Control-Flow Analysis

Paul Gazzillo

Compiler Optimization

- Compiler generates machine code
- Many choices
  - Instruction selection (addition/multiplication, store/load, registers)
  - Register usage
  - Cache utilization
- Free to *optimize*
  - Behavior preserving transformation
  - Faster code, smaller code, fewer memory accesses, more cache usage, etc

Intermediate Representation

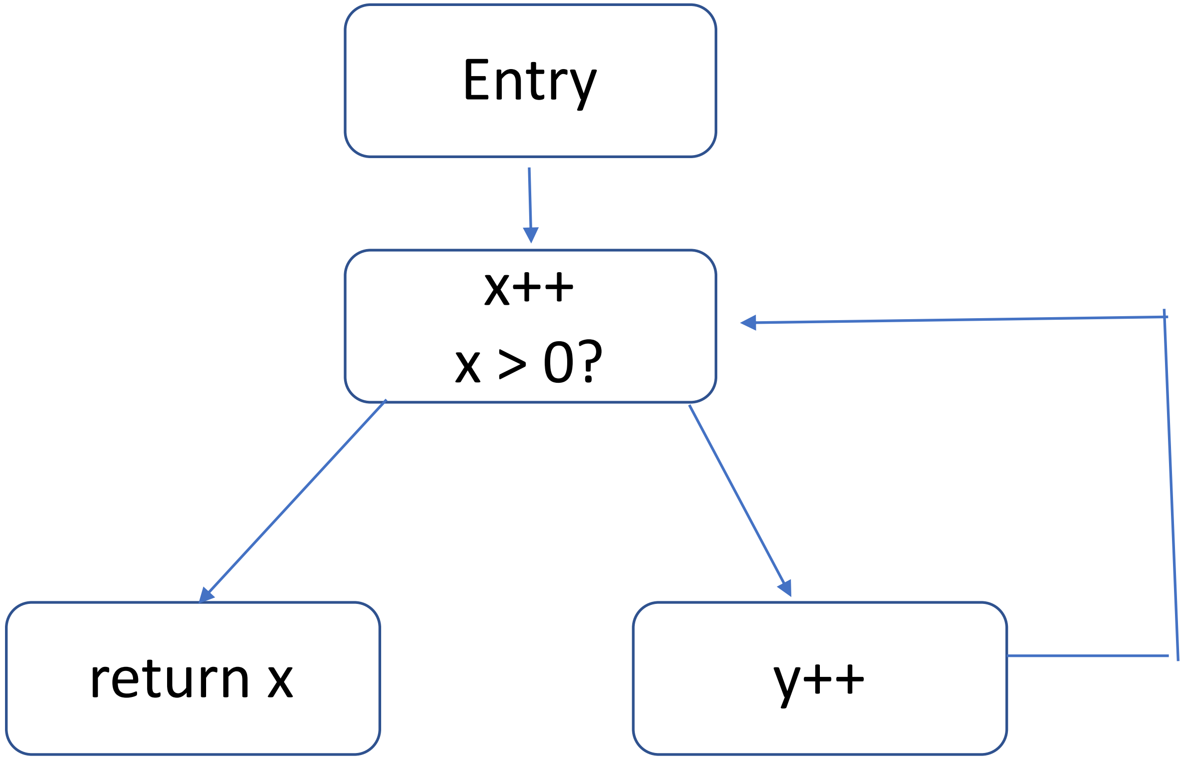
- Modern compilers target many architectures
  - x86
  - ARM
  - etc
- Similar optimization opportunities
- Machine independent machine code
  - Close to assembly
  - Without hardware differences
    - unbounded registers
    - common instruction subset

First Step: Control-Flow

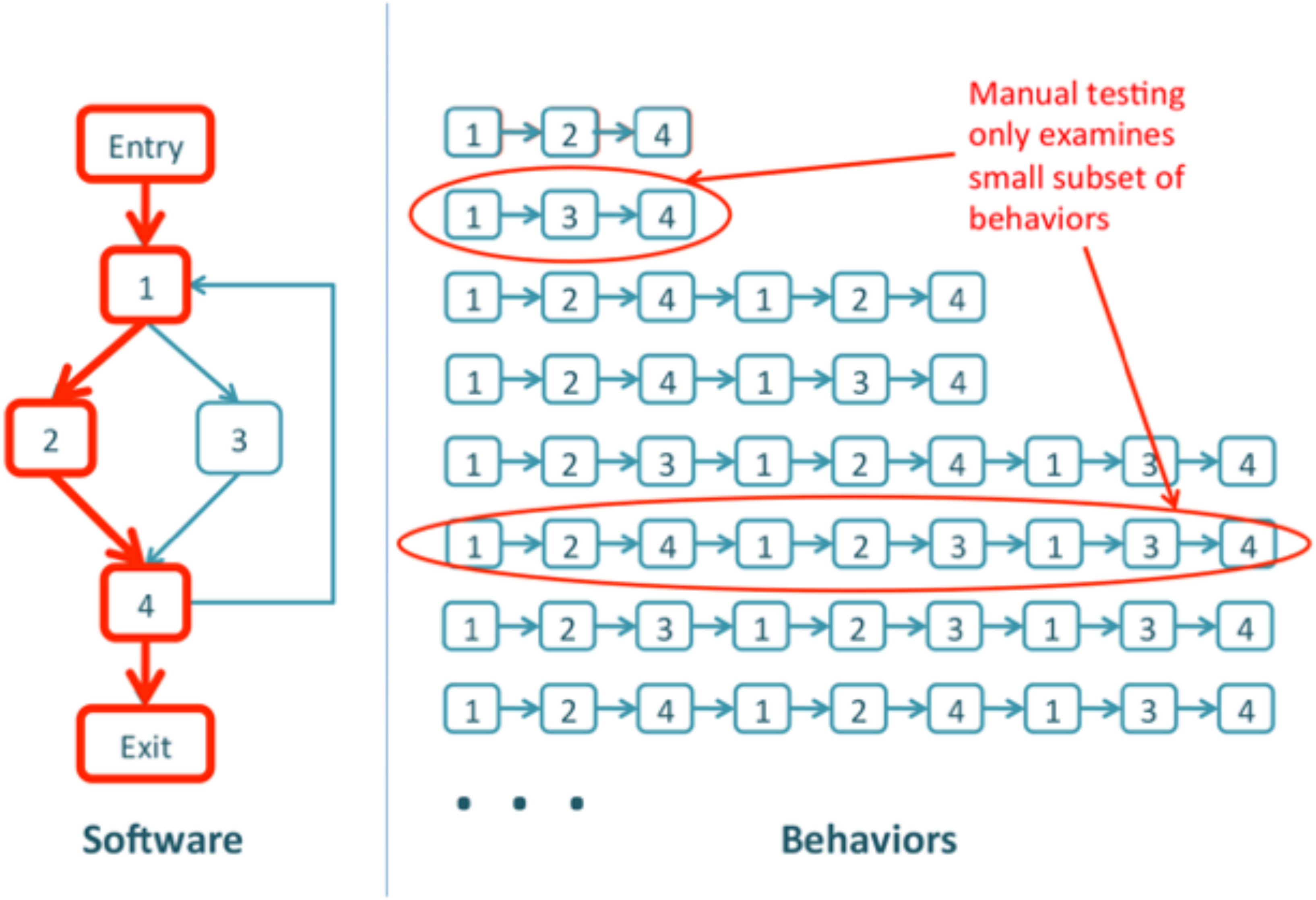
- AST
  - Language-specific
  - Process via tree traversal
  - Little information about what program does
    - e.g., steps of computation
- Control-Flow Graph (CFG)
  - Order of instructions
  - Branching behavior
  - Graph algorithms

Control-Flow Graphs

- Entry node is start of program
- Nodes are straightline code
  - No branching to or from inside
- Edges are branches
  - Ifs and loops
  - Single edge is unconditional branch



State Transitions



Manual testing only examines small subset of behaviors

Thanks to John Mitchell

Control-Flow Analysis

- Automatically construct CFG
  - AST -> CFG
  - IR -> CFG
- Traversals based on classic graph algorithms
  - Depth-/bread-first traversal
- Dominance/post-dominance
  - Which operations always happen before/after others, e.g., for optimization
- Structural analysis
  - Recover syntax tree, e.g., for reverse engineering

Control-Flow Applications: Optimization

- CFG is first step in most optimizations
- Register allocation
  - Avoid slower memory operations
- Removing dead code
  - If (false) ...
- Constant propagation/folding
  - x = 3; y = x; z = x + y; => z = 6
- Common subexpression elimination
  - y = x * 3; z = x * 3; …
- And more

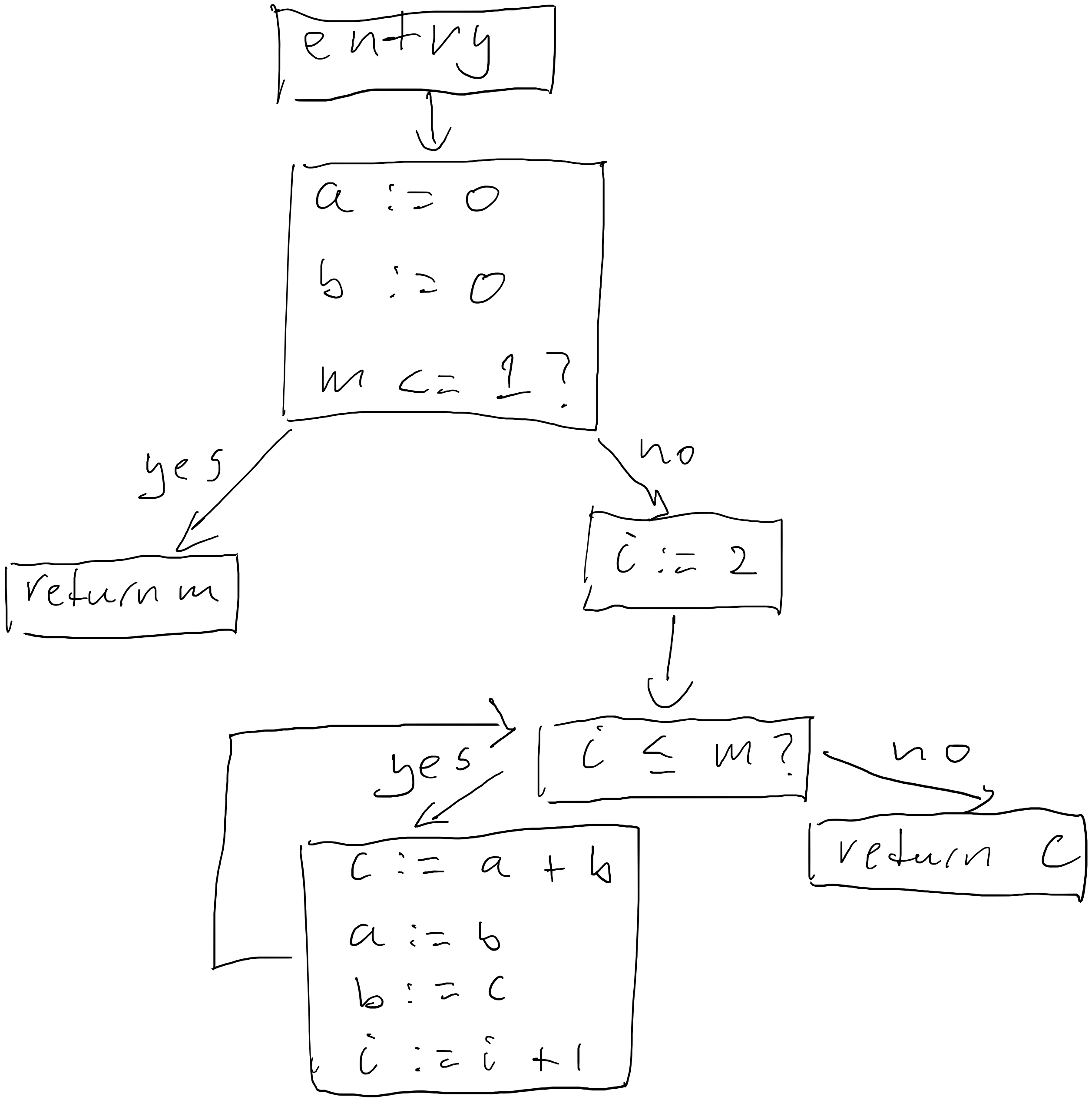
Control-Flow Applications: Program Analysis

- First step in many analyses
- malloc without free
    - is free always after malloc?
- Null pointer error
    - Null flows to dereference?
- Divide-by-zero
- Use-after-free
- Uninitialized variable use
- Array out-of-bounds access
- And more

CFG Analysis

- Dominators
  - Loop identification
  - Code motion
  - Parallelization

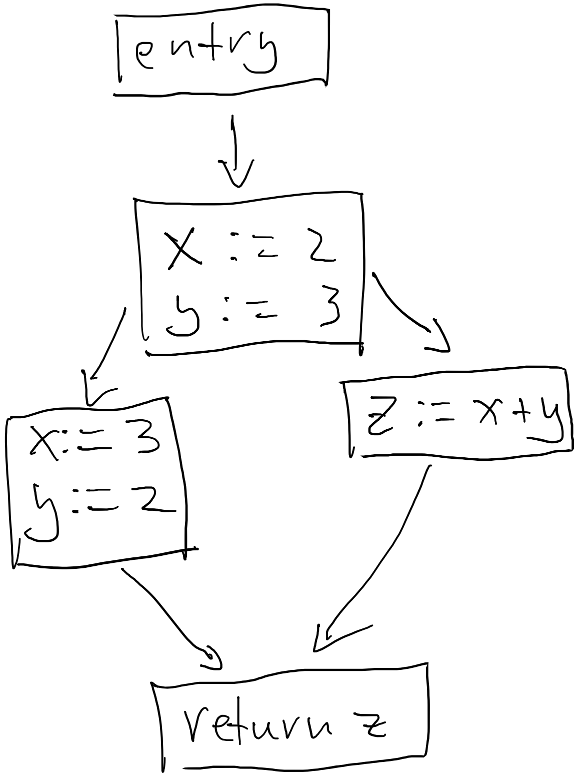
```
function fib(m : int) : int
  var a : int
  var b : int
  var c : int
  var i  : int
  begin
    a := 0
    b := 1
    if (m <= 1) then
      return m
    else begin
      i := 2
      while (i <= m) do begin
        c := a + b
        a := b
        b := c
        i := i + 1
      end
      return c
    end
  end
write fib(10)
```



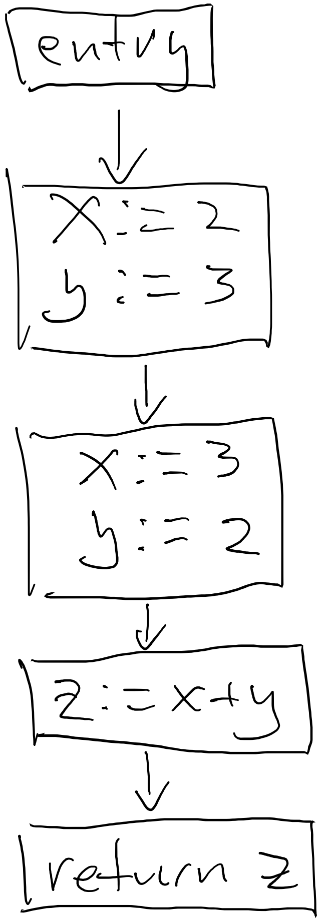
Which CFG?

```
function f(cond : bool) : int
  var x : int
  var y : int
  var z : int
  begin
    if (cond) then begin
      x := 2
      y := 3
    end else begin
      x := 3
      y := 2
    end
    z := x + y
    return z
  end
write f(true)
```

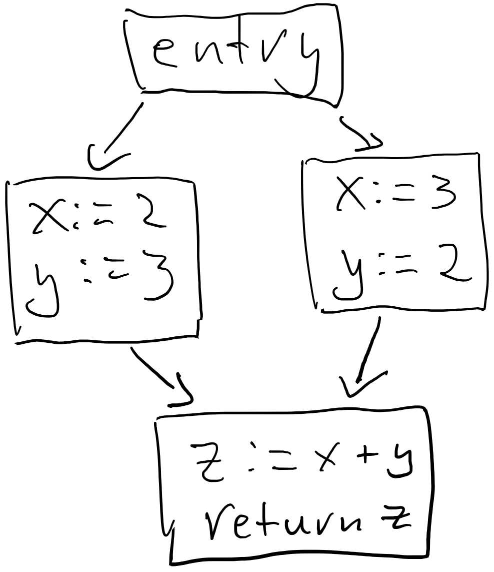
(a)



(b)



(c)



entry
x := 2
y := 3
x := 3
y := 2
z := x + y
return z