软件分析

课程介绍

熊英飞
北京大学

软件缺陷可能导致灾难性事故

2019年波音737Max坠机事件：埃塞俄比亚航空一架波音737 MAX 8型飞机在起飞阶段坠毁，机上人员全数遇难。

事故原因：飞机MCAS防失速自动系统软件存在缺陷

2016年特斯拉车祸：自动驾驶模式下的特斯拉汽车和卡车相撞，导致驾驶员当场丧生

事故原因：在强烈日光条件下，摄像头进入盲区，但软件系统并没有捕获这一情况

2011年亚马逊宕机事故：亚马逊云计算出现了超过2天的宕机事故，造成的资金和信誉损失难以估算

事故原因：软件配置错误导致部分结点请求激增，不断转发请求压垮网络

能否彻底避免软件中出现特定类型的缺陷?

- 缺陷检测问题:
  - 给定某程序 P
  - 给定某种类型的缺陷， 如没有内存泄露
- 输出:
  - 程序P是否存在给定类型的缺陷

- 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：对所有命题，该命题本身或其否定一定能被证明
  - 一致性：任意命题和其否定不能同时被证明
  - 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
- 包含自然数和基本算术运算（如四则运算）的一致系统一定不完备，即包含一个无法证明的定理
  - 完备性：对所有命题，该命题本身或其否定命题一定能被证明
  - 一致性：任意命题和其否定命题不能同时被证明

哥德尔不完备定理与内存泄漏判定

- 主流程序语言能表示自然数和基本运算
  - 注意数学上的自然数是无限的，不等价于Int
- 在现代数学中，任何自动证明算法必须以某种形式系统为基础
  - 对逻辑学不熟悉的同学可以理解为公理+推导规则
- 设在所使用的形式系统中有表达式T不能被证明
  - a=malloc();
  - if (T) free(a);
  - return;
- 若T为永真式，则没有内存泄漏，否则就可能有

哥德尔不完备定理的证明概要

1. 通过某种方式把命题都编码成自然数。
   - 如$\forall a. a \neq 0$编码成$2^1 3^2 5^3 7^2 11^4 13^5$
     - 假设$\forall a. \neq 0$分别对应$1, 2, 3, 4, 5$
2. 通过某种方式把证明的推导过程编码成自然数
3. 证明该编码方式的一种性质
   - 假设a编码成$N(a)$
   - “推导过程x得到结论y”可以写成定义在$N(x)$和$N(y)$上的一个采用基本算术运算的命题，记为prove$(N(x), N(y))$
4. 通过不动点定理证明存在邪恶命题e，满足
   - $N\big(\neg\exists x, prove(x, N(e))\big) = N(e)$
   - 将$N(e)$看做函数输入，等式左边看做函数输出，可以证明该函数有不动点
5. 如果e为假，那么e就可证，推出矛盾，所以e只能为真

停机问题

- 也可以从证明更简单的停机问题来理解软件分析的困难
- 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定算法： 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

注：原本的莱斯定理定义在递归可枚举语言上，这里为了解释方便采用了等价的说法，并默认程序由图灵机定义。

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

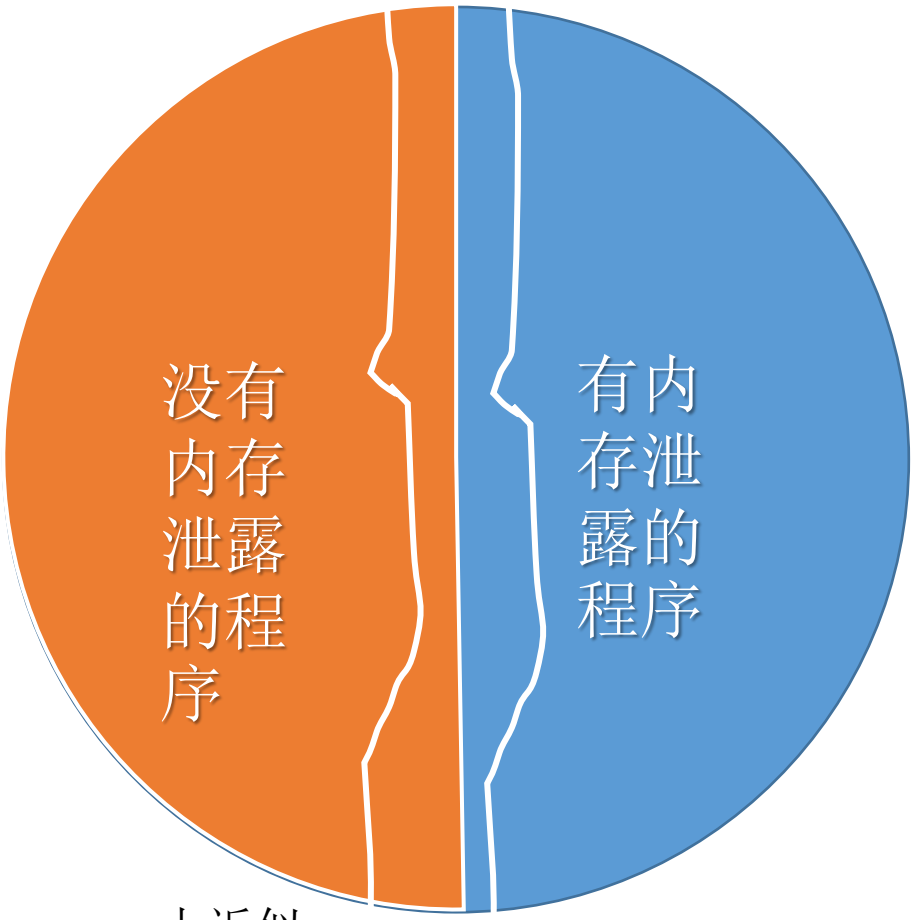
近似求解判定问题

- 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

- 通常可以转换成判定问题看待

- 例：假设正确答案是一个集合S，如判断某个变量执行到某个位置的可能取值
  - 对于每个值，回答判定问题：程序执行到这个位置是否可能出现这个值?
  - 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

- 假设问题为：程序是否有某种类型的Bug?
- 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

另一组术语：正确性和完整性

- 源于逻辑学
  - 正确性Soundness：一个逻辑系统得出结论，那么该结论是对的
  - 完整性Completeness：对于所有对的结论，该逻辑系统都能推出来
- 即
  - 正确性=Must分析
  - 完整性=May分析

另一组术语：正确性和完整性

- 程序分析技术最早源自编译器优化
- 在编译器优化中，正确性定义为编译器做出了一次优化，那么该优化肯定是对的
- 这时程序分析需要保证编译器不出错，则正确性根据需要可对应于must分析和may分析中的一个
- 从这个角度，正确性通常也称为安全性（Safety）

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
  | integer
  | variable

- 和输入的符号，求输出的符号
- 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*负=负
  - 正*零=零
  - 负*正=负
  - 负*负=正
  - 负*零=零
  - 零*正=零
  - 零*负=零
  - 零*零=零
- 对任意抽象输入包括的任意具体输入，其对应具体输出包括在抽象输出中

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}

运算举例

+	正	负	零	猓
正	正			
负	猓	负		
零	正	负	零	
猓	猓	猓	猓	猓

*	正	负	零	猓
正	正			
负	负	正		
零	零	零	零	
猓	猓	猓	零	猓

求近似解基本方法2—搜索

- 判断如下程序是否会有内存泄漏：
  - if (a == b && c == d && a != b) x = malloc();
  - return;
- 遍历a,b,c,d的所有取值，看是否有内存泄漏的情况
- 如果找到一组取值，即存在内存泄漏
- 如果遍历完所有取值，则不存在内存泄露
  - 如果输入是链表等无限长的数据结构，即基本不可能遍历完
- 如果超时，则答案为“不知道”

求近似解基本方法2—搜索

- 判断如下程序是否会有内存泄漏：
  - if (a == b && c == d && a != b) x = malloc();
  - return;
- 直接搜索效率较低，通常会引入各种剪枝和推断的方法
- 用三个布尔变量A, B, C分别表示a==b, c==d, a!=b
- 注意a==b和a!=b互斥，即A==!C
- 遍历所有A, B, C的取值，如果A==!C满足，判断A ∧ B ∧ C是否成立
- 枚举空间大小从$(2^{32})^4$变成8

抽象 vs 搜索

- 抽象通常考虑程序所有的执行，包括整个输入空间和任意长度的执行路径，但给出不精确的结果
- 搜索通常只考虑一部分执行，包括有限的输入空间和有限的执行路径长度，但对于这部分执行给出精确的分析
- 二者也可以结合

本课程《软件分析技术》

- 给定软件系统，回答关于系统行为的问题的技术，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?

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
- 符号执行
  - 基于约束求解对部分执行路径进行程序分析

- 对应基本方法2——搜索

课程内容3：软件分析应用

- 程序合成——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
  - = 缺陷定位+程序（补丁）合成
  - 根据时间决定是否介绍

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

2014-2016

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

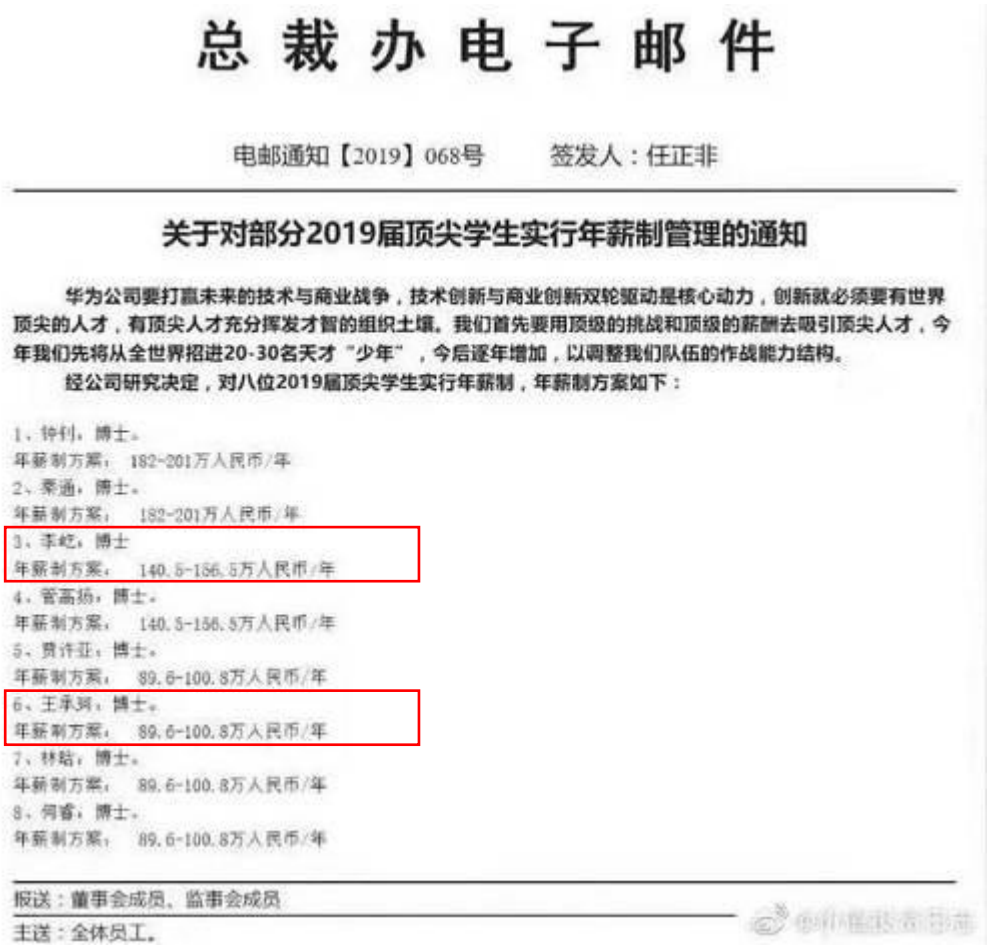
为什么要学习《软件分析》

- 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

2019

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，有顶尖人才充分挥发才智的组织土壤。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、钟钊，博士。
年薪制方案：　182-201万人民币/年
2、秦通，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、王承琰，博士。
年薪制方案：　89.6-100.8万人民币/年
7、林聪，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。

为什么要学习《软件分析》

- 国际形势变化下，IT企业对基础软件的重视达到了空前的高度，正在抢夺软件分析人才
  - 本组从2020届开始：
    - 博士毕业生大都拿到华为天才少年或更高薪水Offer
    - 本科毕业生拿到阿里历史上唯一一个给本科生的阿里星
    - 研究生毕业前通常被各大IT企业排队邀请参观
- 从事软件相关研究的必要条件

如何学习《软件分析》？

- 预备知识
  - 熟悉常见的数学符号
- 关于课程难度
  - 软件分析技术总体是难的
    - 我会尽量用容易的方式介绍
  - 不会为了降低难度删除困难的内容
    - 每年都有完全掌握的同学
    - 困难的内容可能也在某些时刻会用到
    - 本课程的子集在部分高校已经开设
  - 上课会指出哪些是必须掌握的内容
  - 课程项目的设计上不要求全部掌握课程内容
- 课程主页：https://xiongyingfei.github.io/SA/main.htm

参考资料-书和讲义

- 课程讲义——不断完善中，不过预计本年度仍然非常不完整
- 《编译原理》Aho等
  - 易懂，但涉及内容非常少
- 《Lecture notes on static analysis》Moller and Schwartzbach
  - https://cs.au.dk/~amoeller/spa/
  - 易懂，并且作者每年都更新版本，内容也越来越丰富
  - 只覆盖基于抽象解释的分析，且对该部分的高阶知识覆盖不够全面
- 《Program Analysis》Aldrich, Goues, and Padhye
  - https://cmu-program-analysis.github.io/
  - 一开始只有基于抽象解释的分析，后来加了不少内容，整体覆盖范围和本课程类似
  - 基于抽象解释的分析对比上一本，部分地方讲解更简洁清晰，但三名作者都不是静态分析研究人员，部分地方讲得不够专业或有错误
- 《Introduction to Static Analysis》Rival and Yi
  - 一开始就引入抽象解释，比上面两本更深入更系统
  - 一开始的例子就比较复杂，且有些脱离实际，不如上面两本易懂
  - 综合来说不错，推荐给想要深入学习基于抽象解释的分析的同学

参考资料-书和讲义

- 《Program Analysis: An Appetizer》Nielson等
  - https://arxiv.org/ftp/arxiv/papers/2012/2012.10086.pdf
  - 同一批作者著有《 Principles of Program Analysis 》，较为全面，但行文非常不易懂，不推荐初学者
  - 这本书能读懂，但易懂性不如上页前两本，专业性不如上页第三本
- 《Principles of Abstract Interpretation》Cousot等
  - 大师著作，在抽象解释上非常全面
  - 数学符号非常复杂，不易懂，不推荐初学者
- 《Decision Procedures -- An Algorithmic Point of View》Daniel Kroening and Ofer Strichman
  - 全面介绍约束求解算法的经典教材

参考资料—其他课程

- 南京大学《软件分析》课
  - https://tai-e.pascal-lab.net/lectures.html
  - 覆盖主要基于抽象解释的分析技术（但没有抽象解释）
  - 讲解比本课程详细，视频课件都有，本课程学习困难时推荐查阅
  - 本课程实验使用的Tai-e框架来源于南大团队
- 国防科技大学《程序分析》课
  - https://www.educoder.net/classrooms/7759/
  - 陈立前老师是Patrick Cousot共同指导的博士，国内知名抽象解释专家
  - 和上一门课的内容较为互补

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，历任助理教授、副教授
  - 办公室：理科一号楼1431
  - 邮件：xiongyf@pku.edu.cn
- 助教：方屹
  - 博士一年级
  - 邮件：chnfy@stu.pku.edu.cn
- 助教：夏廷轩
  - 本科四年级
  - 邮件：kkogoro@stu.pku.edu.cn

考核与评分

- 课程作业：30分
  - 根据作业的完成质量评分
- 课程项目1：35分
- 课程项目2：35分

- 【新增】课程项目设置指标，达到对应指标获得相应分数
  - 感谢教务部，终于不用内卷了
- 因为客观分无法完全反映同学们学习情况，保留主观调分的可能，一般针对做了很多努力但由于数据集偏差等原因分数特别低的同学，所有主观调分都将公示。

课程项目1

助教升级中，有变动可能，
以后续通知为准

- 实现一个Java上的指针分析系统
  - 分析程序不同位置指针可能指向的值
  - 根据在限定时间内达到的分析精度评分
- 每组提交解决方案和测试样例
- 组队完成（待定，可能变成个人作业），若组队：
  - 3名同学一队
  - 组内贡献不均等的，请在提交的时候说明

感谢唐浩同学帮忙制作开发包和本地评测平台！
感谢肖元安、朱琪豪、吴宜谦帮忙制作在线评测平台！

课程项目2

助教升级中，有变动可能，
以后续通知为准

- 实现一个程序合成工具（可能变成实现某种分析工具）
  - 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
- 组队（待定，可能变成个人作业）要求：
  - 3名同学一队，但队友必须和上一个项目不同

感谢唐浩同学帮忙制作开发包和本地评测平台！
感谢肖元安、朱琪豪、吴宜谦帮忙制作在线评测平台！

教学安排

- 9月
  - 数据流分析
- 10月
  - 过程间分析、指针分析、控制流分析、项目1
- 11月
  - 抽象解释、约束求解、符号执行、程序合成、项目2
- 12月
  - 符号抽象、程序合成、缺陷定位和修复
- 每个项目约4周时间

《软件分析》课历史

- 2012-2013，软件工程研究所读书会
- 2014-2015，和微软亚洲研究院联合开课
  - 54学时，我大概负责2/3，微软负责1/3
  - 微软亚洲研究院主要介绍软件解析学（统计软件分析）上的工作
- 2015，从研究生课改为本研合上课
- 2016之后，完全由我上课
- 2017起，课程的一部分开设为国科大课程《程序分析》（和张路老师共同教授）
- 2021年起，从本研合上课改为本科生课程
- 2022年选课人数超过50人，2023年选课人数超过110人

《软件分析》历年分数

年份	本科	研究生
2024	89.35	
2023	83.71	
2022	87.51	
2021	91.69	
2020	93.75	100
2019	96.51	98.83
2018	99	99
2017	98.53	97.41
2016	95	99.87

品质校园课程评测

难，但其他的都挺好

熊英飞老师 (24-25-1 学期), 2025/01/10

课程听感

作业/任务

关于考试

老师和助教人不错，但课非常硬核

熊英飞老师 (23-24-1 学期), 2024/01/23

课程听感:别看这课叫软件分析技术，实际上和我之前认识的软件分析相差甚大，更多地涉及程序设计语言与程序的分析、抽象解释，以及

累，但极力推荐

熊英飞老师 (22-23-2 学期), 2024/01/15

课程听感:老师讲得很好，收获很大，软件分析还是很有意思的，不签到

作业/任务量:平时每次课有书面作业（有大作业时无书面作业），有时候会比较难，但是能感觉作业设计是比较用心的，比如上课介绍了某个框架，作业

两次大作业，每次3周时间，3人组队，为防止抱大腿要求两次队友不同，每次占总评35%

大作业确实难+卷，大佬不少，但是把上课讲过的算法实现出来，再稍微加入一些优化，就能拿到不错的分数，不要求极高分的话工作量可以接受

队友靠谱很重要，不然可以体验一把单杀三人大作业，参考一下前人的代码也可以帮助找到方向

这课也很适合大四摆烂，两次大作业都是通过基本测试点就给60，不要求分数的话工作量下限低，有挂科风险的老师还会通知补交作业，调分时会保证

关于考试:无考试

关于给分:

每个部分的分数最后会调成均分85标准差12，可能被向下调分（调前优秀率80%超标太多），但是调完还是有总评均分84.4优秀率54%（总评均分不是

大作业不仅看评测结果，也会有手动调整，按照原规则分太低的和有想法但是因为种种原因评测表现不佳的都会加分，如果特判水过去测试点的会扣分

给分非常公开透明，最高分、最低分、平均分、优秀率、调分规则、调前原始分数的平均值和标准差、手动调整以及原因都会公布，非常好

这门课最好的一点是课程的各个地方都能看出老师和助教是为学生着想的，尤其是给分

在某些老师以"分数不重要"为由肆意给学生对未来有极大影响的分数时，这门课却在给分上下大功夫，优秀率在能通过教务的前提下尽可能高，调分规

保证公平性，都看出了这门课是真的考虑了学生的利益的。

本学期计划的改进

- 总体延续课程风格
  - 内容会比其他高校同类课程更深入
  - 要取得高分需要投入大量时间精力
  - 仍然提供摸鱼选项，但摸鱼和成绩不能兼得
- 课程项目去内卷改造
  - 感谢教务部，感谢北大
  - 获得同样分数的难度保持不变或稍微降低
- 课程内容升级
  - 精简前沿内容，特别在大模型时代重要性下降的内容
  - 追踪大模型时代软件分析发展，按需增加内容
  - 优化课程内容讲解方式

作业

- 作业在没有特别指出的情况下，均默认下周二上课前上传教学网
- 假设我们设计一个程序分析来检测程序是否会抛出异常。该分析遍历输入空间中每一个输入，然后执行100条语句，看程序是否抛出异常。对于有限输入空间的程序，该分析是精确分析、上近似、下近似还是不属于以上三类？为什么？
- 假设我们把符号分析的抽象域改成{自然数，负，踩}三个值，其中自然数表示所有正数和零，请写出加法和乘法的计算规则，并给出一个式子，在该抽象域上得到的结果不如{正，负，零，踩}精确。