Retrospective: Feedback-directed Random Test Generation

Carlos Pacheco, Shuvendu Lahiri, **Michael D. Ernst**, Thomas Ball

ICSE 2007 MIP retrospective

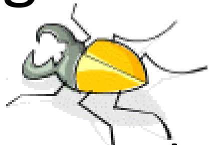
May 26, 2017

Who loves to write tests?

Problem:

- Developers do not love to write tests
- There are not enough tests

Solution:

- Automatically generate tests
- Randoop tool 
  - https://randoop.github.io/randoop/

What is a test?

A test consists of

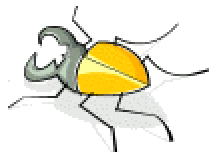
- an input
- an oracle

End-to-end test:

- Batch program: input = file, oracle = expected file
- Interactive program: input = UI events, oracle = windows

Unit test:

- Input = sequence of calls
- Oracle = `assert` statement



Example unit test

```
Object[] a = new Object[];
LinkedList ll = new LinkedList();
ll.addFirst(a);
TreeSet ts = new TreeSet(ll);
Set u = Collections.unmodifiableSet(ts);
```

input

```
assert u.equals(u);
```

oracle

Assertion fails: bug in JDK!

Automatically generated test

- Code under test:

```
public class FilterIterator implements Iterator {
  public FilterIterator(Iterator i, Predicate p) {…}
  public Object next() {…}
  …
}
```

```
/** @throws NullPointerException if either
 * the iterator or predicate are null */
```

- Automatically generated test:

```
public void test() {
  FilterIterator i = new FilterIterator(null, null);
  i.next();
}
```

Throws NullPointerException!

Did the tool discover a bug?

It could be:

1. Expected behavior
2. Illegal input
3. Implementation bug

“Test classification” problem

Challenge: classifying tests

- Without a specification, the tool **guesses** whether a given behavior is **correct**
  - False positives: report a failing test that was due to illegal inputs
  - False negatives: fail to report a failing test because it might have been due to illegal inputs

Test classification is useful for:

- Oracles: A test generation tool outputs:
  - Failing tests – indicates a program bug
  - Passing tests – useful for regression testing
- Inputs: A test generation tool creates input incrementally
  - Should only build on good tests

Example unit test

Previously created

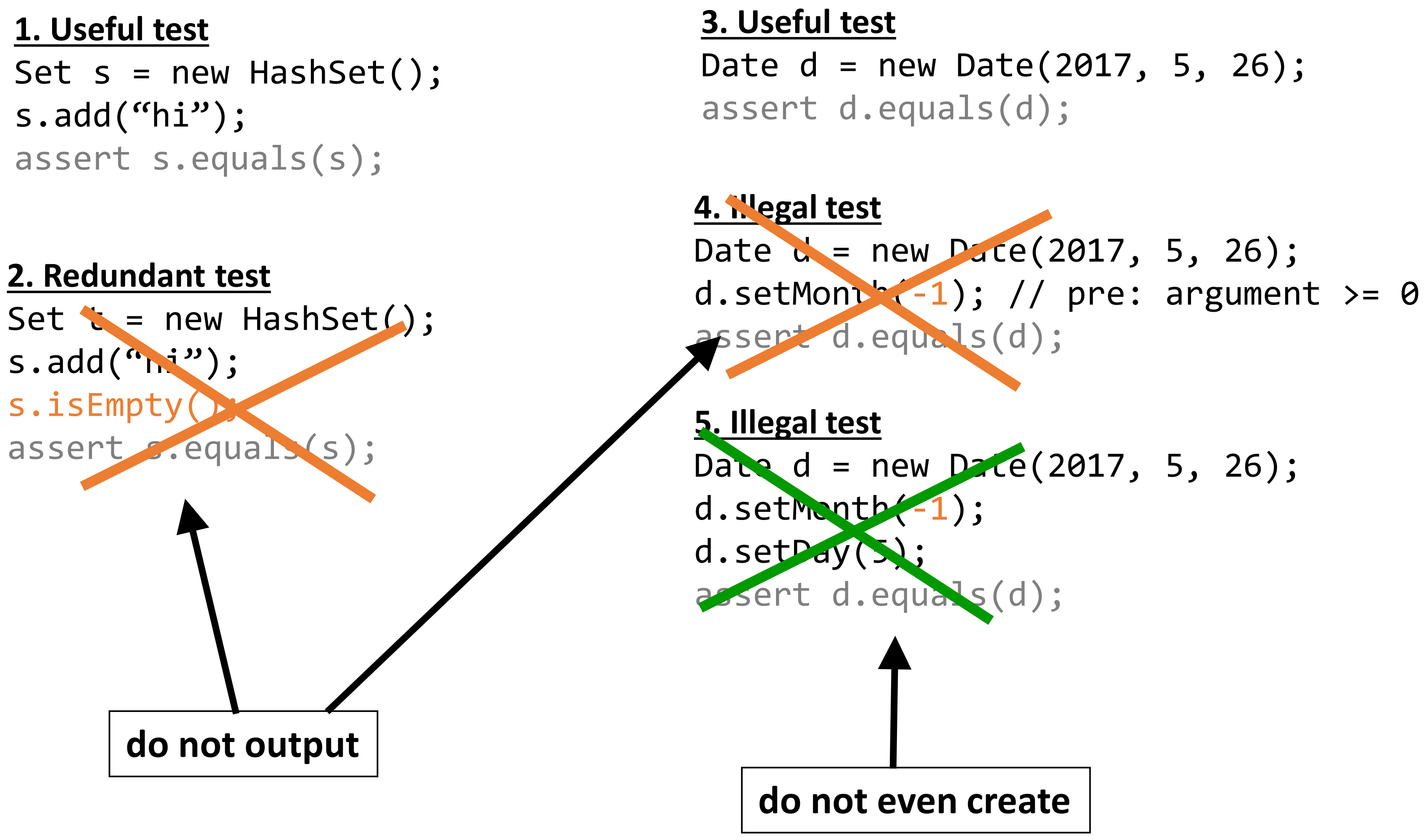
```
Object[] a = new Object[];
LinkedList ll = new LinkedList();
ll.addFirst(a);
TreeSet ts = new TreeSet(ll);
Set u = Collections.unmodifiableSet(ts);
```

input

```
assert u.equals(u);
```

oracle

Pitfalls when extending a test input



1. Useful test

```
Set s = new HashSet();
s.add("hi");
assert s.equals(s);
```

2. Redundant test

```
Set t = new HashSet();
s.add("hi");
s.isEmpty();
assert s.equals(s);
```

3. Useful test

```
Date d = new Date(2017, 5, 26);
assert d.equals(d);
```

4. Illegal test

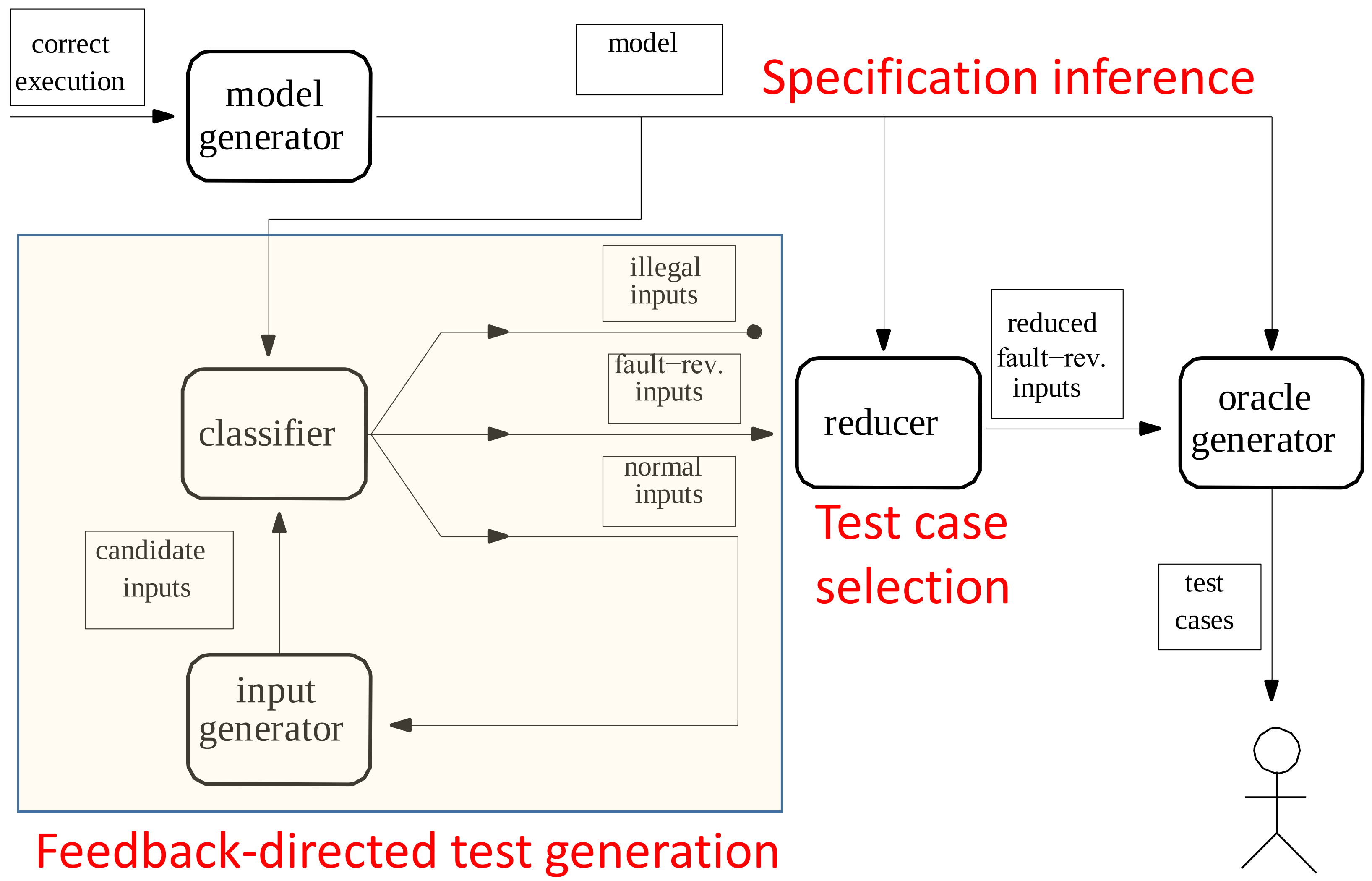
```
Date d = new Date(2017, 5, 26);
d.setMonth(-1); // pre: argument >= 0
assert d.equals(d);
```

5. Illegal test

```
Date d = new Date(2017, 5, 26);
d.setMonth(-1);
d.setDay(5);
assert d.equals(d);
```

Feedback-directed test generation

"Eclat: Automatic generation and classification of test inputs", by Carlos Pacheco and Michael D. Ernst. *ECOOP 2005.*



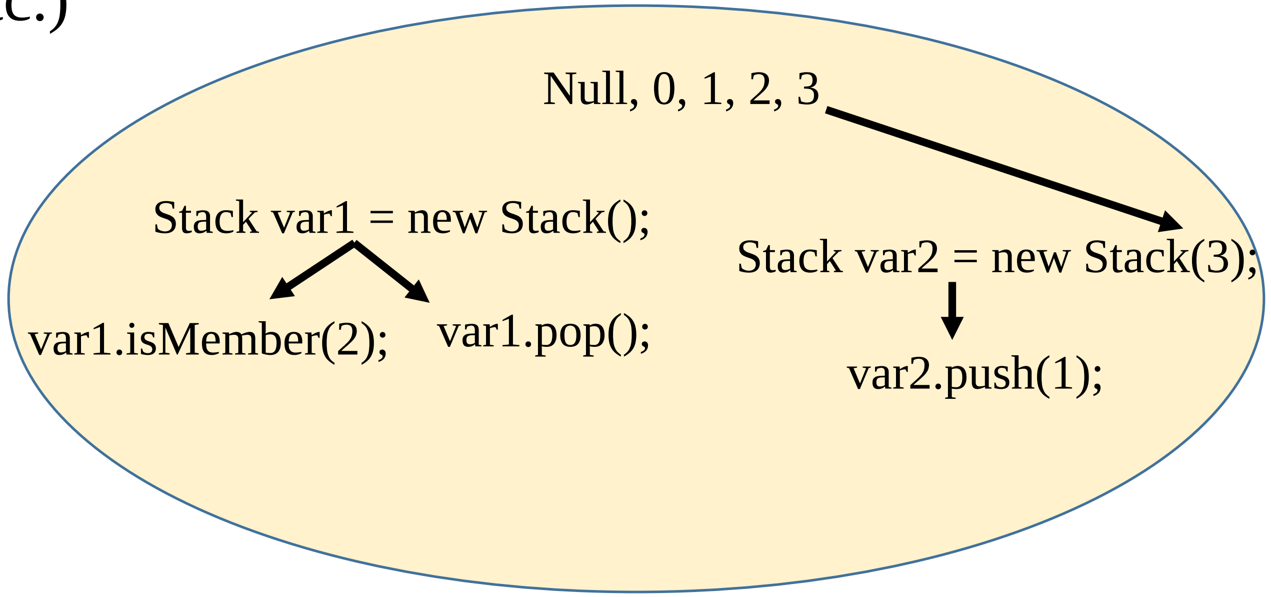
Classifying test behavior

Satisfies precondition?	Satisfies postcondition?	Classification
Yes	Yes	Normal
Yes	No	Fault
No	Yes	Normal (new*)
No	No	Illegal

* For Eclat: outside the domain of existing tests; feedback to test generator
For Randoop: outside the domain of the specification

Test input generator (no oracle yet)

```
1. pool :=  a set of primitives (null, 0, 1, etc.)
2. do N times:
   2.1. create new inputs
        by calling methods/constructors
        using pool values as arguments
   2.2. run the input
   2.3. classify inputs
     2.3.1.  throw away illegal inputs
     2.3.2.  save away fault inputs
     2.3.3.  add normal inputs to the pool
```



Randoop vs. Eclat

Implementations:
1. Eclat
2. Joe
3. Randoop.NET
4. Randoop for Java
   (dozens of releases)

- Test inputs:
  - Randoop: dozens of enhancements:
    richer search space, prune redundancies, …
- Oracles (specifications, assertions):
  - Eclat: generates
  - Randoop: hard-coded library specifications
- Tool output:
  - Eclat: error-revealing tests
  - Randoop: error-revealing tests and regression tests
- Evaluation:
  - Eclat: precision of oracles; code coverage; a few errors revealed
  - Randoop: many errors in real-world programs; outperforms existing techniques

“Feedback-directed Random Test Generation”

- ✓ Feedback-directed
- ❏ Random

Random testing: Obviously a bad idea

- No guarantees about fault detection, coverage

  Systematic techniques give no guarantees

- Cannot cover simple code

  Only 1 in 2^{64} chance to find the crash in:

  ```
  void foo(long x) {
    if (x == 0xBADC0DE)
      crash();
  }
  ```

  Random ≠ black-box

- Many publications show it is inferior [Ferguson 1996, Marinov 2003, Visser 2006, …]

  Small benchmarks, wrong measurements, strawman implementations

- Not complex enough to merit publication

  Say “stochastic” instead of “random”

Arguments in favor of random testing

- Simple to implement
- Fast: generate lots of tests, big tests, many behaviors
- Scalable: works on real programs
- In theory, about as effective as systematic testing [Duran 1984, Hamlet 1990]
- In practice, highly effective

- Randoop chose random because it was the most practical choice
  - I would choose random again today
  - "Feedback-directed unit test generation for C/C++ using concolic execution" [Garg 2013]

Other/better test generation approaches

- Manual test generators: QuickCheck [Claessen 2000]
- Exhausive (model checking): Korat [Boyapati 2002]
- Concolic (concrete + symbolic): DART [Godefroid 2005], CUTE [Sen 2005]
- Symbolic (constraint solving): Klee [Cadar 2008]

- Satisfy input constraints: Csmith [Eide 2008]
- Input similarity metric: ARTOO [Ciupa 2008]
- Search-based: Genetic algorithms EvoSuite [Fraser 2011], MaJiCKe [Jia 2015]
- Better guidance: GRT [Ma 2015]

Randoop evaluation

- Found errors in test program used by 3 previous papers
- Better coverage than systematic techniques
    - on programs they chose for evaluation
- > 200 distinct defects in .NET framework and JDK
    - Other tools did not scale to this code

(Shuvendu will discuss the evaluation further.)

What Randoop is bad at

- Entire programs (some progress: [Robinson 2011])
- Requires tuning
- Tends to get stuck
- Complex, specific inputs
    - Protocols -- make calls in specific order (e.g., database connections)
    - Strings
    - Complex objects
- Tests can be hard to understand
- Focused generation: Top-down vs. bottom-up generation

Still outperforms other techniques and tools.

Perspective

- Why was Randoop successful?
- Advice about your research

How to evaluate a technique

- Your technique is probably better, but show it honestly
- Scientific goal is to evaluate *techniques*, not *tools*
  - Implement every optimization or heuristic for all techniques
  - Avoids confounding factors
  - Enables fair comparison of systematic, symbolic, and random search
  - Evaluate the optimization or heuristic in multiple contexts
- Random approaches are a common whipping boy or strawman
  - It is no surprise and no achievement to beat a dumb implementation

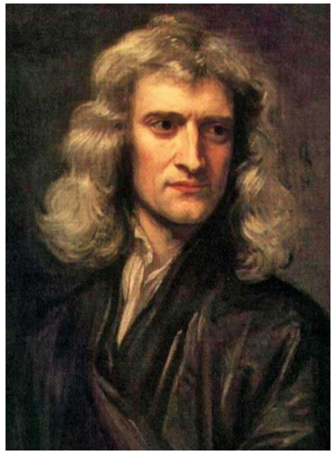
When evaluating an existing tool

- Don't misuse the tool
  - Example: tune one tool or provide it extra information

- Read the manual (Randoop manual offers specific advice)
- Use command-line options (Randoop has 57!)
- Report bugs

Scientific progress requires reproducibility

- Make your work publicly available
  - tool, evaluation scripts & inputs, and outputs
- Extra effort: robust and easy to use, *beyond* the experiments in the paper
- Some people choose to prioritize other factors
  - Money, reputation, scientific advantage, number of publications
- If you prioritize other factors and keep your data secret, you are not fully acting like a scientist



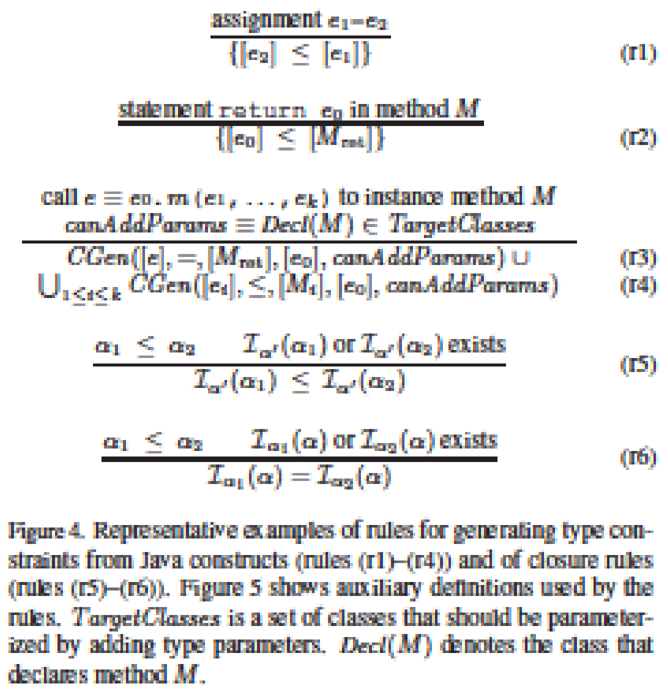
"If I have seen further, it is by standing on the shoulders of giants.“ Isaac Newton, 1676.

Maintain your artifacts

- Other people can compare to, and build on the work
  - Other people can disparage the work or scoop you
  - Distracts from other research
- 10 years later, I still maintain Randoop
  - Bug fixes, new features
  - On average, 1 release per month (version 4 next month)
  - Against the advice of some faculty
- Essential for scientific progress
- Poorly rewarded by the scientific community
  - Pursuing the shiny new thing
  - Valuing novelty over effectiveness
  - Valuing number of papers over scientific value and impact

Don’t give up

- My papers were rejected before being accepted
  - … and became better as a result
  - A paper rejection is a gift
- Eclat paper had limited impact
- ICSE 2007 recognized the value of my work!
  - ACM Distinguished Paper Award
  - Time (and more work!)
    can change people’s opinions
    about what has most impact



$$\frac{\text{assignment } e_1 = e_2}{\{[e_2] \le [e_1]\}} \quad \text{(r1)}$$

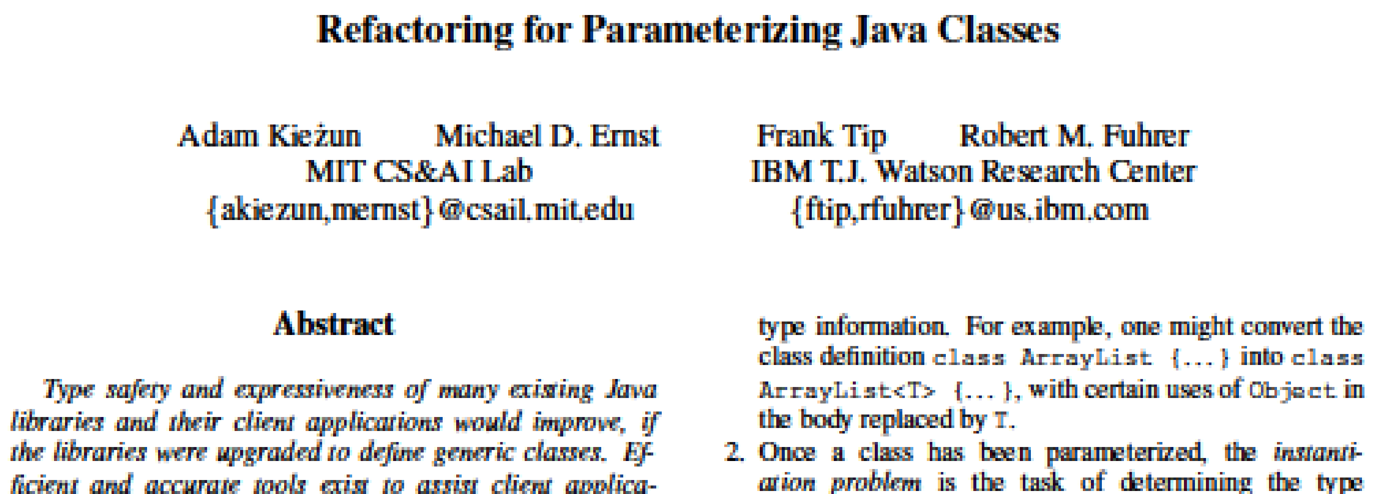
$$\frac{\text{statement } \texttt{return } e_0 \text{ in method } M}{\{[e_0] \le [M_{\text{ret}}]\}} \quad \text{(r2)}$$

$$\frac{\begin{array}{c}\text{call } e \equiv e_0.m(e_1, \ldots, e_k) \text{ to instance method } M \\ canAddParams \equiv Decl(M) \in TargetClasses\end{array}}{\begin{array}{c}CGen([e], =, [M_{\text{ret}}], [e_0], canAddParams) \cup \\ \bigcup_{1 \le i \le k} CGen([e_i], \le, [M_i], [e_0], canAddParams)\end{array}} \quad \begin{array}{c}\text{(r3)}\\ \text{(r4)}\end{array}$$

$$\frac{\alpha_1 \le \alpha_2 \qquad \mathcal{I}_{\alpha'}(\alpha_1) \text{ or } \mathcal{I}_{\alpha'}(\alpha_2) \text{ exists}}{\mathcal{I}_{\alpha'}(\alpha_1) \le \mathcal{I}_{\alpha'}(\alpha_2)} \quad \text{(r5)}$$

$$\frac{\alpha_1 \le \alpha_2 \qquad \mathcal{I}_{\alpha_1}(\alpha) \text{ or } \mathcal{I}_{\alpha_2}(\alpha) \text{ exists}}{\mathcal{I}_{\alpha_1}(\alpha) = \mathcal{I}_{\alpha_2}(\alpha)} \quad \text{(r6)}$$

Figure 4. Representative examples of rules for generating type constraints from Java constructs (rules (r1)–(r4)) and of closure rules (rules (r5)–(r6)). Figure 5 shows auxiliary definitions used by the rules. *TargetClasses* is a set of classes that should be parameterized by adding type parameters. *Decl*(M) denotes the class that declares method M.



Refactoring for Parameterizing Java Classes

Adam Kieżun   Michael D. Ernst
MIT CS&AI Lab
{akiezun,mernst}@csail.mit.edu

Frank Tip   Robert M. Fuhrer
IBM T.J. Watson Research Center
{ftip,rfuhrer}@us.ibm.com

Abstract

Type safety and expressiveness of many existing Java libraries and their client applications would improve, if the libraries were upgraded to define generic classes. Efficient and accurate tools exist to assist client applica-

type information. For example, one might convert the class definition `class ArrayList {...}` into `class ArrayList<T> {...}`, with certain uses of `Object` in the body replaced by `T`.

2. Once a class has been parameterized, the *instantiation problem* is the task of determining the type

We need results, not ideas

Arguments in favor of ideas:

- An imaginative contribution
- Shows connections between areas
- Sparks yet more ideas
- Proposes work for other people to do
- Recognition on CV

Arguments in favor of results:

- Most ideas are worthless
- It’s easy to make up a persuasive argument
- If you aren’t willing to do the work, do you believe in your idea?
- Poor evaluation may be misleading
- Idea papers reward shallow work, inhibit subsequent publication

Your work should be actionable

Implement your idea

- Enables evaluation
    - Essential for understanding the technique
    - Essential for evaluating the technique
    - Essential for evaluating usefulness
    - Always yields surprises (ABB for detouring, Microsoft for discarded tests, …)
- Helps the whole field
    - Others can build on it
    - Others are inspired to do better
    - Enables comparisons

Evaluation: the most important part of a paper

- Don't just show *success*, show *improvement*
  - Requires comparison to previous techniques
  - Requires that previous tools exist or are re-implemented
- Evaluate the whole task, not just part of it
  - Misleading to claim big improvement on a trivial part of the problem
- Measure the right metrics
  - For testing, *not* coverage or mutant kill score
  - Use real defects, such as Defects4J [Just 2014] or CoREBench [Boehme 2014]
- Involve the user
  - Case studies can be more appropriate than controlled experiments
  - Gold standard: real-world use
- What are the most important aspects to be realistic?
- Won't realistic evaluations slow down science?
  - Science is about truth and results, not ideas or publications

Test generation: quality over quantity

- It’s easy to produce a lot of tests
- Previous work (Jov, Jcrasher) produced mostly illegal tests
  - Example: illegal inputs lead to crashes
- We examined the tests: what would a user do?
- Randoop was willing to discard some tests
- Quality metric: reveal real defects
  - Count defects, not failures.
  - Don't be discouraged if the maintainers won't fix them.

Ideas: quality over quantity

- Aimed for simple ideas, concisely explained
- Easy to understand, reproduce, refute
- The best papers have simple ideas
- Simple ideas are harder to produce than complex ones

Automation: quality over quantity

- Human is expected to
  - Examine failures
  - Provide input/guidance to Randoop

- Cooperation between human and machine
  - Each does the tasks it is best suited to

Publications: quality over quantity

- Only 3 Randoop publications
- Despite 15 years of work

Randoop is still finding bugs

- This month, 60 bugs in Apache Commons Math (and many others)
- Randoop remains the easiest to use and best test generator
- There's no good reason not to run Randoop on your program
- Try it today:
  - C#: https://github.com/abb-iss/Randoop.NET
  - Java: https://randoop.github.io/randoop/