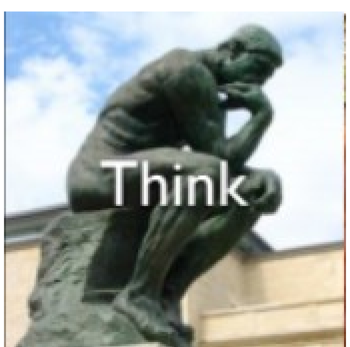
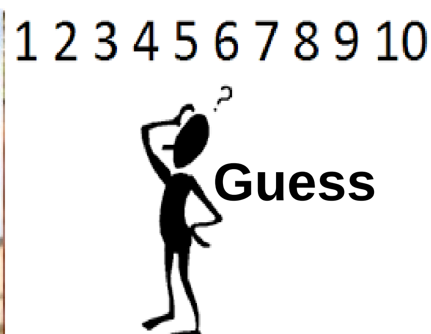
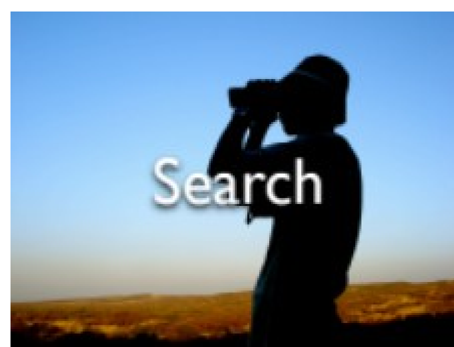
Topics for this Lecture

- Manually Test generation
- Automatic Test Generation Approaches
  - ~~Random Testing~~ ~~(Guess)~~
  - ~~Search Based Software Testing~~ ~~(Search~~)
  - Constraint-Based (Symbolic execution) Testing √√ (Deduce)







Oregon State University

Constraint-Based Testing (CBT)

- **Constraint-Based Testing (CBT)** is the process of generating test cases against a testing objective by using constraint solving techniques.
- **Test objective**: generating a test case that covers a given testing criterion (such as statement, branch, batch, etc.)
- Constraints on inputs: if inputs satisfy constraints, then testing objective will be satisfied
- **CBT i**mproves significantly code-coverage (as constraints capture hard-to-reach test objectives).
- **CBT** is fully automated test data generation methods.
-

Oregon State University

Symbolic Execution

- **Symbolic execution** is a program analysis technique that analyzes a program's code to automatically generate test data for the program.

- **Symbolic execution** analysis program's code with unspecified inputs.

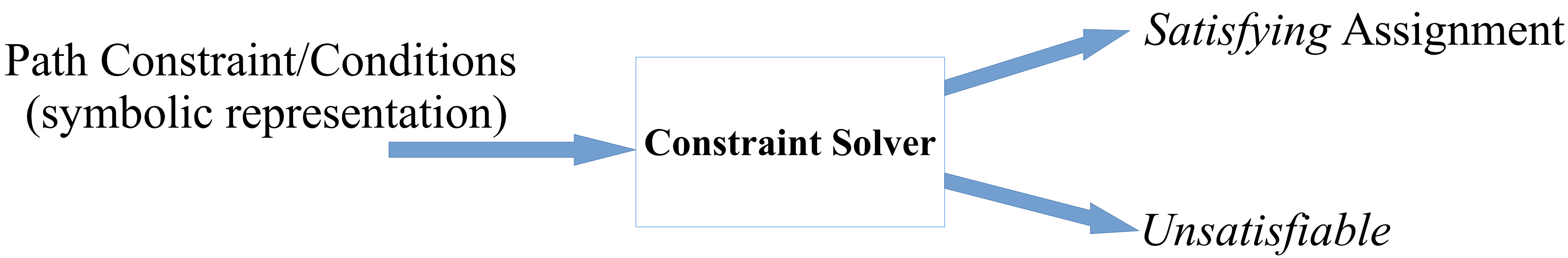
- **Symbolic execution** uses symbolic values, instead of concrete values, as program inputs, and represents the values of program variables as symbolic expressions of those inputs.

- For each path, Symbolic Execution builds a path constraint/path condition.

- The **path constraint/path condition** (PC) is a Boolean formula over the symbolic inputs, which is an accumulation of the constraints that the inputs must satisfy for an execution to follow that path.

Oregon State University

Constraint Solvers



- A path condition is ***satisfiable*** **(i.e., feasible)** if there is a way to assign values to variables and make the **path condition** (i.e., boolean formula) true.
  e.g., $(x > 0 \;\&\&\; x < 20 \;\&\&\; x == y + y)$ is satisfied by $(x{:}10, y{:}5)$

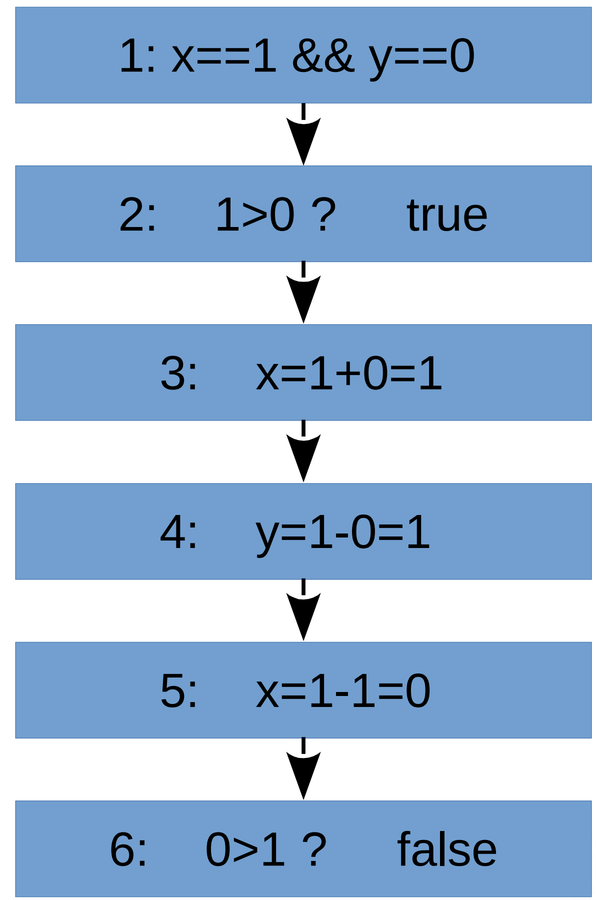
- A path condition is ***unsatisfiable*** **(i.e., infeasible)** if every assignment of values to variables makes the formula false, which means that there is no program input for which the program will take that (infeasible) path.
  e.g., $(x > 0 \;\&\&\; x < 20 \;\&\&\; x == y + y \;\&\&\; x\%2 == 1)$ is unsatisfiable.

Oregon State University

Standard Execution

```
int x,y;
1:    if (x>y) {
2:          x = x+y;
3:          y = x-y;
4:          x=x-y;
5:          if (x>y)
6:                assert false;
7     }
8:  print x,y
```

Code that swaps two integers
variables x and y



Symbolic Execution

```
   int x, y;
1  if(x > y){
2     x = x+y;
3     y = x-y;
4     x = x-y;
5     if(x - y > 0)
6        assert false;
7  }
8  print(x, y)
```

(a)

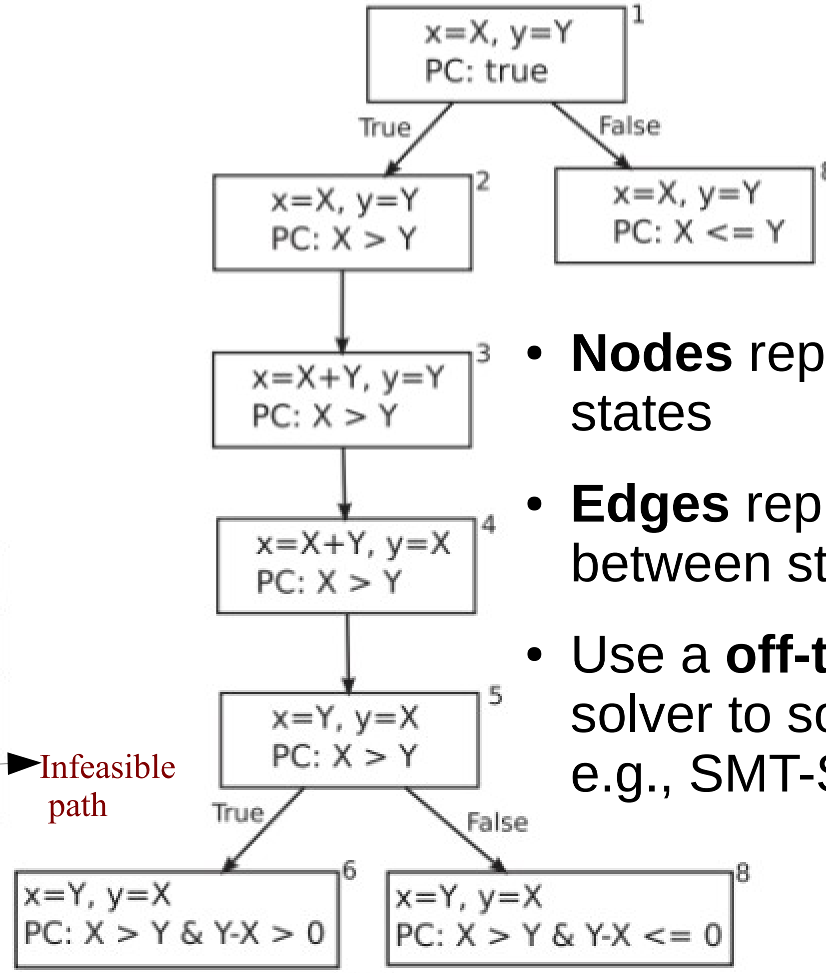
Code that swaps two integers

Path	PC	Program Input
1,8	X <= Y	X=1 Y=1
1,2,3,4,5,8	X>Y & Y-X<=0	X=2 Y=1
1,2,3,4,5,6	X>Y & Y-X>0	none

Infeasible path

(c)

The PC's and their solutions (if they exist)



(b)

The symbolic execution tree for the code

- **Nodes** represent program states
- **Edges** represent transitions between states
- Use a **off-the-shelf** constraint solver to solve path conditions, e.g., SMT-Solver / SAT-Solver

Oregon State University

Symbolic Execution Limitations

- The technique suffers from some problems that limit its effectiveness on real world software:
  1. **Path explosion**: most real world software have an extremely large number of paths, and it is difficult to analyze all program paths.
  2. **Complex constraints**: it is difficult to analyze constraints involving mathematical functions such as sin() and log()

```
double example(int x, int y, double z) {
  boolean flag = y > 1000;
  // ...
  if(x + y == 1024)
    if(flag)
      if(Math.cos(z)−0.95 < Math.exp(z))
        // target branch
  // ...
}
```

The third branch contains a problematic non-linear constraint, and if the **Math** library is not available in source code or bytecode, then deriving constraints can be difficult in the first place.

Native code: no access to source code

Oregon State University

Tools

- Symbolic Execution - Java PathFinder (NASA)
  http://javapathfinder.sourceforge.net/extensions/symbc/doc/

- SAGE (Microsoft)
  https://patricegodefroid.github.io/public_psfiles/ndss2008.pdf

- PEX (C#) https://pexforfun.com/default.aspx

- CUTE(C)
  http://osl.cs.illinois.edu/publications/conf/sigsoft/SenMA05.html

- Z3 is one of the most powerful SMT solvers currently available.
  http://channel9.msdn.com/posts/Peli/The-Z3-Constraint-Solver/

Oregon State University

References:

Anand, Saswat, et al. "An orchestrated survey of methodologies for automated software test case generation." Journal of Systems and Software 86.8 (2013): 1978-2001.

Gotlieb, Arnaud. "Euclide: A constraint-based testing framework for critical c programs." Software Testing Verification and Validation, 2009. ICST'09. International Conference on. IEEE, 2009.

Oregon State University