Automated Test Case Generation

or: How to not write test cases

Stefan Klikovits

EN-ICE-SCD
Université de Genève

28^{th} September, 2015

Reminder: Testing is not easy

Credit: *HBO*

Overview: Automated Testing

Automated . . .

- test execution
  - setup
  - program execution
  - capture results
- result checking
- reporting

Overview: Automated Testing

Automated . . .

- test input generation
- test selection
- test execution
- results generation (oracle problem)
- result checking
- reporting

Random Testing

Random testing

- input domains form regions [8]
- input represents the region around it
- maximum coverage through maximum diversity [2]
- but: random input is. . . well, random!

Adaptive Random Testing

NON-Random random testing (?!)

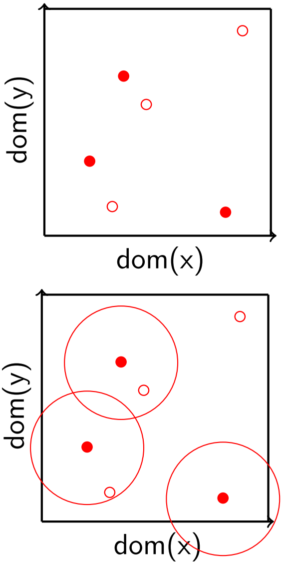
Credit: *https://sbloom2.wordpress.com/category/evaluations/*

Adaptive Random Testing

NON-Random random testing (?!)

- evaluate previous TCs before generating a new one
- choose one that is as different as possible
- various strategies [1]

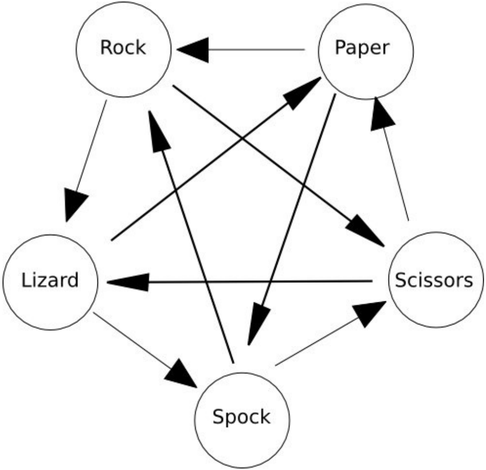
ART strategies



Criticism

- non-determinism
- input data problems (e.g. ordering in discrete domains)
- computationally expensive (time, memory)
- unrealistic scenarios [2]
  - too high defect rates
  - no actual SUT

Combinatorial Testing



Combinatorial Testing

- Idea: test all possible input (combinations)
- large number of TCs
- (slight) improvement: *Equivalence classes*!
  - $(5 < \text{uint a} < 10) \Rightarrow \{[0..5], [6..9], [10..\text{maxInt}]\}$
- still large TC sets
  - 5 parameters – 3 EC each $\Rightarrow$ 243 TC
  - plus *boundary values*, exceptions, etc.

Orthogonal/Covering Arrays

Orthogonal arrays (OA)

- test each pair/triple/... of parameters
- restriction: every τ-tuple has to be tested equally often

Covering arrays (CA)

- ...every pair (τ-tuple) has to appear at least once
- logarithmic growth [3]

Working principle

Scenario:

- 3 Parameters (OS, Browser, Printer)
- 2 values each ({W, L}, {FF, CH}, {A, B})

OS	Browser	Printer
W	FF	A
W	FF	B
W	CH	A
W	CH	B
L	FF	A
L	FF	B
L	CH	A
L	CH	B

Table: Pairwise testing

Working principle

Scenario:

- 3 Parameters (OS, Browser, Printer)
- 2 values each ({W, L}, {FF, CH}, {A, B})

OS	Browser	Printer
W	**FF**	**A**
W	FF	B
W	CH	A
W	**CH**	**B**
L	FF	A
L	**FF**	**B**
L	**CH**	**A**
L	CH	B

Table: Pairwise testing

Criticism

- computationally expensive
- NP-hard [3]
- test case prioritisation

Industry measurements:

- 70 % pairwise; 90 % threeway [6]
- 97 % of medical devices with pairwise tests [5]

Examples

Scenario 2:

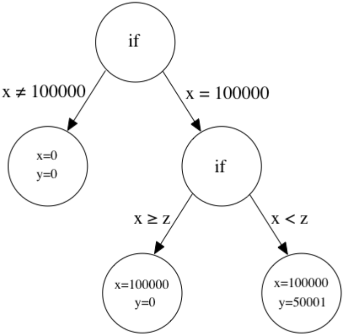
- 4 parameters - 3 values each
- exhaustive tests: $3^4 = 81$
- TCs to cover all pairs: **9**

Examples

Scenario 3:

- 10 parameters - 4 values each
- exhaustive tests: $4^{10} = 1{,}048{,}576$
- TCs to cover all pairs: **29**

Symbolic Execution

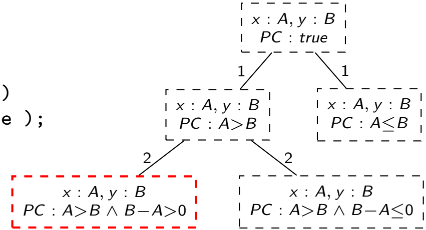


Symbolic execution

- build execution tree
- use symbols as input
- sum up *Path constraints* (PCs)
- use constraint solvers

Example *Execution tree* and *Path constraints* [7]

```
    int x, y;
1:  if ( x > y ) {
2:    if ( y - x > 0 )
3:      assert ( false );
    }
```



Difficult constraints? Concolic Execution!

Idea:

- use symbolic values as long as possible
- switch to real values when necessary

Example:

```
int obscure(int x, int y) {
  if (x == hash(y)) return -1;   // error
  return 0;                      // ok
}
```

Figure: Example of Concolic Execution [4]

Real life application

Whitebox fuzzying [4]

1. start with *well-formed* inputs
2. record all the individual constraints along the execution path
3. one by one negate the constraints, solve with a constraint solver and execute new paths

Properties:

- highly scalable
- focus on *security vulnerabilities* (buffer overflows)
- no need for a *test oracle* (check for system failures & vulnerabilities)

Found one third of all bugs discovered in Windows 7!

Model-based TC generation

Credit: *http://formalmethods.wikia.com/wiki/Centre_for_Applied_Formal_Methods*

- automatic/manual model generation
- three approaches
- Axiomatic | FSM | LTS

Model-based test case generation

TC selection:

- offline/online test selection

Modeling notations (textual & graphical):

- Scenario-, State-, Process-oriented

Criticism

- state space explosion
- complex model generation
- defining a “good” model is non-trivial
- requires knowledge of modeling

Summary

- **(Adaptive) Random Testing** (BB):
  cheap generation; non-deterministic; (hit & miss)
- **Combinatorial Testing** (BB):
  expensive; many TCs
- **Symbolic/Concolic Execution** (WB):
  problematic constraints; path explosion
- **Model-based** (WB)
  not "just" coding; need a "good" model (complex);
  state space

Summary

- **(Adaptive) Random Testing** (BB):
  cheap generation; non-deterministic; (hit & miss)
- **Combinatorial Testing** (BB):
  expensive; many TCs
- **Symbolic/Concolic Execution** (WB):
  problematic constraints; path explosion
- **Model-based** (WB)
  not "just" coding; need a "good" model (complex); state space

Summary

- **(Adaptive) Random Testing** (BB):
  cheap generation; non-deterministic; (hit & miss)
- **Combinatorial Testing** (BB):
  expensive; many TCs
- **Symbolic/Concolic Execution** (WB):
  problematic constraints; path explosion
- **Model-based** (WB)
  not "just" coding; need a "good" model (complex); state space

Summary

- **(Adaptive) Random Testing** (BB):
  cheap generation; non-deterministic; (hit & miss)
- **Combinatorial Testing** (BB):
  expensive; many TCs
- **Symbolic/Concolic Execution** (WB):
  problematic constraints; path explosion
- **Model-based** (WB)
  not "just" coding; need a "good" model (complex); state space

References

[1] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, and Phil Mcminn.
An orchestrated survey of methodologies for automated software test case generation.
J. Syst. Softw., 86(8):1978–2001, August 2013.

[2] Andrea Arcuri and Lionel C. Briand.
Adaptive random testing: an illusion of effectiveness?
In *ISSTA*, pages 265–275, 2011.

[3] Charles J. Colbourn.
Combinatorial aspects of covering arrays.
Le Matematiche (Catania), 58, 2004.

References (cont.)

[4] Patrice Godefroid.
Test generation using symbolic execution.
In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 24–33. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[5] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, Jr.
Software fault interactions and implications for software testing.
IEEE Trans. Softw. Eng., 30(6):418–421, June 2004.

[6] D. Richard Kuhn and Michael J. Reilly.
An investigation of the applicability of design of experiments to software testing.
In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, SEW '02,

References (cont.)

pages 91–, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Corina S. Pasareanu and Willem Visser.
A survey of new trends in symbolic execution for software testing and analysis.
STTT, 11(4):339–353, 2009.

[8] L.J. White and E.I. Cohen.
A domain strategy for computer program testing.
IEEE Transactions on Software Engineering, 6(3):247–257, 1980.