(x,y).$$

$$\begin{aligned} \mathrm{pr}(x) &= \begin{cases} 1 & \text{若}\mathrm{D}(x) = 2(\text{即}x\text{大于1且只有1和}x\text{除尽}x); \\ 0 & \text{否则}. \end{cases} \\ &= \overline{\mathrm{sg}}(|\ \mathrm{D}(x) - 2\ |). \end{aligned}$$

p_x. 它的含义是第 x 个素数; 它可定义为

$$\begin{cases} p_0 = 0; \\ p_{x+1} = \mu z \le (\mathrm{p}_x! + 1)[z > \mathrm{p}_x \text{且} z \text{是素数}]. \end{cases}$$

所以 $\mathrm{p}_0 = 0, \mathrm{p}_1 = 2, \mathrm{p}_2 = 3, \cdots$, 它们由 p_1 开始枚举一切素数.

$(x)_y$. 它的含义是 x 的素数分解中第 y 个素数的幂指数. 它可以定义为: $(x)_y = \mu z < x(p_y^{z+1}$ 不能除尽 $x)$, 即 p_y^z 是 x 的因数, 但 p_y^{z+1} 不是.

以后我们经常要用一组配对函数 $\pi(x,y), \pi_1(z), \pi_2(z)$ 来建立 N 和 $N \times N$ 之间的一一对应, 即

$$\pi(\pi_1(z), \pi_2(z)) = z.$$

我们可以用如下定义的 π, π_1, π_2 作为 **配对函数**, 显然它们都是原始递归函数,

$$\begin{aligned}
&\pi(m,n) = 2^m(2n+1) - 1,\\
&\pi_1(p) = \mu x \leq p[(\exists y \leq p)\pi(x,y) = p],\\
&\pi_2(p) = \mu y \leq p[(\exists x \leq p)\pi(x,y) = p].
\end{aligned}$$

以后我们有时以 $\langle x,y\rangle$ 表示 $\pi(x,y)$.

既然上面这么多的函数都是原始递归函数, 会不会一切算法可计算函数都是原始递归函数? 1928 年, 阿克曼 (Ackermann) 给出了一个算法可计算函数并不是原始递归函数.

我们可以引进由 $n+1$ 元函数 $f(y,x_1,\cdots,x_n)$ 到 n 元函数

$$g(x_1,\cdots,x_n)$$

的 **求最小数算子**, 记为

$$g(x_1,\cdots,x_n) = \mu y[f(y,x_1,\cdots,x_n) = 0].$$

求最小数算子也称为 $\mu-$ **算子**.

函数 $g(x_1,\cdots,x_n)$ 可以是一个部分可定义函数, 即对某些

$$x_1,\cdots,x_n,$$

函数值 $g(x_1,\cdots,x_n)$ 无定义, 也就是说, 不存在 y 满足

$$f(y,x_1,\cdots,x_n) = 0.$$

原始递归函数类加上 $\mu-$ 算子生成的最小函数类称为 **部分递归函数类**, 因为其中有的函数不一定对每一个非负整数都有定义, 即不是全函数. 如果一个部分递归函数还是全函数 (即定义域为非负整数集), 则称它为 **递归函数**.

在 20 世纪 30 年代, 人们在算法可计算函数的研究中取得了一些结果. 首先彻奇 (Church) 定义了 $\lambda-$ 演算, 他和他的学生克林尼 (Kleene) 定义了 $\lambda-$ 可定义函数, 把它作为算法可计算函数的定义. 1934 年哥德尔 (Gödel) 在普林斯顿做讲演, 讲解他的著名的 **不完全性定理** 时, 提到了算法可计算函数的定义, 讲了经他改正的耳布朗 (Herbrand) 的想法. 克林尼当时为哥德尔演讲做记录. 其后, 克林尼定义出了递归演算系统, 由此定义了递归函数类, 其后克林尼又证明了它和在原始递归函数类之外加上 $\mu-$ 算子后定义出的函数类等价.

还有其他的一些著名的定义算法可计算函数的方法. 例如图灵机、马尔可夫正规算法、波斯特演算.

现简单地讲一下 **图灵机** 的办法. 当美国人给出各种算法可计算函数的定义方法时, 在英国, 图灵 (Turing) 也在进行同样的工作. 图灵分析了人类进行算法地计算的过程, 定义一个机器来模拟人的工作过程. 当人们在用纸和笔进行计算时, 要在纸的一定部位写上或擦去所用的符号, 人在计算过程中眼光在纸上移动以进行计算. 人们还要经常地提醒自己现在正要做的事, 即随时确定自己在计算过程中的哪种情况之下, 换言之, 要决定自己处于什么状态之下, 下一步要干什么事.

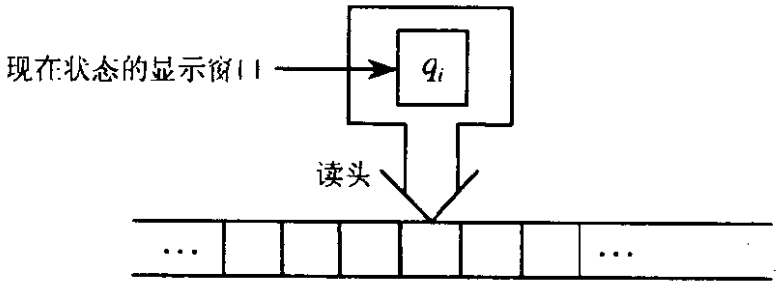


图1.1

按此想法，图灵定义了图灵机. 这机器有一个有穷字母表 $S=\{s_0,s_1,\cdots,s_n\}$，一个有穷的状态集 $Q=\{q_1,\cdots,q_m\}$. 机器要处理的信息放在一个两端趋于无穷，被分成一个一个小格的带上. 通常在这带上只有有穷多个格子里不是放字母 s_0，即不是空格. 也就是说只有有穷多个格子里放有 $j>0$ 的 s_j(其中 $1\leq j\leq n$). 机器有一个 **读头**，它在任何时刻注视带子上的某个小格，其结构如图 1.1.

机器做的事可以是，在注视着的格子内擦去原有的符号，写上另一符号；把读头左移一格或右移一格. 机器在每一时刻都处于某个状态，并要执行某个指令，当执行完这指令后进入另一状态. 机器的指令是如下形式的四元组：

$$\langle q_i,s_j,s_k,q_l\rangle, \tag{1.1}$$

或

$$\langle q_i,s_j,R,q_l\rangle, \tag{1.2}$$

或

$$\langle q_i,s_j,L,q_l\rangle. \tag{1.3}$$

(1.1) 的含意是：机器当前处于状态 q_i，注视的格子里的符号是 s_j，这时才可执行指令 (1.1). 它的动作是在注视着的格子里，擦去 s_j，写上 s_k，然后进入状态 q_l. (1.2) 的含意是：机器当前处于状态 q_i，注视的格子里的符号是 s_j，动作里读头向右移一格，然后进入状态 q_l. (1.3) 的含意类似 (1.2)，只是改右移为左移.

给定一图灵机，即给定一指令集 P. 图灵机在进行计算前，机器的带子上被放上一定的信息，例如是要计算的函数的输入值，机器也注视着某个特定的方格. 图灵机的计算如下：若机器正处于状态 q_i，读头注视的方格内符号为 s_j，那么机器在 P 中找头两个符号为 q_is_j 的指令. 通常用的图灵机是确定性图灵机，即其指令集 P 中的任二不同指令的头两个符号必需不完全相同，即至少

有一个不相同. 所以在计算的每一步, 若机器的状态为 q_i, 读头注视的符号为 s_j, 那么 P 中最多有一个指令的头两个符号为 q_is_j. 当有一步找不到可以执行的指令时, 机器停止计算, 此时图灵机带上的数据即为计算的结果. 一般我们称计算开始前, 带上的信息为机器的 **输入**; 计算结束时, 带上的信息为机器的 **输出**. 在通常情况下, 图灵机的字母表为 $\{0,1\}$, 即 $s_0=0, s_1=1$. 数字 n 则是以两边为 0, 当中为 $n+1$ 个 1 所组成的符号串来表示. 图灵机的计算可能会出现 "死循环", 例如当机器进入状态 $\langle q_i, s_j, q_i, s_j\rangle$, 则机器永远执行下去而得不到任何输出, 这时称该计算为 **发散**. 一个计算发散的情形有多种形式. 图灵机的计算只当机器停止 (即找不到合用指令) 后才产生一个 **输出**, 它定义为该停机状态下带子上 "1" 的个数. 如果一个图灵机, 例如 M, 对输入 n 停机, 则称为 M 对输入 n **收敛**. 因此通常情况下, 一个图灵机计算的函数是 $N^+=\{0,1,2,\cdots\}$ 上的一个部分函数. 一数论函数 $f(x)$ 称为是 **部分可计算函数**, 若有一图灵机使得, $f(n)=m$ 当且仅当, 当图灵机的输入为 n 时, 机器一定停机, 且其输出为 m. 一个部分可计算函数若还是全函数, 则称其为 **可计算函数**.

可以证明, 部分递归函数的概念和部分可计算函数的概念是等价的. 此外它们还和用 λ- 演算、马尔可夫正规算法、波斯特演算等系统定义的部分可计算函数类都等价. 那么上述定义是否都可以作为能行可计算函数的数学定义呢? 这点是无法严格地证明的, 因为人们无法证明有严格的数学定义的概念和只有直观含义而无精确数学描述的概念的等价性. 这点只能靠数学家的数学实践来检验. 为了说明这些严格定义的函数类的任一种, 如递归函数类和算法可计算函数是一致的, 彻奇提出了它的著名的论题:

彻奇论题: 一切算法可计算函数都是递归函数.

由于显然一切递归函数都是算法可计算函数, 所以彻奇论题肯定了两个函数类是相同的. 彻奇提出了他的论题后, 开始时, 许多人怀疑它的正确性. 例如, 他的学生克林尼就怀疑过, 他极力想找出一个反例, 即找出一个不是递归函数的算法可计算函数,