Introduction to Program Analysis

Julian Cohen
HockeyInJune@isis.poly.edu
Manual and Automatic Program Analysis

Program Analysis

- pro·gram /ˈprōˌgram,-grəm/ (noun):
  a sequence of instructions that a computer can interpret and execute
- a·nal·y·sis /əˈnaləsis/ (*noun*):
  detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation

Common Themes

Fuzzy Analysis

versus

Sound and Complete Analysis

The Halting Problem

- Given a computer program
- Determine if the program halts or runs forever

```
srand( time( NULL ) );
while ( true ) {
     if (rand() == 1) {
          exit();
     }
}
```

```
int i = 10;
while ( true ) {
     if ( i == 1 ) {
          exit();
     }
     i++;
}
```

Rice's Theorem

- Given a property and a program
- Determine if the program implements the property

Property: Addition
Program:

```
int add(int a, int b) {
     return a + b;
}
```

Property: JavaScript
Program: Internet Explorer

Code Analysis

- IDA Pro / IDA Python
- BAP
- radare
- ERESI
- Coverity, Fortify, SonarQube
- LLVM, BitBlaze, Insight, Jakstab, PANDA, DECAF

Program Introspection

- Debugging, Instrumentation, Emulation
- Windbg
- Immunity Debugger
- gdb / ptrace
- vtrace / vdb / vivisect
- PIN, DynamoRIO
- qemu

Automatic Testing

- Fuzzing

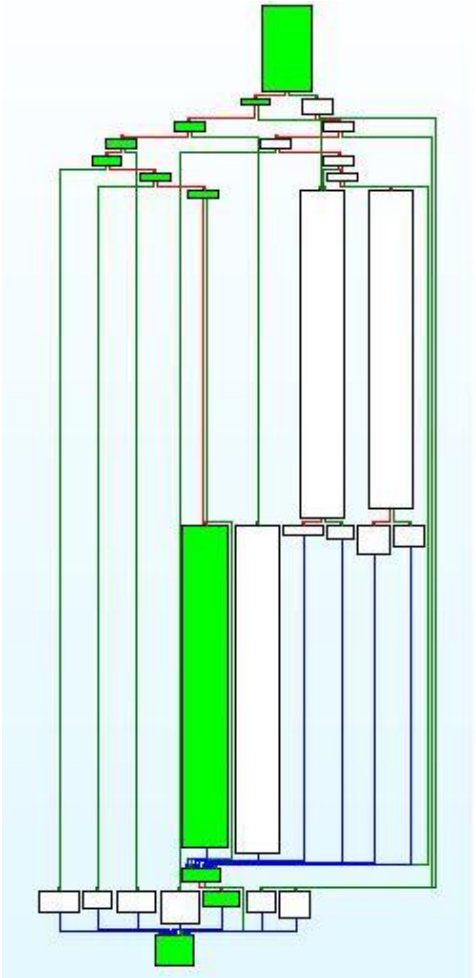
Code Translation and Representation

- Compilation, Optimization, Transformations

```
addr 0x0 @asm "add %rax,%rbx“
label pc_0x0
T_t1:u64 = R_RBX:u64
T_t2:u64 = R_RAX:u64
R_RBX:u64 = R_RBX:u64 + T_t2:u64
R_CF:bool = R_RBX:u64 < T_t1:u64
R_OF:bool = high:bool((T_t1:u64 ^ ~T_t2:u64) & (T_t1:u64 ^ R_RBX:u64))
R_AF:bool = 0x10:u64 == (0x10:u64 & (R_RBX:u64 ^ T_t1:u64 ^ T_t2:u64))
R_PF:bool =
       ~low:bool(let T_acc:u64 := R_RBX:u64 >> 4:u64 ^ R_RBX:u64 in
              let T_acc:u64 := T_acc:u64 >> 2:u64 ^ T_acc:u64 in
              T_acc:u64 >> 1:u64 ^ T_acc:u64)
R_SF:bool = high:bool(R_RBX:u64)
R_ZF:bool = 0:u64 == R_RBX:u64
```

Code Coverage

- Code Executed / Code Available



Data Flow Tracking

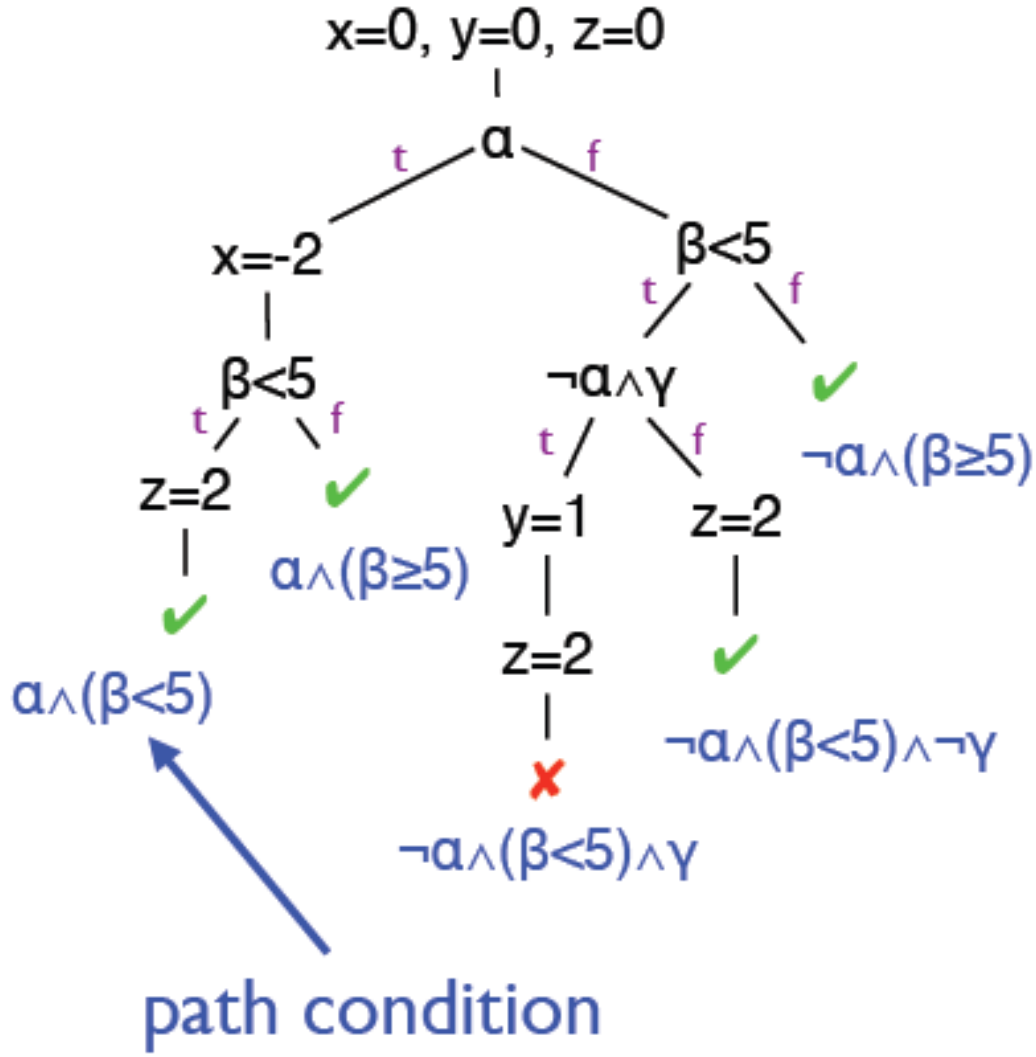
- Follow data flow from source to sink

Abstract Interpretation

- To describe a program in another language

Symbolic Execution

```
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)
```



Constraint Solving

```
(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Real)
(declare-const e Real)
(assert (> a (+ b 2)))
(assert (= a (+ (* 2 c) 10)))
(assert (<= (+ c b) 1000))
(assert (>= d e))
(check-sat)
(get-model)
```

```
sat
(model
  (define-fun c () Int
    (- 5))
  (define-fun a () Int
    0)
  (define-fun b () Int
    (- 3))
  (define-fun d () Real
    0.0)
  (define-fun e () Real
    0.0)
)
```

Formal Verification

- To mathematically prove a program “correct”

Homework

http://www.cs.umd.edu/class/fall2011/cmsc631/lectures/sym.pdf

http://akira.ruc.dk/~madsr/webpub/absint.pdf

- Section 1

http://www.cs.unm.edu/~moore/tr/02-12/zovi.pdf

- Optional

http://users.ece.cmu.edu/~ejschwar/papers/oakland10.pdf