软件分析

课程介绍

熊英飞
北京大学
2017

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碰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
  - 保守型：任何抽象域导出来的具体结论可以直接在具体域中证明
  - 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
- 蕴含皮亚诺算术公里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

- 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
- 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部分函数（Partial Function），该函数描述了程序的行为
- 关于程序行为的任何非平凡属性，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：给定函数上的性质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。

- 编写如下函数来检测程序p是否停机

```
Bool halt(Program p) {
  void evil(Input n) {
     Output v = ok_prog(n);
     p();
     return v;
  }
  return P_holds(evil);
}
```

- 如果P_holds存在，则我们有了检测停机的算法

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

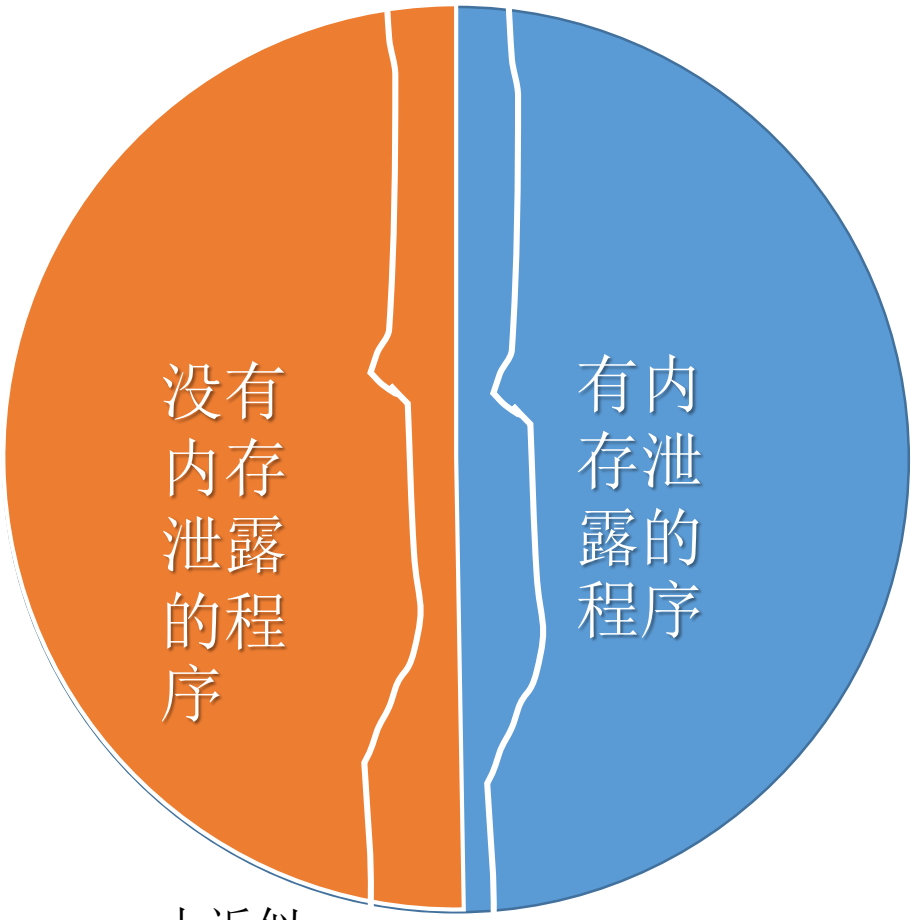
近似求解判定问题

- 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⊆S,
    - NEVER∩S=∅,
    - S⊆MUST∪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：安全性

- 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
- 安全性：根据分析结果所做的优化绝对不会改变程序语义
- 安全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
- 安全性有时也被成为健壮性(soundness)、正确性(correctness)
- 健壮性的反面有时也被成为完整性(completeness)
  - 如果健壮性对应must-analysis，则完整性对应may-analysis

求近似解基本原则——抽象

- 给定表达式语言

  term := term + term
  | term – term
  | term * term
  | term / term
  | integer

- 判断结果的符号是正是负

- 限制条件：分析在一台只有一个两位寄存器，没有内存的机器上进行

限制条件

- 无限制条件：直接求出表达式的值，然后查看符号位

- 有限制条件：
  - 无法分析出精确的答案
  - 将结果映射到一个可分析的抽象域

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}
  - 踝={所有的整数和NaN}

运算举例

+	正	负	零	鞡
正	正			
负	鞡	负		
零	正	负	零	
鞡	鞡	鞡	鞡	鞡

/	正	负	零	鞡
正	正	负	零	鞡
负	负	正	零	鞡
零	鞡	鞡	鞡	鞡
鞡	鞡	鞡	鞡	鞡

测试和类型检查中的抽象

- 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
  - 把程序输入抽象成“待测试”（有限集合）和“其他”（可能无限）两个集合，并只在“待测试”集合上检查
- 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
  - 把程序抽象成只包含throw语句、try/catch语句、函数声明和调用的抽象程序，然后在抽象程序上分析

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
- 约束求解和符号执行
  - 如何确定某条路径的可执行性

问题一：程序分析常常面临大量未知量

- File a = File.open(…)
- File.open是一个没有源代码的系统调用或网络服务
- 如何能知道File.open做了什么事？

- 其他常见未知量
  - 应用软件的具体需求
  - 具体软件设计上的约束
    - API方法的输入范围、调用顺序等
    - 某方法调用者对该方法的期望

问题二：安全性保障真的那么重要么？

- 实践证明，追求安全性往往会导致精度严重下降
  - 假设指向分析
  - 因为File.open没有源码，那么要保证安全，我们需要假设该方法可能任意修改任何内存位置
  - 于是任意指向关系都有可能，其他位置的分析精度变得无关紧要
- Soundiness宣言（http://soundiness.org/）
  - 不追求绝对的Soundness
  - 而是明确给出哪些地方可能Unsound，并衡量其影响

启发式规则Heuristic Rules

- 基于部分信息或者不完整分析做出的判断，通常不保证正确性或者精度
- 针对File.Open的启发式规则
  - 如果被调用函数名为java.io.File.Open，则假设该函数返回一个指向新内存位置的指针，并且不影响已有指向关系
- 该规则不能保证正确
  - 如果用户用了不同的JDK实现
  - 如果用户自己的函数命名为java.io.File.Open
  - 如果Java未来升级新版本引入了不同的行为

通过统计获得启发式规则

- 启发式规则通常不容易定义
  - File.Open只是JDK的一个函数，人工为JDK所有函数建模需要大量时间
  - 对于涉及具体应用程序的未知量无法提前定义
    - 应用程序需求、软件设计上的约束
- 采用统计的方法获得启发式规则
  - 例1：对于File.Open函数的大量执行进行统计，发现该函数始终不改变任何已有指向关系，于是认为该函数不改变指向关系
  - 例2：对于大量代码分析发现99.9%的情况下调用File.Open()之后都会调用File.Close()，于是可以将该要求记录为一条设计约束，剩下0.1%的情况就是潜在Bug

课程内容2：
数据驱动的软件分析

- 常见统计方法及其在软件分析上的应用
  - 有监督的机器学习——统计得到判定问题的启发式规则
    - K近邻法
    - 朴素贝叶斯
    - 决策树
    - 支持向量机
    - 神经网络
  - 频繁模式挖掘——挖掘出现次数足够多的模式集合
  - 聚类——利用某种距离度量将数据分类
- 代码上的统计方法
  - 树卷积神经网络
  - DEEPSYN/TGEN

调查：需要介绍常见统计方法吗？

- 有监督的机器学习——统计得到判定问题的启发式规则
  - K近邻法
  - 朴素贝叶斯
  - 决策树
  - 支持向量机
  - 神经网络
- 频繁模式挖掘——挖掘出现次数足够多的模式集合
- 聚类——利用某种距离度量将数据分类

- 去年的两条意见
  - “后半部分内容感觉理解起来很困难。”
  - “前半学期讲的可以再详细一些，后面几节课可以删除一些。”

优化问题

- 优化问题：给定某种度量，返回解空间中尽可能优的一组解
- 应用了近似法之后，软件分析问题可以看做优化问题
  - 在满足安全性的前提下，给出近似程度最高的一组解
- 同时很多软件分析问题天然是优化问题
  - 例1：找到令当前程序误差最大的一组输入？
  - 例2：对当前代码片段是否存在语义等价执行效率更高的书写方法？
- 因为软件本身的复杂性和规模，线性规划/动态规划等寻找最优方案的算法常常不工作

课程内容3：
面向优化问题的软件分析

- 元启发式搜索算法
  - 爬山法
  - 模拟退火
  - 粒子群算法
  - 遗传算法
- 面向优化问题的软件分析案例
  - 如何将问题转为搜索问题
  - 如何利用元启发式搜索来优化

课程内容4：
软件分析的典型应用

- 缺陷定位——确定程序有Bug后，如何知道Bug在哪里
- 程序综合——如何让电脑自动编写程序
- 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
  - 安全性不再作为第一位强调（Soundiness宣言）
  - 数据驱动的软件分析大发展（软件仓库挖掘、软件解析学）
  - 基于启发式搜索的软件分析变得更流行（基于搜索的软件工程）

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（主要指内容1）
  - 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》

- 大公司核心部门的就业机会
  - 微软、IBM、谷歌、Oracle、Facebook的开发工具部门
  - 大公司的内部工具部门
    - “谷歌最强的人都在开发内部工具。”—某网友
  - 企业研究院
- 中国企业
  - 中国企业已经发展到了需要自己的开发工具的阶段，但仍缺乏合适的人才
  - “目前的白盒工具的市场上，基本都是国外的产品。”
    --HP某售前工程师，2015
  - 华为、阿里、360等企业的软件分析团队每年找我定向推荐
- 科学研究

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：梁晶晶
  - 计算机软件与理论方向博士二年级
  - 办公室：1434
  - 邮件：ljjaxeabc@126.com

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
- 《机器学习》周志华

考核与评分（暂定）

- 课堂表现：20分
- 课程作业：30分
- 课程项目：50分

- 课程项目：利用所学知识完成一个程序分析工具，具有一定的实用价值