



软件分析

课程介绍

熊英飞
北京大学
2019

软件缺陷可能导致灾难性事故



2003年美加停电事故：由于软件故障，美国和加拿大发生大面积停电事故，造成至少11人丧生



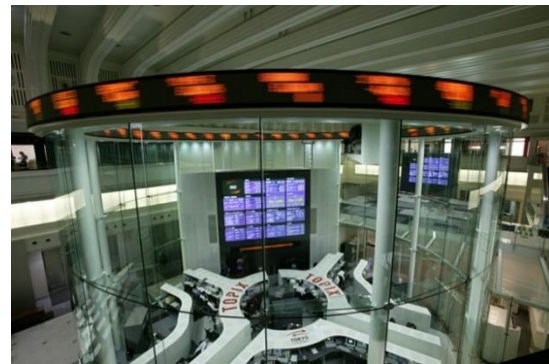
事故原因：电网管理软件内部实现存在重大缺陷，无法正确处理并行事件。

2006年巴西空难：由于防撞系统问题，巴西两架飞机相撞，造成154名人员丧生



事故原因：软件系统没有实现对防撞硬件系统故障的检测

2005年，东京证券交易所出现了人类历史上最长停机事故，造成的资金和信誉损失难以估算



事故原因：由于输入错误的升级指令，导致软件版本不匹配

能否彻底避免软件中出现缺陷？



- 问题：
 - 给定某程序 P
 - 给定某种类型的缺陷， 如内存泄露
- 输出：
 - 该程序中是否含有该类型的缺陷
- 是否存在算法能给出该判定问题的答案？
 - 软件测试
 - “Testing shows the presence, not the absence of bugs.” -- Edsger W. Dijkstra



库尔特·哥德尔(Kurt Gödel)

- 20世纪最伟大的数学家、逻辑学家之一
- 爱因斯坦语录
 - “我每天会去办公室，因为路上可以和哥德尔聊天”
- 主要成就
 - 哥德尔不完备定理



希尔伯特计划

Hilbert's Program



- 德国数学家大卫·希尔伯特在20世纪20年代提出
- 背景：第三次数学危机
 - 罗素悖论： $R = \{X \mid X \notin X\}$, $R \in R$?
- 目标：提出一个形式系统，可以覆盖现在所有的数学定理，并且具有如下特点：
 - 完备性：所有真命题都可以被证明
 - 一致性：不可能推出矛盾，即一个命题要么是真，要么是假，不会两者都是
 - 可判断性：存在一个算法来确定任意命题的真假

哥德尔不完备定理

Gödel's Incompleteness Theorem



- 1931年由哥德尔证明
- 蕴含皮亚诺算术公理的一致系统是不完备的
- 皮亚诺算术公理=自然数
 - 0是自然数
 - 每个自然数都有一个后继
 - 0不是任何自然数的后继
 - 如果 b, c 的后继都是 a , 则 $b=c$
 - 自然数仅包含0和其任意多次后继
- 对任意能表示自然数的系统, 一定有定理不能被证明

哥德尔不完备定理与内存泄露判定



- 主流程序语言的语法+语义=能表示自然数的形式系统
- 设有表达式T不能被证明
 - `a=malloc()`
 - `if (T) free(a);`
 - `return;`
- 若T为永真式，则没有内存泄露，否则就可能有



停机问题

- 据说哥德尔不完备性定理的证明和停机问题的证明非常类似
- 停机问题：判断一个程序在给定输入上是否会终止
 - 对应希尔伯特期望的第三个属性
- 图灵于**1936**年证明：不存在一个算法能回答停机问题
 - 因为当时还没有计算机，就顺便提出了图灵机



停机问题证明

- 假设存在停机问题判断算法: `bool Halt(p)`

- `p`为特定程序

- 给定某邪恶程序

```
void Evil() {  
    if (!Halt(Evil)) return;  
    else while(1);  
}
```

- `Halt(Evil)`的返回值是什么?

- 如果为真, 则`Evil`不停机, 矛盾
- 如果为假, 则`Evil`停机, 矛盾

是否存在确保无内存泄露的算法？



- 假设存在算法： `bool LeakFree(Program p)`

- 给定邪恶程序：

```
void Evil() {  
    int a = malloc();  
    if (LeakFree(Evil)) return;  
    else free(a);  
}
```

- `LeakFree(Evil)`产生矛盾：

- 如果为真，则有泄露
- 如果为假，则没有泄露



术语：可判定问题

- 判定问题（Decision Problem）：回答是/否的问题
- 可判定问题（Decidable Problem）是一个判定问题，该问题存在一个算法，使得对于该问题的每一个实例都能给出是/否的答案。
- 停机问题是不可判定问题
- 确定程序有无内存泄露是不可判定问题



练习

- 如下程序分析问题是否可判定？假设所有基本操作都在有限时间内执行完毕，给出证明。
 - 确定程序使用的变量是否多于50个
 - 给定程序，判断是否存在输入使得该程序抛出异常
 - 给定程序和输入，判断程序是否会抛出异常
 - 给定无循环和函数调用的程序和特定输入，判断程序是否会抛出异常
 - 给定无循环和函数调用的程序，判断程序是否在某些输入上会抛出异常
 - 给定程序和输入，判断程序是否会在前50步执行中抛出异常（执行一条语句为一步）



问题

- 到底有多少程序分析问题是不可判定的？



莱斯定理(Rice's Theorem)

- 我们可以把任意程序看成一个从输入到输出上的函数（输入输出对的集合），该函数描述了程序的行为
- 关于该函数/集合的任何非平凡属性，都不存在可以检查该属性的通用算法
 - 平凡属性：要么对全体程序都为真，要么对全体程序都为假
 - 非平凡属性：不是平凡的所有属性
 - 关于程序行为：即能定义在函数上的属性

运用莱斯定理快速确定可判定性



- 给定程序，判断是否存在输入使得该程序抛出异常
 - 可以定义： $\exists i, f(i) = EXCPT$
- 给定程序和输入，判断程序是否会抛出异常
 - 可以定义： $f(i) = EXCPT$
- 确定程序使用的变量是否多于50个
 - 涉及程序结构，不能定义
- 给定无循环和函数调用的程序，判断程序是否在某些输入上会抛出异常
 - 只涉及部分程序，不符合定理条件（注意：不符合莱斯定理定义不代表可判定）



莱斯定理的证明

- 反证法：给定函数上的非平凡性质 P 。
- 首先假设空集（对任何输入都不输出的程序）不满足 P 。
 - 因为 P 非平凡，所以一定存在程序使得 P 满足，记为`ok_prog`。
 - 假设检测该性质 P 的算法为`P_holds`。
- 我们可以编写如下函数来检测程序 q 是否停机

```
Bool halt(Program q) {
  void evil(Input n) {
    Output v = ok_prog(n);
    q();
    return v; }
  return P_holds(evil); }
```
- 如果空集满足 P ，将`ok_prog`换成一个让 P 不满足的程序，同样推出矛盾



刚刚说的都是真的吗？
世界真的这么没希望吗？



一个检查停机问题的算法

- 当前系统的状态为内存和寄存器中所有Bit的值
- 给定任意状态，系统的下一状态是确定的
- 令系统的所有可能的状态为节点，状态A可达状态B就添加一条A到B的边，那么形成一个有向图（有限状态自动机）
- 如果从任意初始状态出发的路径都无环，那么系统一定停机，否则可能会死机
 - 给定起始状态，遍历执行路径，同时记录所有访问过的状态。
 - 如果有达到一个之前访问过的状态，则有环。如果达到终态，则无环。
- 因为状态数量有穷，所以该算法一定终止。

哥德尔、图灵、莱斯错了吗？



- 该检查算法的运行需要比被检查程序p更多的状态

```
void Evil() {  
    if (!Halt(Evil)) return;  
    else while(1);  
}
```

- Halt(Evil)无法运行，因为Halt(Evil)的运行需要比Evil()更多的状态空间，而Evil()的运行又需要比Halt(Evil)更多的状态空间
- 然而一般来说，不会有程序调用Halt
 - 对这类程序该算法可以工作



模型检查

- 基于有限状态自动机抽象判断程序属性的技术
- 被广泛应用于硬件领域
- 在软件领域因为状态爆炸问题（即状态数太多），几乎无法被应用到大型程序上

所以，世界还是没有希望了吗？



- 近似法拯救世界
- 近似法：允许在得不到精确值的时候，给出不精确的答案
- 对于判断问题，不精确的答案就是
 - 不知道

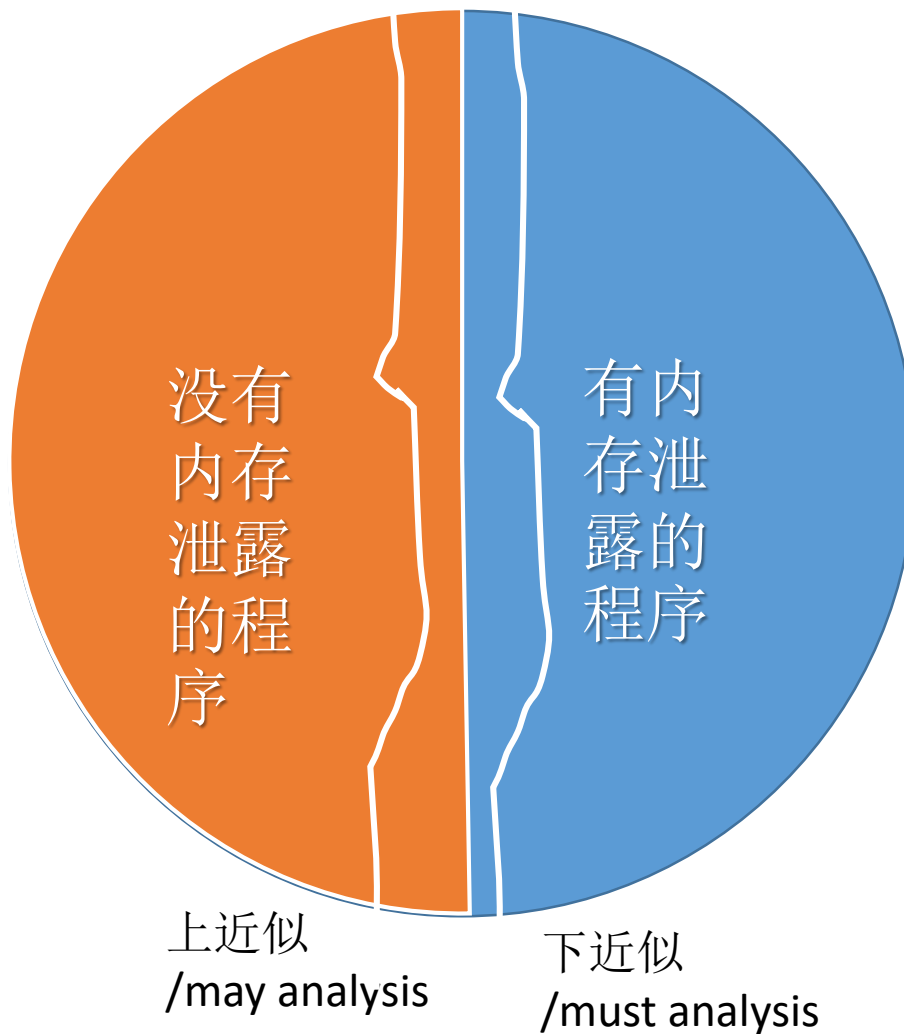


近似求解判定问题

- 原始判定问题：输出“是”或者“否”
- 近似求解判定问题：输出“是”、“否”或者“不知道”
- 两个变体
 - 只输出“是”或者“不知道”
 - must analysis, lower/under approximation (下近似)
 - 只输出“否”或者“不知道”
 - may analysis, upper/over approximation (上近似)
- 目标：尽可能多的回答“是”、“否”，尽可能少的回答“不知道”



近似法判断内存泄露





非判定问题

- 近似方法、**must**分析和**may**分析的定义取决于问题性质
- 例：假设正确答案是一个集合S
 - **must**分析：返回的集合总是S的子集
 - **may**分析：返回的集合总是S的超集
 - 或者更全面的分析：返回不相交(Disjoint)集合**MUST**,**MAY**,**NEVER**，其中
 - $MUST \subseteq S$,
 - $NEVER \cap S = \emptyset$,
 - $S \subseteq MUST \cup MAY$
- **must**和**may**的区分并不严格，可以互相转换
 - 将判定问题取反
 - 对于返回集合的问题，将返回值定义为原集合的补集



练习

- 测试属于must分析还是may分析？
- 类型检查属于must分析还是may分析？



答案

- 例：利用测试和类型检查回答是否存在输入让程序抛出异常的问题
- 测试：给出若干关键输入，看在这些输入上是否会抛出异常
 - 如果抛出异常，回答“是”
 - 如果没有抛出以后，回答“不知道”
 - must分析
- 类型检查：采用类似Java的函数签名，检查当前函数中所有语句可能抛出的异常都被捕获，并且main函数不允许抛出异常
 - 如果通过类型检查，回答“否”
 - 如果没有通过，回答“不知道”
 - may分析



另一个术语：健壮性

Soundness

- 程序分析技术最早源自编译器优化
- 在编译器优化中，我们需要保证决定不改变程序的语义
- 健壮性：分析结果保证不会改变程序语义
- 健壮性的定义和具体应用场景有关，但往往对应于 `must` 分析和 `may` 分析中的一个
- 健壮性有时也被成为安全性(Safety)、正确性(correctness)
- 健壮性的反面有时也被称为完整性(completeness)
 - 如果健壮性对应 `must-analysis`，则完整性对应 `may-analysis`



求近似解基本方法1—抽象

- 给定表达式语言

term := term + term

| term - term

| term * term

| term / term

| integer

| variable

- 比如: $a+b*c$
- 如果输入都为正数, 结果也一定是正数吗?



抽象域

- 正 = {所有的正数}
- 零 = {0}
- 负 = {所有的负数}

• 乘法运算规则:

- 正 * 正 = 正
- 正 * 零 = 零
- 正 * 负 = 负

- 负 * 正 = 负
- 负 * 零 = 零
- 负 * 负 = 正

- 零 * 正 = 零
- 零 * 零 = 零
- 零 * 负 = 零



问题

- 正+负=?
- 解决方案：增加抽象符号表示“不知道”
 - 正={所有的正数}
 - 零={0}
 - 负={所有的负数}
 - 躲={所有的整数和NaN}



运算举例

+	正	负	零	罊
正	正			
负	罊	负		
零	正	负	零	
罊	罊	罊	罊	罊

/	正	负	零	罊
正	正	负	零	罊
负	负	正	零	罊
零	罊	罊	罊	罊
罊	罊	罊	罊	罊



求近似解基本方法2—搜索

- $a+b*c$
- 以某种顺序遍历所有的 a,b,c
- 检查 $a > 0 \wedge b > 0 \wedge c > 0 \rightarrow a + b * c > 0$
- 如果不成立，得知原结论不成立
- 如果遍历完所有的 a,b,c 值，则原结论成立
- 如果超时，则答案为“不知道”

- 优化：在搜索过程中
 - 用启发式方法定义搜索顺序
 - 根据运算符的性质剪枝



本课程 《软件分析技术》

- 给定软件系统，回答关于系统性质的问题的技术，称为软件分析技术
 - 该软件的运行是否会停机？
 - 该软件中是否有内存泄露？
 - 该软件运行到第10行时，指针x会指向哪些内存位置？
 - 该软件中的API调用是否符合规范？

课程内容1：基于抽象解释的程序分析



- 数据流分析
 - 如何对分支、循环等控制结构进行抽象
- 过程间分析
 - 如果对函数调用关系进行抽象
- 指针分析
 - 如何对堆上的指向结构进行抽象
- 抽象解释
 - 对于抽象的通用理论
- 抽象解释的自动化
- 对应基本方法1——抽象

课程内容2：基于约束求解的程序分析



- SAT
 - 基础可满足性问题
- SMT
 - 通用可满足性问题
- 霍尔逻辑
 - 表达程序规约的方法和相应推导方法
- 符号执行
 - 基于约束求解的路径敏感分析
- 对应基本方法2——搜索



课程内容3： 软件分析应用

- 程序综合——如何让电脑自动编写程序
- 缺陷定位——确定程序有Bug后，如何知道Bug在哪里
- 缺陷修复——找到Bug后，如何让电脑自动修复程序中的Bug



软件分析发展历史

- 软件分析最早随着编译技术一起发展起来
 - 早期的发展主要是基于抽象的技术，强调安全性
- 2000年以后，几个主要变化
 - 编译器越来越成熟，软件分析技术更多用于软件工程工具而非编译器
 - 随着互联网的发展，可用的数据空前增多
 - 计算机速度有了大幅提高，使得很多以前不能做的智能分析成为可能
 - SAT求解算法获得突飞猛进
- 形成了几大新趋势
 - 健壮性不再作为第一位强调（Soundness宣言）
 - 基于约束求解的方法大量增多
 - 智能化软件分析蓬勃发展，形成若干子领域
 - 软件仓库挖掘、软件解析学、基于搜索的软件工程
 - 然而这部分总的来说还缺乏系统的理论，不在课上介绍



为什么要开设《软件分析》

- 重要性
 - 几乎所有的编译优化都离不开软件分析
 - 几乎所有的开发辅助工具都离不开软件分析
 - 更好的理解计算和抽象的本质与方法
- 学习难度
 - 历史长
 - 方法学派多
 - 缺乏易懂的教材
 - 传统上采用大量数学符号



为什么要学习《软件分析》

- IT企业对软件分析人才求贤若渴
- 从事软件相关研究的必要条件

总裁办电子邮件

电邮通知【2019】068号 签发人：任正非

关于对部分2019届顶尖学生实行年薪制管理的通知

华为公司要打赢未来的技术与商业战争，技术创新与商业创新双轮驱动是核心动力，创新就必须要有世界顶尖的人才，有顶尖人才充分发挥才智的组织土壤。我们首先要用顶级的挑战和顶级的薪酬去吸引顶尖人才，今年我们先将从全世界招进20-30名天才“少年”，今后逐年增加，以调整我们队伍的作战能力结构。

经公司研究决定，对八位2019届顶尖学生实行年薪制，年薪制方案如下：

- 1、钟利，博士。
年薪制方案：182-201万人民币/年
- 2、梁通，博士。
年薪制方案：182-201万人民币/年
- 3、李屹，博士。
年薪制方案：140.5-156.5万人民币/年
- 4、管高扬，博士。
年薪制方案：140.5-156.5万人民币/年
- 5、贾许亚，博士。
年薪制方案：89.6-100.8万人民币/年
- 6、王承河，博士。
年薪制方案：89.6-100.8万人民币/年
- 7、林皓，博士。
年薪制方案：89.6-100.8万人民币/年
- 8、何睿，博士。
年薪制方案：89.6-100.8万人民币/年

报送：董事会成员、监事会成员

主送：全体员工。





教学团队

- 教师：熊英飞
 - 2009年于日本东京大学获得博士学位
 - 2009-2011年在加拿大滑铁卢大学从事博士后研究
 - 2012年加入北京大学，任“百人计划”研究员
 - 办公室：1431
 - 邮件：xiongyf@pku.edu.cn
- 助教：朱琪豪
 - 硕士一年级
 - 办公室：1537
 - 邮件：zhuqh@pku.edu.cn



预备知识

- 编译器前端知识
- 熟悉常见的数学符号



课程主页

- <http://sei.pku.edu.cn/~xiongyf04/SA/main.htm>



参考书

- 课程课件
- 《编译原理》 Aho等
- 《Lecture notes on static analysis》 Moller and Schwartzbach
 - <https://cs.au.dk/~amoeller/spa/>
- 《Principle of Program Analysis》 Nielson等
- 《Decision Procedures -- An Algorithmic Point of View》 Daniel Kroening and Ofer Strichman



考核与评分

- 课堂表现：20分
- 课程作业：20分
- 课程项目1：30分
- 课程项目2：30分



课程项目1

- 感谢唐浩同学帮忙制作开发包和评测平台！
- 实现一个Java上的指针分析系统
- 要求：
 - 无法在测试程序上正常运行的不合格
 - 如：超时（3分钟），崩溃
 - 在测试程序上能输出结果，但结果不健壮(unsound)，60分
 - 结果健壮，根据精度分数在70-100之间
 - 代码提交作为评分参考
- 组队完成：
 - 2-3名同学一个小组
 - 组内贡献不均等的，请在提交的时候说明
- 每组提交解决方案和一个测试程序



课程项目2

- 实现一个程序综合工具
 - 根据规约自动编写程序
 - 根据在限定时间内求解出的样例个数评分
- 每组提交解决方案和一个测试程序



教学安排

- 9月
 - 数据流分析、稀疏分析
- 10月
 - 过程间分析、指针分析、控制流分析、抽象解释
- 11月
 - 约束求解、符号执行、项目1
- 12月
 - 软件分析应用、项目2
- 每个项目约4周时间