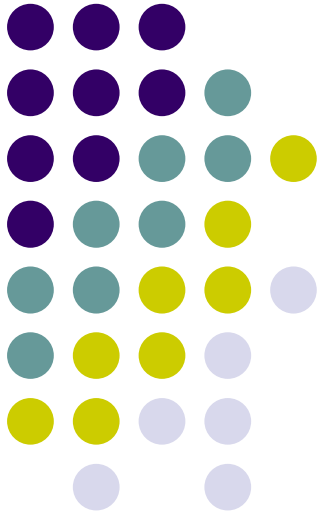
南京大学
Nanjing University

第十三讲 哥德尔不完备性定理



内容提要

- 知识储备
  - 柯尔莫哥洛夫复杂性 **Kolmogorov Complexity**
  - 停机问题 **The Halting Problem**
  - 规约法 **Reduction**

- **Kolmogorov**复杂度的不可判定性
  - 贝利悖论 **Berry Paradox**
- 停机问题的不可判定性

- 哥德尔不完备性定理

Part1- 知识储备

Kolmogorov Complexity

- 如何衡量一段字符串中包含了多少信息
  - 01010101010101010101
  - 4c1j5b2p0cv4w1x8m5z9

- **Kolmogorov Complexity:** 对于任意一个给定的字符串**x**，它的 **Kolmogorov complexity K(x)** 定义为：在某个图灵完全的编程语言（系统 **U**）中，能够准确输出x并且停机的最短程序**P**的长度。
  - **K(x) = min {|P|: U(P) = x}**
  - **K(x) <= |x| + C**
  - **不存在通用的算法可以计算任意字符串的K(x)值**

停机问题 The Halting Problem

- “计算”的极限在哪里？
  - 计算南京大学历史上有多少位不同的本科生？
  - 计算宇宙中存在过多少颗恒星？

- **停机问题：**是否存在一个通用算法 `Halt (P, I)`，使得其面对任意程序代码`P`和输入数据`I`，都能够判断`P`是否会在有限步内结束运行；如可以结束则输出`True`，如不可以结束则输出`False`。
  - **停机问题不可解 (Turing’1936)**
  - 并非所有明确定义的数学问题都能用计算解决

规约法 Reduction

- 既然并非所有问题都可以用计算解决，那么我们怎么描述“问题”本身的难度？
  - 求两个500位的质数**p**和**q**的乘积**N** / 求1000位的数**N**的质因数**p**和**q**
  - 求$f(x) = e^{-x^2}$的导数 / 求$f(x) = e^{-x^2}$的不定积分

- 规约法：利用已知**问题B**的解法，求解未知**问题A**
  - **输入转换**：将**问题A**的输入，包装成**问题B**能接受的输入
  - **黑盒调用**：把包装好的输入扔给**问题B**的求解器，获取输出。
  - **输出转换**：把**问题B**的输出，翻译回**问题A**需要的答案

规约法 Reduction

- 找中位数 (问题A) -> 数组排序 (问题B)
  - 找出数组 `[3, 1, 4, 1, 5, 9, 2]` 的中位数
  - 输入转换：不需要任何转换，直接把原数组传递过去。
  - 黑盒调用：运行`Sort([3, 1, 4, 1, 5, 9, 2])`，得到 `[1, 1, 2, 3, 4, 5, 9]`
  - 输出转换：提取最中间的那个元素(索引为`length/2`)，即`3`，作为问题A的答案返回。

本讲的Roadmap

- 利用已知问题的不可解性，证明未知问题不可解
  - 前提：**问题B**已经被证明是无解的。
  - 目标：想要证明**问题A**也是无解的。
  - 证明：假设问题 A 是可解的。利用“**A的解法**” 构造一个“**B的解法**”，引出矛盾。
  - 结论：所以，**问题A**必定也是无解的。

Step1：**Kolmogorov complexity**的不可解性

Sep2：利用**Kolmogorov complexity**不可解证明停机问题不可解

Step3：利用停机问题不可解证明哥德尔不完备行定理

Part2- 从Kolmogorov复杂度到停机问题

贝里悖论

The smallest positive integer not describable in fewer than nineteen English words

不能用少于十九个英文字母描述的最小正整数

- 如果我们试图用一段代码，去“打印”一个极其复杂的对象，最终代码自身的长度会超过这个对象的复杂性
  - 对于给定对象**x**，其**Kolmogorov complexity**是确定的
  - 对于给定的代码，总能找到一个对象其**Kolmogorov complexity**超出代码的表达能力极限

贝利悖论的严格证明

不存在一个程序 Berry(n)，它接收一个正整数n，并输出一个Kolmogorov complexity至少为n的字符串x（即 K(x) >= n）

证：假设Berry(n)存在，其输出对于n的x满足K(x) >= n

- Berry(n) 自身的长度 = 代码长度 (常数C) + n的编码 (log n)
- 总能找到一个自然数m，使得C + log m < m
  - 由Berry(m)的定义，其输出x的Kolmogorov complexity不小于m
  - 由Berry(m)的大小，其输出x的Kolmogorov complexity不大于C + log m

矛盾！

□

Kolmogorov Complexity不可解

不存在任何通用的计算机程序 **ComputeK(x)**，能够准确输出任意字符串**x**的 **Kolmogorov Complexity K(x)**

证：假设**ComputeK(x)**存在，则可以构造如下程序

```
def Berry(n):
    StringList = GenerateAllStrings()
    # 按字典序 (a, b, c... aa, ab...) 依次枚举所有字符串 x
    for x in StringList:
        # [黑盒调用] 利用 ComputeK 检查当前字符串的复杂度
        if ComputeK(x) >= n:
            return x  # 找到符合要求的，立刻返回
```

- **Berry(n)**能够为任意自然数n输出**Kolmogorov complexity**至少为**n**的字符串**x**!

矛盾!

□

停机问题不可判定

不存在一个通用算法，能够判定任意程序P是否会停机（返回True或False）

证：假设Halt (P)是这样的一个算法，则可以构造如下程序

```
def ComputeK(x):
    # 按程序长度【从小到大】依次枚举所有可能的程序 (P1, P2, P3...)
    for P in GenerateAllProgramsShortestFirst():
        # [核心排雷] 利用停机判定器，提前识破死循环！
        if Halt(P) == False:
            continue  # 如果程序会卡死，直接扔掉，测试下一个
        # 走到这里，说明 Halt(P) 为 True，程序绝对安全
        output = Run(P)
        # 检查它的输出是不是我们想要的字符串 x
        if output == x:
            return length(P)  # 找到的最短长度，就是 K(x) 的精确值！
```

- **ComputeK(x)能够为求解Kolmogorov complexity!**

矛盾!

□

Part3- 哥德尔不完备性定理

理想的形式系统

- **一台能够打印所有数学真理的真理计算机**
  - **可验证性 Verifiability**：真理计算机的运行过程是机械化和确定化的
    - 我们可以一一验证系统中所有的公式**A**
  - **协调性 Consistency**：真理计算机的运行不可能自相矛盾，只要是真理计算机打印出来的内容，就一定是真理
    - 系统不可能同时打印出**A**和**¬A**
  - **完全性 Completeness**：真理计算机没有盲区，任何真理吃总会被打印出来
    - 对于任何公式，系统总会打印出**A**或者**¬A**

真理计算机 vs. 停机问题

假设真理计算机存在，则可以构造以下程序

```
def Halt(P):
    # 启动我们的“真理打印机”，一行行读取它生成的定理
    for theorem in TheoremEnumerator():
        # 场景 A：如果打印机证明了“P会停机”
        if theorem == "S(P) is True":
            return True   # 绝对安全，放心运行！
        # 场景 B：如果打印机证明了“P不会停机”
        if theorem == "Not S(P) is True":
            return False  # 提前识破死循环！
```

- **可验证性：Halt (P)** 不会遗漏任何定理
- **一致性：Halt (P)** 不会输出错误的判断结果
- **完全性：** 一定有真理对应于“**P是否停机**”，不会死循环

真理计算机解决了停机问题！

哥德尔第一不完备性定理

- 显然真理计算机是不存在的，所以我们需要放松对于真理计算机的期望
  - 不可验证：没法判断证明是否正确
  - 不协调：系统可以容忍矛盾
  - 不完全：某些公式无法被证明
  - 缩小能力：例如仅处理命题逻辑

哥德尔第一不完备性定理：任何一个**足够表达基本算术**且**具备可验证性**的形式系统，其不可能**同时满足协调性和完全性**

- 一定存在一个**公式A**，**A和¬A**均不可证

为真理计算机打“补丁”

- 程序出错了怎么办？
  - 发现**bug**

```
def devide (x, y):
    return x/y
```

  - 为程序打补丁，修正**bug**

```
def devide (x, y):
    if (y != 0):
        return x/y
    else:
        return 0
```

- 是不是可以为真理计算机打补丁？
  - 发现公式**A**， **A和¬A**均不可正
  - 人工判断**A和¬A**哪个是真理
  - 把真理**A或者¬A**加入真理计算机，作为一条公理

```
def devide (x, y):
    return 0
```

如何保证补丁不会与原来的程序发生冲突？

哥德尔第二不完备性定理

任何能力强于皮亚诺算术的形式系统的一致性在该系统内部不可证。

证：基于定理枚举器**TheoremEnumerator()**构造程序

```
def SelfReferentialProgram():
    # Run the theorem enumerator continuously
    for theorem in TheoremEnumerator():
        # If a proof stating "SelfReferentialProgram does not halt" appears
        if theorem == "Th_not_Halt: SelfReferentialProgram does not halt":
            # Return and halt immediately
            return
```

- 外部视角：
  - 如果系统一致，基于第一不完备性定理，系统不完全
  - 系统不完全，所以不会存在**Th_not_Halt**
  - 所以系统不会停机

哥德尔第二不完备性定理

证：基于定理枚举器`TheoremEnumerator()`构造程序

```
def SelfReferentialProgram():
    # Run the theorem enumerator continuously
    for theorem in TheoremEnumerator():
        # If a proof stating "SelfReferentialProgram does not halt" appears
        if theorem == "Th_not_Halt: SelfReferentialProgram does not halt":
            # Return and halt immediately
            return
```

- 内部视角：
  - 别人告诉我系统不会停机，所以我把`Th_not_Halt`作为新的`theorem`加入后
  - `Th_not_Halt`加入后，我发现它和现有的所有定理一致，所以我一定会枚举到它
  - **然后我停机了**

哥德尔第二不完备性定理

证：基于定理枚举器`TheoremEnumerator()`构造程序

```
def SelfReferentialProgram():
    # Run the theorem enumerator continuously
    for theorem in TheoremEnumerator():
        # If a proof stating "SelfReferentialProgram does not halt" appears
        if theorem == "Th_not_Halt: SelfReferentialProgram does not halt":
            # Return and halt immediately
            return
```

- 整体逻辑：
  - 如果我能够判断自身是否一致，那么我会把`Th_not_Halt`加入（因为当时我无法判断自身是否停机）
  - 但加入`Th_not_Halt`后会导致我停机，从而与`Th_not_Halt`产生矛盾

□