软件分析

课程介绍

熊英飞
（代表全体课程团队）
北京大学
2014

Microsoft Research
微软亚洲研究院

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

Microsoft Research 微软亚洲研究院

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, i)
  - p为特定程序，i为输入
- 给定某邪恶程序

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

- Halt(Evil, 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(p) {
    int a = malloc();
    if (LeakFree(p)) 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？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？

Microsoft Research
微软亚洲研究院

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：$\exists i, f(i) = EXP$
- 给定程序和输入，判断程序是否会抛出异常
  - 可以定义：$f(i) = EXP$
- 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是否在输入i上停机

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

- 如果P_holds存在，则我们有了检测停机的算法

世界果真没有希望了吗？

- 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
  - 只输出“否”或者“不知道”
    - may analysis, upper/over approximation

- 目标：尽可能多的回答“是”、“否”，尽可能少的回答“不知道”

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}
  - 粿={所有的整数和NaN}

运算举例

+	正	负	零	窳
正	正			
负	窳	负		
零	正	负	零	
窳	窳	窳	窳	窳

/	正	负	零	窳
正	正	负	零	窳
负	负	正	零	窳
零	窳	窳	窳	窳
窳	窳	窳	窳	窳

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

程序分析

- 对程序代码进行分析，获得关于程序行为的信息
- 程序分析的基本原则就是做抽象，将无法分析的内容抽象为可分析的内容

上半学期：程序分析

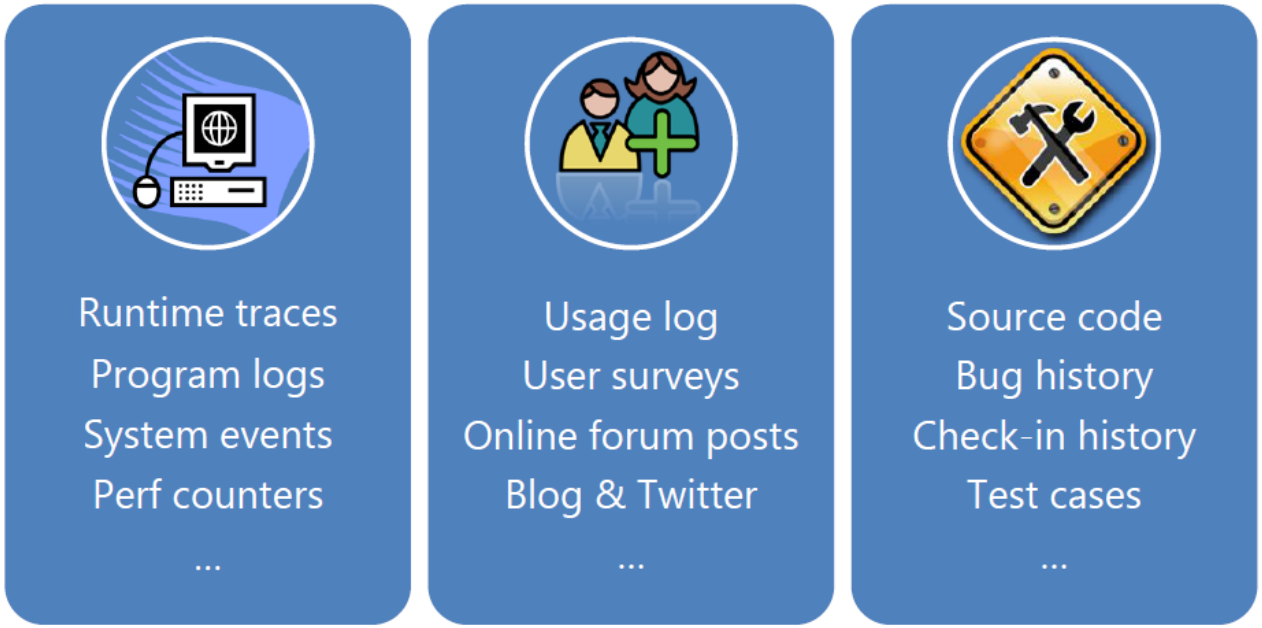
- 数据流分析
  - 将程序抽象成一张控制流图，忽略掉跳转条件等信息
- 过程间分析
  - 将函数调用关系抽象成一张调用图，忽略掉调用堆栈等信息
- 路径敏感分析
  - 将程序抽象成一条条的执行路径，并只考虑有限的路径集合
- 模型检查（待定）
  - 将程序抽象成有限状态自动机
- 抽象解释（待定）
  - 关于抽象的理论

问题一

- 程序分析技术可以给出关于程序行为的近似论断
- 问题一：如何知道什么论断需要给出？
  - 我们现在考虑的都是通用问题
    - 如：无内存泄露、无未捕获的异常
  - 每个程序有自己特定的论断
    - 如：排序程序要保证输出从大到小排列
  - 程序应该满足的论断称为程序的规约（specification）
  - 如何知道每个程序特定的规约？

解决方案：

- 软件开发中产生大量制品



- 通过数据分析，可以从各种制品中获得规约信息

数据分析举例

- 通过分析代码注释，获取对应函数的规约
- 通过分析代码库帮助文件，获取代码库实现的规约
- 对于程序优化任务，没有优化的程序就是规约
- 需求和设计文档里面包含了程序的规约
- 实现相似功能的代码可能包含了规约
- ……

如何做数据分析

- 大量现有可用技术
  - 信息检索
    - 数据表示、特征匹配
  - 数据挖掘
    - 频繁模式挖掘
  - 机器学习
    - 聚类算法、分类器学习
  - 自然语言处理
    - 分词、句子结构分析、语义理解
  - 数据库
    - 海量数据处理

问题二

- 问题二：安全性保障真的那么重要么？

- 人是会犯错误的，人工完成的任何任务往往没有安全性保障
- 由人组成的社会虽不完美，却正常运转

- 绝对正确性不是必须的，很多时候我们追求“足够好”的结果

没有安全性的分析

- 数据分析技术通常不保证安全性，但在各领域广泛使用
  - 例：百度返回的结果既不保证覆盖所有相关的网页也不保证返回的网页一定相关
- 更多的一些智能技术
  - 启发式搜索
  - 模糊分析和概率分析

更多应用

- 除了衡量和提高软件质量，基于数据的智能化软件分析还有更多应用
  - 通过分析大量程序日志，优化系统和硬件
  - 通过分析开发人员的通信记录，优化管理方式
  - 通过分析相似软件的行为，发现恶意软件
  - 通过分析代码和缺陷报告的关系，自动分配开发人员修复
  - ……

历史

- 历史上，软件分析技术主要做眼于具有严格的安全性的分析
- 2000年以后，两个主要变化
  - 随着互联网的发展，可用的数据空前增多
  - SAT求解器速度有了大幅提高，使得很多以前不能做的智能分析成为可能
- 研究人员开始大量采用数据驱动的智能化软件技术
- 若干新学术领域的形成
  - 软件仓库挖掘
  - 基于搜索的软件工程
- 2009年，微软亚洲研究院张冬梅研究员提出“软件解析学”，来概括这一新的软件分析子学科
- 在北大师兄、UIUC谢涛副教授的推广下，“软件解析学”已经在软件工程领域广泛所知

软件解析学

- 通过收集和分析数据，为软件开发和使用过程中的各种任务提供可行的有价值的信息。

下半学期：软件解析学和相关智能分析技术

- 黑盒分析
  - 通过智能搜索技术获取软件的性质
- 软件解析学概念和基础技术
  - 软件解析学的各项数据收集、统计和挖掘基础知识
- 软件代码和缺陷解析技术
  - 如何利用数据自动查找代码的缺陷、处理缺陷报告
- 在线服务解析技术
  - 针对软件服务数据的解析技术
- 数据驱动的操作系统性能分析
  - 利用数据查找系统性能瓶颈代码
- 软件解析结果的可视化
  - 如何展示数据分析的结果

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
  - 没有好的教材
  - 传统上采用大量数学符号

感谢参加软工所读书会的同学们！

Microsoft Research
微软亚洲研究院

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，但没有合适的人才
  - “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
- 科学研究

任课团队

- 熊英飞　　北京大学“百人计划”研究员
- 张路　　北京大学教授、杰青
- 张冬梅　　微软亚洲研究院研究员
- 张海东　　微软亚洲研究院架构师
- 楼建光　　微软亚洲研究院研究员
- 韩石　　微软亚洲研究院研究员
- 张洪宇　　微软亚洲研究院研究员

助教

- 王杰
  - 之前在读书会上惨遭虐待的同学之一
  - 办公室：1434
  - 邮件：yjxxtd@yeah.net

预备知识

- 编译器前端知识
- 熟悉常见的数学符号

Microsoft Research
微软亚洲研究院

参考书

- 课程课件
- 《编译原理》
- 《Lecture notes on static analysis》
- 《Principle of Program Analysis》

考核与评分

- 课堂表现：10-20分
  - 课堂参与度、随堂小测验等
- 课程作业：30-40分
- 课程项目：40-50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