Feedback-controlled Random Test Generation

Kohsuke Yatoh[1*], Kazunori Sakamoto[2], Fuyuki Ishikawa[2], Shinichi Honiden[12]

1:University of Tokyo, 2:National Institute of Informatics

* He is currently affiliated with Google Inc., Japan. All work is done in Univ. Tokyo and nothing to do with Google.

My First Motivation

Software testing

- Very important
- Tedious, labor-intensive and error-prone

I want someone ELSE to write tests for me!

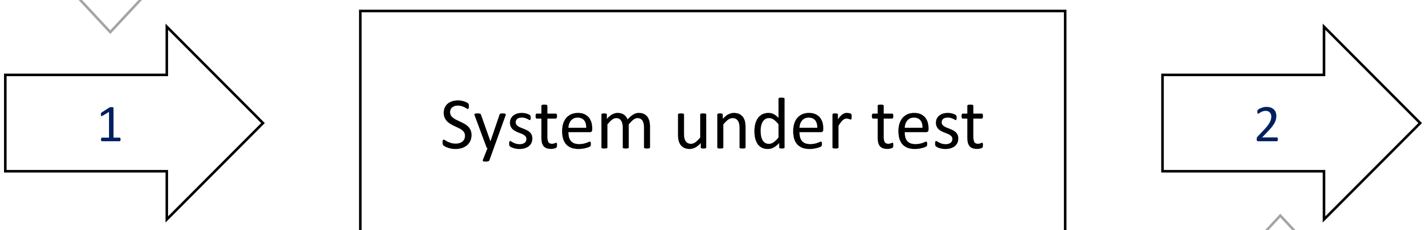
→ Automatic Test Generation

Two Sides of Automated Test Generation

This paper

1. Input generation (data)

Generating interesting test data



2. Output verification (assertions)

Oracles – specifications, domain specific knowledge

Background

Feedback-directed random test generation (FDRT) [Pacheco.07]

Random test generation for OOP languages

Usage

- Test by contracts [Pacheco.07]
- Regression test gen. [Robinson.11]
- Specification mining [Pradel.12]
- Test by property [Yatoh.14]
- Combination with other automated test generation [Garg.13, Zhang.14]

Example

Input: Class list

```
class AddressBook {
 AddressBook(int capacity) {
  assert capacity >= 0;
  …
 }
 void add(Person person) {…}
}


class Person {
 Person(String name) {
  assert name != null;
  …
 }
}
```

Output: Method sequences

```
AddressBook a1 =
 new AddressBook(10);

Person p1 =
 new Person("foo");

a1.add(p1);

//AddressBook a2 =
// new AddressBook(-1);

//Person p2 =
// new Person(null);

Person p3 =
 new Person("bar");

a1.add(p3);

a1.add(p1);
```

FDRT Pros & Cons

Good — Applicable to wider range of SUT
than other methods like symbolic execution

Bad — Coverage of generated tests are low and unstable
→ less possibility to detect faults

Our Contributions

1. Analyzed characteristics of FDRT
   and found one cause of low and unstable coverage
2. Proposed a new method to mitigate the low coverage
   (Feedback-controlled Random Test Generation)
   → **2x - 3x coverage** for utility libraries

FDRT Algorithm

Classes Under Test

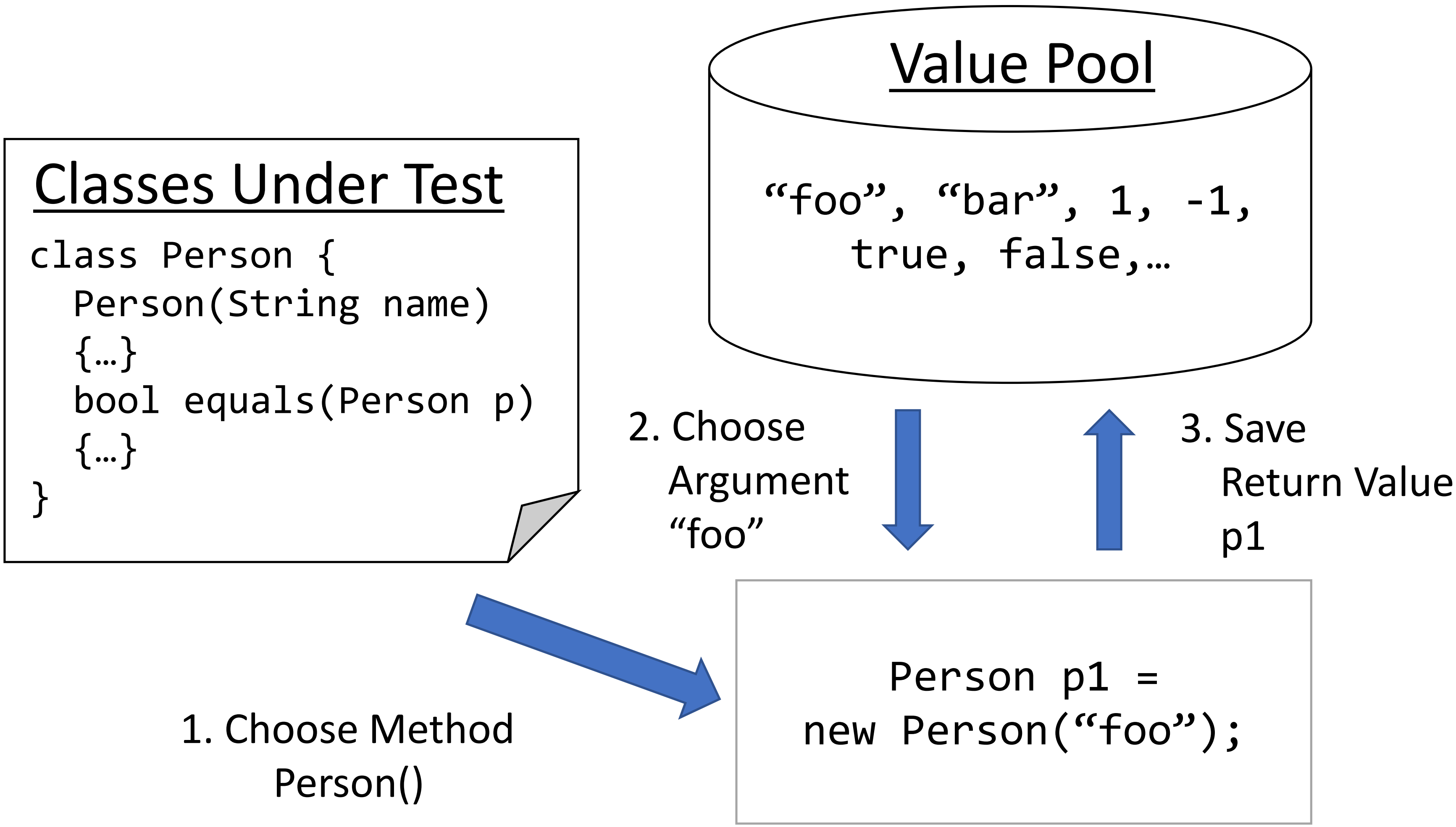
```
class Person {
  Person(String name)
  {…}
  bool equals(Person p)
  {…}
}
```

Value Pool

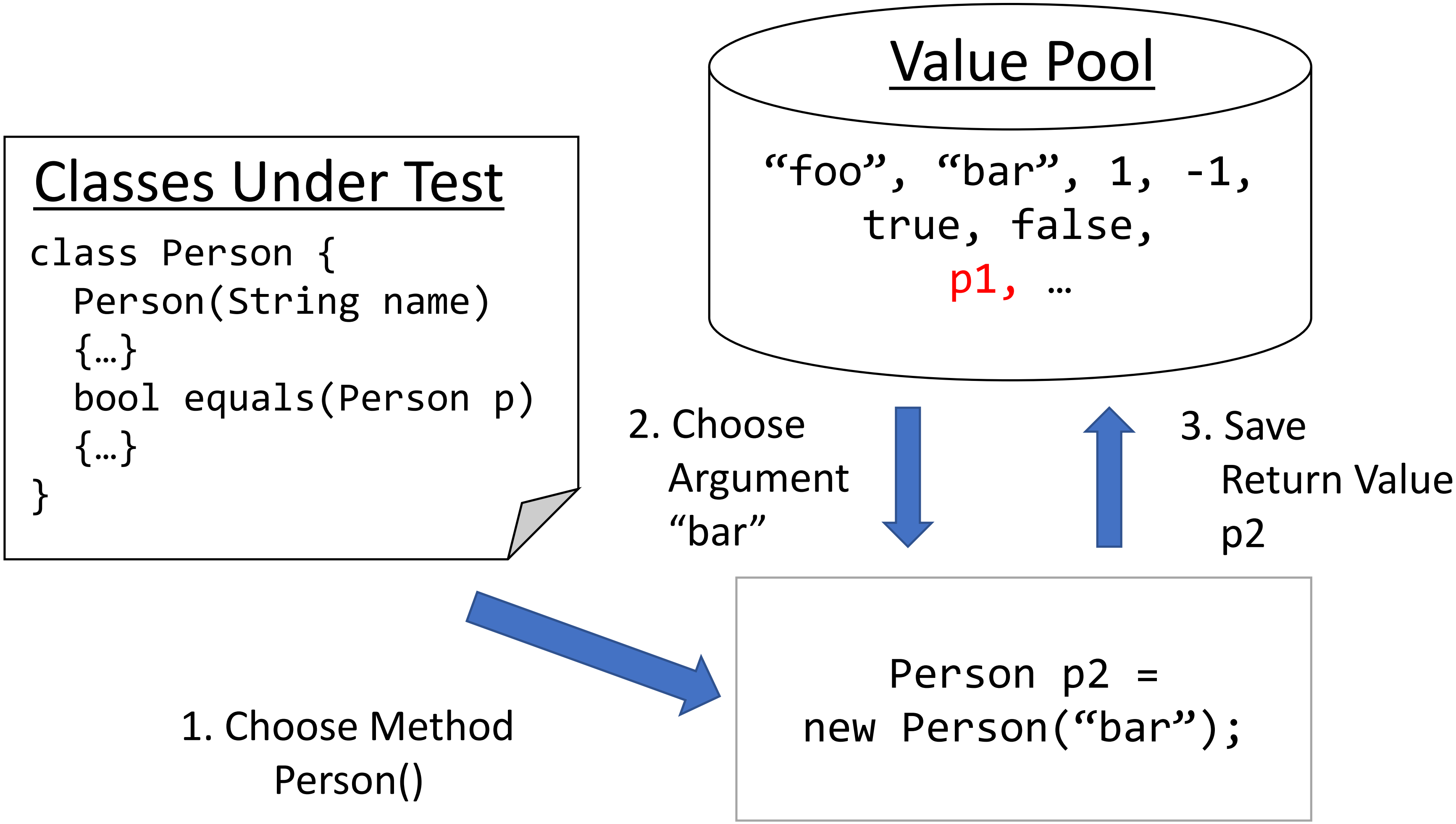
“foo”, “bar”, 1, -1,
true, false,…

Pool of Candidate Arguments
(Initialized with random primitives)

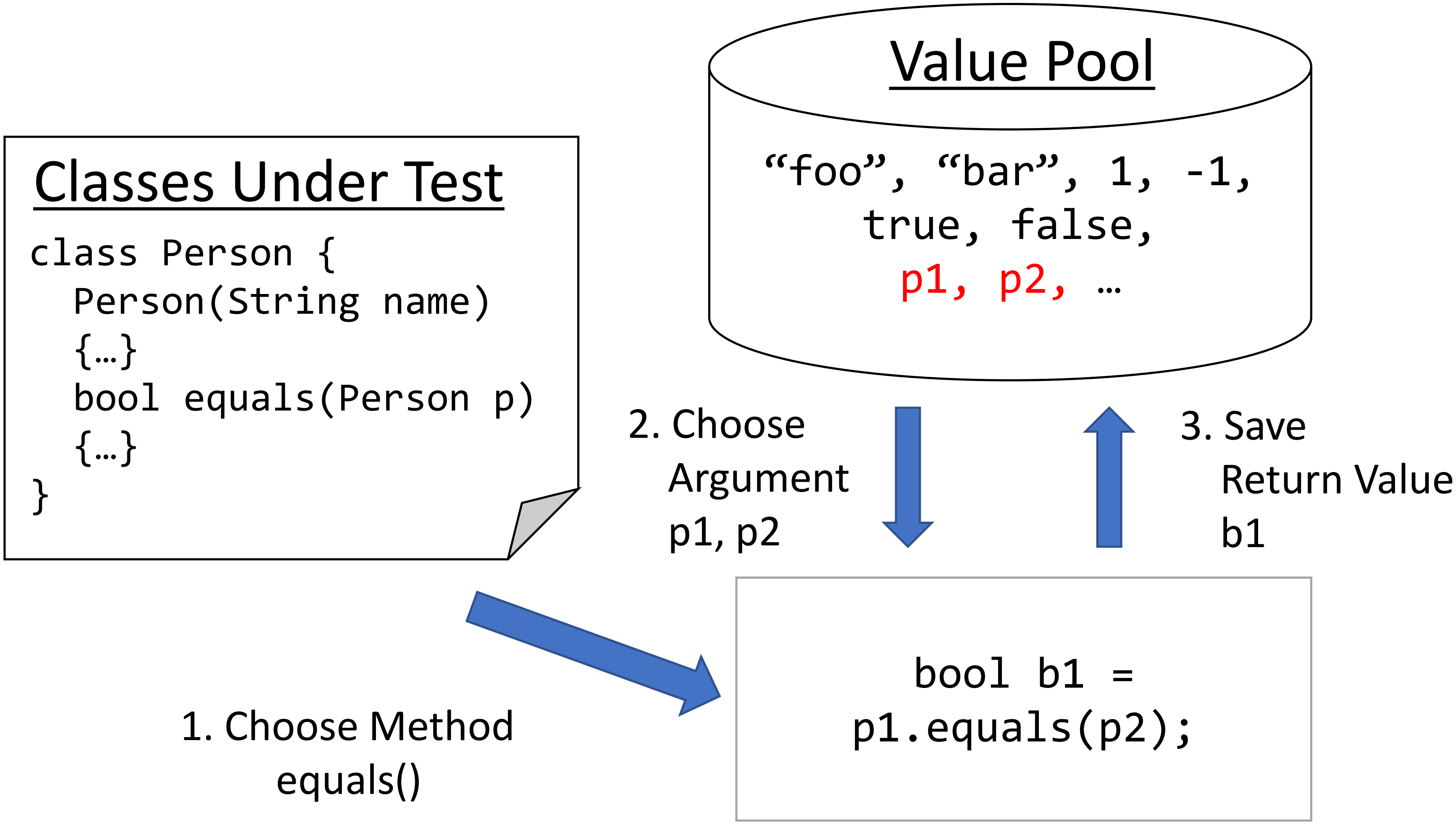
FDRT Algorithm



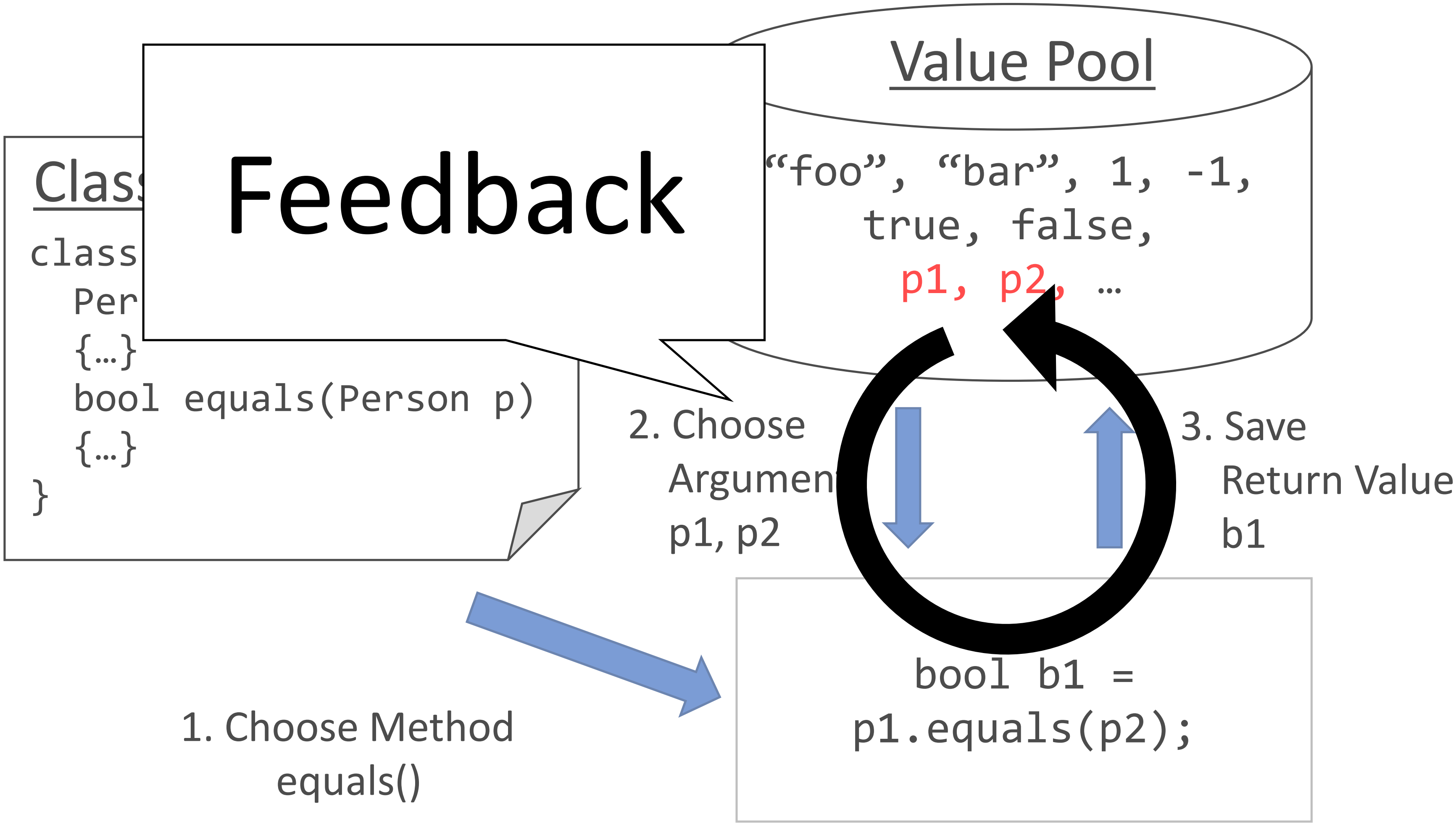
FDRT Algorithm



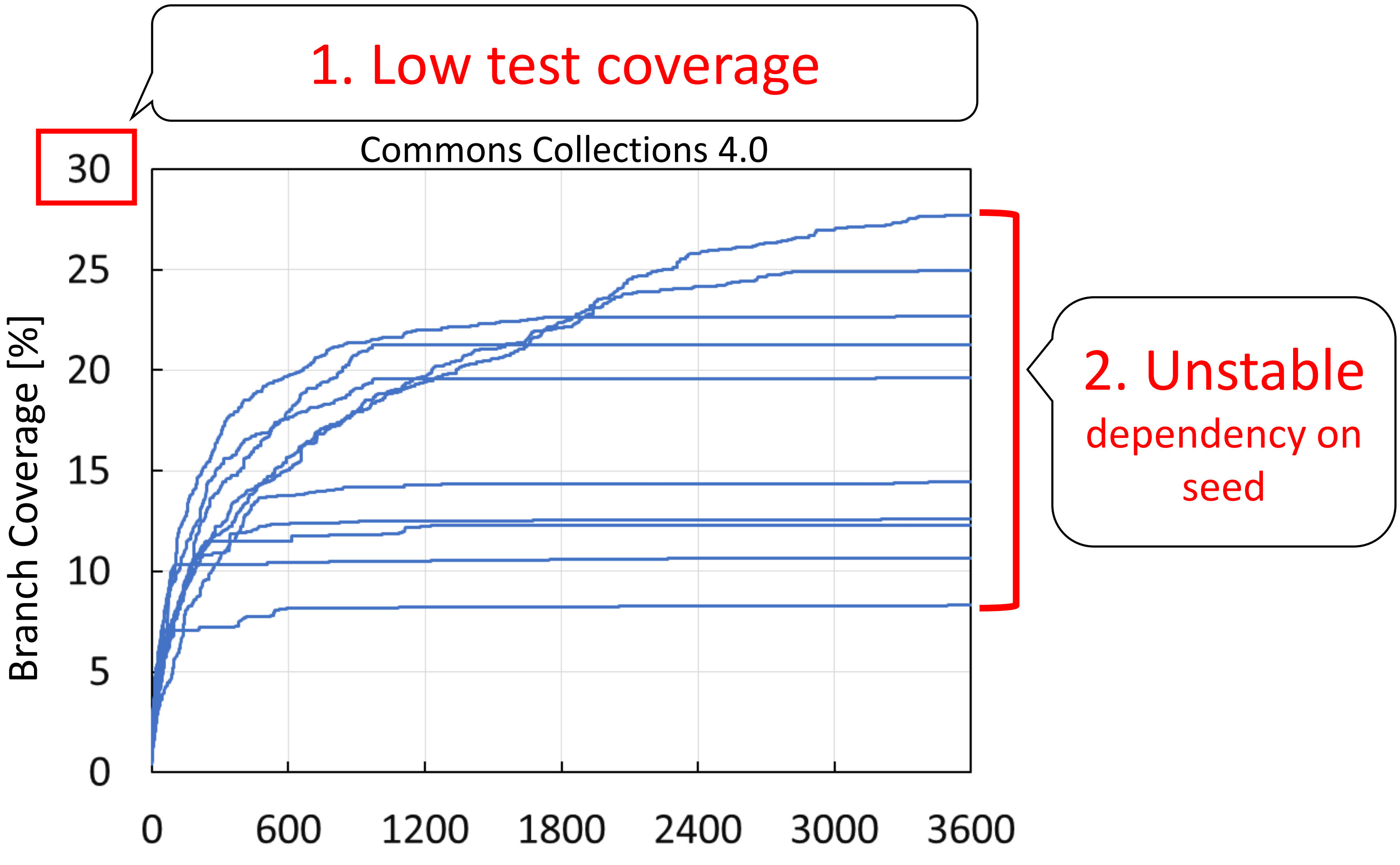
FDRT Algorithm



FDRT Algorithm



Problems When Applying to Real Libraries



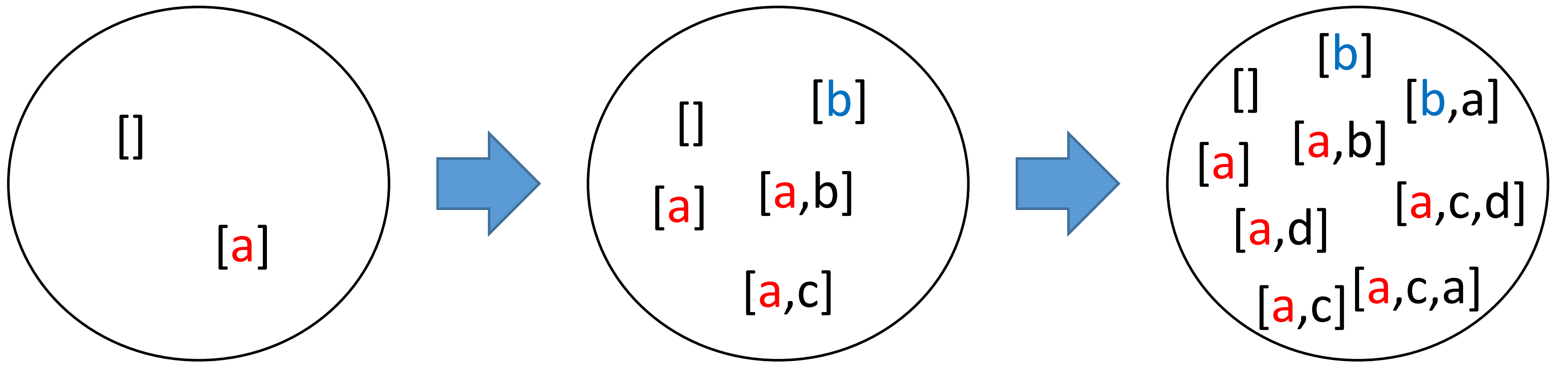
Cause of Low and Unstable Coverage

Positive feedback loop of FDRT

⇒Bias grows in pool

⇒Less diversity of generated tests

Bias in pool is amplified by feedback (e.g. List)



Proposed Method

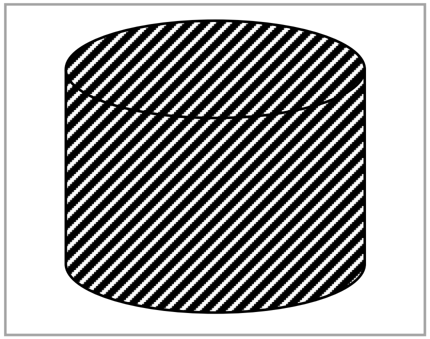
Feedback-controlled Random Test Generation

- Keep diversity by multiple pools
  - Hold multiple pools at the same time
  - Use multiple pools concurrently
- Promote diversity by manipulating pools
  1. Select pool
  2. Add pool
  3. Delete pool
  4. Global reset

Keep Diversity by Multiple Pools

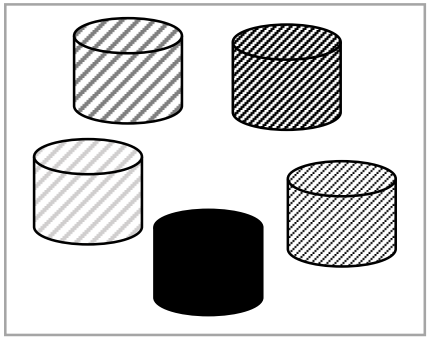
- Hold multiple pools at the same time
  - Each pool may be biased, but keep diversity as whole
- Use multiple pools concurrently (in turn)
  - Enable pool manipulation described later

Single pool



Original method

Set of pools



Proposed method

Promote Diversity by Manipulating Pools

1. Select pool

   Prioritize pools by ‘score’ function

   (High priority for pools that are likely to archive higher coverage)

2. Add pool

   Add new pools dynamically

3. Delete pool

   Delete similar pools using ‘uniqueness’ function

4. Global reset

   Reset all pools + Restart JVM

See the paper for the definition of score and uniqueness function

Evaluation

Compared 3 methods

- baseline   FDRT, one run
- reset   FDRT, reset every 100 sec.
- control   Proposed method

SUT

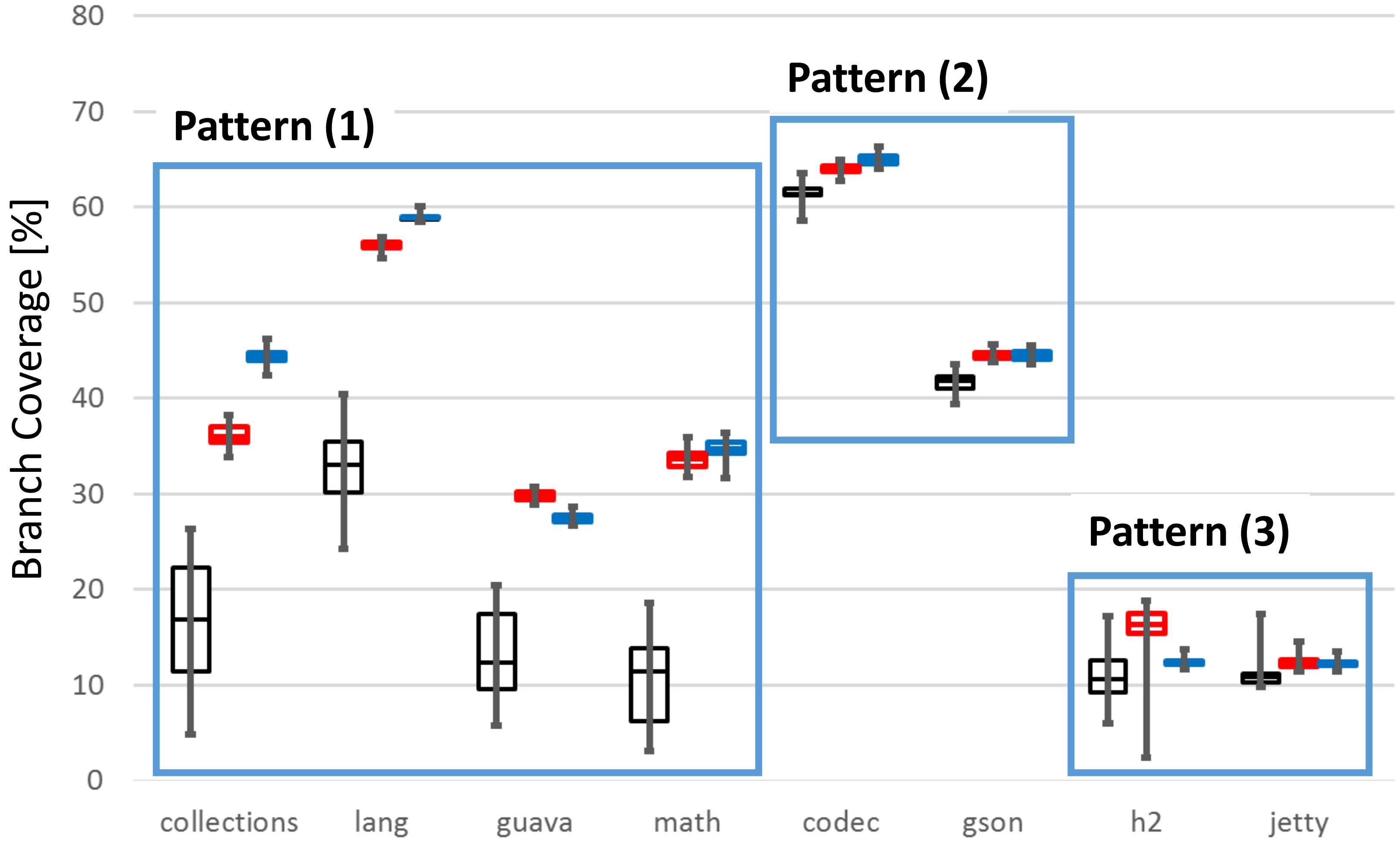
- 8 popular Java libraries from MVNRepository

Configuration

- Generate tests using 3600 sec. and record coverage of generated tests
- Conduct experiments with 30 different random seeds

Xeon X5650 (2.67GHz), 100GB RAM, CentOS 7.0
Isolated by Docker Ubuntu 14.04 w/ OpenJDK 1.7

Results – after 3600 seconds



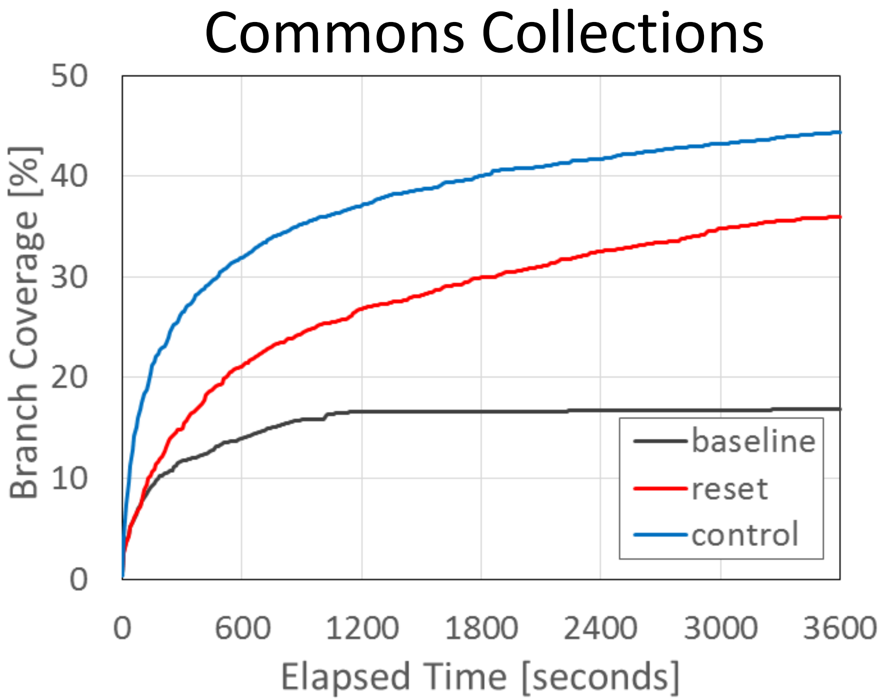
8 Libraries x 3 methods (baseline, reset, control)

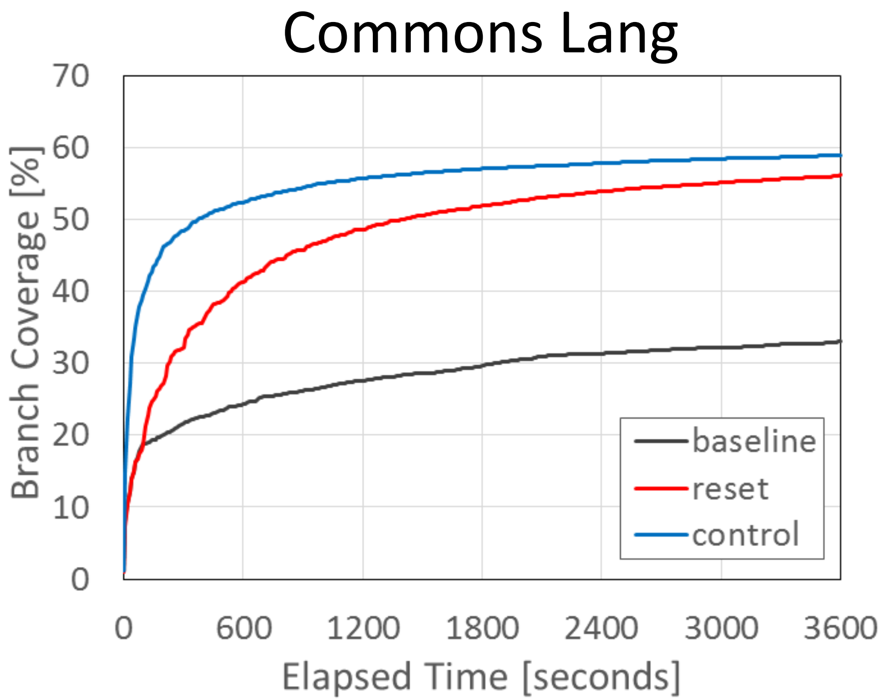
(1) Large Utility Libraries

Random testing is semantically suitable for this kind of libraries

4 utility libraries with 50K ～200K LOC

Large improvement on average and variance of coverage





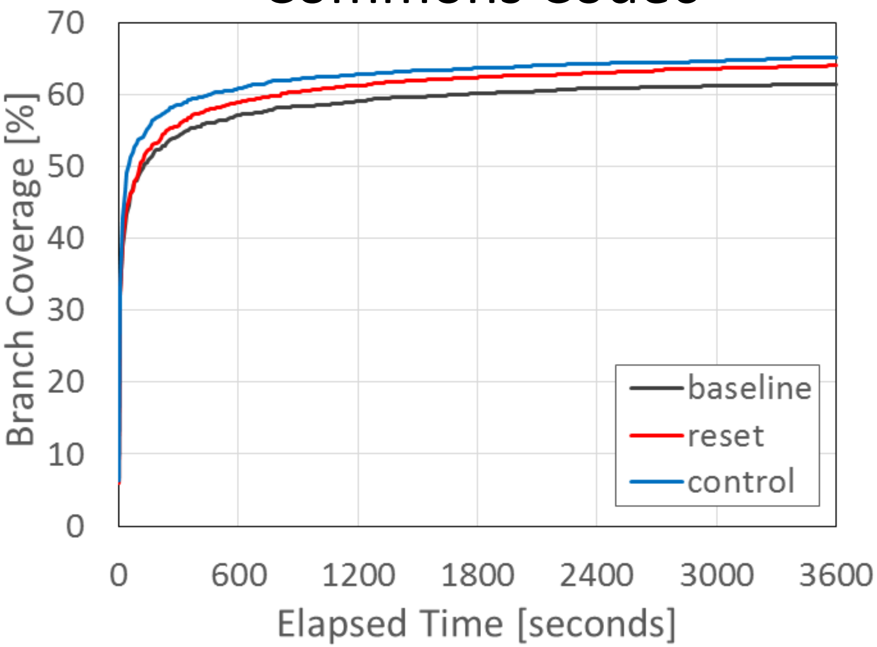
(2) Small Libraries

2 libraries with 10K LOC

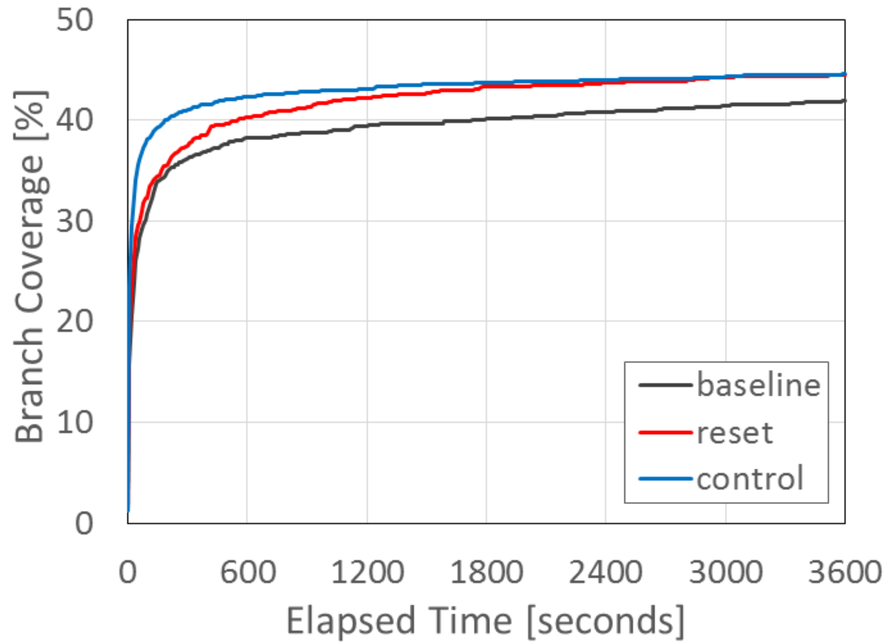
Small improvement, as the original FDRT do very well

Improvement on increase speed

Commons Codec



Gson

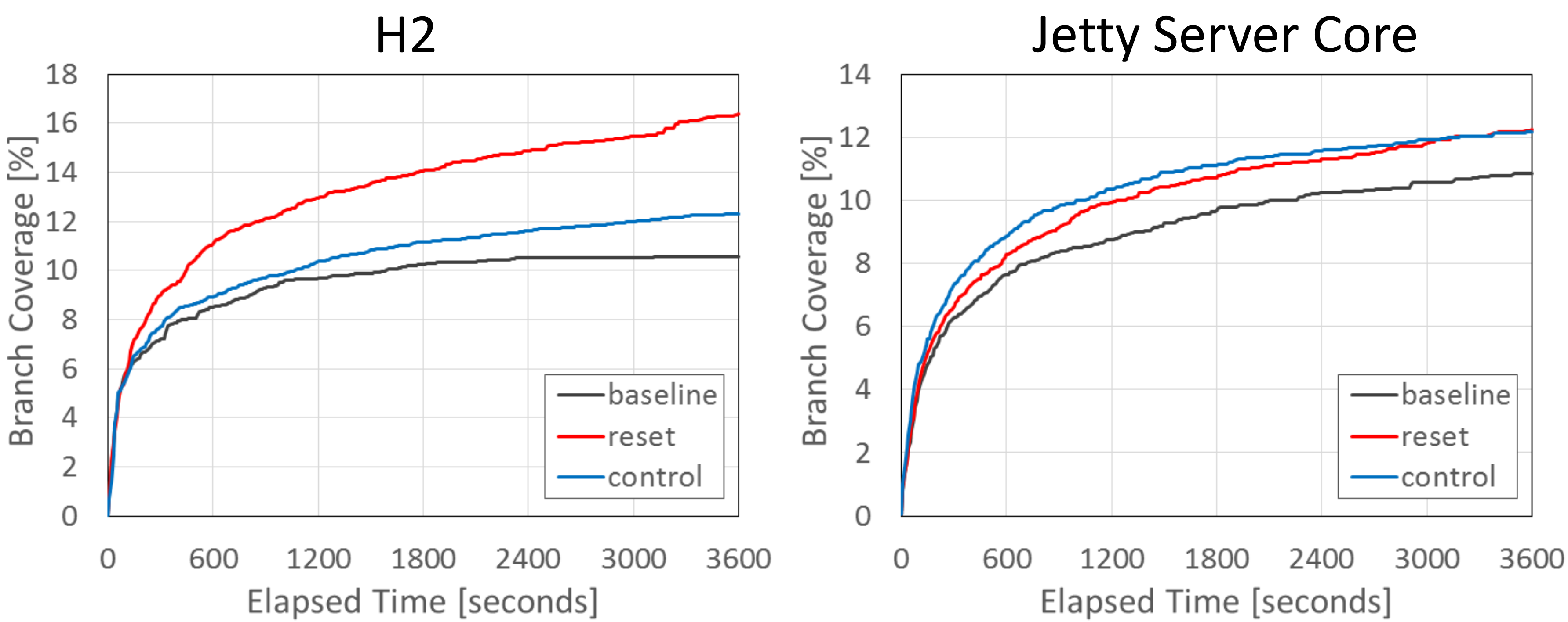


(3) Configuration-intensive Libraries

2 libraries (Database / Web server)

No improvement, very low coverage

Needs careful configuration to work properly



Summary

Problem

Low and unstable coverage of FDRT

Cause: Bias of pool due to positive feedback loop

Method

Feedback-controlled Random Test Generation

- Keep diversity by multiple pools
- Promote diversity by pool manipulation

Result

Three result patterns depending on SUT

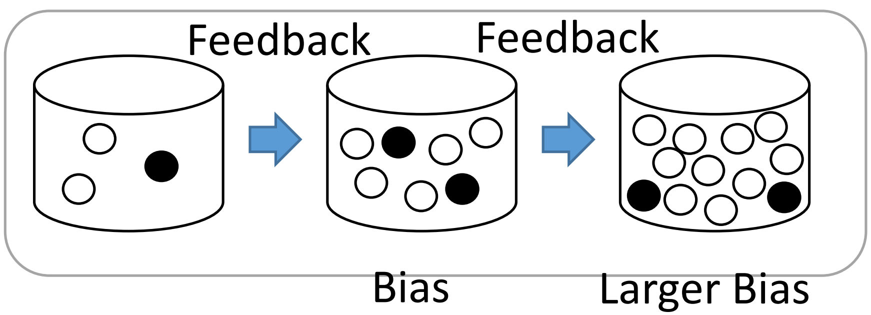
- Large utility libraries: Large improvement
- Small libraries: Small improvement, Less time for fixed coverage
- Configuration-intensive libraries: No changes

Appendix

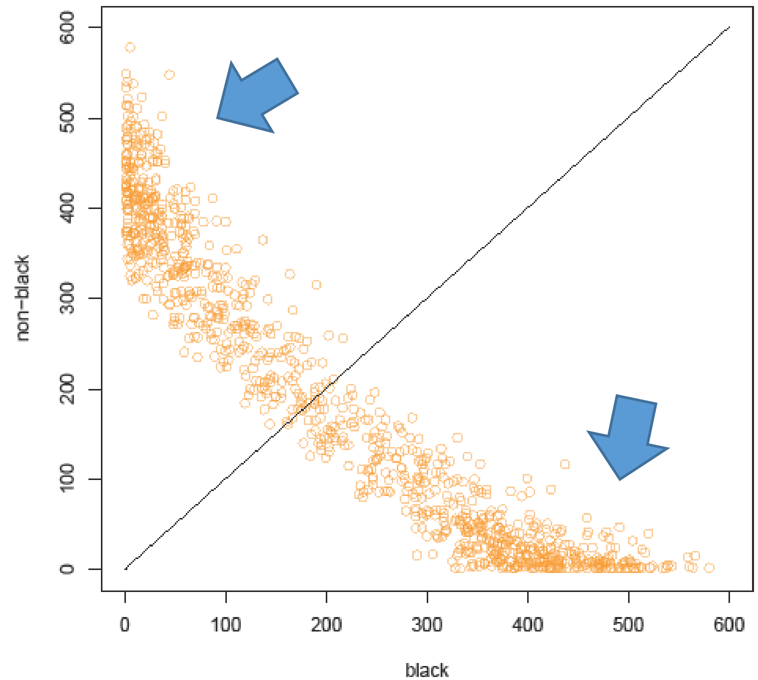
Bias and Limited Diversity

e.g. Black or non-black stone

```
class Stone {
  bool black;
  Stone(bool black) {…}
  bool isBlack() {…}
  Stone clone() {…}
}
```



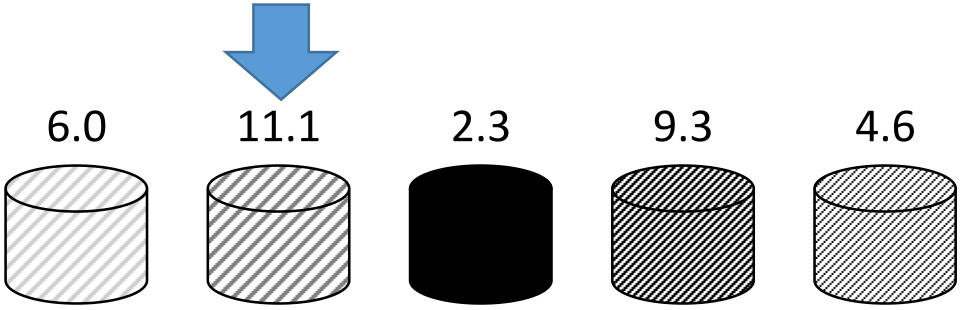
# of generated stones



1. Select Pool

- Select pool that is most likely to increase coverage
- Scoring function

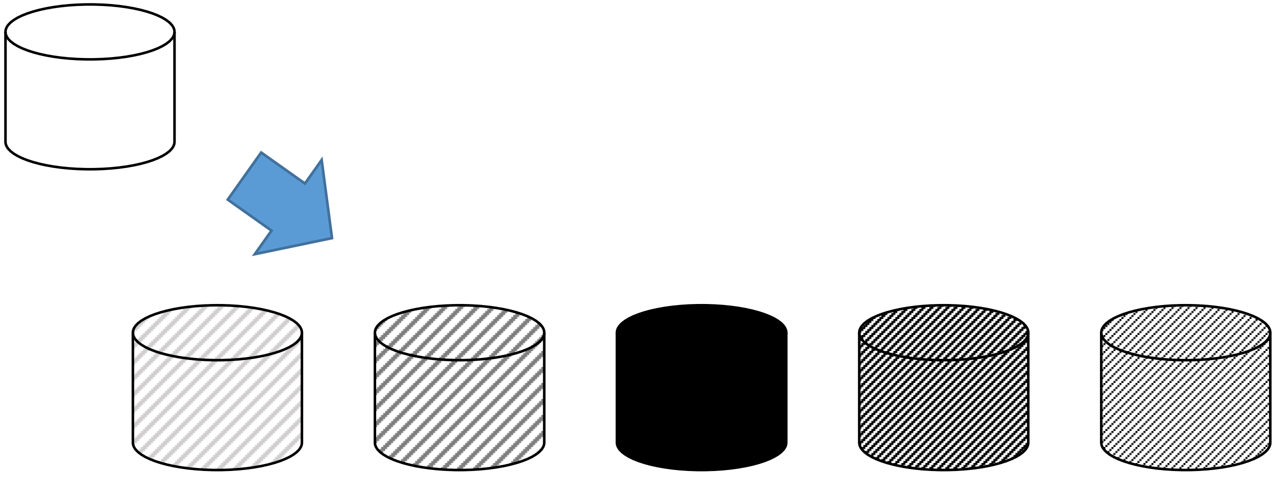
$$score(pool) = \begin{cases} \frac{coverage(pool)}{consumedTime(pool)} & (coverage(pool) > 0) \\ \infty & (coverage(pool) = 0) \end{cases}$$



Improves average coverage

2. Add Pool

- Add a new pool every 1 second



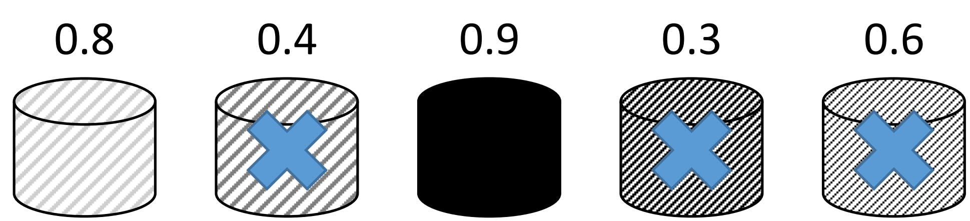
3. Delete Pool

- Delete pools with similar contents, when #pools exceeds a threshold
- Uniqueness function

$$uniqueness(pool) = \frac{\sum_{c \in covered(pool)} uniqueness(pool, c)}{|covered(pool)|}$$

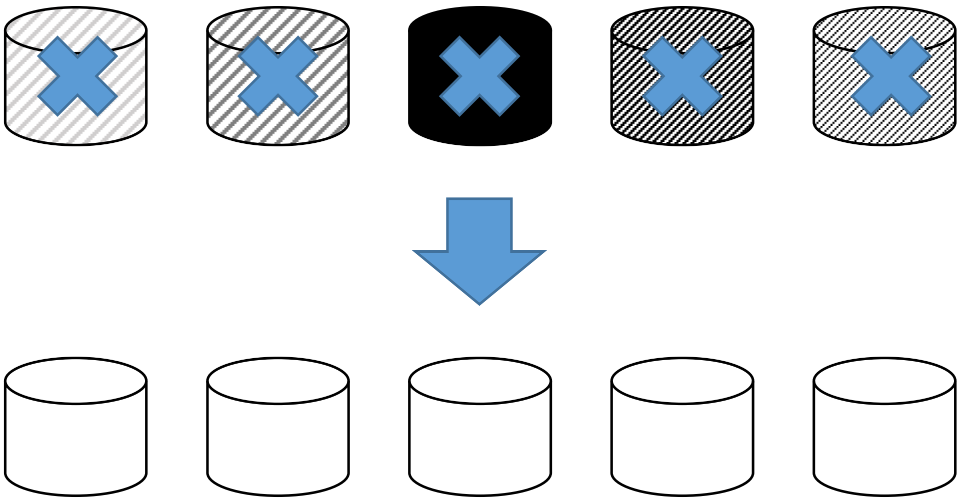
$$uniqueness(pool, c) = \frac{count(c, pool)}{\sum_{p \in allpools} count(c, p)}$$

Improves (decreases) Variance of coverage



4. Global Reset

- Reset every pool and restart JVM
- In order to remedy effect of nondeterministic behaviors and JVM instability



Results

3 result patterns, depending on SUT property

	Name	LOC	Category
(1)	Commons Collections	58,186	Collections
	Commons Lang	66,628	Core Utilities
	Guava	129,249	Core Utilities
	Commons Math	202,839	Math Libraries
(2)	Commons Codec	13,948	Base64 Libraries
	Gson	12,216	JSON Libraries
(3)	H2 Database Engine	158,926	Embedded SQL Databases
	Jetty Server Core	32,316	Web Servers

Related Work

- Adaptive random testing [Ciupa.08]
  - Similar concept as our approach
    （Avoid testing with similar values）
  - Heavy computation cost due to calculating distances between every generated values [Arcuri.11]

- Combination with Dynamic Symbolic Execution (DSE)
  - Use FDRT to create seed sequences for DSE [Bounimova.13, Zhang.14]
  - Alternatively execute FDRT and DSE [Garg.13]

  Replacing FDRT with our approach would improve the effectiveness and efficiency of these techniques