CS 5154
Automatic Test Generation

Spring 2021

Owolabi Legunsen

Announcements and Reminders

- Homework 3 has been released
  - More testing of Commons Math
  - Hope it helps with your course project

- I’ll announce times for project clinics (likely hold them next week)

- Keep working on your projects…

Testing: review of basic testing concepts

- **Test case:** Something used to test for bugs

  Input + expected output (i, eo)

- **Test oracle:** decides if a test passes

  expected output of a test.

  tells what correct output of test should be

- **Test suite:**

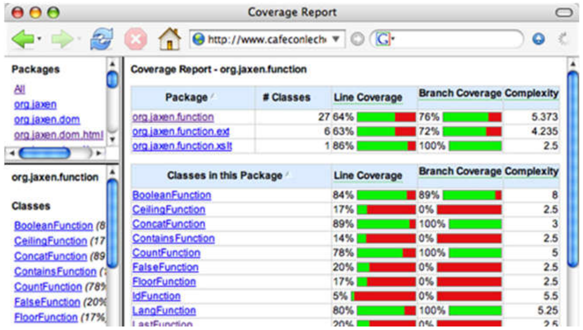
  Group of test cases (possibly related)

- **Test adequacy:**

  covering the criteria (100%)

Testing: basic concepts

- **Test case** (or, simply **test**): an execution of the software with a given test input, including:
  - Input values
  - Sometimes include execution steps
  - Expected outputs (**test oracle**)
- **Test suite**: a finite set of tests
  - Usually can be run together in sequence
- **Test adequacy**: a measurement to evaluate the test quality
  - Such as code coverage



Coverage Report - org.jaxen.function

Package	# Classes	Line Coverage	Branch Coverage	Complexity
org.jaxen.function	27	64%	76%	5.373
org.jaxen.function.ext	6	63%	72%	4.235
org.jaxen.function.xslt	1	86%	100%	2.5

Classes in this Package	Line Coverage	Branch Coverage	Complexity
BooleanFunction	84%	89%	8
CeilingFunction	17%	0%	2.5
ConcatFunction	89%	100%	3
ContainsFunction	14%	0%	2.5
CountFunction	78%	100%	5
FalseFunction	20%	0%	2.5
FloorFunction	17%	0%	2.5
IdFunction	5%	0%	5.5
LangFunction	80%	100%	5.25
LastFunction	20%	0%	2.5

Thought experiment

- Given a class C, how would you go about automatically creating a test suite for C?

```
public class HashSet extends Set{
  public boolean add(Object o){…}
  public boolean remove(Object o){…}
  public boolean isEmpty(){…}
  public boolean equals(Object o){…}
  ...
}
```

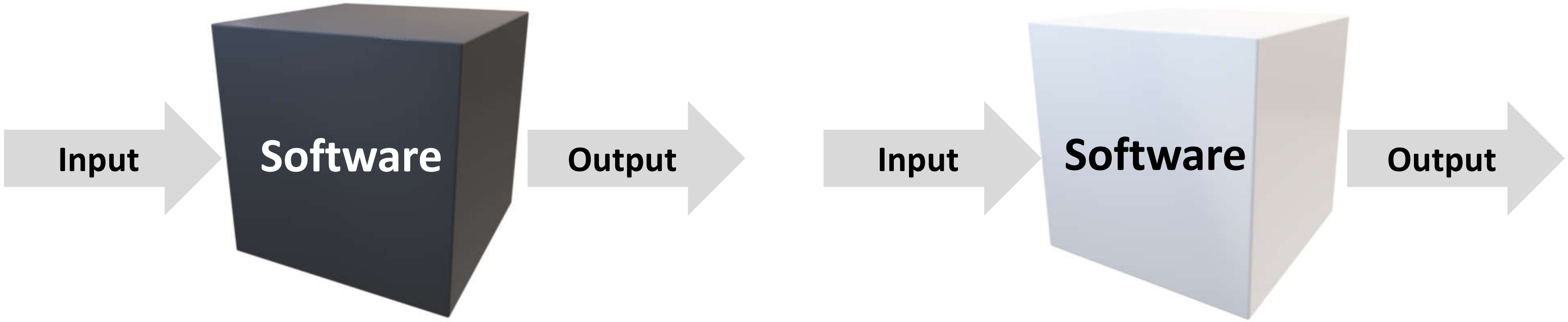
- Alternatively, what are the pieces that you need to create a test suite for C?

specification (what's expected)

input space

Types of test generation

- Functional vs. structural testing



Input Software Output

Input Software Output

- **Functional test generation**: generates tests based on the functionality of the program
- **Structural test generation**: generates tests based on the source-code structure of the program

Structural generation granularities

- Projects providing a number of public APIs for external use
  - **Method-level test generation**: consider various method invocation sequences to expose possible faults

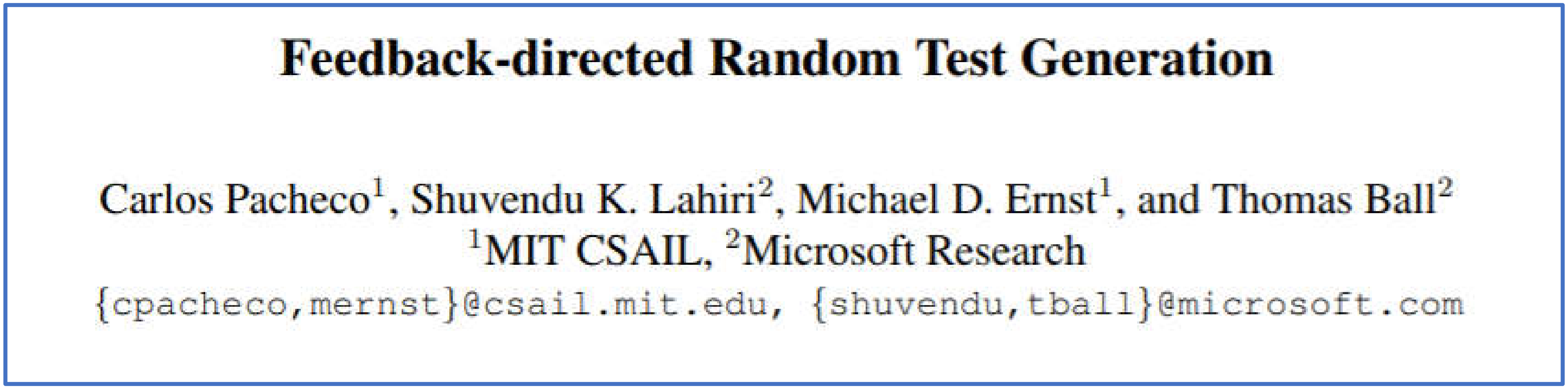
Guided unit test generation (this lecture and the next)

- Projects usually used as a whole
  - **Path-level generation**: consider all the possible execution paths to cover most program elements

Symbolic execution (??)

This lecture

- Feedback-directed Random Test Generation (ICSE'07)
  - The intuitions
  - The tool
  - Read the paper for more details!



Feedback-directed Random Test Generation

Carlos Pacheco[1], Shuvendu K. Lahiri[2], Michael D. Ernst[1], and Thomas Ball[2]
[1]MIT CSAIL, [2]Microsoft Research
{cpacheco,mernst}@csail.mit.edu, {shuvendu,tball}@microsoft.com

Problem: unit test generation

Program under test:

```
public class Math{
  public static int sum(int a, int b){
    return a+b;
  }
  …
}
```

Example JUnit test:

```
public class MathTest{
  @Test
  public void testSum (){
    int a=1;
    int b=1;
    int c=Math.sum(a, b);
    assertEquals(2,c);
  }
  …
}
```

Input values

Execution steps

Test oracle

Is this an important problem?

JIRA

THE APACHE SOFTWARE FOUNDATION

Commons-Math

84,377 lines of **source code** src/main

86,924 lines of **unit-test code** src/test

How to do random structural test generation?

```
public class HashSet extends Set{
  public boolean add(Object o){…}
  public boolean remove(Object o){…}
  public boolean isEmpty(){…}
  public boolean equals(Object o){…}
  ...
}
```

Program under test

- Need to generate a random sequence of invocations, where each requires
  - A random method
  - Some random parameters
  - A random receiver object
    - Not required for static methods

Generation

```
Set s = new HashSet();
s.add(“hi”);
```

Generated test *t1*

```
Set s = new HashSet();
s.add(“hi”);
s.remove(null);
```

Generated test *t2*

```
Set s = new HashSet();
s.isEmpty();
s.remove(“no”);
s.isEmpty();
s.add(“no”);
s.isEmpty();
s.isEmpty();
...
```

Generated test *t3*

…

Your turn…

- What are the limitations of random method-sequence generation?

  - oracle not generated

Random method-sequence generation: limitations

- Does not have test oracles
  - E.g., an ideal test oracle for the test below: **assertEquals(1, s.size())**
- Cannot generate complex tests
  - E.g., the parameters of some method invocations can be generated by other method invocations
- Can have many redundant or illegal tests

```
Set s = new HashSet();
s.isEmpty();
s.remove("no");
s.isEmpty();
s.add("no");
s.isEmpty();
s.isEmpty();
```

A random test

Random method-sequence generation: redundant and illegal tests

```
1. Useful test:
Set s = new HashSet();
s.add("hi");
```

```
2. Redundant test:
Set s = new HashSet();
s.add("hi");
s.isEmpty();
```

Should not output

```
3. Useful test:
Date d = new Date(2006, 2, 14);
```

```
4. Illegal test:
Date d = new Date(2006, 2, 14);
d.setMonth(-1); // pre: argument >= 0
```

Should not output

```
5. Illegal test:
Date d = new Date(2006, 2, 14);
d.setMonth(-1); // pre: argument >= 0
d.setDay(5);
```

Should not even generate

Randoop: feedback-directed (adaptive) random test generation

- Use code contracts as test oracles

- Build test inputs incrementally
  - New test inputs extend previous ones
  - In this context, a test input is a method sequence

- As soon as a test is created, use its execution results to guide generation
  - away from redundant or illegal method sequences
  - towards sequences that create new object states

Randoop input/output

- **Input**: TreeSet
  - Classes under test
  - Time limit
  - Set of contracts = assumptions
    - Method contracts (e.g. “o.hashCode() throws no exception”)
    - Object invariants (e.g. “o.equals(o) == true”)
- **Output**: contract-violating test cases

```
HashMap h = new HashMap();
Collection c = h.values();
Object[] a = c.toArray();
LinkedList l = new LinkedList();
l.addFirst(a);
TreeSet t = new TreeSet(l);
Set u = Collections.unmodifiableSet(t);
assertTrue(u.equals(u));
```

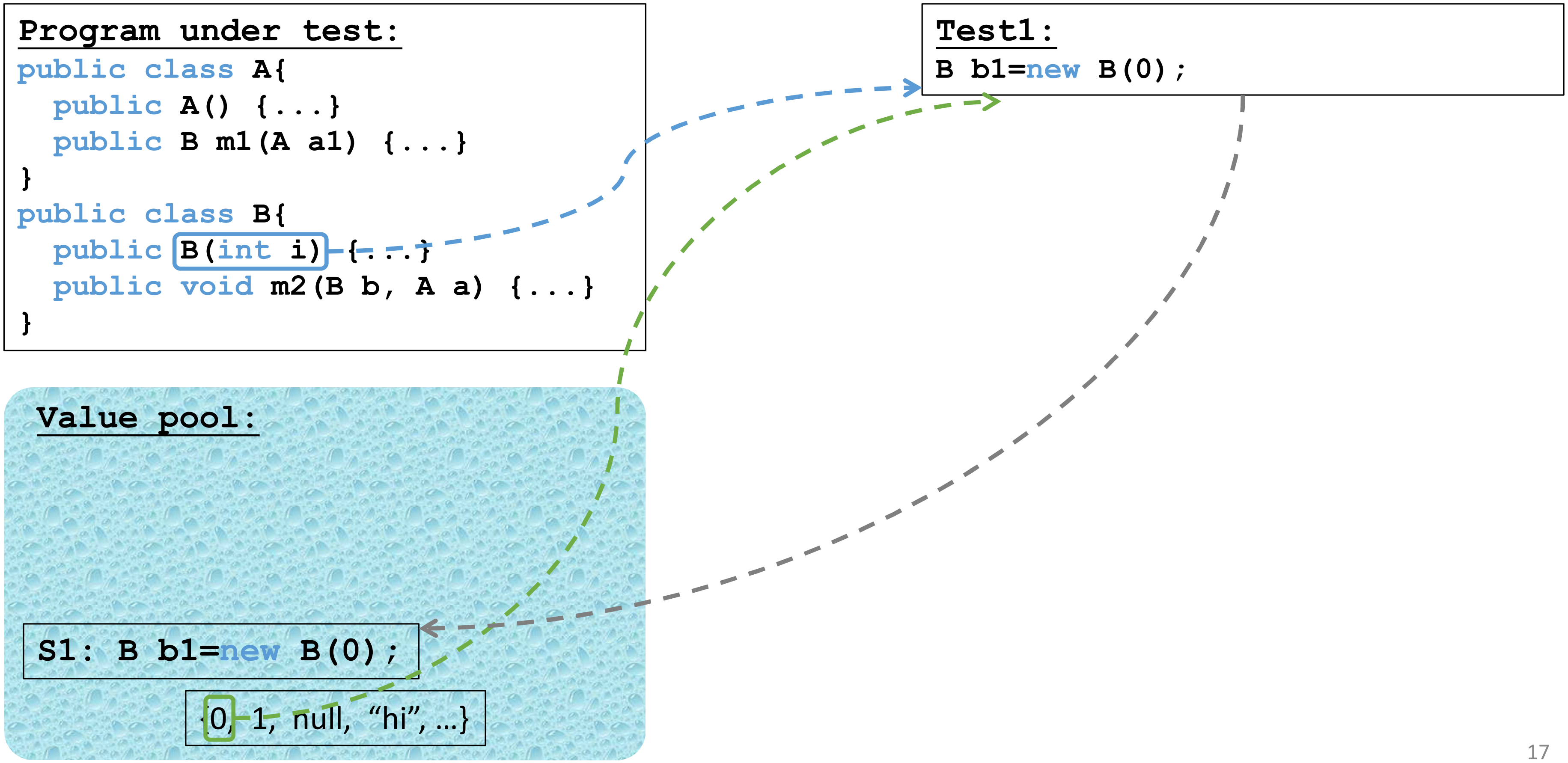
fails on Sun’s JDK 1.5/1.6 when executed

Randoop: algorithm

- Seed value pool for primitive types
    - pool = { **0, 1, true, false, “hi”, null** ... }
- Do until time limit expires:
    - Create a new sequence
        - Randomly pick a method call $\mathbf{m}(\mathbf{T_1}...\mathbf{T_k})/\mathbf{T_{ret}}$
        - For each input parameter of type $\mathbf{T_i}$, randomly pick a sequence $\mathbf{S_i}$ from the value pool that constructs an object $\mathbf{v_i}$ of type $\mathbf{T_i}$
        - Create new sequence $\mathbf{S_{new}} = \mathbf{S_1}$; ... ; $\mathbf{S_k}$; $\mathbf{T_{ret}}\ \mathbf{v_{new}} = \mathbf{m}(\mathbf{v_1}...\mathbf{v_k})$;
        - if $\mathbf{S_{new}}$ was previously created (lexically), go to first step
    - Classify the new sequence $\mathbf{S_{new}}$
        - May discard, output as test case, or add to pool

Method
Parameter
Receiver object

Randoop: example



```
Program under test:
public class A{
  public A() {...}
  public B m1(A a1) {...}
}
public class B{
  public B(int i) {...}
  public void m2(B b, A a) {...}
}
```

```
Test1:
B b1=new B(0);
```

```
Value pool:

S1: B b1=new B(0);
```

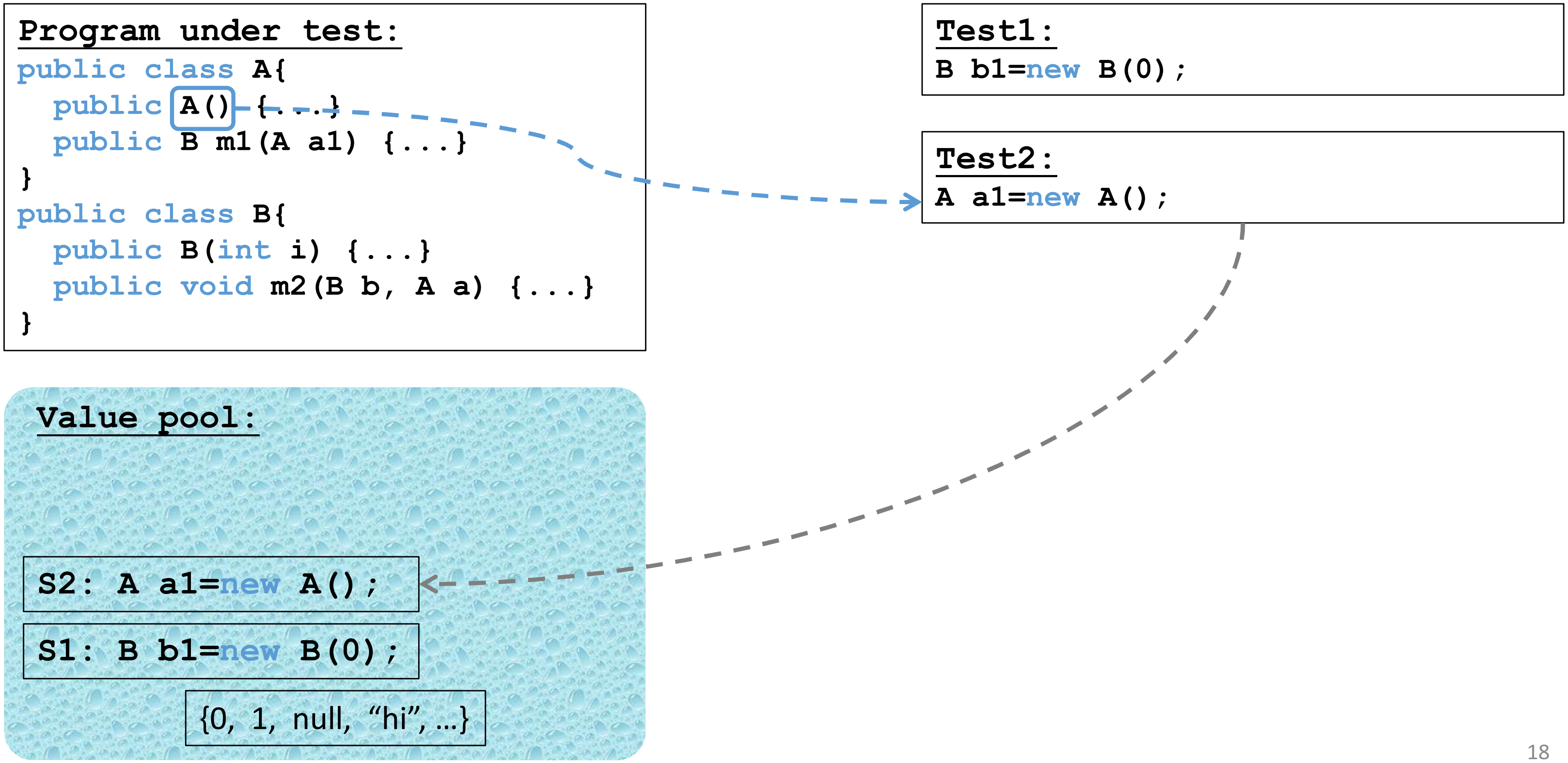
{0, 1, null, “hi”, …}

Method

Parameter

Receiver object

Randoop: example



Program under test:

```
public class A{
  public A() {...}
  public B m1(A a1) {...}
}
public class B{
  public B(int i) {...}
  public void m2(B b, A a) {...}
}
```

Test1:

```
B b1=new B(0);
```

Test2:

```
A a1=new A();
```

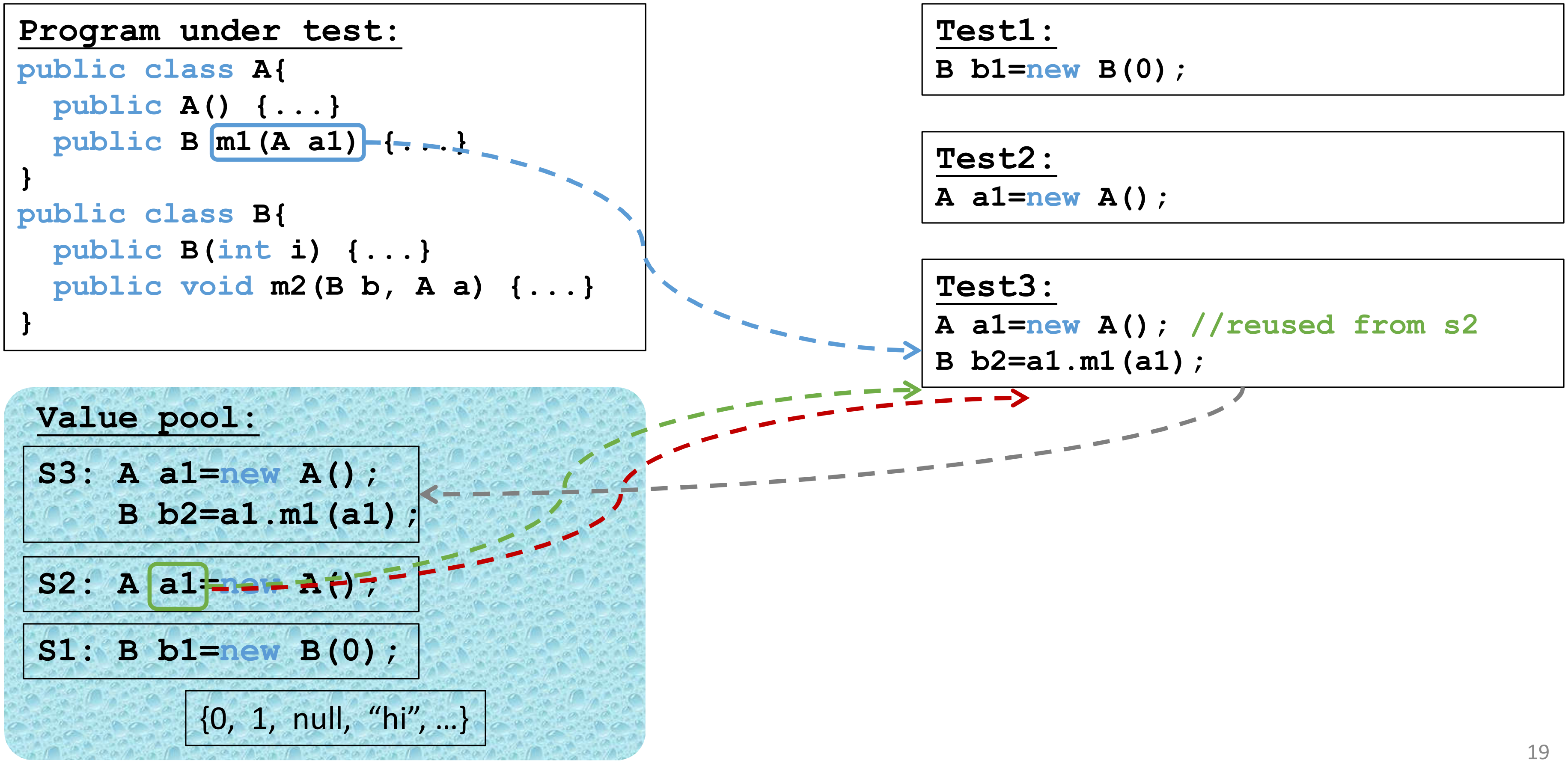
Value pool:

```
S2: A a1=new A();
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

⇢ Method
⇢ Parameter
⇢ Receiver object

Randoop: example



Program under test:

```
public class A{
  public A() {...}
  public B m1(A a1) {...}
}
public class B{
  public B(int i) {...}
  public void m2(B b, A a) {...}
}
```

Value pool:

```
S3: A a1=new A();
    B b2=a1.m1(a1);
```

```
S2: A a1=new A();
```

```
S1: B b1=new B(0);
```

{0, 1, null, “hi”, ...}

Test1:

```
B b1=new B(0);
```

Test2:

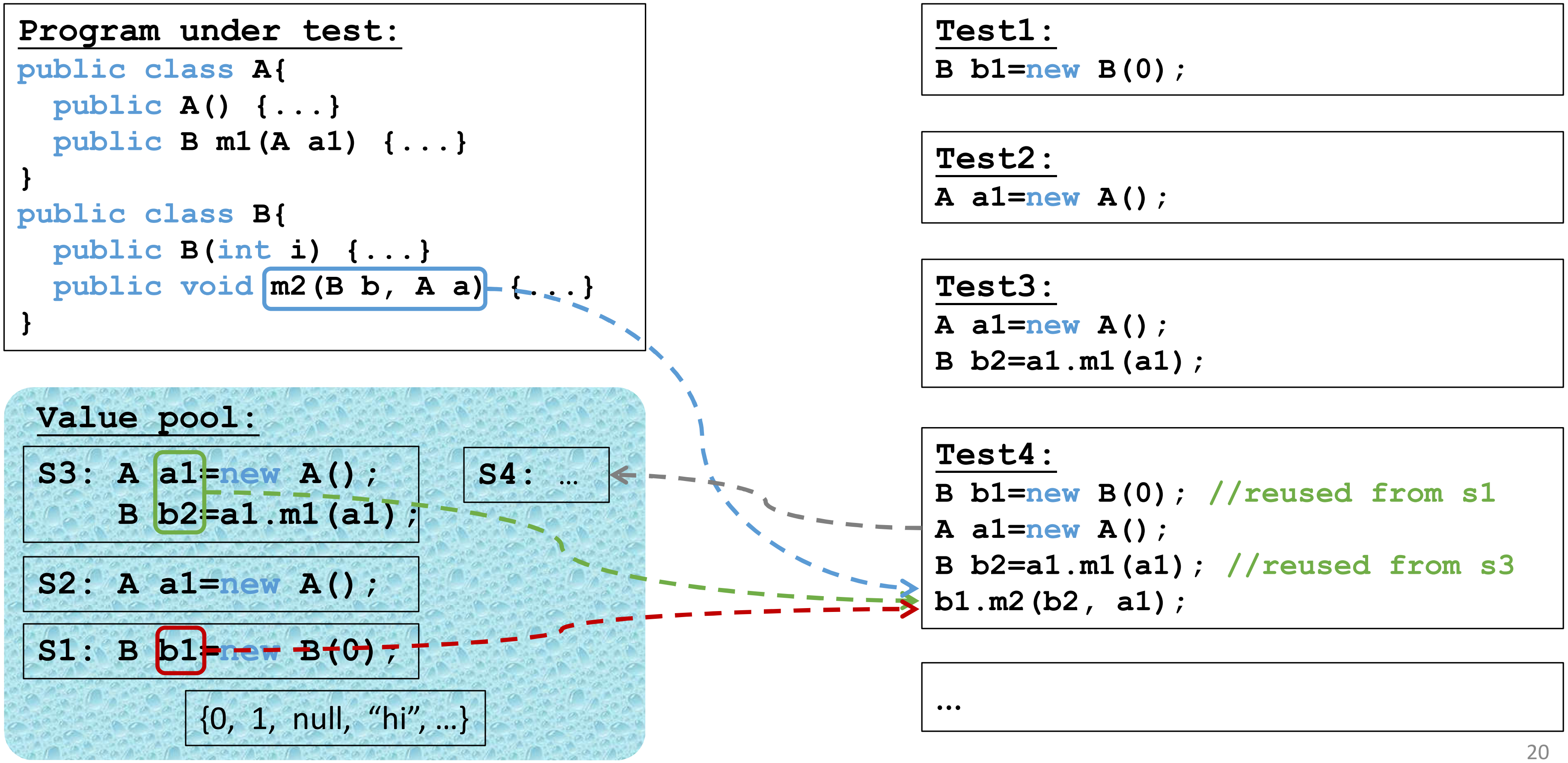
```
A a1=new A();
```

Test3:

```
A a1=new A(); //reused from s2
B b2=a1.m1(a1);
```

- - → Method
- - → Parameter
- - → Receiver object

Randoop: example



```
Program under test:
public class A{
  public A() {...}
  public B m1(A a1) {...}
}
public class B{
  public B(int i) {...}
  public void m2(B b, A a) {...}
}
```

```
Value pool:
S3: A a1=new A();
    B b2=a1.m1(a1);
S4: …
S2: A a1=new A();
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

```
Test1:
B b1=new B(0);
```

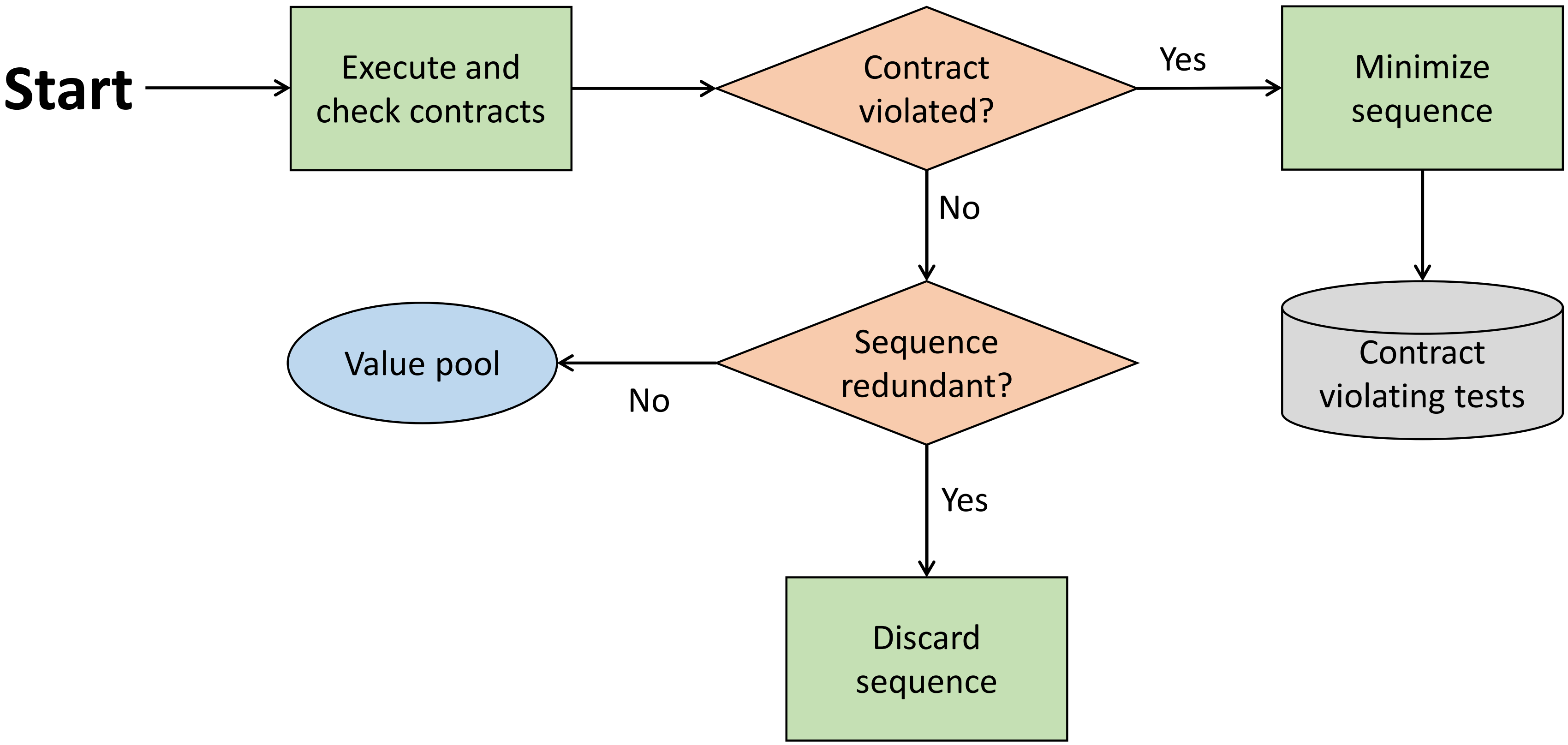
```
Test2:
A a1=new A();
```

```
Test3:
A a1=new A();
B b2=a1.m1(a1);
```

```
Test4:
B b1=new B(0); //reused from s1
A a1=new A();
B b2=a1.m1(a1); //reused from s3
b1.m2(b2, a1);
```

…

Classifying a sequence



Redundant sequences

- During generation, maintain a set of all objects created
- A sequence is redundant if all the objects created during its execution are members of the above set (using *equals* to compare)
- Could also use more sophisticated state equivalence methods
  - E.g. heap canonicalization

Tool support

- **Input**:
  - An assembly (for .NET) or a list of classes (for Java)
  - Generation time limit
  - Optional: a set of contracts to augment default contracts
- **Output**: a test suite (JUnit or Nunit) containing
  - Contract-violating test cases
  - Normal-behavior test cases

Randoop
Automatic unit test generation

Randoop outputs oracles

- Oracle for contract-violating tests:

```
Object o = new Object();
LinkedList l = new LinkedList();
l.addFirst(o);
TreeSet t = new TreeSet(l);
Set u = Collections.unmodifiableSet(t);
assertTrue(u.equals(u));//expected to fail
```

Find current bugs

- Oracle for normal-behavior tests (regression tests):

```
Object o = new Object();
LinkedList l = new LinkedList();
l.addFirst(o);
l.add(o);
assertEquals(2, l.size());//expected to pass
assertEquals(false,l.isEmpty());//expected to pass
```

Find future bugs

Some Randoop options

- Avoid use of null

```
Statically:
Object o = new Object();
LinkedList l = new LinkedList();
l.add(null);
```

```
Dynamically:
Object o = returnNull();
LinkedList l = new LinkedList();
l.add(o);
```

- Bias random selection
  - Favor shorter sequences
  - Favor methods that have been less covered
  - Use constants mined from source code
- Source code available:
  - https://randoop.github.io/randoop/

Code coverage by Randoop

Data structure programs	Time (s)	Branch cov.
Bounded stack (30 LOC)	1	100%
Unbounded stack (59 LOC)	1	100%
BS Tree (91 LOC)	1	96%
Binomial heap (309 LOC)	1	84%
Linked list (253 LOC)	1	100%
Tree map (370 LOC)	1	81%
Heap array (71 LOC)	1	100%

Bug detection by Randoop: subjects

Subjects	**LOC**	**Classes**
JDK (2 libraries) (java.util, javax.xml)	53K	272
Apache commons (6 libraries) (logging, primitives, chain, jelly, math, collections)	114K	974
.Net libraries (6 libraries)	615K	3455

Bug detection by Randoop: methodology

- Ran Randoop on each library
  - Used default time limit (2 minutes)
- Contracts:
  - **o.equals(o)==true**
  - **o.equals(o)** throws no exception
  - **o.hashCode()** throws no exception
  - **o.toString()** throw no exception
  - No null inputs and:
    - Java: No NPEs
    - .NET: No NPEs, out-of-bounds, of illegal state exceptions

Bug detection by Randoop: subjects

Subjects	Failed tests	Unique failed tests	Error-revealing tests	Distinct errors
JDK	613	32	29	8
Apache commons	3,044	187	29	6
.Net framework	543	205	196	196
Total	4,200	424	254	210

Errors found: examples

- JDK Collections classes have 4 methods that create objects violating **o.equals(o)** contract
- Javax.xml creates objects that cause **hashCode** and **toString** to crash, even though objects are well-formed XML constructs
- Apache libraries have constructors that leave fields unset, leading to NPE on calls of **equals**, **hashCode** and **toString** (this only counts as one bug)
- .Net framework has at least 175 methods that throw an exception forbidden by the library specification (NPE, out-of-bounds, of illegal state exception)
- .Net framework has 8 methods that violate **o.equals(o)**
- .Net framework loops forever on a legal but unexpected input

Has Randoop been compared to existing solutions?

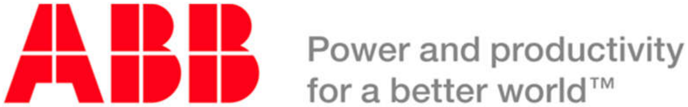
- Systematic testing:
  - Java PathFinder (JPF)
  - jCUTE
- Undirected Random testing:
  - Randoop-feedback
  - JCrasher

Regression testing scenario

- Randoop can create regression oracles
- Generated test cases using JDK 1.5
  - Randoop generated 41K regression test cases
- Ran resulting test cases on
  - JDK 1.6 Beta
    - 25 test cases failed
  - Sun’s implementation of the JDK
    - 73 test cases failed
  - Failing test cases pointed to 12 distinct errors
  - These errors were not found by the extensive compliance test suite that Sun provides to JDK developers

```
Object o = new Object();
LinkedList l = new LinkedList();
l.addFirst(o);
l.add(o);
assertEquals(2, l.size());//expected to pass
assertEquals(false,l.isEmpty());//expected to pass
```

Randoop: applications



Discussion

- Strengths
- Limitations
- Future work