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，该命题本身或其逆命题一定能被证明
  - 一致性：任意命题和其逆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
   - $N\left(\neg \exists x, prove(x, N(e))\right) = N(e)$
   - 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

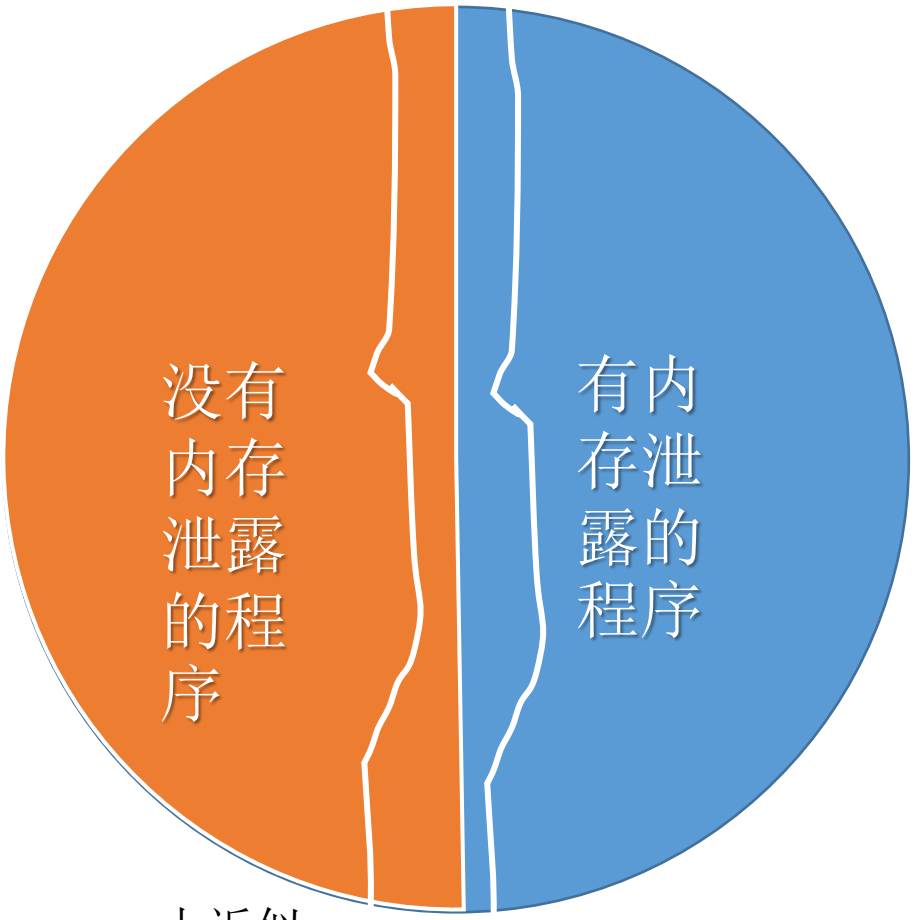
近似求解判定问题

- 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

- 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：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：分析结果对应的优化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

- 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*零=零
  - 正*负=负
  - 负*正=负
  - 负*零=零
  - 负*负=正
  - 零*正=零
  - 零*零=零
  - 零*负=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={所有的整数和NaN}

运算举例

+	正	负	零	粿
正	正			
负	粿	负		
零	正	负	零	
粿	粿	粿	粿	粿

/	正	负	零	粿
正	正	负	零	粿
负	负	正	零	粿
零	粿	粿	粿	粿
粿	粿	粿	粿	粿

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

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
- 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

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

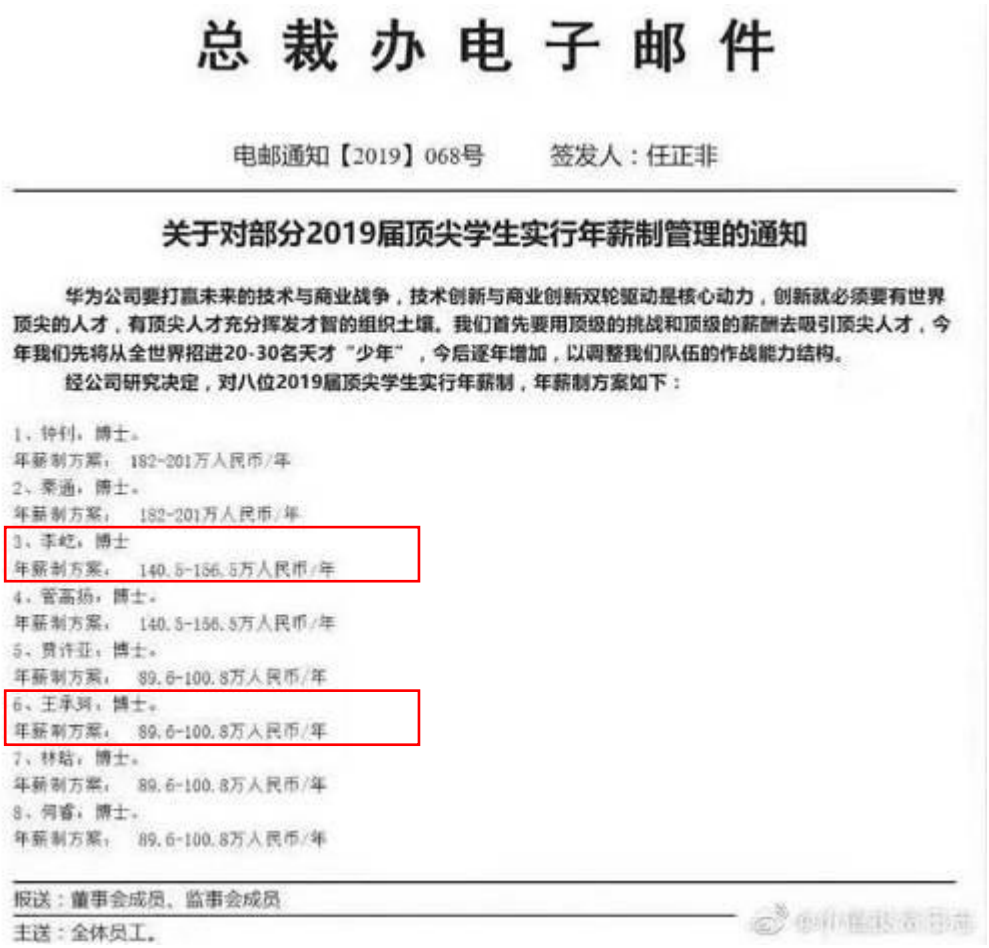
为什么要学习《软件分析》

- 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
    - 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

《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年选课人数突破50人，2023年选课人数突破110人

《软件分析》历年分数

年份	本科	研究生
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

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：肖元安
  - 博士二年级
  - 办公室：昌平文德楼
  - 邮件：xmcp@pku.edu.cn
- 助教：练琪灏
  - 博士一年级
  - 办公室：昌平文德楼
  - 邮件： mepy@stu.pku.edu.cn
- 助教：张钊
  - 博士一年级
  - 办公室：昌平文德楼
  - 邮件：zhangzhao2019@pku.edu.cn

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

参考书

- 课程课件、讲义
- 《编译原理》Aho等
- 《Lecture notes on static analysis》Moller and Schwartzbach
  - https://cs.au.dk/~amoeller/spa/
- 《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 》，但过于形式化，不推荐
- 《Principles of Abstract Interpretation》Cousot等
- 南京大学《软件分析》课
  - https://tai-e.pascal-lab.net/lectures.html
- 国防科技大学《程序分析》课
  - https://www.educoder.net/classrooms/7759/
- 《Decision Procedures -- An Algorithmic Point of View》 Daniel Kroening and Ofer Strichman

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

- 各部分分数会做标准差-平均分的正规化，确保标准差和平均分一致
- 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

- 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)，1分
  - 结果健壮，根据精度分数在1-2之间
  - 代码提交作为评分参考
  - 提交自己编写的测试样例，包括代码和标准输出
- 最终给分：
  - 公开的两个测试用例为60分，剩余测试用例为40分。
- 组队完成：
  - 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！
感谢朱琪豪帮忙制作在线评测平台！感谢吴宜谦的改进！

课程项目2

- 实现一个程序合成工具
  - 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
- 组队要求：
  - 3名同学一队，但队友必须和上一个项目不同

感谢曾沐焓、吉如一同学帮忙制作开发包和本地评测平台！
感谢朱琪豪帮忙制作在线评测平台！感谢吴宜谦的改进！

本学期计划的变化

- 拟撰写课程讲义，包括课程要点、关键概念、算法、证明等
- 拟扩大基础部分讲解，适当减少前沿部分内容
- 重组课程内容，以抽象解释和方程求解贯穿内容
- 拟增加作业量
- 取消平时成绩
- 课程项目推荐框架从Soot改为太阿

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

作业

- 停机问题的证明定义在没有输入的函数上，能否改成在带输入的函数上？注意这时Halt(p, i)函数接受两个参数，其中i是输入。
- 假设我们把符号分析的抽象域改成{自然数，负，槑}三个值，其中自然数表示所有正数和零，请写出加法和除法的计算规则，并给出一个式子，在该抽象域上得到的结果不如{正，负，零，槑}精确。