中国科学技术大学
University of Science and Technology of China

符号执行

《程序设计语言理论》

张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

静态分析

- 能分析程序中所有可能的运行

  - 有许多有趣的想法和工具
  - 很多只是在论文上展示有好的效果
  - 学术界推出的被企业认可的商用工具寥寥无几

    Dawson Engler: coverity [CACM2010]

但是，开发者使用起来....

- 不容易，论文中的结果描述的是静态分析专家所用的
- 有生命力的商用工具：要能处理误报(false positives)、开发者的困惑、错误管理、...

[CACM2010] A few billion lines of code later: using static analysis to find bugs in the real world , Communication of the ACM, 53(2):66-75, 2010.

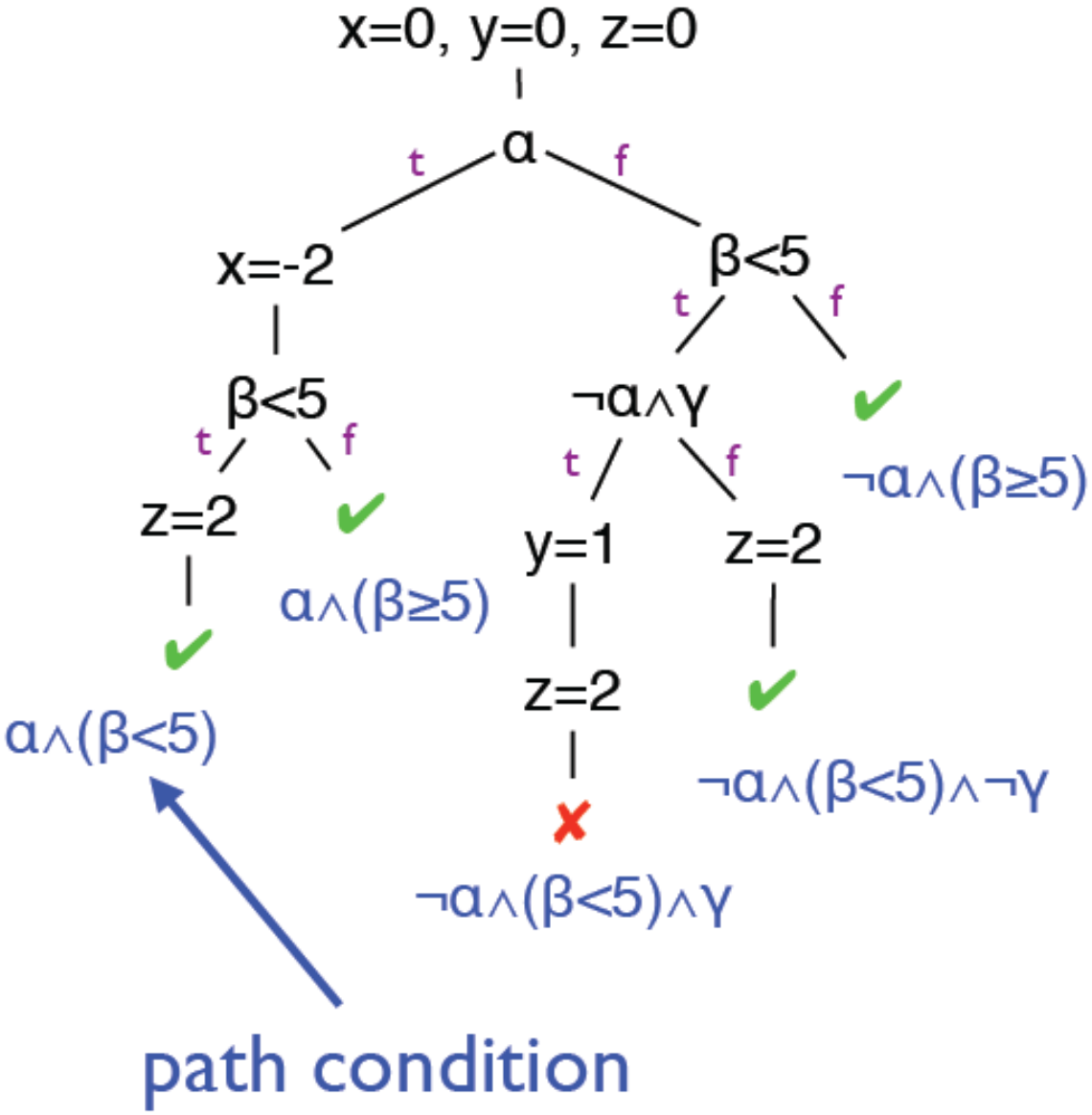
符号执行的引入

- 抽象的作用
  - 让静态分析对所有可能的运行进行建模，但引入保守性
  - *-敏感性方法试图改进之，但是远远不够
- 静态分析的抽象 ≠ 开发者的抽象
- 测试
  - 每个测试只能考察一种可能的执行
  - 希望测试用例更具有一般性，但是却没有保障
- 符号执行：对测试的泛化，引入符号值来计算

[计算机学报2015] 软件安全漏洞检测技术，38(4):717-732, 2015.

符号执行举例

```
1.  int a = α, b = β, c = γ;
2.  // symbolic
3.  int x = 0, y = 0, z = 0;
4.  if (a) {
5.      x = -2;
6.  }
7.  if (b < 5) {
8.      if (!a && c) { y = 1; }
9.      z = 2;
10. }
11. assert(x+y+z!=3)
```



符号执行

- 每个符号执行路径代表多个实际的程序运行
  - 实际运行的具体值满足符号执行的路径条件

  相比测试，符号执行可以覆盖更多的程序执行空间
- 符号执行的发展
  - 早期的工作：James C. King. Symbolic execution and program testing. CACM, 19(7):385–394, 1976. (高引)
  - 问题：1980's 当时的机器内存小且慢
  - 符号执行代价极高：大量可能的执行路径，需要求解器判断哪些路径是可行的，程序状态有许多位

符号执行现状

- 计算机：速度快、内存便宜
- 有非常强大的SMT/SAT求解器
  - **SMT**= Satisfiability Modulo Theories =**SAT++**
  - 快速解决非常多的问题：检查断言、削减可行路径
  - 求解器：Z3(已集成到LLVM中)、STP、Yices等
- 近10年来的有代表性论文
  - KLEE（OSDI2008, C. Cadar、D.Engler等,用于LLVM）
  - [CACM2013] Symbolic execution for software testing(30y+)
  - [CSUR2018] A Survey of Symbolic Execution Techniques

简单命令式语言IMP的符号执行

```
a ::= n | X | -a | a0+a1 | a0-a1 | a0×a1 | a0/a1
b ::= bv | ¬b | b0∧b1 | b0∨b1 | a0=a1 | a0<a1 | a0>a1
c ::= skip | input(s) | X:=a | if b then c else c
    | c0; c1 | while b do c | assert b
```

- n是整数、X是变量、bv是布尔值
- c是命令、a和b分别是整型和布尔型表达式
- Sym-while的OCaml实现：https://github.com/Isweet/sym-while
  - syswhile.ml 包含总控程序，即main，具体执行或符号执行
  - lexer.mll(词法描述)、parser.mly(语法描述)、ast.ml(AST)
  - concrete.ml: 解释执行，状态是变量到整数的映射，Imp.run s
  - symbol.ml, symbolic.ml: 符号执行，状态时变量到符号表达式和路径条件的映射

符号表达式

- 符号表达式可能包含变量 ast.ml

```
type arith =
  | AEVar of string
  | AENum of int
  | AENegate of arith
  | AEPlus of arith * arith
  | AEMinus of arith * arith
  | AEMult of arith * arith
  | AEDiv of arith * arith
```

```
type boolean =
  | BETrue
  | BEFalse
  | BENot of boolean
  | BEAnd of boolean * boolean
  | BEOr of boolean * boolean
  | BELT of arith * arith
  | BEGT of arith * arith
  | BEEq of arith * arith
```

符号状态

- **具体状态：** 变量到整数 concrete.ml

```
type conc_state = int StringMap.t
```

- **符号状态：** 变量到符号表达式和路径条件 symbolic.ml

```
type sym_state = (Symbol.int_t StringMap.t) * Symbol.t
```

  - **Symbol.t:**布尔型符号表达式类型
  - **Symbol.int_t:**整型符号表达式类型

基于fork的符号执行

- 如何判断哪个分支可行(feasible)？

```
let rec eval (s : stmt) (s_st : sym_state) : answer =
  let (env, pc) = s_st in
```

```
| SIf (b, s1, s2) ->
  let l = t_of_boolean b env in              (* branch cond *)
  let cond_true = LAnd (l, pc) in            (* … and path cond *)
  let cond_false = LAnd ((LNot l), pc) in
  let sat_true = check (z3_of_t cond_true) in
  let sat_false = check (z3_of_t cond_false) in
(match sat_true with                         (* might do both branches *)
  | Some _ -> …(eval s1 (env, cond_true))
  | None -> …);
  (match sat_false with
  | Some _ -> …(eval s2 (env, cond_false))
  | None -> …);
```

符号执行策略

- 顶层策略
  - 初始化状态：pc=0, 路径条件为true，状态为empty
  - 对每条语句符号求值
  - 一旦执行分叉，则对两个分支都分别求值（DepthFS）
  - 执行完后，返回多个符号状态
- 路径爆炸(path explosion)
  - 分支、循环
- 搜索策略
  - 基本算法：DFS（容易在某部分stuck）、BFS

符号执行策略

- 搜索策略
  - 优先权搜索：更可能有断言错误的、在给定时长运行
  - 将程序执行看成是DAG
    - 结点：程序状态；边：状态之间的迁移
    - 图搜索策略
  - 随机性(randomness)
    - 随机地选下一条路径(Random path)
    - 如果没有遇到感兴趣的，则随机地重新启动搜索
    - 当有多条相同优先权的路径时，随机地选一条

  缺点：可复现性 reproducibility

符号执行策略

- 覆盖引导的启发式方法
  - 主要思想：尽可能地访问以前未访问的语句
  - 方法：语句评分- 访问频次，选分值最低的语句
  - 可行之处：错误常位于程序中难以到达的地方
  - 不可行之处：前条件的设置会影响语句的可达性

  KLEE：随机路径+覆盖引导
- 分代搜索：BFS+覆盖引导
  - 第0代：随机选一条路径，运行到结束
  - 第1代：从0代获得路径，去除路径上的一个分支条件得到新的路径前缀，再对该前缀求解得到结果路径

SMT求解器的性能

- SAT求解器是SMT求解器的核心
  - SMT(可满足性模理论)：接受不同格式的等式系统
  - SAT：必须是合取CNF范式的布尔等式
  - 理论上，所有SMT查询可以约减到SAT查询
  - 实践上，SMT和高级优化是关键的

  - 简单的等式 x+0=x
  - 数组理论： read(x, write(42, x, A)) =42
    - 42是下标，A是数组，x是元素
  - 缓存：记住求解器的查询； 删除无用变量

符号执行重生

- 两个关键的系统
  - DART (Godefroid and Sen, PLDI 2005)
  - EXE (Cadar, Ganesh, Pawlowski, Dill, and Engler， CCS2006)
- SAGE：Microsoft的concolic执行器(动态符号执行)
- Mayhem(CMU), Angr(UCSB), Triton
- Java Symbolic PathFinder
- KLEE