Topics for this Lecture

- Automatic Test Generation Approaches
  - ~~Random Testing~~
  - ~~Evolutionary Search~~
  - Constraint-Based (symbolic execution) Testing

Constraint-Based Testing (CBT)

- **Constraint-Based Testing (CBT)** is the process of generating test cases against a testing objective by using constraint solving techniques

- Although the CBT technique was first proposed in the mid seventies, the technique has received much attention recently from researchers.

- 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.

Test data generation by 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** uses symbolic values, instead of concrete values, as program inputs, and represents the values of program variables as symbolic expressions of those inputs.

- The **path constraint** (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.

- **Symbolic Execution** was designed for accomplishing this data generation task automatically with the help of *constraint solvers*.

Test data generation by symbolic execution

- A symbolic executor first identifies a program path in its control flow graph (CFG)

- Select one or several paths → **Path selection** step

- Generate the path conditions → **Symbolic evaluation** techniques

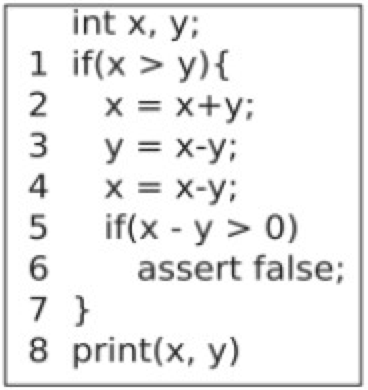
- Solve the path conditions to generate test data that activate the selected paths → **Constraint solving**

- The symbolic executor asks a *constraint solver* if the collected constraint has a solution, and The *constraint solver* either answers:

  1. "**yes**" and gives a possible assignment of the variables.
  2. "**not**" (proving the corresponding path is not *feasible*)

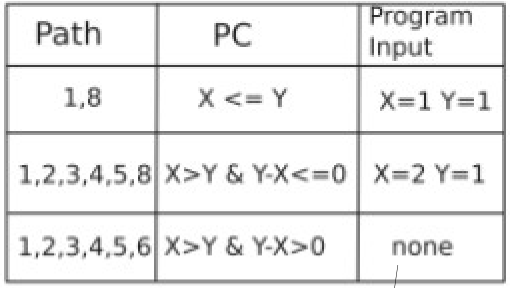
Test data generation by symbolic execution

- (**a**) Code that swaps x and y, when the initial value of x is greater than the initial value of y.
- (**b**) the corresponding *symbolic execution tree*, and
- (**c**) test data and *path constraints* corresponding to different program paths



```
   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)
```

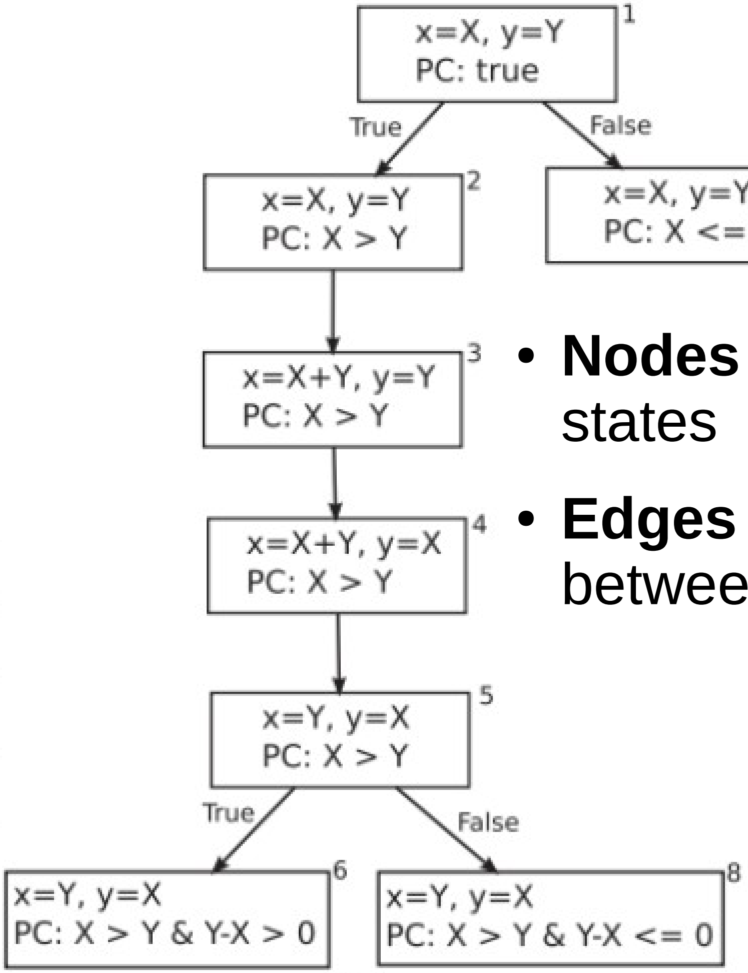
(a)



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

(c)

Infeasible path



(b)

- **Nodes** represent program states
- **Edges** represent transitions between states

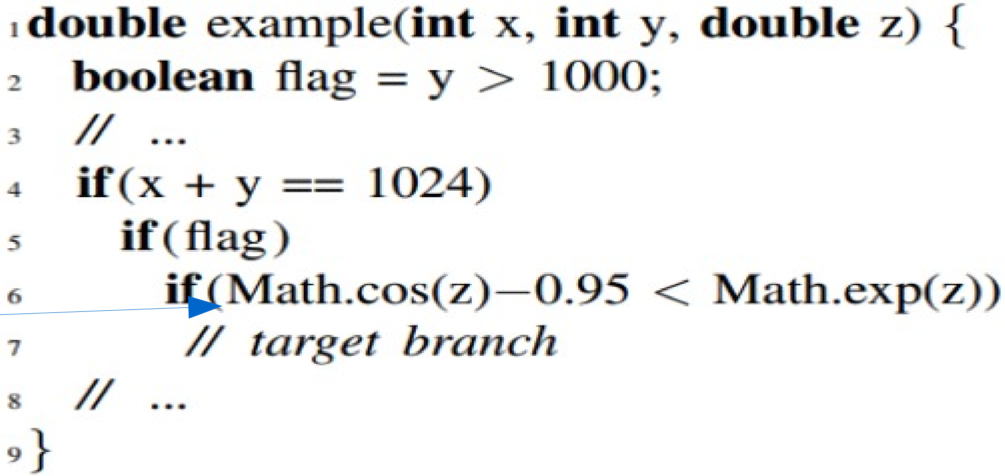
Problems for symbolic evaluation techniques

- Explosion of paths (heuristics are needed to explore the search space)

- Number of iterations in loops must be selected prior to any symbolic execution

- Environment Interactions

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.



```
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
  // ...
}
```

Tools

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

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

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.