COMP5111 – Fundamentals of Software Testing and Analysis

Random Testing (Feedback Directed)

Shing-Chi Cheung

Computer Science & Engineering

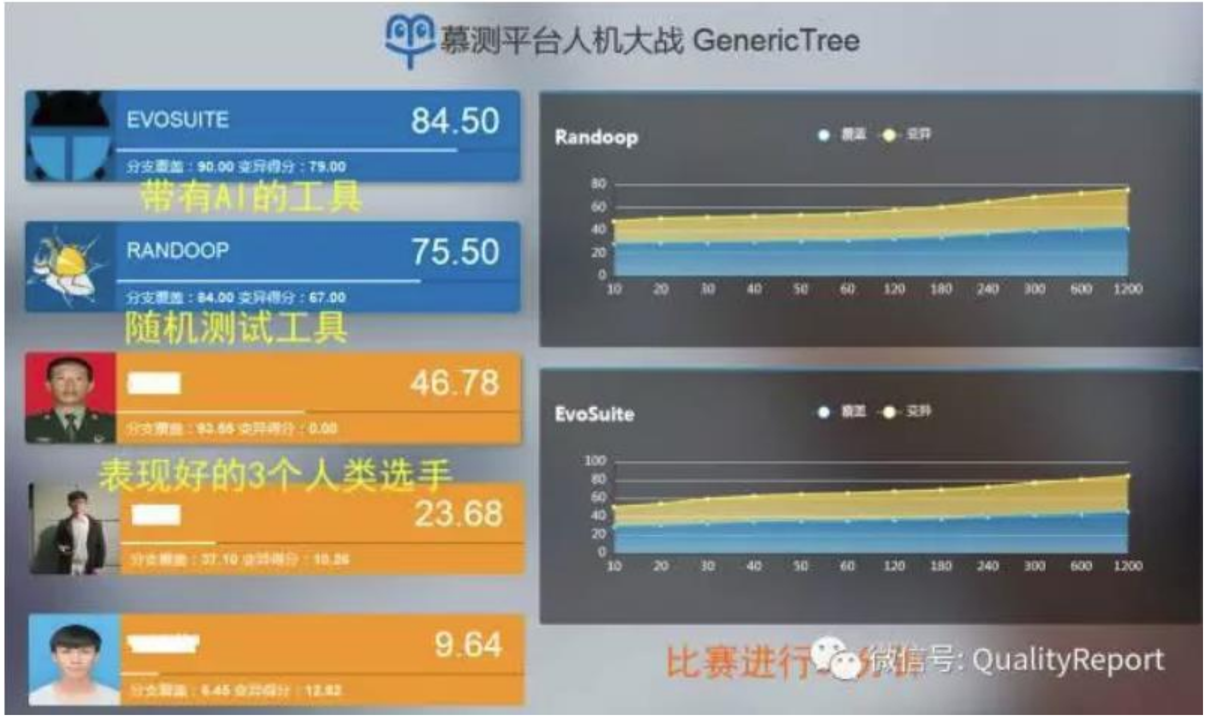
HKUST

Some slides are adapted from https://randoop.github.io/randoop/files/thesis_talk_post.pdf

Dream of a developer…

人機世紀之戰
Google
AlphaGo
VS.
李世石
3/9 (三)12:00
Round 1
必POTV
LIVE
全程中文直播

Three selected top testers among a few thousand contest participants vs two test generation algorithms in 2017



慕测平台人机大战 GenericTree

EVOSUITE 84.50

分支覆盖：90.00 变异得分：79.00

带有AI的工具

RANDOOP 75.50

分支覆盖：84.00 变异得分：67.00

随机测试工具

[redacted] 46.78

分支覆盖：93.56 变异得分：0.00

表现好的3个人类选手

[redacted] 23.68

分支覆盖：37.10 变异得分：10.26

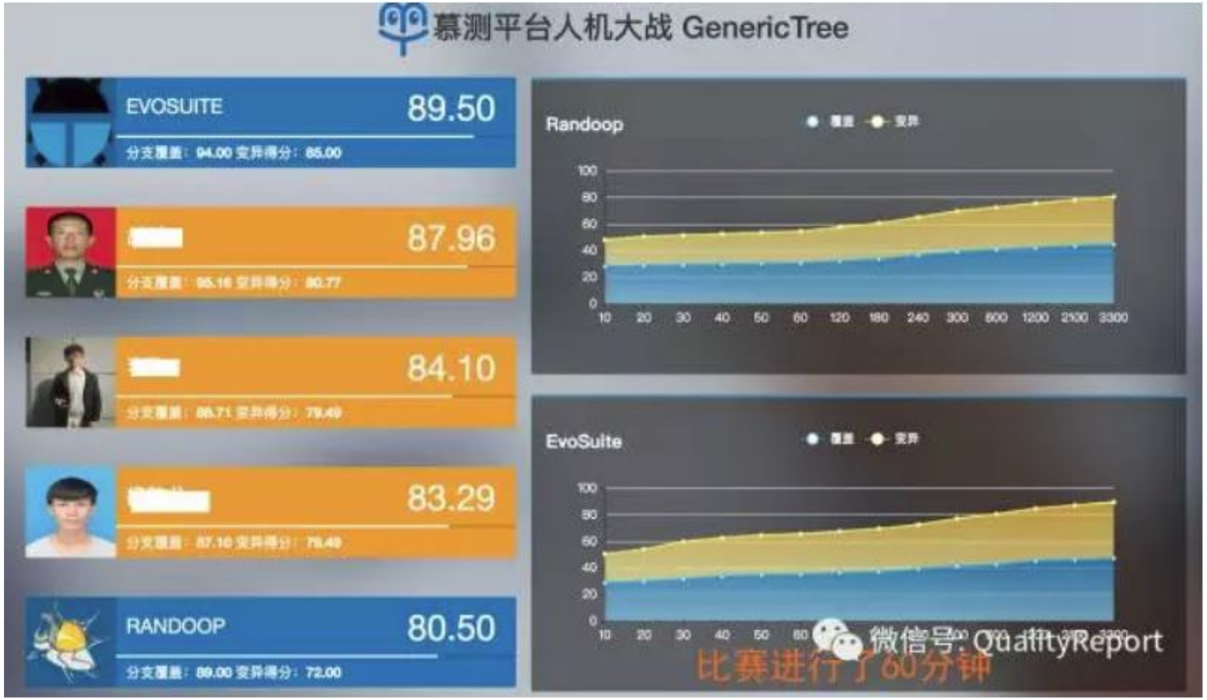
[redacted] 9.64

分支覆盖：6.45 变异得分：12.82

比赛进行 微信号: QualityReport

After 30 mins

After 60 mins



慕测平台人机大战 GenericTree

EVOSUITE 89.50

分支覆盖：94.00 变异得分：85.00

[redacted] 87.96

分支覆盖：95.16 变异得分：80.77

[redacted] 84.10

分支覆盖：88.71 变异得分：79.49

[redacted] 83.29

分支覆盖：87.10 变异得分：79.48

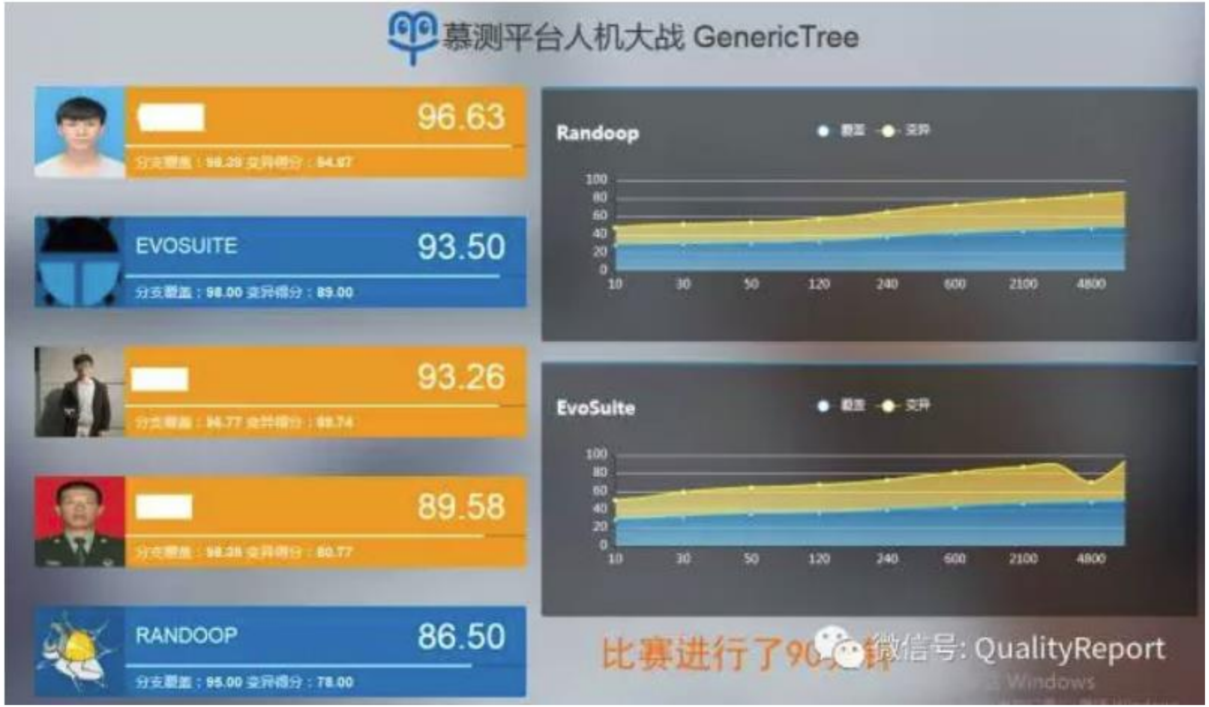
RANDOOP 80.50

分支覆盖：89.00 变异得分：72.00

微信号: QualityReport

比赛进行了60分钟

After 90 mins



慕测平台人机大战 GenericTree

[redacted] 96.63
分支覆盖：98.29 变异得分：94.97

EVOSUITE 93.50
分支覆盖：98.00 变异得分：89.00

[redacted] 93.26
分支覆盖：94.77 变异得分：89.74

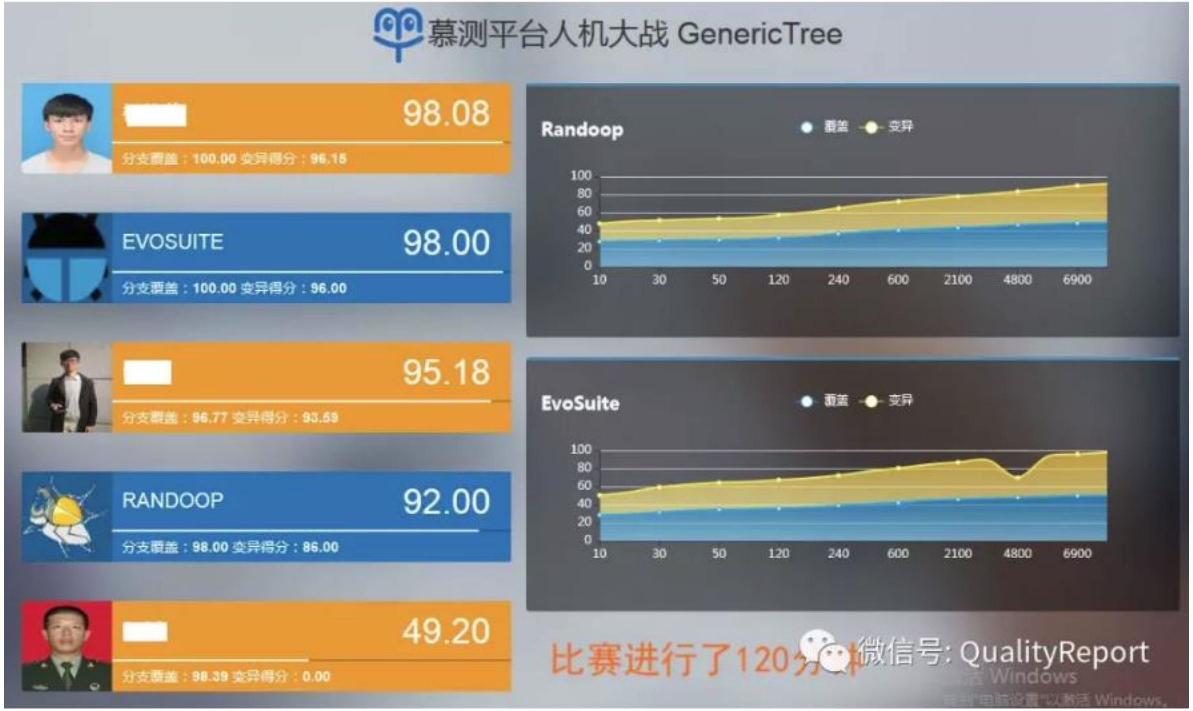
[redacted] 89.58
分支覆盖：98.28 变异得分：80.77

RANDOOP 86.50
分支覆盖：95.00 变异得分：78.00

比赛进行了90

微信号: QualityReport
Windows

After 120 mins



慕测平台人机大战 GenericTree

[redacted]	98.08	分支覆盖：100.00 变异得分：96.15
EVOSUITE	98.00	分支覆盖：100.00 变异得分：96.00
[redacted]	95.18	分支覆盖：96.77 变异得分：93.59
RANDOOP	92.00	分支覆盖：98.00 变异得分：86.00
[redacted]	49.20	分支覆盖：98.39 变异得分：0.00

比赛进行了120分钟

微信号: QualityReport

激活 Windows
转到"电脑设置"以激活 Windows。

Automatic Software Testing

- Random testing
- Symbolic analysis
- Concolic testing
- Search-based testing

Can unit tests be automatically generated without program spec?

Can generated tests detect real faults?

Random Testing

```
foo (int &x, int &y) {
 if (x>y) {
  x = x + y;
  y = x - y;
  x = x - y;
  if (x - y > 0) {
      assert (false); // bug
  }
 }
}
```

Options:

- Random input data generation
- Random user interaction sequence
- Random data selection from database
- Combinations of all above

1st trial: x = 1321, y = 456;

2nd trial: x = -2908, y = 89;

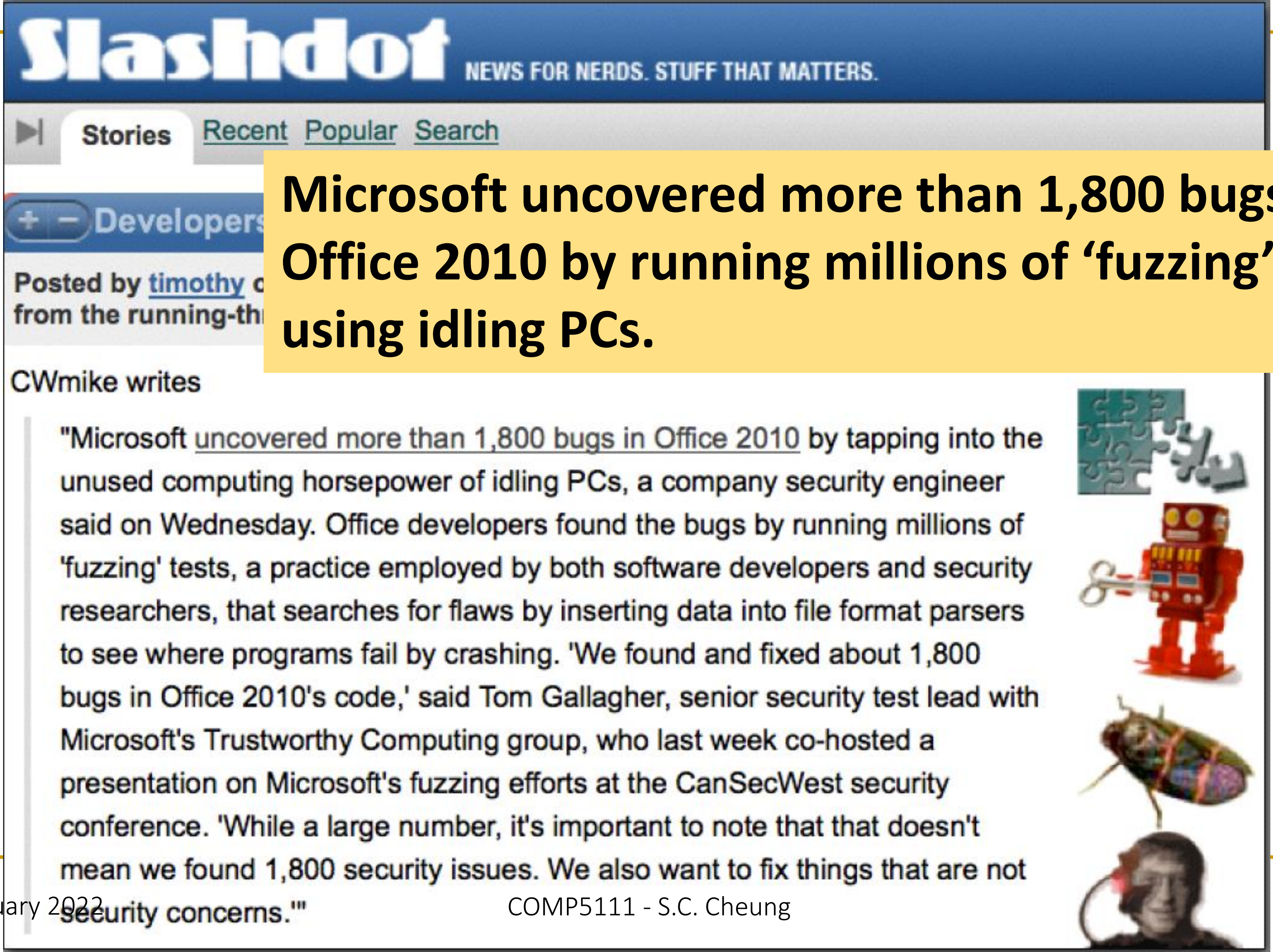
…

nth trial: …

Random Testing

- Mentioned first time by Glenford J. Myers in 1979.
- Popularly used by industry as fuzzing tests.

Microsoft uncovered more than 1,800 bugs in Office 2010 by running millions of ‘fuzzing’ tests using idling PCs.



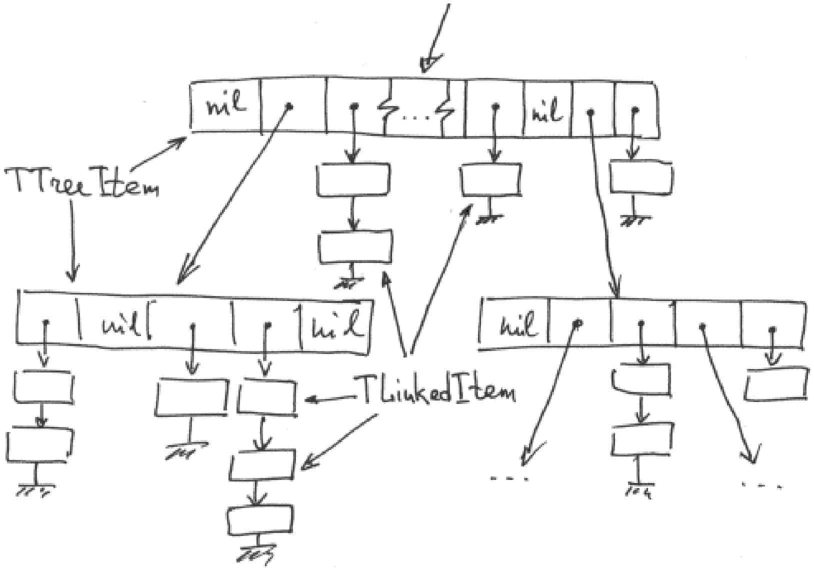
Posted by timothy o
from the running-th

CWmike writes

> "Microsoft uncovered more than 1,800 bugs in Office 2010 by tapping into the unused computing horsepower of idling PCs, a company security engineer said on Wednesday. Office developers found the bugs by running millions of 'fuzzing' tests, a practice employed by both software developers and security researchers, that searches for flaws by inserting data into file format parsers to see where programs fail by crashing. 'We found and fixed about 1,800 bugs in Office 2010's code,' said Tom Gallagher, senior security test lead with Microsoft's Trustworthy Computing group, who last week co-hosted a presentation on Microsoft's fuzzing efforts at the CanSecWest security conference. 'While a large number, it's important to note that that doesn't mean we found 1,800 security issues. We also want to fix things that are not security concerns.'"

Random Unit Test Generators

- Challenge 1: Generate complex data structures
- Challenge 2: Avoid generating redundant tests
- Challenge 3: Generate test oracles (i.e., the assert statements)



A Polynomial Library

Dynamic data structure

```
class Poly {
  List <Mono> elements;
  Poly() { … }
  Poly plus(Mono m) { … }
  Poly mult(Poly p) { … }
  Poly sum(Poly p) { … }
  Poly deriv() { … }
  …
}
```

```
class Mono {
  int num, den, exp;
  Mono(int num, int den, int exp) {
    …
  }
}
```

$$\frac{num}{den}\; x^{exp}$$

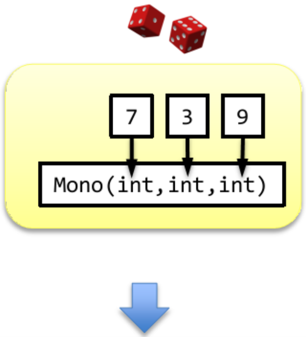
How can we generate tests with complex data structures?

Insight: Complex data structures can be built incrementally from primitive values

Common Generation Strategy

operation	input	output
`Mono(int,int,int)`	`3 ints`	`a new Mono`
`Poly()`	`none`	`a new Poly`
`Poly plus(Mono)`	`a Poly, a Mono`	`a new Poly`

random terms



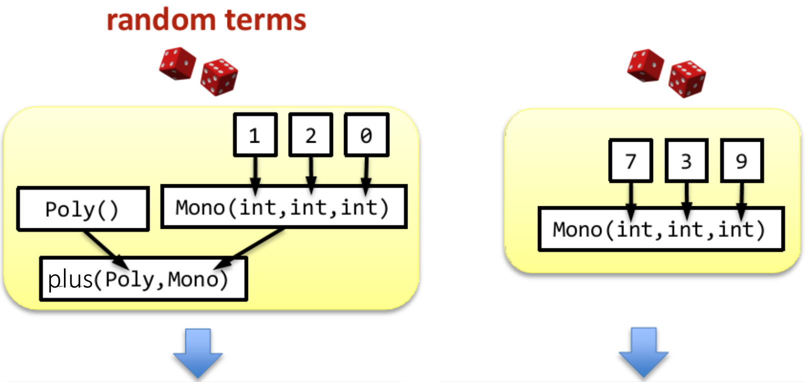
```
public void test1() {

  p = new Poly()
      .mult(new Poly());

  checkInvariant(p);
}
```

```
public void test2() {

  p = new Poly()
      .plus(new Mono(1,2,0));

  checkInvariant(p);
}
```

```
public void test3() {

  m = new Mono(7,3,9);


  checkInvariant(m);
}
```

Common Generation Strategy

operation	input	output
Mono(int,int,int)	3 ints	a new Mono
Poly()	none	a new Poly
Poly plus(Mono)	a Poly, a Mono	a new Poly



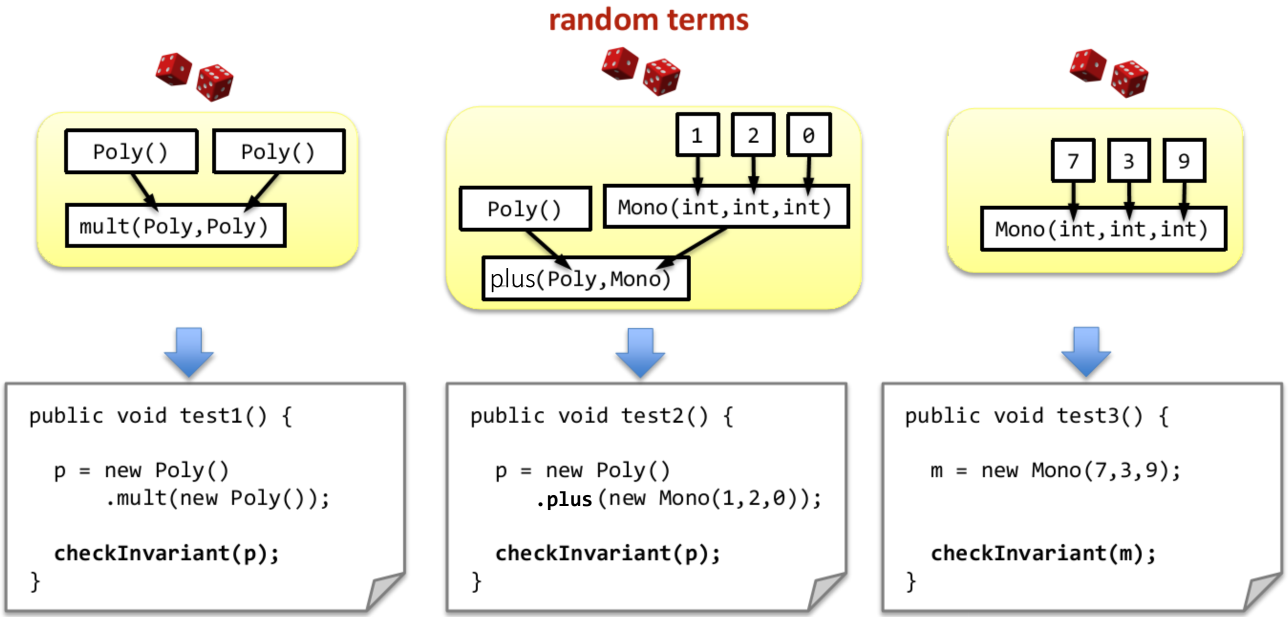
```
public void test1() {

  p = new Poly()
      .mult(new Poly());

  checkInvariant(p);
}
```

```
public void test2() {

  p = new Poly()
     .plus(new Mono(1,2,0));

  checkInvariant(p);
}
```

```
public void test3() {

  m = new Mono(7,3,9);


  checkInvariant(m);
}
```

Common Generation Strategy

operation	input	output
Mono(int,int,int)	3 ints	a new Mono
Poly()	none	a new Poly
Poly plus(Mono)	a Poly, a Mono	a new Poly



```
public void test1() {

  p = new Poly()
      .mult(new Poly());

  checkInvariant(p);
}
```

```
public void test2() {

  p = new Poly()
     .plus(new Mono(1,2,0));

  checkInvariant(p);
}
```

```
public void test3() {

  m = new Mono(7,3,9);


  checkInvariant(m);
}
```

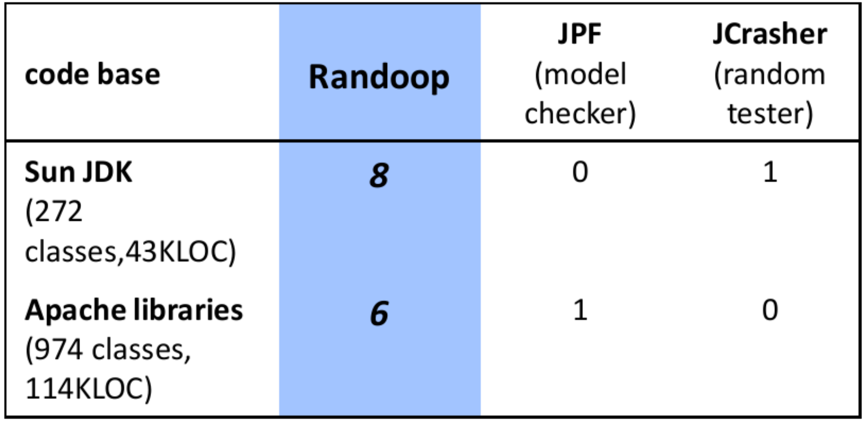
How can we build complex data structures mechanically?

Insight: Feed generated data structures back to a repository

Feedback Directed Random Test Generator

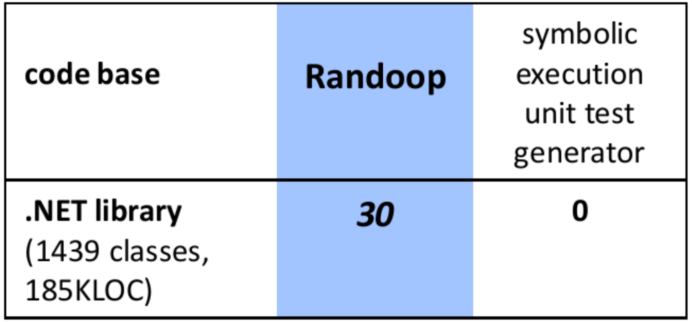
- Carlos Pacheco [ICSE 2007, ISSTA 2008]
- Randoop (https://code.google.com/p/randoop/)
- Able to reveal unknown faults in widely used libraries

distinct errors revealed (Java)



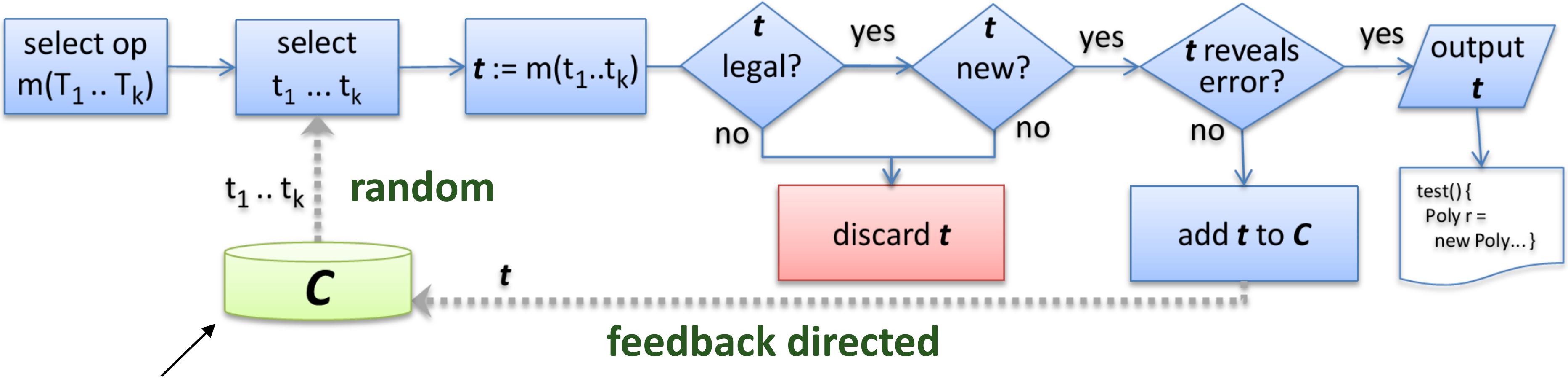
code base	**Randoop**	**JPF** (model checker)	**JCrasher** (random tester)
Sun JDK (272 classes,43KLOC)	***8***	0	1
Apache libraries (974 classes, 114KLOC)	***6***	1	0

distinct errors revealed (.NET)



code base	**Randoop**	symbolic execution unit test generator
.NET library (1439 classes, 185KLOC)	***30***	**0**

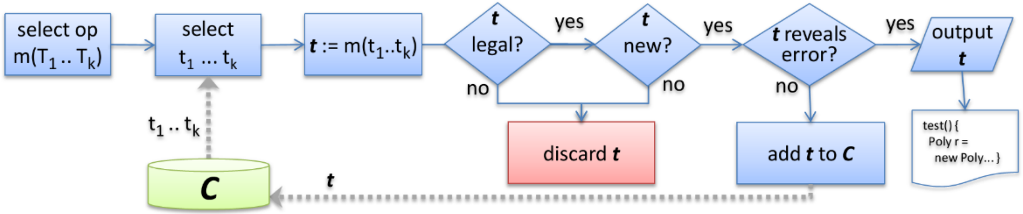
Feedback Directed Random Test Generator



- *C* is a repository containing possible terms used by the Test Generator.

Feedback Directed Random Test Generator

- Generate tests with simple inputs randomly from repository *C*.



select op m(T_1 .. T_k)

select t_1 … t_k

t := m(t_1..t_k)

t legal?

yes

no

t new?

yes

no

discard *t*

t reveals error?

yes

no

add *t* to *C*

output *t*

test() {
Poly r =
new Poly...}

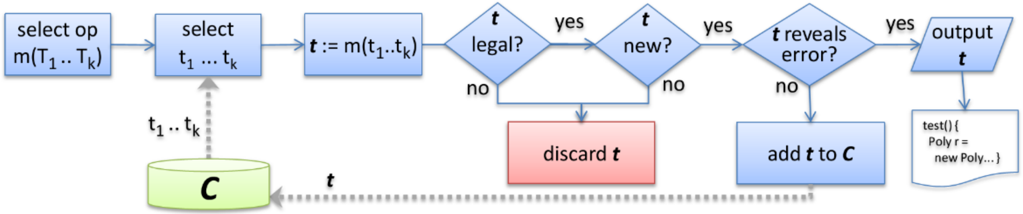
t_1 .. t_k

C

t

Feedback Directed Random Test Generator

- Generate tests with simple inputs randomly from repository *C*.
- Build new test inputs incrementally from previous ones.



select op $m(T_1 .. T_k)$

select $t_1 ... t_k$

t := $m(t_1..t_k)$

t legal?

yes

no

t new?

yes

no

t reveals error?

yes

no

output ***t***

discard ***t***

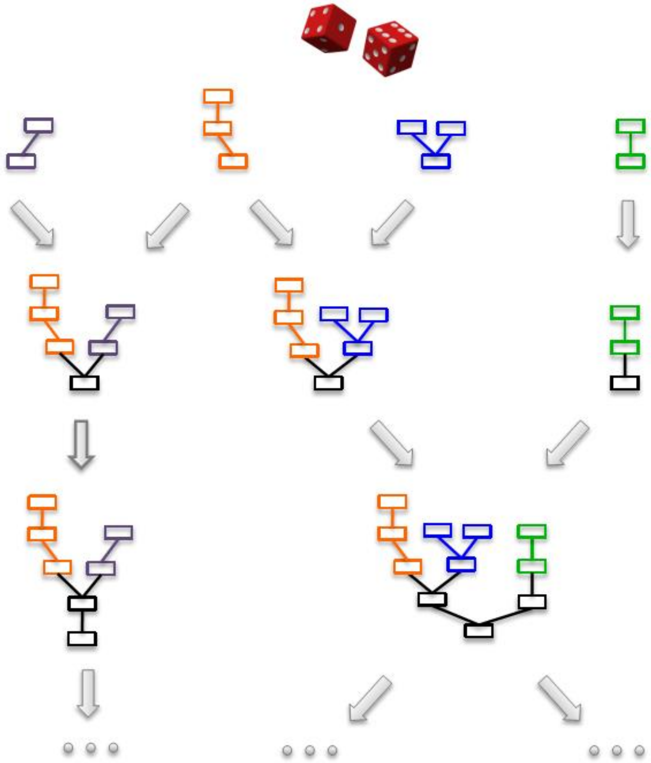
add ***t*** to ***C***

$t_1 .. t_k$

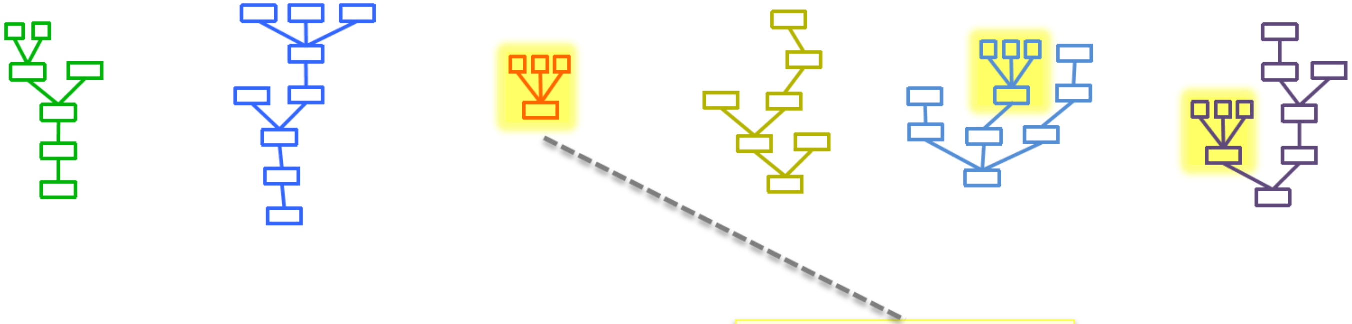
C

t

```
test() {
 Poly r =
   new Poly...}
```



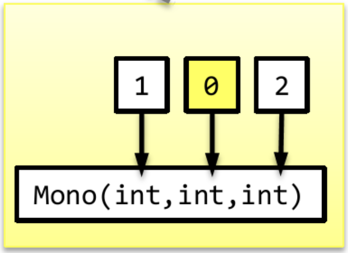
But …



generates useless inputs

- illegal, repetitive

$$\frac{num}{den}\; x^{exp}$$



illegal input to Mono

throws IllegalArgException

Mono(int num, int den, int exp) { … }
Expects den ≠ 0 and exp ≥ 0

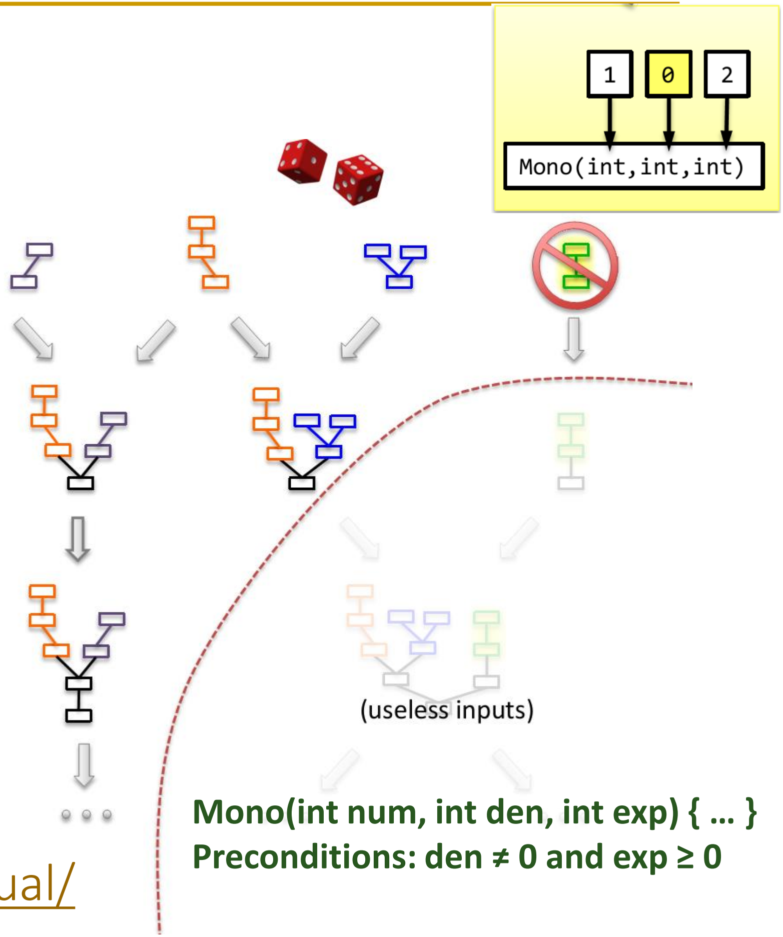
→ **Input Space Pruning**

How can we avoid redundant tests?

Insight: Prune the input space with pre-conditions and equivalence

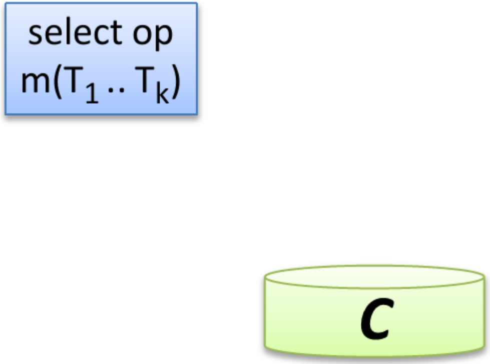
Pruning Input Space

- Executes inputs
- Discards the ones useless for extension
  - illegal, redundant
- Prune input space
  - Specify pre-conditions on method parameters
  - See method pre-conditions at https://randoop.github.io/randoop/manual/



Mono(int num, int den, int exp) { … }
Preconditions: den ≠ 0 and exp ≥ 0

Example: Mono(int, int, int)



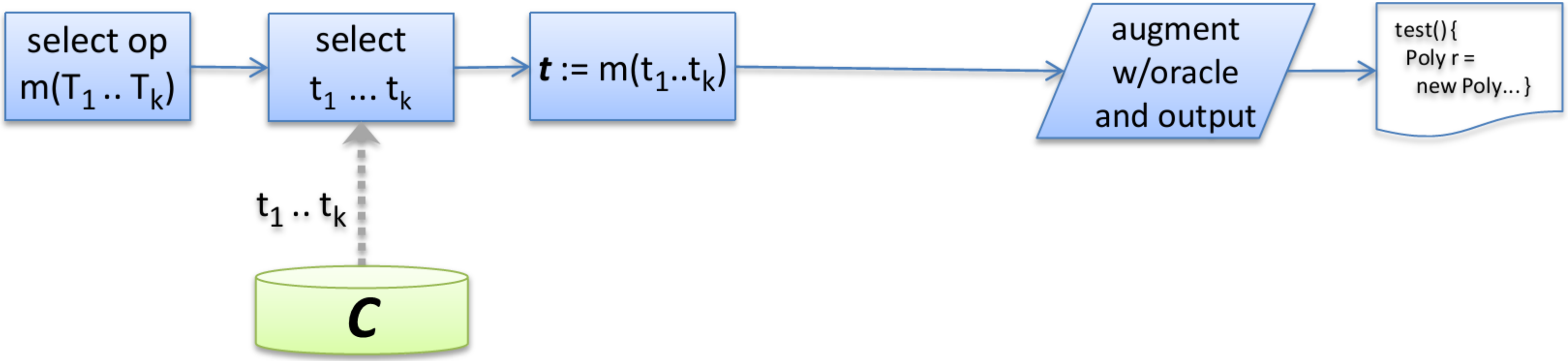
select op $m(T_1 .. T_k)$

C

component set of terms

`C = { 0, 1, 2, null, false, etc. }`

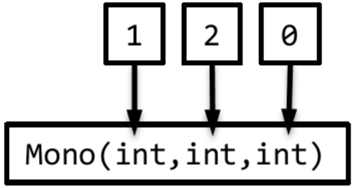
Example: Mono(int, int, int)



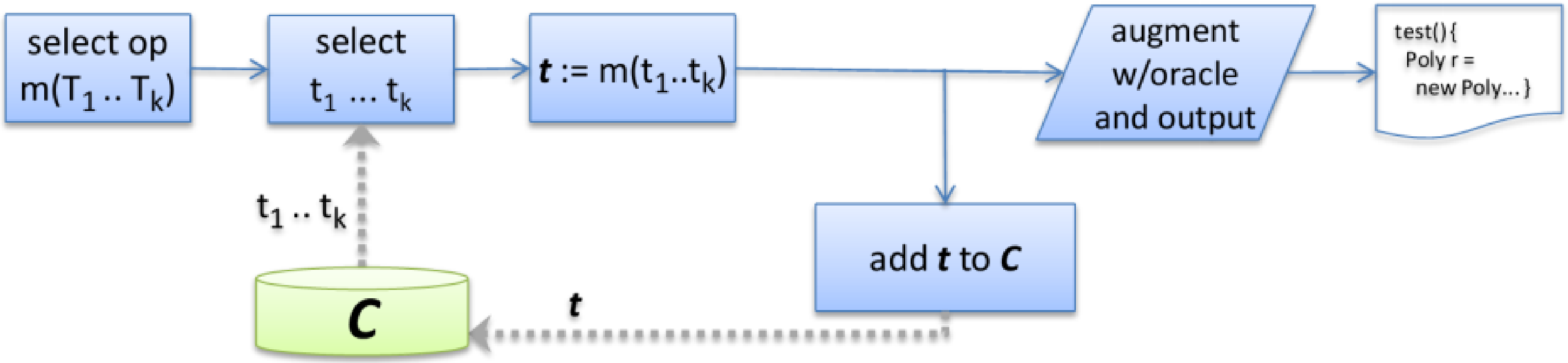
component set of terms

C = { `0, 1, 2, null, false, etc.` }

Example:



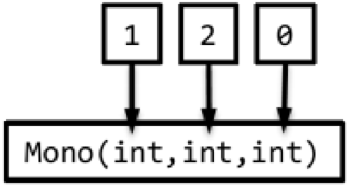
Example: Mono(int, int, int)



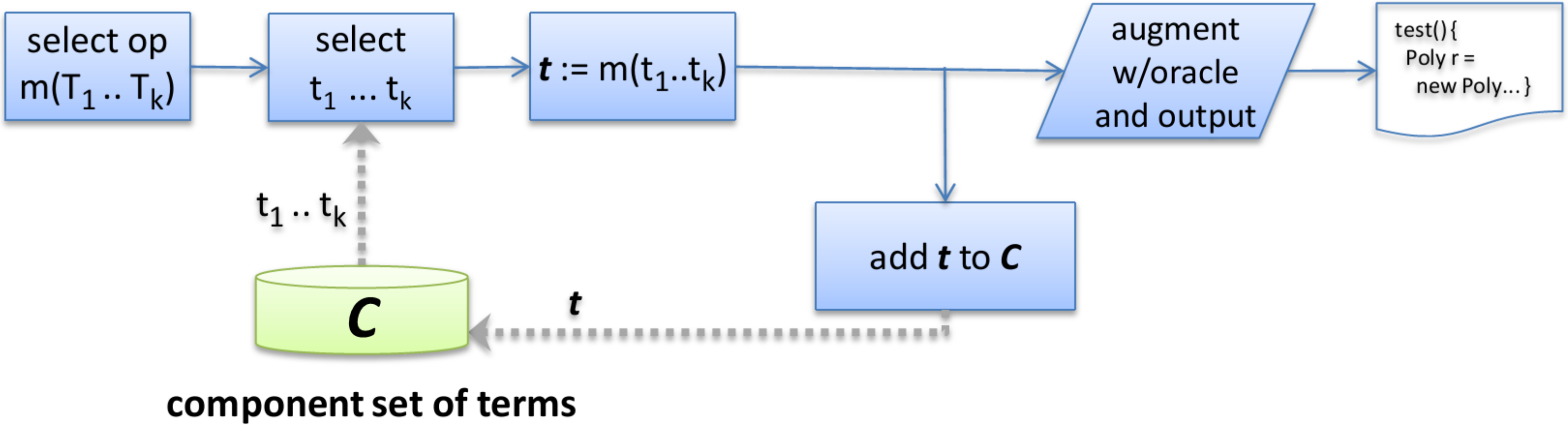
component set of terms

C = { `0, 1, 2, null, false, Mono(1,2,0)` }

Example:



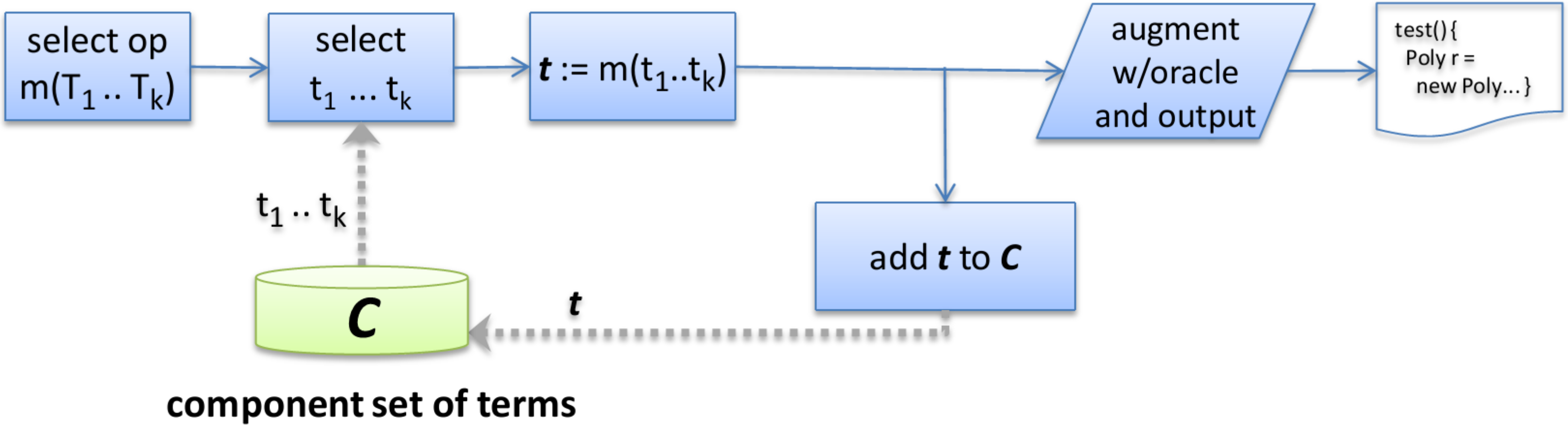
Example: Poly( )



C = { `0, 1, 2, null, false, Mono(1,2,0), Poly()` }

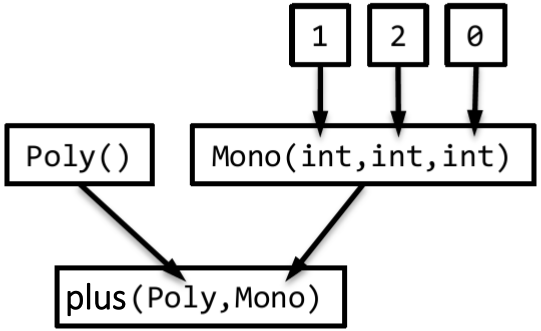
Example: `Poly()`

Example: plus(Poly, Mono)

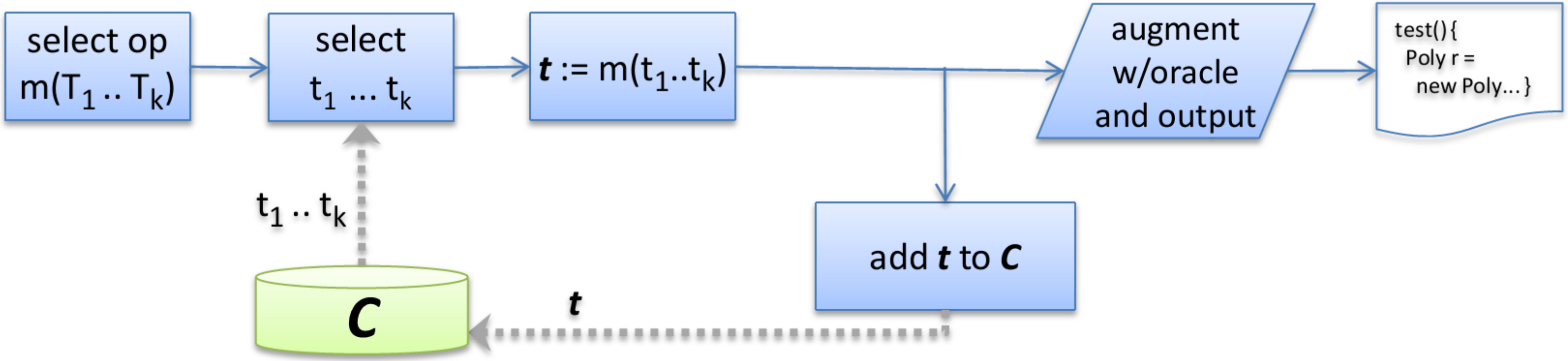


C = { 0, 1, 2, null, false, Mono(1,2,0), **Poly()** }

Example:



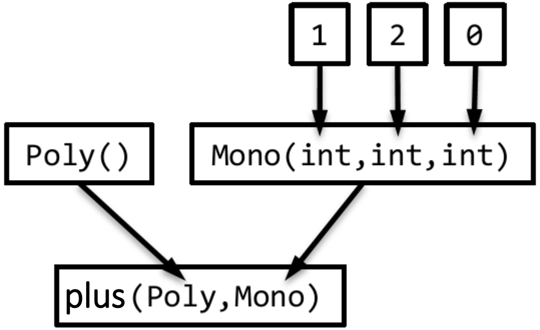
Example: plus(Poly, Mono)



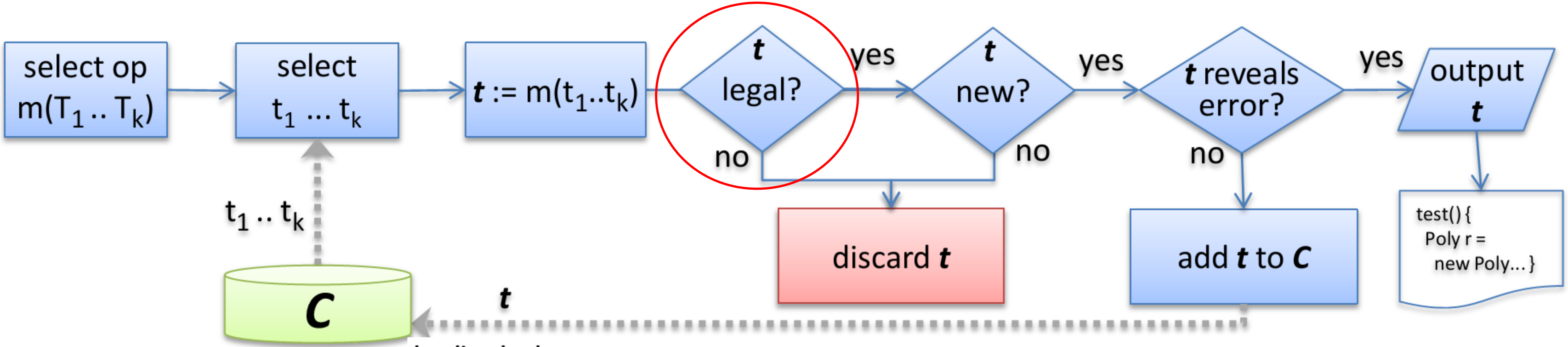
component set of terms

C = { `0, 1, 2, null, false, Mono(1,2,0), Poly(),` plus`(Poly(),Mono(1,2,0))` }

Example:



Guided Generator - Legality

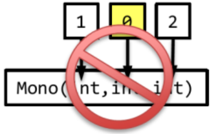


```
test(){
 Poly r =
  new Poly...}
```

legality checker

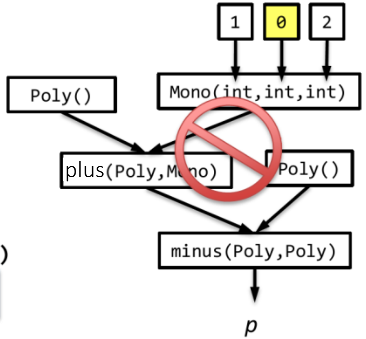
- determines if a term is legal/illegal
- discard illegal terms

Violate pre-conditions of m()

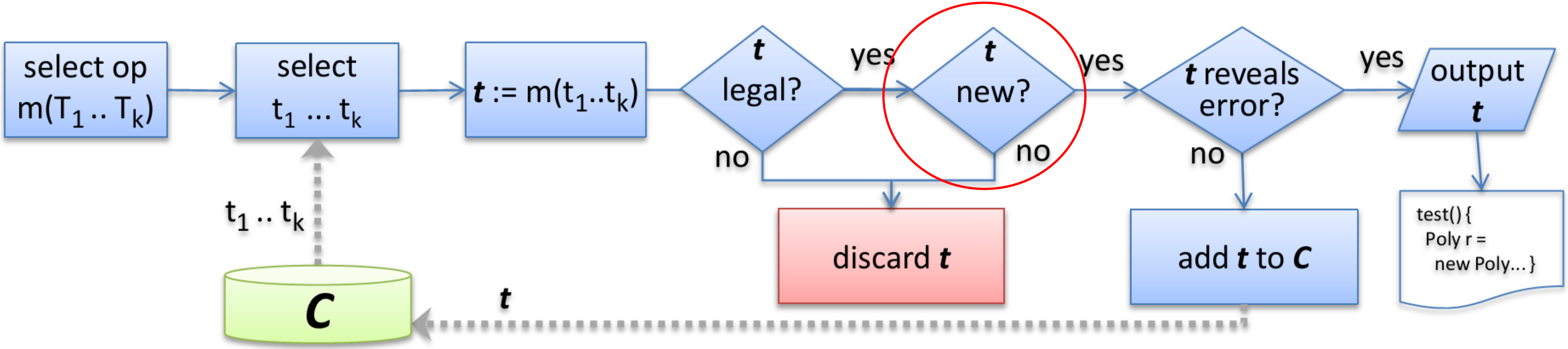


`Mono(int num, int den, int exp)`

Expects `den != 0` and `exp >= 0`.



Guided Generator - Equivalence



- Determines if two terms are equivalent
  - e.g., Mono(1,2,1) and Mono(2,4,1)
- Discard a term if equivalent to one in *C*
- Implement equivalence as t.equals(t')

Applied Randoop to 13 libraries

- built-in oracles
- heuristic guidance
- default time limit
  (2 minutes/library)

				tests	errors
	library	**LOC**	**classes**	**output**	**revealed**
JDK	java.util	39K	204	20	6
	java.xml	14K	68	12	2
Apache commons project	chain	8K	59	20	0
	collections	61K	402	67	4
	jelly	14K	99	78	0
	logging	4K	9	0	0
	math	21K	111	9	2
	primitives	6K	294	13	0
.NET	mscorlib	185K	1439	19	19
	system.data	196K	648	92	92
	system.security	9K	128	25	25
	system.xml	150K	686	15	15
	web.services	42K	304	41	41
	TOTAL	***750K***	***4451***	***411***	***206***

Outputs one test per violating method

.NET libraries specification:

"no method should throw NPEs, assertion violations, or IllegalMemAccess exception"

How can we define test oracles?

Insight: Define generic oracles and regression oracles

Five Built-in Oracles

- Contracts over Object.equals()
  - Reflexivity: o.equals(o) == true
  - Symmetry: o1.equals(o2) == o2.equals(o1)
  - Transitivity: o1.equals(o2) && o2.equals(o3) implies o1.equals(o3)
  - Equals to null: o.equals(null) == false
  - It does not throw an exception

Five Built-in Oracles

- Contracts over Object.hashCode()
  - o1.equals(o2) == true implies o1.hashCode() == o2.hashCode()
  - It does not throw an exception
- Contracts over Object.toString()
  - It does not return null
  - It does not throw an exception

Five Built-in Oracles

- Contracts over Object.clone()
  - It does not throw an exception, including CloneNotSupportedException
  - It does not throw an exception

Five Built-in Oracles

- Contracts over Comparable.compareTo() and Comparator.compare()
  - Reflexivity: o.compareTo(o) == 0
  - Anti-symmetry: sgn(o1.compareTo(o2)) == -sgn(o2.compareTo(o1))
  - Transitivity: o1.compareTo(o2)>0 && o2.compareTo(o3)>0 implies o1.compareTo(o3)>0
  - Substitutability of equals: x.compareTo(y) == 0 implies sgn(x.compareTo(z)) == sgn(y.compareTo(z))
  - Consistency with equals(): x.compareTo(y)==0 implies x.equals(y)
  - It does not throw exception

Regression Test Oracle

- Generate assertion using the current test output

```
public class ClassExampleWithFailure {
  public static int twice(int x) { return x+x;}
  public static int foo(int x, int y) {
    int z = twice(x);
    if (z == 144 && y > 20) {
      assert(false);  // assert failure
    }
    return y*z;
  }
}
```

```
@Test
  public void test022() throws Throwable {
    …
    int i2 = ClassExampleWithFailure.foo(24832, 388);
    org.junit.Assert.assertTrue(i2 == 19269632);
  }
```

Evaluation and Industry Adoption

Comparison

JDK

- 6 methods that create objects violating reflexivity of equality
- 2 well-formed XML objects cause **`hashCode/toString NPEs`**

Apache

- 6 constructors leave fields unset, leading to NPEs

.NET

- 175 methods throw forbidden exceptions
- 7 methods that violate reflexivity of **`equals`**

.NET

- library hangs given legal sequence of calls

without guidance

none revealed

66% fewer revealed

70% fewer revealed

not revealed

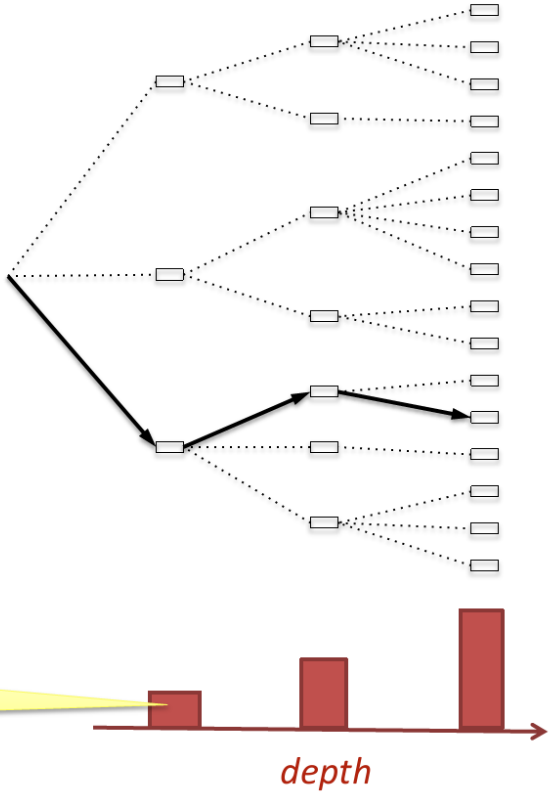
Comparison

- Why is Randoop more effective?
  - Prune useless inputs
  - Generates longer tests
  - Regression oracles

Test Length vs. Test Effectiveness

random testing is more effective when generating **long chains** of operations

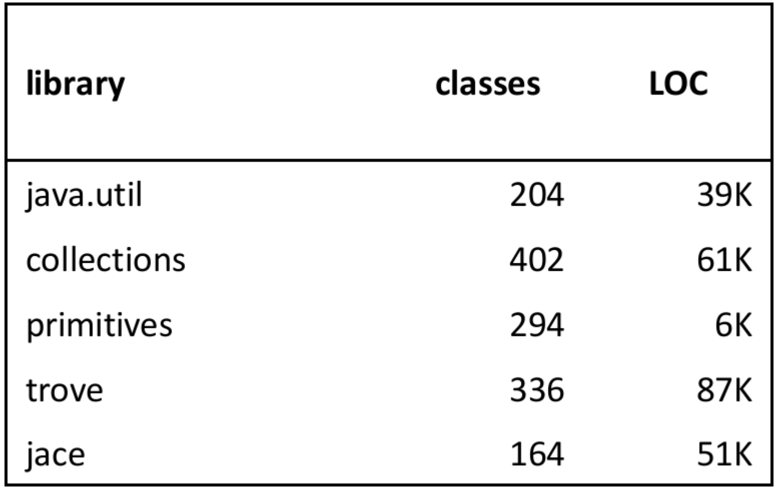
```
m=Mono(1,1,1)

p=Poly()

p2=p.add(m)
```



chances of an operation revealing an error

An Experiment

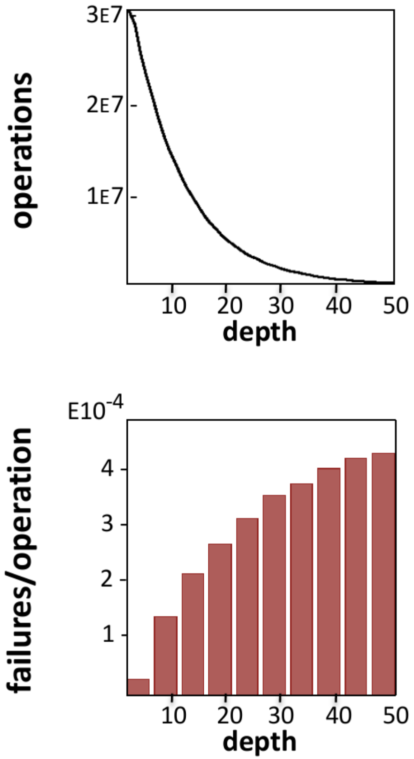
- Pure random testing
  - Start from empty sequence
  - Take random steps
  - Restart if error or exception
- Exercise 10M operations per library
  - Take several days



library	classes	LOC
java.util	204	39K
collections	402	61K
primitives	294	6K
trove	336	87K
jace	164	51K

Results

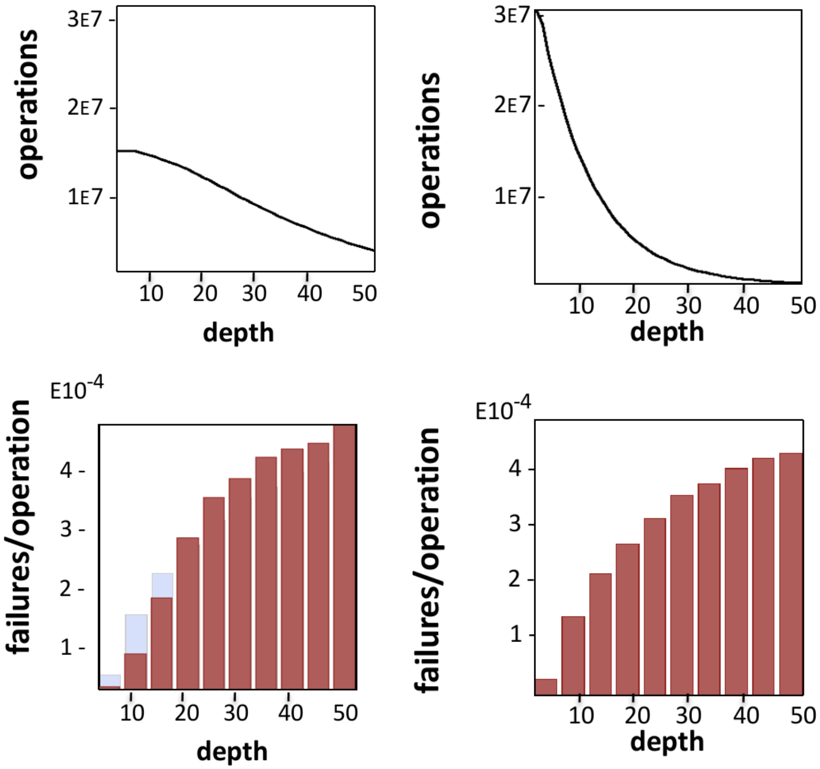
- Fail to create long chains (due to exceptions)
- Failure rate is higher at greater depths
- Ineffective → performs most operations where failure rate is the lowest



pure random

Using Randoop (i.e., Feedback Directed Random)

- More tests with longer chains
- Able to reveal more failures (under the same budget)



Randoop

pure random

Comparison

library	random walk	Randoop only leg.	Include also equivalence checking Randoop
java.util	20	21	27
collections	28	37	48
primitives	16	13	19
trove	20	27	27
jace	15	15	26
TOTAL	**99**	**113**	**147**

Failures detected

Other findings …

- Can generated tests detect real bugs?
- Is automated test generation cost effective?

Case study [ISSTA 2008]

- Microsoft test team
- Randoop (.NET version)
- Applied to highly-tested library
  - Tested over 5 years by 40 engineers

Findings: revealed more errors in 15 hours than the team typically discovers in 1 person-year of effort

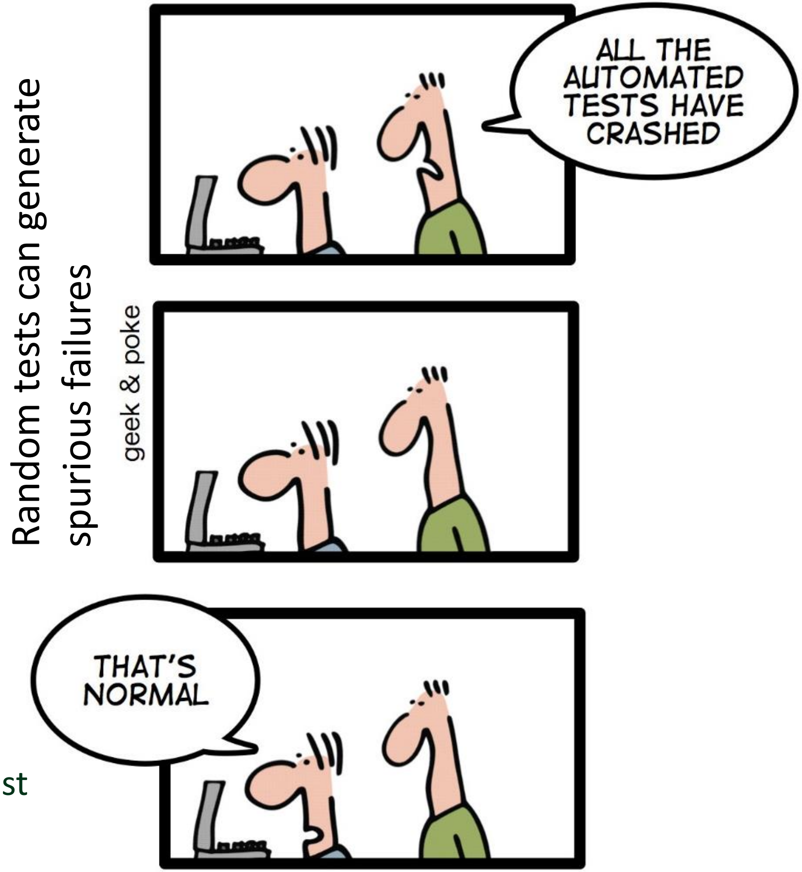
Case Study Statistics

- Facts
  - Human time interacting with Randoop: 15 hours
  - CPU time running Randoop: 150 hours
  - Total distinct method sequences: 4 million
  - New errors revealed: 30
- Randoop
  - 30 new errors in 15 hours of human effort
  - 1 new error for ½ hour effort
- Existing team methods
  - 20 new errors per year
  - 1 new error for 100 hours human effort

Interacting with Randoop
Inspecting the resulting tests
Discarding redundant failures

Limitations

- Can generate spurious failures
  - Unaware of implicit pre-conditions*
- Weak assert statements
- Low coverage (< 50%) for reactive programs
  - Android apps, GUI applications
  - Not driven by coverage

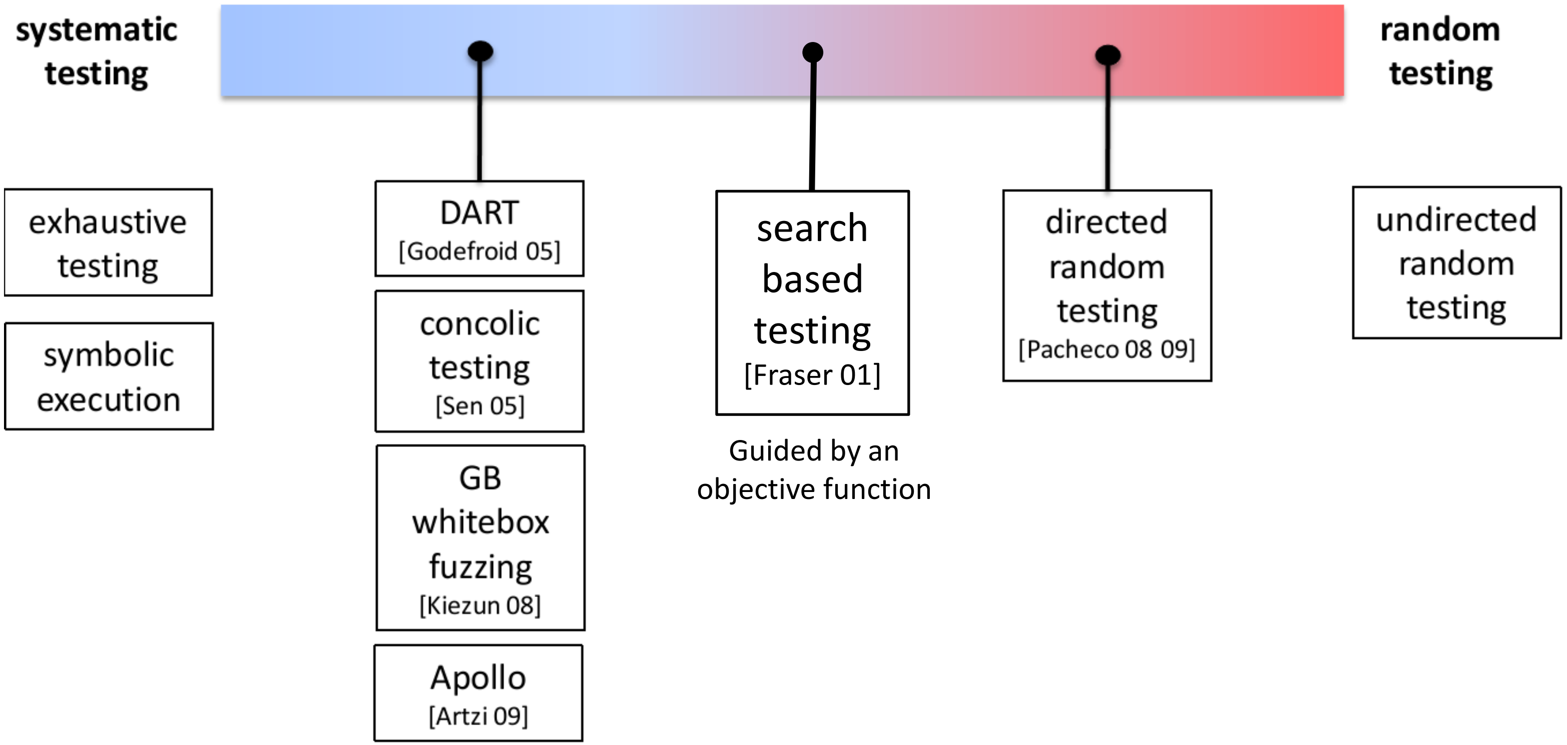


Random tests can generate spurious failures

geek & poke

*Mijung Kim, Shing-Chi Cheung, Sunghun Kim. Which Generated Test Failures Are Fault Revealing? Prioritizing Failures Based on Inferred Precondition Violations using PAF. In ESEC/FSE 2018.

Spectrum of Testing Techniques



systematic testing

random testing

exhaustive testing

symbolic execution

DART [Godefroid 05]

concolic testing [Sen 05]

GB whitebox fuzzing [Kiezun 08]

Apollo [Artzi 09]

search based testing [Fraser 01]

Guided by an objective function

directed random testing [Pacheco 08 09]

undirected random testing

Using Randoop

https://randoop.github.io/randoop/manual/index.html

Video: https://www.youtube.com/watch?v=nPdb-72-EJY

Further Readings

- Other popular random testing tools
  - Jubula for GUI testing (http://www.eclipse.org/jubula/)
  - Monkey & Stoat for Android software testing (http://developer.android.com/tools/help/monkey.html) (https://github.com/tingsu/Stoat)
  - Sapienz for Android random testing (https://github.com/Rhapsod/sapiens)
    - Facebook prototype to be replaced by an official release in 2019
- API invariance inference
  - Robillard, M.P.; Bodden, E.; Kawrykow, D.; Mezini, M.; Ratchford, T., "Automated API Property Inference Techniques," *IEEE Transactions on Software Engineering,* vol.39, no.5, pp.613-637, May 2013.