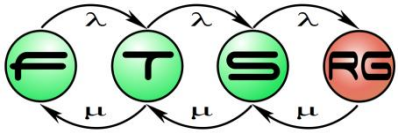
Symbolic execution based testing Java Path Finder

Csaba Debreceni



Budapest University of Technology and Economics
Department of Measurement and Information Systems

Introduction

- Software Testing
  - 50% of total development costs
- White box testing vs Black box testing
- Solutions
  - Model checking
    - ☺ automatic, exhaustive <> ☹ scalability issues
  - Static analysis
    - ☺ scalable <> ☹ spurious warnings
  - Dynamic analysis
    - ☹ Incomplete -> missing important errors

Symbolic execution

- King, A new approach to program testing, 1975
  - Symbolic values <> Concrete values
  - Symbolic expressions
  - Symbolic tree
    - Path constraint (PC)
    - Program counter
  - Generate test inputs
- At each branch point PC updated
  - Unsatifiable –> Stop
  - Satisfiable -> Fork

Example

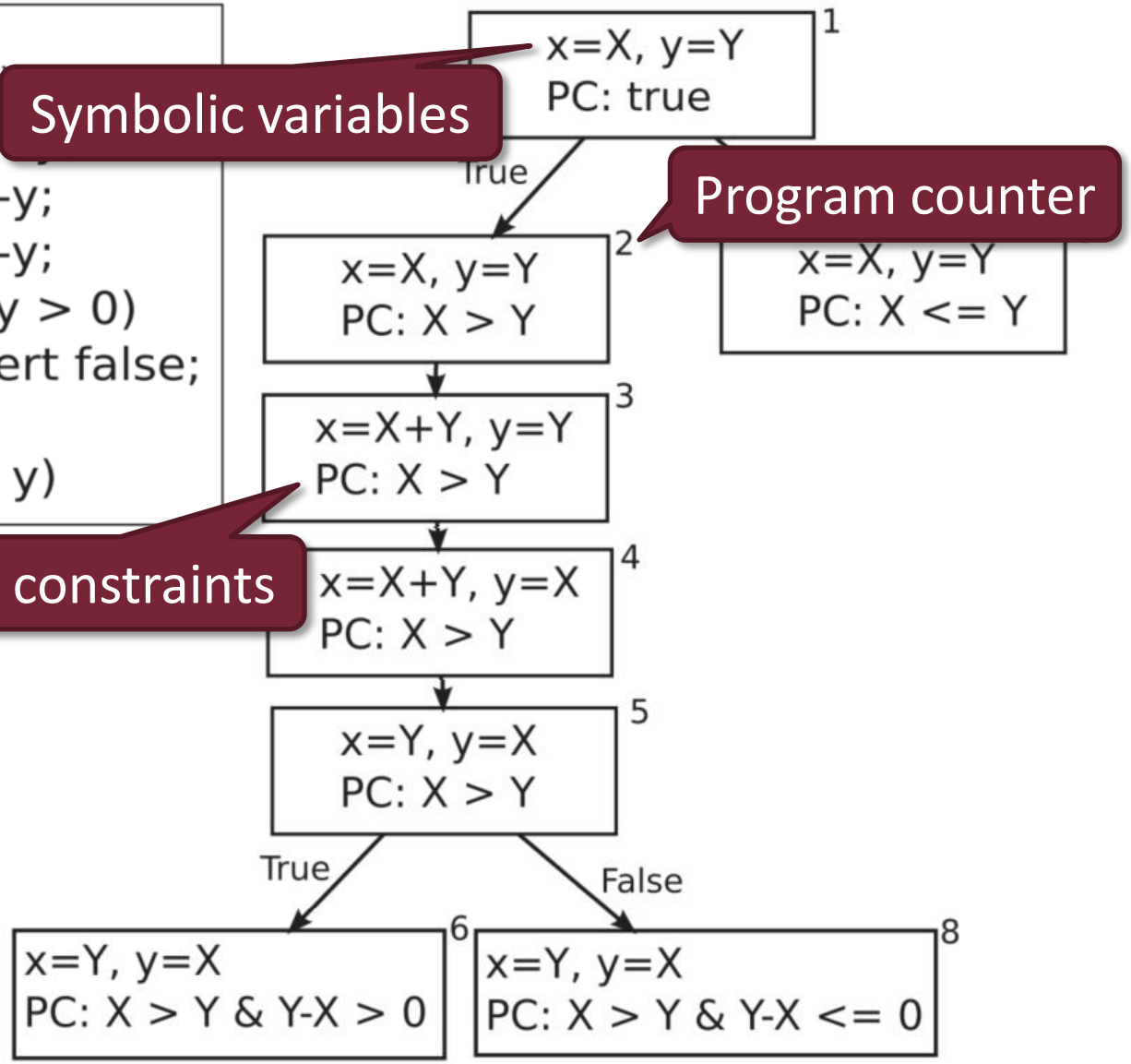
```
    int x, y;
1   if(x > y){
2      x = x+y;
3      y = x-y;
4      x = x-y;
5      if(x - y > 0)
6         assert false;
7   }
8   print(x, y)
```

Input

Program counter

Example

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



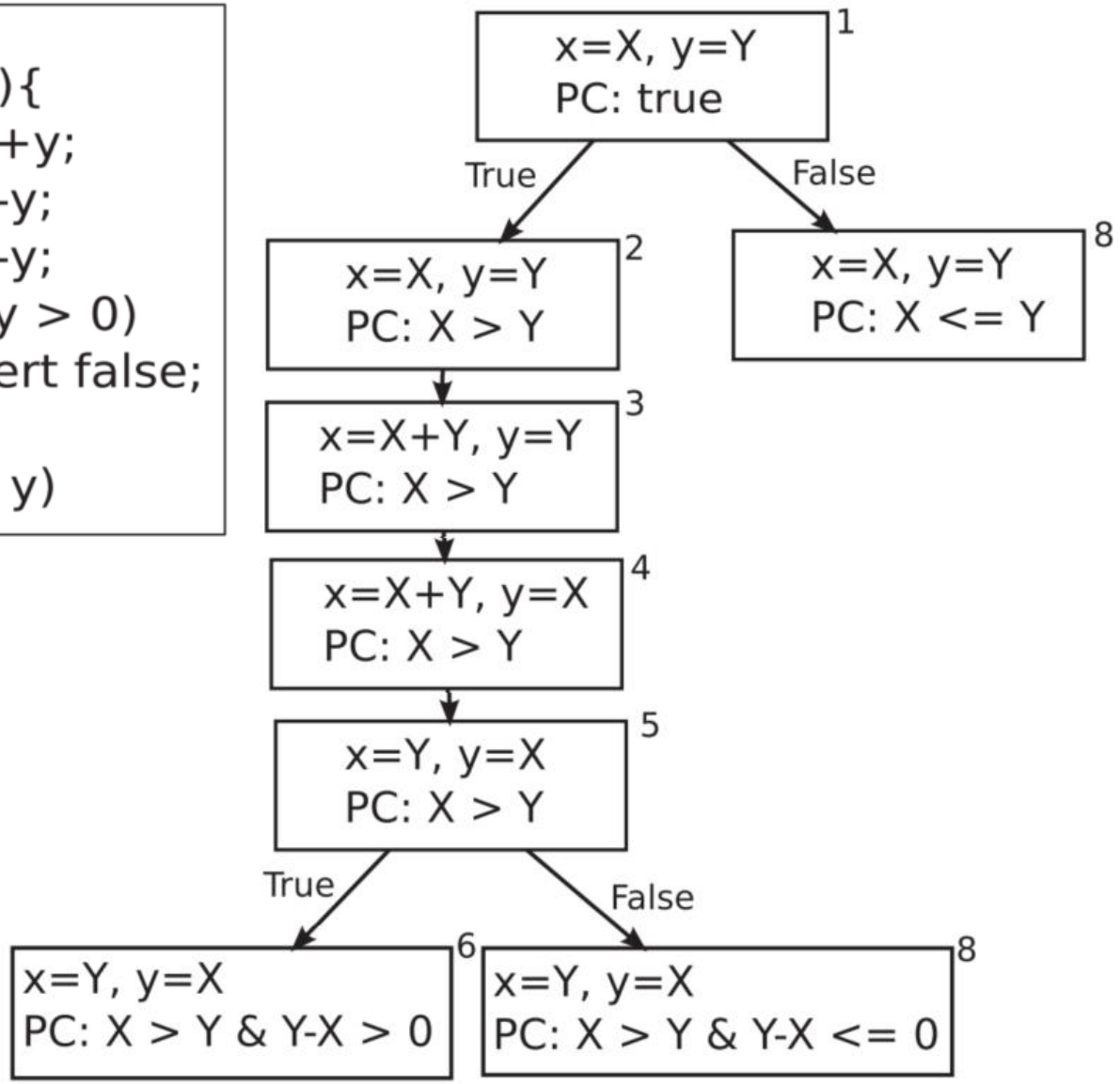
Symbolic variables

Program counter

Path constraints

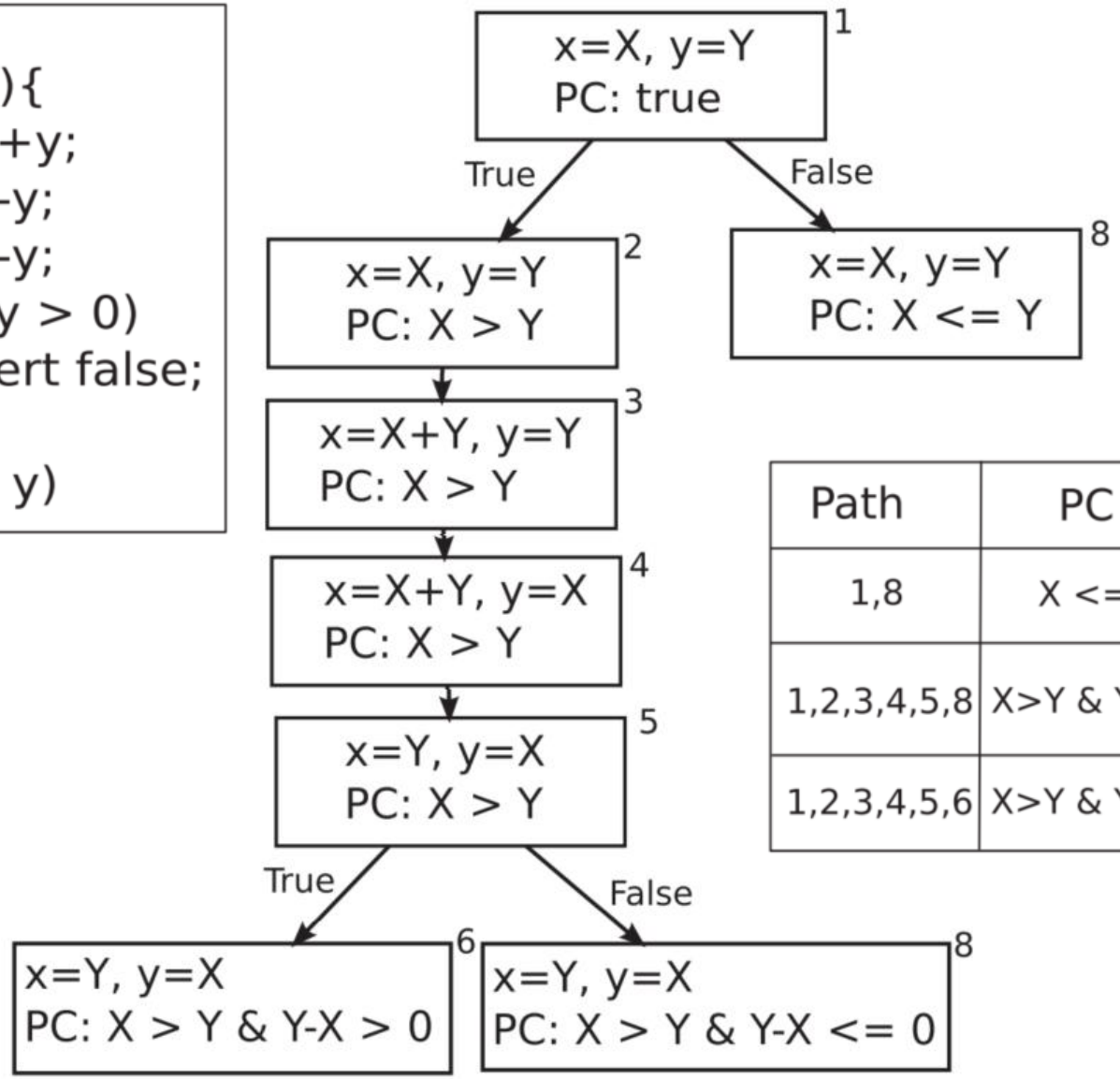
Example

```
    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)
```



Example

```
    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)
```



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

Why now?

- Requires large and complex constraints
  - SAT/SMT solvers
  - Z3, Alloy, STP, Yice…
- Computationally expensive
  - significantly lower execution time
  - but still limited

Possible uses

- Improve code coverage
- Expose software bugs
- Error reporting
- Security exploits
- Robustness testing
- Testing of graphical UI

Fundamental open problems

- Path explosion
  - Extremely large number of paths -> computational overhead
  - BFS, DFS, RR
- Path divergence
  - Multiple programming “languages”
  - Manual effort is required
- Complex constraints
  - Concolic
  - Simplify constraints

Hivatkozás

- Cseppentő Lajos
  - Szimbolikus végrehajtást használó tesztgeneráló eszközök egységes összehasonlítása
  - Kari TDK 2. helyezés

Név	Platform	Hozzáférés	Megj. éve	Bemenet	Kimenet
CATG	Java	Nyílt forráskód	2012	Bájtkód	Tesztbemenet
CodePro AnalytiX	Java	Zárt forráskód	2001	Forráskód	JUnit tesztesetek
CREST	C	Nyílt forráskód	2008	Forráskód	Tesztesetek
CUTE & jCUTE	C & Java	Nem elérhető	2005	Forráskód/bájtkód	Tesztbemenet
DART	C	Nem elérhető	2005	Forráskód	?
EXE	C	Nem elérhető	2006	Forráskód	?
LCT	Java	Nyílt forráskód	2010	Bbájtkód	Tesztbemenet
KLEE	C	Nyílt forráskód	2008	LLVM bájtkód	Tesztbemenet
Palus	Java	Nyílt forráskód	2010	Bájtkód	Tesztesetek
PET/jPET	Java	Nylt forráskód	2009	Forráskód	Tesztesetek
PEX	.NET	Zárt forráskód	2008	Bájtkód	Tesztesetek
SPF	Java	Nyílt forráskód	2012	Bájtkód	Tesztbemenet

Java PathFinder

http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-symbc

Java PathFinder

- Nasa
  - 1999: begin
  - 2005: open source
  - Mars Rover and Honeywell's DEOS (RTOS)
    - Automated verification of time partitioning
- Model checker
  - Set of properties
  - Counter examples
  - Over custom-made JVM
- Install
  - Mercurial repository
  - Eclipse plugin
  - https://www.youtube.com/watch?v=6vVAYT4yMfw

Symbolic PathFinder

- Released in 2013
- Over JPF model checker
  - model checker -> explores symbolic tree
- Several built-in SMT solver
  - Choco, IASolver, CVC3

- Only “main” methods
- JPF configuration
  - concolic = concrete + symbolic

DEMO Symbolic PathFinder

```java
package hu.bme.mit.inf.symbolic.example.source.classes;

public class Simple1 {

    public static void main (String[] args) throws Exception {
        Simple1 s = new Simple1();
        s.simple1(1, 2);
    }

    public void simple1(int x, int y) throws Exception {
        if(x > y) {
            x = x + y;
            y = x - y;
            x = x - y;
            if(x - y > 0)
                throw new Exception();
        }
        System.out.println(String.format("%s,%s", x,y));
    }
}
```

```
====================================================== system under test
hu.bme.mit.inf.symbolic.example.source.classes.Simple1.main()

====================================================== search started: 2014.12.11. 11:45
Symbolic string analysis
Symbolic string analysis
0,0
Symbolic string analysis
Symbolic string analysis
0,0

====================================================== Method Sequences
[simple1(-999999,-1000000)]
[simple1(-1000000,-1000000)]
```

```
target=hu.bme.mit.inf.symbolic.example.source.classes.Simple1

classpath=C:\\Eclipse\\Eclipse Symbolic\\workspace\\hu.bme.mit.inf.symbolic.example.source\\bin

symbolic.method=hu.bme.mit.inf.symbolic.example.source.classes.Simple1.simple1(sym#sym)

listener=gov.nasa.jpf.symbc.sequences.SymbolicSequenceListener

vm.storage.class=nil

search.multiple_errors=true
```

Summary

- Symbolic Execution
  - from the mid of seventies
  - symbolic values <> concrete data
  - test data from SMT
- Java PathFinder
  - from NASA
  - model checker
  - several module
- Symbolic PathFinder
  - based on JPF