Constraint-Based Reasoning in Static Analysis and Testing

Jian Zhang(张健)

Chinese Academy of Sciences

July 1, 2014

Test Data Generation & Static Analysis Based on Symbolic Execution & Constraint Solving

Testing & Static Analysis

Testing:

- Test data/case generation/preparation ***
- Test case execution
- Test result analysis / fault localization …

Static Analysis (of source code) ***

Path-oriented/sensitive Analysis

* program path ➔ path condition (PC)

↑

symbolic execution

(or backward substitution)

* The path is feasible (or executable) iff the PC is satisfiable.

Constraint Solving

Symbolic Execution

giving *symbolic* values as input to the program, and simulating the program's behaviour

- [Boyer et al. 1975] [King 1976] [Clarke 1976]
- Many research groups working on this technique ...
- **Path feasibility analysis and constraint solving [Zhang 2000] [Zhang-Wang 2001]**
- **Constraint Solving and Symbolic Execution [Zhang VSTTE2005]**

A path in bubble-sort

- i = n-1;
- @ i > 0;
- indx = 0;
- j = 0;
- @ j < i;
- @ a[j+1] < a[j];
- temp = a[j];
- a[j] = a[j+1];
- a[j+1] = temp;
- indx = j;   j = j+1;
- @ j >= i;
- i = indx;
- @ i <= 0;

- Path condition:

  n-1 > 0

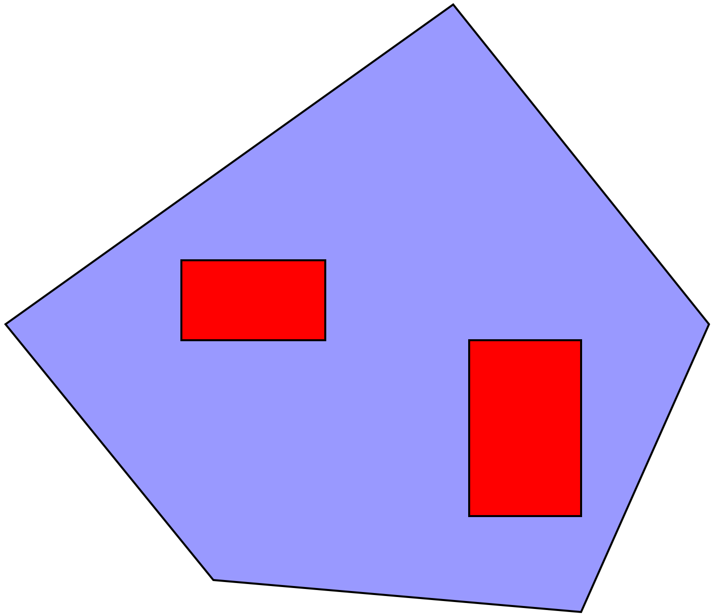
  a[1] < a[0]

  n-1 <= 1

Input data: n = 2

a[]: { 3, 2 } or { 8, 1} or …

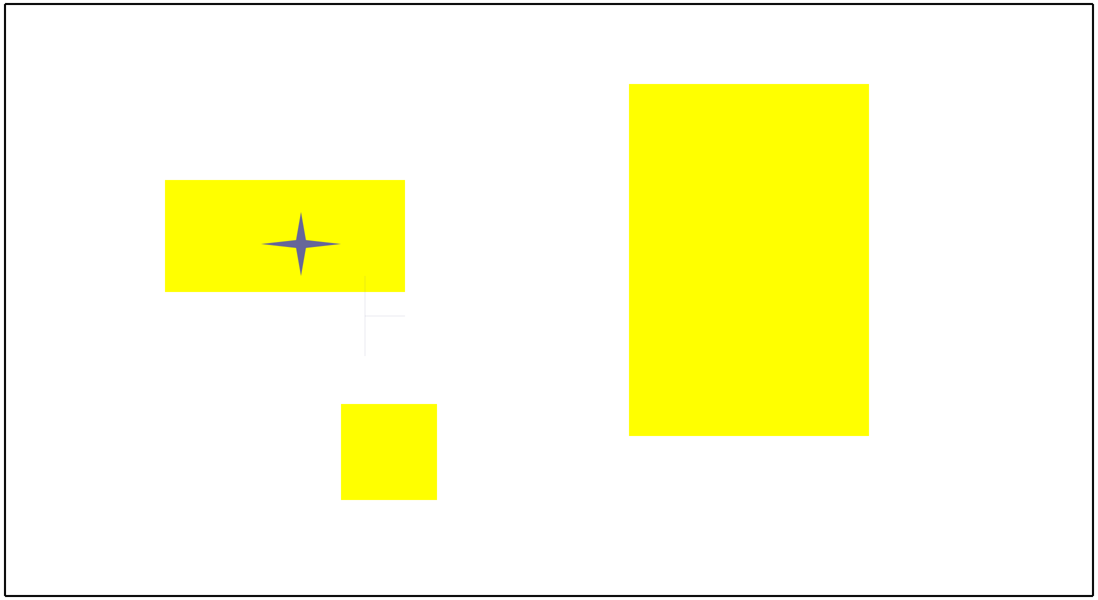
Symbolic Execution



Checking **areas** (rather than **points**) in the input space.

Test Generation and Bug Finding via Symbolic Execution

- Finding one point in any area



Satisfiability Checking (SAT solving)

- Example Input:

```
p cnf 3 3
1 2 3 0
-1 -2 0
-2 -3 0
```

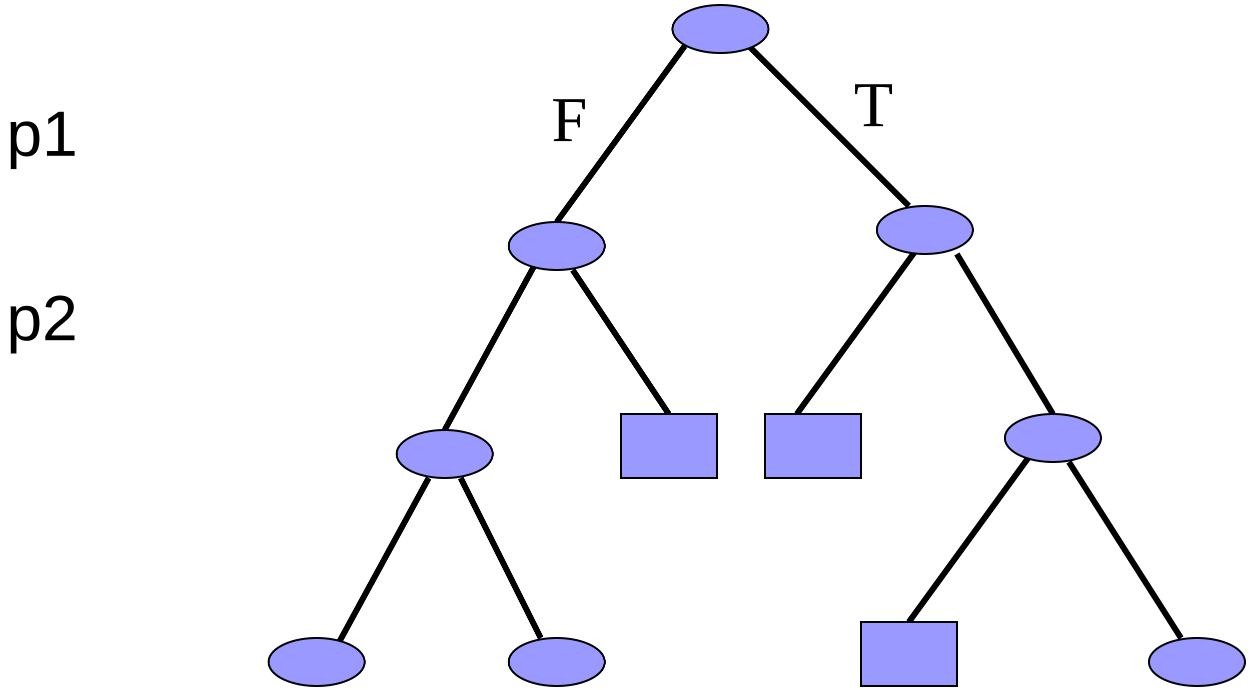
denoting the following formula:

(p1 OR p2 OR p3)   AND

(NOT p1 OR NOT p2) AND

(NOT p2 OR NOT p3)

Binary search tree



Satisfiability Modulo Theories (SMT) solvers – CVC3/CVC4, Yices, Z3, …

- x3, x2, x1, x0: INT;

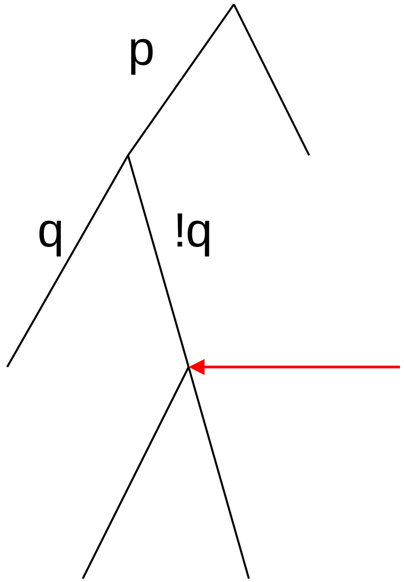
  CHECKSAT (x0 >= 0 AND x0 <= 9 AND x1 >= 0 AND x1 <= 9 AND x2 >= 0 AND x2 <= 9 AND x3 >= 0 AND x3 <= 9 AND (x0 > 0 OR x1 > 0 OR x2 > 0 OR x3 > 0) AND 1000*x0 + 100*x3 + 10*x2 + x1 = 2000*x3 + 200*x2 + 20*x1 + 2*x0);

Constraint Solver: BoNuS (1999-2001) [Zhang 2000] [ZhangWang 2001]

- Example.

```
enum { Male, Female } gender;
int age;
bool b = (age > 18);
bool married;
{
    and(not(b), married);
}
```

Search proc. (simple SMT solving)



p: (x>3)
q: (2*x > 5)

Check the feasibility of:
x>3; 2*x <= 5

Linear programming:
lp_solve

Constraint Solving + Symbolic Execution [Zhang VSTTE 2005 (LNCS 4171)]

- verify – or find bugs in – certain programs
- check the error messages produced by other static analyzers, to eliminate some false alarms
- automate an important part of unit testing, i.e., generating test cases (input data) for the program
- generate test cases for black-box testing or model-based testing, if a proper specification (like EFSM) is provided.

Unit Testing – Stmt/branch coverage Examples: GNU coreutils **[XZ 2006]**

- **remove_suffix()** in basename.c
- **cat()** in cat.c
- **cut_bytes()** in cut.c
- **parse_line()** in dircolors.c
- **set_prefix()** in fmt.c
- **attach()** in ls.c
- **bsd_split_3()** and **hex_digit()** in md5sum.c

Example. GNU make: dir.c

```
char *dosify(char filename[20]) {
    …
    for (i = 0; *filename != '\0' && i < 8 &&…; i++) {
      *df = *filename;  df++;  filename++;
    }
    if (*filename != '\0') {
      *df = *filename;  df++;  filename++;
    for (i = 0; *filename != '\0' && i < 3 &&…; i++) {
      *df = *filename;  df++;  filename++;
    }
}
```

Test suite: 3 test cases → 100% branch coverage

Static Analysis of C programs

- inter-procedural, path sensitive tools for finding *memory leak*
  - **[Xu-Zhang, 2008]** on top of LLVM
  - **[Xu-Zhang-Xu, 2011] melton**, on top of Clang static analyzer
- Canalyze -- a tool for finding various kinds of bugs (e.g., NULL pointer dereferencing; undefined return value; …)

Bugs Found in Open Source Software

Software	KLoC	Undef. value	NULL ptr	Mem. leak	Use after free
libosip2-4.0.0	28.9		√	√	√
libosip2-3.6.0	29.0			√	
lighttpd-1.4.32	46.3	√		√	√
Openssh-5.9p1	89.8			√	
wget-1.13	91.8	√		√	
sqlite-3.7.11	139.2			√	
Coreutils-8.15	202.3		√		
Coreutils-8.17	211.9				√
sed-4.2	30.4	√			
glibc-2.15	1020.5		√		

Error in Openssh-5.9p1

```
//in file sshconnect2.c
authmethod_get(...) {
    ...
    for( ; ;) {
        if ((name = match_list(…)) == NULL){//Allocate heap space to name
        }
        …
        if (…) {
            …
            xfree(name);
            return …;
        }
    }//end for
    if (name) xfree(name);
}
```

Ex. bug report -- bftpd

From: "Jesse Smith" <jessefrgsmith@yahoo.ca>
Date: 2013-5-28
Subject: Re: Some potential bugs in bftpd-3.8
To: "Zhenbo Xu" <zhenbo1987@gmail.com>

I had a chance to examine your bug reports for Bftpd. All of the problems you reported are correct. The memory handling for bftpd_cwd_mappath() was an especially bad bug.

All of these bugs have been fixed in my copy of the code and I will be releasing a new version of Bftpd soon …

Finding Bugs Related to Floating-point numbers

- Divide by zero
- One operand much larger than the other
- …

To detect such problems, we may need to solve constraints like:

$$((11 * a2 * b2 - b6 - 121 * b4)*16777216) < 3$$

Finding witnesses for data race bugs

Data race – severe concurrency bug

- [Said et al. 2011]
- …
- [Huang-Meredith-Rosu 2014]

We need to solve constraints like:

$$X10-X3 = 1;\quad X5<X7 \text{ or } X9<X2;\ \dots$$

Detecting Resource Leak in Android Apps

Resources:

- exclusive (e.g., Camera)
- memory consuming (e.g., MediaPlayer)
- energy consuming (e.g., SensorManager)

Resource request and release operations

Tool: **Relda** [Guo et al. 2013]

Benchmarks: Baidu, Taobao, Tencent, …

Example.

```
private void initCamera() throws IOException
  {  if(!bIfPreview)
        { //If the camera is not in preview mode, turn it on.
          mCamera = Camera.open();   }
        if (mCamera != null && !bIfPreview)
        { ...... mCamera.startPreview();  bIfPreview = true; }
  }
private void resetCamera()
  {  if (mCamera != null && bIfPreview)
        {  mCamera.stopPreview();
           mCamera.release();
          ...... }
  }
```

Technology Transfer

- need from industry
- power of the tool

- Commercialization … ??

Combinatorial Testing

- Black-box testing technique
- The system-under-test (SUT) has a set of parameters/components, each of which can take some values.
- Example:
  - Browser: IE, Chrome, Firefox, ...
  - Operating system: Linux, Windows XP, ...
  - Manufacturer: Dell, Lenovo, HP, ...

Constraints in CT

- Example of constraints:

  not ((Browser==IE) && (OS==Linux))

- [Arcuri and Briand 2012] "in the presence of constraints, random testing can be worse than combinatorial testing"

Constraints used everywhere

- static program analysis (bug finding)
- combinatorial testing

- [DeMillo-Offutt ~1990], …
- KLEE, SAGE, …
- Workshop on the Constraints in Software Testing, Verification and Analysis (CSTVA)

All kinds of Constraints

- Linear inequalities: $x+2y < 3$
- Integer difference constraints: $x-y < 2$
- Non-linear constraints: $2xy+z = 8$
- Propositional formulas: $(p \,||\, \sim q) \;\&\&\; r$
- SMT formulas: $(x-y>5) \,||\, (x+2y<16)$
- …

Extensions to SMT Solving

Extension to SMT solving (I)

	Finding 1 solution	Finding the best solution
Logic + arithmetic	SMT	SMT-opt
Linear Inequalities	LP, …	Linear programming

Optimization w.r.t. complex constraints

- Linear Programming

  min. f(X)

  s.t. Ax <= b

- SMT optimization

  min. f(X)

  s.t. constraints in SMT form

A Simple Example

min. $x - y$

subject to:

$(((y + 3x < 3) \rightarrow (30 < y)) \vee (x <= 60))$

$\wedge ((30 < y) \rightarrow \neg (x > 3) \wedge (x <= 60))$

Stress Testing – test data gen.

- Extract paths from some model (e.g., activity diagram).
- From the path, obtain resource consumption information.
- Generate constraint solving/optimization problems.
- Solve them !

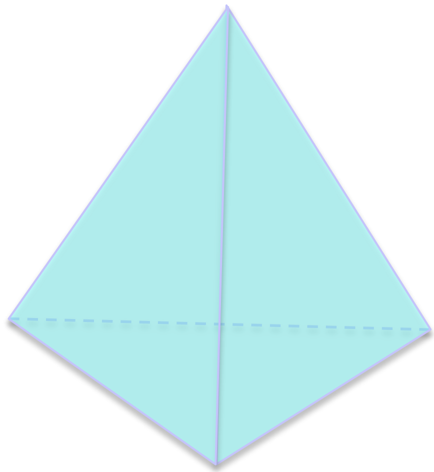
[Zhang-Cheung 2002]

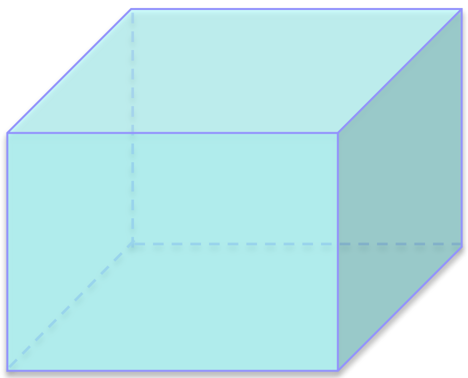
min:-100 (x00-x01) -1200 (x10-x11) -1800 (x20-x21) -56 (x30-x31) -1500 (x40-x41) -150 (x50-x51) -1440 (x60-x61) -25 (x70-x71) -200 (x80-x81) -230 (x90-x91)

u1 = 1; r1 = 1; v0 <= 25; s0 <= 25; u0 = v1; r0 = s1; u1 <= u0; x01 = u1 or x11 = u1 or x21 = u1; x00 <= u0; x10 <= u0; x20 <= u0; x01 = x21; x10 = x20; x00 - x01 <= 5; x10 - x11 <= 6; x20 - x21 <= 15; v1 <= v0; x31 = v1 or x41 = v1; …

Extension to SMT solving (II)

	solution?	#of solutions
Logic+ arithmetic	SMT	Vol.comput.
prop. Logic	SAT	model counting





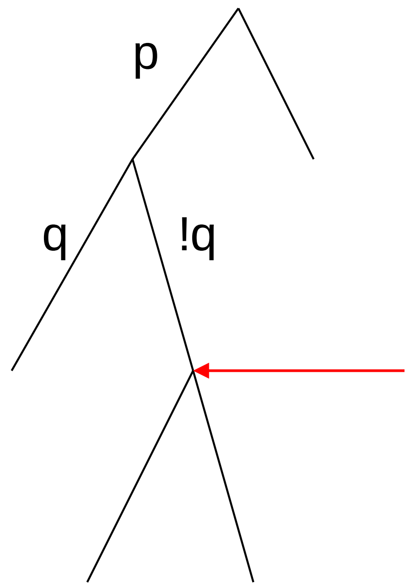
Compute volume / solution density

- Given an SMT formula (a set of constraints), compute the volume of its solution space (or its solution density).
- Example. $\Phi :=$

$$(((y+3x<1) \rightarrow (30<y)) \vee (x \leq 60)) \wedge ((30<y) \rightarrow \neg(x>3) \wedge (x \leq 60))$$

- High complexity: #p-hard even for a single convex polyhedron

Solution counting [MaLiuZhang 2009]



p: (x>3)
q: (2*x > 5)

Check the #of solutions of:
x>3; 2*x <= 5

Vol. computing for polytopes:
vinci

Estimate the volume of polytopes

- Simple Monte-Carlo algorithm [Liu-Zhang-Zhu 2007]
- *PolyVest* [Ge-Ma-Zhang 2013]

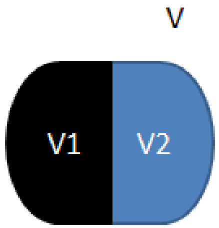
A Testing Problem as a By-product

- How do we know the “reliability” of the method?
- How do we know the accuracy of the results?

Testing PolyVest

Relation r1:

V= V1+V2



Results:

$$Deviation(V,V') = \frac{V' - V}{V}$$

Dimension	Vol(V)	Vol(V1)	Vol(V2)	Vol(V1+V2)	Deviation
5	814.03	573.792	254.765	828.557	+1.78%
8	829.167	323.116	435.406	758.522	-8.52%
14	16961.6	8594.56	8302.33	16896.89	-0.38%
20	1.101e+12	2.412e+11	8.332e+11	1.074e+12	-2..45%

Testing PolyVest

Relation r2:

V1: Ax <= b
V2: Ax <= nb
Dimension of x is m

Volume relation: V2= V1 * n^m

Results:

$$Deviation(V,V') = \frac{V' - V}{V}$$

Dimension	Vol(V1)	n	n^m	expected	Vol(V2)	Deviation
5	848.758	5	3125	2.652e+06	2.620e+06	-1.21%
8	815.157	3	6561	5.348e+06	5.259e+06	-1.66%
14	15631.2	3	4.782e+06	7.485e+010	7.585e+10	+1.34%
20	1.048e+12	2	1.049e+06	1.099e+18	1.135e+18	+3.28%

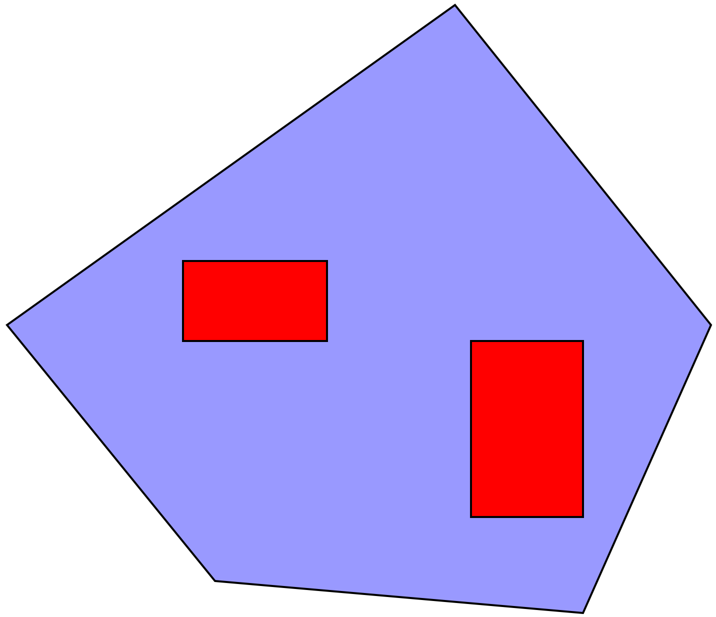
Constraint Solving and Symbolic Execution [Zhang VSTTE2005]

- Verification
- Static analysis
- Testing

"We can also perform **other kinds of analysis** which are not so related to the correctness of programs." (page 544)

Branch/Path Execution Frequency Computation

Symbolic Execution



Checking **areas** (rather than **points**) in the input space.
What are the sizes of the areas? How much is covered?

Branch selection--Example

```
int x;
@ ((x <= 100) && (x > 20))
{
  x = x - 10;
  if (x > 30)
     ... //TRUE branch
  else
     ... //FALSE branch
}
```

- TRUE branch
  75% (3/4)
- FALSE branch
  25% (1/4)

Constraints:

- (a <=100)&&(a > 20)
  (a-10 > 30)
- (a <=100)&&(a > 20)
  (a-10 <= 30)

```
int x;
@ ((x <= 50) && (x > 20))
{
  x = x - 10;
  if (x > 30)
    ... //TRUE branch
  else
    ... //FALSE branch
}
```

- TRUE branch: 1/3
- FALSE branch: 2/3

Constraints:

- (a <= 50)&&(a > 20)
  (a-10 > 30)
- (a <= 50)&&(a > 20)
  (a-10 <= 30)

Path execution frequency -- Example.

```
int getop(s,lim)
    char s[];    int  lim;
{
    int i, c;
    while ((c=getchar()) == ' ' || c == '\t' || c == '\n')     ;
    if (c != '.' && (c < '0' || c > '9'))    return(c);
    s[0] = c;
    for(i = 1; (c=getchar()) >= '0' && c <= '9'; i++)
        if (i < lim)     s[i] = c;
    if (c == '.') {      if (i < lim)    return(c);     …    }
}
```

- Path 1

  $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 27.$

  XP(Path1) ≈ 0.945

- Path 2

  $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 11 \rightarrow 19 \rightarrow 20 \rightarrow 27.$

  XP(Path2) ≈ 0.021

Performance Estimation Based on Symbolic Execution & Volume Computing

Estimating A Program's Performance

- *δ(P)* -- the number of solutions of the path condition (→ path exec. frequency).
- Symbolic benchmarking
  - Generate some paths;
  - Calculate the performance of each path -- PIND(Pi)
  - Estimate Performance of the program:

$$PIND(P1)^* \, \delta(P1) + PIND(P2)^* \, \delta(P2) + \ldots$$

Example. Bubble sort

```
for(i = 0; i < N-1; i++)
    for(j = N-1; j > i; j--) {
        if (a[j-1] > a[j])
            swap(a[j-1], a[j]);
    }
```

Analysis of bubble sort

N=4: 24 paths. (δ: ~ the same for each path)

- For each path, the number of comparisons is the same (6).
- But the number of swaps is different, ranging from 0 to 6. The total number of swaps for all the 24 paths is 72.
- **On average** (when N=4), it needs 3 swaps of array elements and 6 comparisons between array elements.

Example. FIND [Hoare 1971]

Input: array A; size N; int f;

Output:

A[0], A[1], ..., A[f-1]

<= A[f]

<= A[f+1], ..., A[N-1].

benchmarking -- I

- Randomly choose some paths

(N=8, f=3)

Path	nComp	nSwap	Delta(P)
w11	9	3	16303680
w16	9	2	12191040
w18	9	3	16303680
w20	9	2	12191040

benchmarking -- II

Randomly generate some input data

- { -2, 5, 6, 3, 1, 0, -7, 6 };
- { 2, 0, -2, -8, 4, -4, 5, 1 };

Path	nSwap	Delta(P)
R1	4	4075920
R6	6	87516

- Average #of swaps: 4.04

$$(4075920*4 + 87516*6) / (4075920+87516) \approx 4.04$$

Symbolic Benchmarking

- Symb. execution → symb. benchmarking
  - one symb. exec. == many conc. exec.

- Given the program
  - generate paths from the flow graph; or run the program a number of times
  - calculate the PIND and Delta values
  - get the estimated performance of the prog. (weighted sum of the PINDs)

Reliability of
Component-Based Software Systems

Reliability of Component-based Syst.

- reliability of components → reliability of systems ?
  - execution frequency calculation
  - …

- Existing Works:

  [Hamlet-Mason-Woit 2001] Theory of software reliability based on components
  - System design – control structures: sequences, conditionals, loops, …
  - ("A supporting tool would …")

- Existing Works:

  [Palviainen, Evesti, Ovaska 2011] The reliability estimation, prediction and measuring of component-based software

  - Model-based reliability prediction/measuring
  - State transition matrix – each element of the matrix P*sa* denotes the probability of moving from state *s* to state *a*.
  - A tool chain …

Summary

- All kinds of constraints
  - Linear/non-linear inequalities
  - SAT ---- (p or not q) and (not p or q)
  - SMT ---- (x+y >= 3) and good and (x-y != 8)

  - Decision problem → counting, optimization

- Testing, analysis, reliability, security …

Acknowledgement

- Jun Yan
- Feifei Ma
- Zhongxing Xu
- Zhenbo Xu
- Zhiqiang Zhang
- Tianyong Wu
- Xingming Wu
- Cunjing Ge
- Chaorong Guo
- Yanli Zhang

THANK YOU !