

# 图灵的停机问题及其对角线证法研究

杜立智,陈和平,符海东

(武汉科技大学 计算机科学与技术学院,湖北 武汉 430081)

**摘要:**停机问题是计算机科学领域的最经典问题之一,被认为是不可解的。证明停机问题不可解的方法主要包括对角线法和判定程序法,其中对角线法是康托尔对角线法的延伸。通过对康托尔对角线法、图灵关于停机问题不可解的对角线证法以及判定程序证法的深入分析,揭示了判定程序证明的本质,指出了在不影响判定程序设计初衷(即拥有对所有其他程序是否停机的判定功能)的前提下,该证明否定不了这样的判定程序存在性。同时揭示了对角线证明方法的根本缺陷和谬误。

**关键词:**康托尔;对角线;停机问题;图灵;不可解问题

中图分类号:TP31

文献标识码:A

文章编号:1673-629X(2016)12-0064-05

doi:10.3969/j.issn.1673-629X.2016.12.014

## Research on Turing's Halting Problem and Diagonal Method

DU Li-zhi, CHEN He-ping, FU Hai-dong

(College of Computer Science and Technology, Wuhan University of  
Science and Technology, Wuhan 430081, China)

**Abstract:** Halting problem is one of the most classical ones in computer science, which has been considered unresolvable. The methods of testification on the unresolvability of halting problem are composed of diagonal proof and that of program, in which the former is a stretch of canter's diagonal method. By deep study on Cantor's diagonal method, Turing's unresolvable proofs of the halting problem, and both the program proof and the diagonal proof, the essence of the program proof is revealed. It is pointed out that without loss of its original function, the judge program may exist. In addition, the essential deficiency and fault of the diagonal method, both Cantor's and Turing's, is revealed.

**Key words:** Cantor; diagonal; halting problem; turing; unresolvable problems

## 0 引言

计算机算法和计算复杂性理论是计算机科学中最重要的组成部分<sup>[1]</sup>。根据计算复杂性理论<sup>[2]</sup>,自然界和科学理论上的所有问题可以分为三大类:多项式问题,即可以在多项式时间内得到解答的问题;指数型问题,即可以在指数型时间内得到解答的问题;不可解问题,无论花多少时间,都不可能得到解答的问题<sup>[3-4]</sup>。对前两类问题,可以很容易找到确切的例子来进行说明,但对于最后一类,则很难找到具体的例子来说明。不可解问题与问题本身无解是两个不同的概念。例如,可以很容易地设计出一个方程,使得该方程无解,也能很容易地判断该方程无解。而不可解问题指的是,问题本身可能是有解的,只是无论花多长时间都得不到这个解。例如,停机问题就被证明是不可解的<sup>[5]</sup>,

著名的  $3x + 1$  问题到目前为止亦被认为是不可解的<sup>[6]</sup>。

所谓停机问题指的是:任给一个程序  $f$  和一个输入  $x$ ,是否存在一个判断程序  $P$  能判断  $f$  对  $x$  的计算是否停止。

停机问题最先是计算机科学史上最伟大的天才阿兰-图灵提出的,被称作计算机之父的冯-诺依曼曾说,图灵才是真正的计算机之父。图灵首先提出了停机问题并率先用对角线法证明:停机问题是不可解的。因此,停机问题得到了广泛研究,关于其不可解性,有两种不同的权威经典证明。一种是图灵的对角线证法,另一种方法是判定程序证法。

然而,图灵的对角线证法受到了质疑,不少人认为,这一证明实际上是存在逻辑问题的。但判定程序

收稿日期:2015-09-07

修回日期:2015-12-23

网络出版时间:2016-11-21

基金项目:2014年湖北省自然科学基金项目(2014CFC1121)

作者简介:杜立智(1964-),男,硕士,副教授,研究方向为计算机算法、计算数学、计算复杂性。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20161121.1633.008.html>

证明法则一直被认可。并且,这两种证明一直被大量地引用。文中通过详细分析这两个证明过程,包括对角线方法的历史,指出它们的根本缺陷或局限性。

### 1 康托尔与对角线方法

谈到停机问题,不能不提到集合理论的发明者康托尔及其对角线法。康托尔是用对角线法证明全体实数的个数大于全体自然数的个数。这个方法影响非常深远,直到后来的图灵停机问题、哥德尔定理其实都是该方法的不同延伸。

康托尔的主要贡献就是他的无穷集合理论<sup>[7]</sup>,无穷集合理论是建立在一一对应的思维方式上的。注意一一对应本身只是一种思维方式,并无实质价值。若不是他用对角线法将无穷集合分级,那一一对应就是无聊的了,因为那样的话,所有无穷集合都是一一对应的。也就是说,推翻了对角线法,就相当于推翻了康托尔的整个理论大厦,包括他的一一对应理论。

康托尔在研究无穷集合时,富有洞察性地看到了对无穷集合的大小问题<sup>[8]</sup>,不能再使用直观的“所含元素的个数”来描述,于是他创造性地将一一对应引入进来,两个无穷集合“大小”一样当且仅当它们的元素之间能够构成一一对应。这是一个非常直观的概念,一一对应,当然个数相等。然而这同时就是它不直观的地方了。对于无穷集合,日常的所谓“个数”的概念不管用了,因为无穷集合里面的元素个数本就是无穷多个。例如,说自然数集合能够跟偶数集合构成一一对应,从而自然数集合跟偶数集合里面元素“个数”是一样多的。怎么可能?偶数集合是自然数集合的真子集,所有偶数都是自然数,但自然数里面还包含奇数呢,说起来应该是二倍的关系吧?不是!只要这样来构造一一对应:

1 2 3 4 ...  
2 4 6 8...

用函数来描述就是 $f(n) = 2n$ 。检验一下是不是一一对应的。不可思议对吗?用这种一一对应的手法还可以得到很多惊人的结论,如一条直线上所有的点跟一个平面上所有的点构成一一对应(也就是说复数集合跟实数集合构成一一对应)<sup>[9]</sup>。以致连康托尔自己都不敢相信自己的眼睛了,这也就是为什么他在给戴得金的信中会说“我看到了它,却不敢相信它”的原因。然而,除了一一对应之外,还有没有不能构成一一对应的两个无穷集合呢?有。实数集合就比自然数集合要“大”,它们之间实际上无法构成一一对应。这就是康托尔的对角线方法要解决的问题。

证明实数的个数比自然数多,等价于证明实数集不可列,那么实数为什么不能与自然数建立一一对应

呢?康托尔的证明如下<sup>[10]</sup>:

假定(0,1)之间的实数可列,全部列出如下:

第一个数记为: $A_1 = 0.A_{11}A_{12}A_{13}\cdots$

第二个数记为: $A_2 = 0.A_{21}A_{22}A_{23}\cdots$

依此类推, $A_{ij}$ 代表列出来的第*i*个数的第*j*位小数,是个0到9之间的整数。同时,左边与这个数列一一对应的是全体自然数:1,2,...,*n*,...

下面构造一个属于(0,1)的实数 $B = 0.B_1B_2B_3\cdots$ ,不等于所列出的任何一个。

只要使 $B_1$ 不等于 $A_{11}$ (这样 $B$ 就不等于 $A_1$ ), $B_2$ 不等于 $A_{22}$ (这样 $B$ 就不等于 $A_2$ ).....依此, $B_i$ 不等于 $A_{ii}$ .....

这样构造的数 $B$ 是(0,1)中的实数而没有被列出来,于是实数不可列。于是认为此证明具有大问题。

(1)是逻辑上的问题,当他试图构造一个完全不一样的实数的时候,这一试图本身就与前面已经列出了“所有”的实数相矛盾,或者说,这一试图本身的前提是,前面并没有完全列出“所有”的实数。若是在有穷领域,当然可以通过这样的方式来反证,因为有穷领域是很直观的、看得很清的。但请注意这是在无穷领域,跟有穷领域的思维方式是不一样的。逻辑上无穷领域只要包含了“所有”,就无法再构建“新的”了。这个如同下述问题,上帝能制造一块连他自己都搬不动的石头吗?此问题常被用来“证明”不存在万能的上帝。但因前提已假设了结论(“自己都搬不动”),故这样的证明是无聊的。如同“上帝能搬动一块连他自己都搬不动的石头吗?”一样的无聊,为了避免后者明显的荒谬可笑,将其中之一的“搬动”改成了“制造”,但本质是一样的。既然假定了上帝万能(无穷),就不能再在其他前提表述中暗含上帝不能。请注意,从逻辑上,万能(无限能)是压倒一切的,正如康托尔那个假设包含了“所有”本身就意味着压倒了一切一样。

(2)他的对角线法是建立在假定宽和高严格相等的前提下,而在无穷领域,这样的假设有问题。即使是同级无穷大,这样的假设也说不过去。因为你是要构造一个具体的“对角线数”,以两个无穷领域的元素个数严格相等为前提基础来构建一个有穷领域的具体的数,逻辑上是站不住脚的,因为有穷领域的任何常数个数等同于无穷领域的0个数。他用增加了一行来反证原假设不成立,就好像在说,无穷大甲比无穷大乙多一个,这样的说法无疑是荒谬的。换句话说,康托尔的对角线法,相当于主张如下不等式成立: $\infty + 1 > \infty$ 。而这样的不等式显然是错误的。并且直觉上,他那个列表的高明显大于宽,否则实数就不连续了,也就是说不包含全部了。而为了使宽等于高,注意这里因为要构造一个具体的“对角线数”,宽和高就必须严格相等,为

了达到这一目的,惟一的方法就是补零。一旦补零,其潜在的含义就是,那个列表并没有包含全部的实数。

(3)他那个列表假定左边是全部的自然数,与右边全部的实数一一对应。既然右边假定实数全部列出,然后构建一个新的实数来否决这个假定,那左边为什么不能这样做呢?假定左边的自然数也全部列出,这个时候,若按照这个荒谬的对角线法,将自然数全部用二进制表示。同样,直觉上,或者说按照有限领域的理解,高明显大于宽,解决的办法是左边全部补零,这样问题就来了,也可以在对角线上(按取反)构建一个与前面所有自然数都不相同的自然数。总之,由于左边的全部自然数的宽和高也都是无穷的,因而也可以用对角线法予以否定。请注意,在宽和高的关系上,左边的自然数和右边的实数是有一定不同的,但你不能以此不同为根据来认定左边无对角线。而右边有对角线,否则就意味着前提里面包含了结论(自然数可数而实数不可数的结论)。

(4)再从另一个角度来看康托尔证法的荒谬。假定右边那个阵列列出了全部的小数,并且是从小到大排序,第一个小数各位全部是0,“最后”一个小数各位全部是9,中间无缝连续。请注意,不能否定这个假设,若是认为这个假设不成立,那就意味着前提里面包含了结论(不可列的结论)。所以,到目前为止,这个假设没有任何问题。此时能不能找到一个不在这个阵列当中的小数呢?毫无疑问不能。那么,康托尔为什么能构建出这样一个小数呢?关键是他设想了一条荒谬的根本不存在的对角线。在有穷领域,对角线必须是宽和高严格相等。而在无穷领域,宽和高相等的含义是什么呢?答案是:宽等于高是相等,宽等于高加1和高等于宽加1其实也是相等的。也就是说,在无穷领域,根本不存在一条确定的对角线,若一定要说有对角线,那对角线也是不确定的。用不存在或不确定的对角线来构造一个确定的小数无疑是荒谬的。

(5)再从有穷领域看看康托尔对角线法的诡辩性。任何一个宽和高也就是行和列相等的方阵,假定其每一行都不相同。那么,这样的方阵永远不可能包含用那些元素构成的所有行,因为都可以通过对角线取反得到一个与前面所有行不同的行。但要建立一个由那些元素组成的包含所有不同行的阵列是很容易办到的,只要不限定高必须与宽相等就行。为什么在无穷领域,康托尔既要假定阵列包含所有的行,同时又要规定阵列的宽和高必须严格相等呢?要知道,后一个规定本身就使得前一个假设不可能成立啊!

(6)他假定那个小数阵列的宽和高完全相等,实际上是运用了自己的无穷集合元素一一对应的理论,而一一对应理论之所以有效,取决于对角线证明的结

论,也就是说,他使用的是循环证法。

(7)无穷大无形状定理:无穷大没有确定的形状。

因为,假定哲学上的宇宙空间是无穷大,也就是说没有比它更大的空间,显然,这个假设没有任何问题。那么这个无穷大的形状是什么呢?若是方体,取其对角线作为新的边,构造另一个方体,比原方体大,与假设矛盾。同样,若是设那个无穷大是球体,以球的直径为边长构建另一方体,又比原球体大,矛盾。所以说,无穷大无形状。

康托尔对角线法的根本错误在于:他那个实数阵列,宽即实数的位数是无限的,高即实数的个数也是无限的,他假定这两个无限构成了一个正方形的形状,因之才有了对角线,这个假定违背了无穷大没有确定的形状,从而也是错误的。

请注意,这里只是否定他的对角线法,并不意味着否定实数是比自然数更大级别的无穷大。

康托尔对角线法还有另一种表述方式,即所谓排中律方式,其本质与上述对角线方式是一样的。

著名数学家布劳威尔认为,在无穷领域,应该从根本上禁用排中律<sup>[11]</sup>。也就是说,他认为排中律在无穷领域是不成立的。笔者上述第二条从否定 $\infty+1>\infty$ 着手否定康托尔的对角线,以及第七条的无穷大无形状定理,从实质上与布劳威尔认为排中律不适用无穷大领域的观点是一致的。

还有一种方式是通过改变概念和范畴来得出与康托尔相反的结论,从而否定他的理论。这样的方式争议很大,并没有得到公认。哲学家维特根斯坦从哲学的范畴质疑康托尔的对角线<sup>[12]</sup>。文中方法与这些有着本质的不同。文中是在传统数学和逻辑观念的范畴内来论及康托尔的对角线法的。

## 2 图灵关于停机问题的对角线证法

如前所述,图灵停机问题是指:存在不存在一个程序比如说 $P$ ,能够判断出任意一个程序 $X$ 是否会在输入数据 $Y$ 的情况下停机。不妨设 $P(X,Y)$ 表示 $P$ 判断程序是 $X$ ,数据是 $Y$ 的结果。如果停机,那么 $P(X,Y)$ 输出一个yes;如果不停机,那么 $P(X,Y)$ 输出一个no。这就是图灵停机问题。这样的程序 $P$ 是否存在。

图灵证明,这样的程序 $P$ 是不存在,他用反证法。

图灵证明这个问题其实是运用了康托尔的对角线删除法。

假设这样的 $P(X,Y)$ 存在。而己知程序 $X$ 本身是可以被编码的。也就是可以为所有的程序进行编号: $x_1, x_2, \dots$ 。而数据 $Y$ 本身也是这样的编号: $y_1, y_2, \dots$ 。因而就可以把每一对 $X$ 和 $Y$ 的组合排列在一张表上。比如横行表示的是数据 $Y$ ,而纵行表示的是程序 $X$ ,



这样  $P(X,Y)$  的值也就是 yes 或 no,可以写在第  $X$  行第  $Y$  列的对应位置上,表示  $P(X,Y)$  判断出的  $X$  作用在输入  $Y$  上是否会出现死循环的结果。示例如下:

	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$\cdots$	$Y_n$	$\cdots$
$x_1$	:yes	no	no	yes	yes	no	$\cdots$	yes	$\cdots$
$x_2$	:no	no	yes	yes	no	yes	$\cdots$	no	$\cdots$
$x_3$	:yes	yes	no	no	yes	no	$\cdots$	no	$\cdots$
$\cdots$	$\cdots$								
$x_n$	:no	yes	no	yes	yes	no	$\cdots$	yes	$\cdots$
$\cdots$	$\cdots$								

上表中的每一行都表示一个确定的程序作用到不同的数据上所得到的结果。比如程序  $X_n$  作用在  $y_1 y_2 y_3 y_4 y_5 y_6 \cdots Y_n \cdots$  上给出的结果就是一个序列: no yes no yes yes no  $\cdots$  yes  $\cdots$

下面把对角线上的元素挑出来。也就是专门找那些第 1 行第 1 列,第 2 行第 2 列,第 3 行第 3 列 $\cdots$ 这样的元素就可以得到一个序列:yes,no,no,  $\cdots$

而根据这个序列完全可以构造这样一个反序列: no,yes,yes,  $\cdots$

也就是说这个序列在每一个位上都与前一个对角线序列不同!这个序列就称为对角线删除序列。那么这个对角线删除序列是否在上表中的某一行上呢?答案是否定的!这个序列必然不在表中。因为它的第一个元素与 1,1 不同,第二个元素与 2,2 不同,第三个元素与 3,3 不同 $\cdots$ 所以这个构造出来的对角线删除序列不在表中。

完全可以设计出一段程序  $Q$  使得  $Q$  作用在数据上产生的序列就是对角线删除序列,而对角线删除序列不在表中就意味着程序  $P$  并不能判断出程序  $Q$  作用在任意输入上是否停机。用对角线删除的证明方法来看究竟哪里出错。错误就出在假设程序  $P$  能够得到这样一张完整的表,这张表对所有的程序计算能得到是否停机的答案。

可以看到,首先,图灵的对角线证明包含了康托尔证明的全部谬误,其中最根本的谬误一是假定宽和高严格相等,二是暗含了这样荒谬的不等式: $\infty + 1 > \infty$ 。并且,图灵的对角线证明其荒谬性比康托尔的更明显,因为图灵这个列表的高度,也就是  $x$  的个数更随意,能用对角线得到一行新的  $X$  本身就说明那个列表并不包含全部。请注意,完全可以在某种限定条件下设计一套简明的程序和输入,其个数也是无穷的,在此前提下,完全可以设计出一个判定程序  $P$ ,对所有这些程序和输入做出正确的判断。这在逻辑上简单明了,没有任何问题,但此时若是用对角线法,就会立马否定  $P$  的存在。仅此一点就足以表明,图灵的对角线证法是完全荒谬的。

### 3 停机问题的判定程序证法及分析

如前所述,图灵停机问题是指:存在不存在一个程序比如说  $P$ ,能够判断出任意一个程序  $X$  是否会在输入数据  $Y$  的情况下停机。不妨设  $P(X,Y)$  表示  $P$  判断程序是  $X$ ,数据是  $Y$  的结果。如果停机,那么  $P(X,Y)$  就输出一个 yes;如果不停机,那么  $P(X,Y)$  就输出一个 no。这就是图灵停机问题。这样的程序  $P$  存在吗?

可以设计一个程序来证明这样的程序  $P$  是不存在的,用反证法证明如下:

假设程序  $P$  存在。那么可以根据  $P$  设计一个新的程序  $Q$ :

$X$  是任何一段程序的编码:

```
Program Q(X) {
    m = P(X,X)
    do while ( m =yes)
        ...
    end do
    if m = no then return }
```

这段程序通俗来讲就是:输入任何一段程序  $X$ ,调用函数  $P(X,X)$  并得到返回值  $m$ ,如果  $m = \text{yes}$ ,也就是说根据  $P$  的定义, $P$  判断出程序  $X$  作用到它自己身上时停机。那么  $Q$  就不停地做 do while 和 end do 之间的语句。如果  $m = \text{no}$ ,表示  $P$  判断出程序  $X$  在  $X$  上不停机,就返回,结束该函数。

可以看到,这样定义的函数  $Q(X)$  是没有问题的。下面就进入关键时刻了:问  $Q$  这个程序作用到  $Q$  自身的编码上也就是  $Q(Q)$  会不会死循环呢?当然可以运用强有力的函数  $P(Q,Q)$  来计算这个问题。

假设  $Q(Q)$  会发生死循环,那么  $P(Q,Q)$  就返回 no。然而根据  $Q$  函数的定义,把  $X = Q$  代入其中会发现由于  $P(Q,Q)$  返回的是 no,也就是  $m = \text{no}$ ,因此  $Q$  函数会马上结束,也就是程序  $Q(Q)$  没有发生死循环。然而如果假设  $Q(Q)$  不发生死循环,那么  $P(Q,Q)$  应该返回 yes,这样根据  $Q$  函数的定义,把  $X = Q$  代入  $Q(Q)$  之中会得到  $m = \text{yes}$ ,这样程序就会进入 do while 循环,而这个循环显然是一个死循环。因而  $Q(Q)$  发生了死循环!这又导致了矛盾。

无论  $Q(Q)$  会不会发生死循环,都会产生矛盾,然而哪里错了呢?答案只能是最开始假设的前提错了,也就是说最开始的假设  $P(X,Y)$  能够判断任意程序  $X$  在输入  $Y$  的时候是否死循环是错误的!也就是说这样的程序  $P(X,Y)$  不存在。对于此证明,分析如下:

(1) 可以看到,该证明本质上是沿用了集合悖论的思维方式。集合悖论可以通过某种约定或限定来解决,那就是慎用集合的集合。同样,若是限定被判定的

程序不能调用判定程序,那上述证明就不成立。

(2)集合的各元素之间是没有时间方向的,而判定程序和被判定程序之间有时间方向,判定程序在层次上高于被判定程序,在时间上延后于被判定程序,就像法官能判决犯人,而犯人不能反过来判决法官一样,因而能自然地绕开上述矛盾。也就是说,上述矛盾或悖论实质上是不成在的。

(3)此段程序形式上构成了一个悖论,用程序来设计悖论大约是创新,当然,危险也就来了,因为合法的程序有严格要求。程序是递归的,下面是笔者对递归程序的理解:

许多复杂问题应用递归方法易于解决,否则极难。计算机课程的许多部分都用到递归,因而理解和掌握递归精髓极具意义<sup>[13]</sup>。这里要强调的是,能对较复杂的递归在头脑中形成清晰的思维脉络是对思维能力的极大锻炼,其中多重循环里的条件递归最难,完成它需要极强的算法构思力。递归设计有三个关键点:一是降阶,二是传值,三是保护现场。所谓降阶就是每递一次,总阶数应降一次,否则就会死循环,或导致无意义的交错重复。所谓传值,就是递归的各子函数之间如何进行传值联系。所谓保护现场,实质上也就是将所有递归过程串成一个整体思路。调用递归函数前后的代码越复杂,这第三点也就越难。因此应尽可能使递归函数前后的代码简单化。解决了这三点,递归也就设计好了。一般通过递归函数的形参解决前两点问题。也有通过全局变量来传值的。此时应注意内存的使用效率以及变量值的覆盖顺序是否会容易错等。此外,在降阶的同时,递归程序通常还要对某个基数进行直接计算,若是输入小于等于这个基数,直接计算,大于这个基数,就递归。

若不按这些规则去做,会导致无聊的程序,例如:

```
bool M ( int n )
{
    bool OK = ! M(n) ; //递归,符号! 表示 Not(否定)
    return OK;
}
```

此程序之所以无聊,关键是参数  $n$  没有降阶,并且程序中没有对基数进行计算。若是作如下修改,程序就是有意义的了:

```
bool M ( int n )
{if( n <= 1 ) return TRUE;
    bool OK = ! M(n - 1) ; //递归,符号! 表示 Not(否定)
    return OK;
} 万方数据
```

上述关于停机问题的程序  $Q$  实际上是暗含了递归,但没有包含降阶和初始计算,从而这样的递归程序是有问题的。若是对递归程序进行规范限定,则上述证明就不成立。

## 4 结束语

文中对停机问题不可解的两种证明方法,即对角线法和判定程序法,进行了详细分析。结论是对角线证明法根本不成立;若是进行某种逻辑上或程序规范方面的限制,这样的限制并不影响判定程序正常的判定功能,则判定程序证明法亦不成立。

认为这两种证明从本质上讲是不相同的。判定程序法要求被判定者必须反调用判定程序本身,由此推出悖论。假使不要求被判定者调用判定者的话,该方法就证明不了停机问题的不可解。换言之,若是限定被判定者不能调用判定程序,则判定程序是可能存在的。而对角线法则没有要求被判定者必须反调用判定程序本身,也就是说,若成立的话,对角线法具有更强的证明性。上述分析可见,对角线法存在根本谬误。

## 参考文献:

- [1] Shaffer C A. A practical introduction to data structures and algorithm analysis[M]. Java ed. [s. l.]: Pearson Education, 1998.
- [2] 张云泉,孙家昶,唐志敏,等. 数值计算程序的存储复杂性分析[J]. 计算机学报, 2000, 23(4): 362-373.
- [3] Hein J L. Discrete structures, logic, and computability[M]. Sudbury, MA: Jones and Bartlett, 1995.
- [4] Binder P. Theories of almost everything[J]. Nature, 2008, 455: 884-885.
- [5] 王 柏, 杨 娟. 形式语言与自动机[M]. 北京: 北京邮电大学出版社, 2003.
- [6] Lagarias J C. The  $3x+1$  problem and its generalization[J]. American Mathematical Monthly, 1985, 92: 3-23.
- [7] Mayberry J P. The foundations of mathematics in the theory of sets[M]. Cambridge: Cambridge University Press, 2000.
- [8] Zenkin A. Logic of actual infinity and G. cantor's diagonal proof of the uncountability of the continuum[J]. The Review of Modern Logic, 2004, 9(3-4): 27-82.
- [9] Frege G. The foundations of arithmetic[M]. 2nd ed. Xi'an: Northwestern University Press, 1980.
- [10] Kline M. Mathematics; the loss of certainty[M]. Oxford: Barnes & Noble Rediscoverers, 1982.
- [11] 康斯坦丝·瑞德. 希尔伯特[M]. 上海: 上海科技出版社, 2006.
- [12] Wittgenstein L. Remarks on the foundations of mathematics[M]. 3rd ed. [s. l.]: MIT Press, 2001.
- [13] 徐宝文, 张 挺, 陈振强. 递归子程序的依赖性分析及其应用[J]. 计算机学报, 2001, 24(11): 1178-1184.