CS 5154: Software Testing

Automated Test Generation

Instructor: Owolabi Legunsen

Fall 2021
Review of the six CS5154 themes

1. How to automate the execution of tests? ✓
2. How to design and write high-quality tests? ✓
3. How to measure the quality of tests? ✓
4. How to automate the generation of tests? ←
5. How to reduce the costs of running existing tests?
6. How to deal with bugs that tests reveal? [??]
Why care about automated test generation?

• You learned how to design and write high-quality tests

  • Hypothetical task: test a project with 80k lines of code in one week

  • Test suites can have more lines than code under test, e.g., hw0, hw1, hw2

  Commons-Math

  84,377 lines of source code

  86,924 lines of unit-test code

  How would you proceed?
On automated test suite generation

• Today: fundamental concepts, alternative approaches

• Next: hands-on demo
Testing: review of basic testing concepts

• Test case:

• Test oracle:

• Test suite:

• Test adequacy:
Testing: basic concepts

- **Test case** (or, **test**): executes the code under test and includes
  - Input values
  - Execution steps (most times)
  - Expected outputs

- **Test oracle**: compares observed and expected outputs

- **Test suite**: a finite set of tests
  - Usually, can be run together in sequence

- **Test adequacy**: a measurement of test quality
  - e.g., code coverage
Different approaches target these concepts

• Input value generation, e.g., fuzzing, symbolic execution

• Test suite generation, e.g., Randoop, EvoSuite

• Test oracle generation is very hard

• Test Adequacy: used to evaluate automated tests
Who is using automated test generation

• Randoop:
Who is using automated test generation? (2)

• Type in your favorite search engine:
  • “fuzzing at Google”
  • “fuzzing at Microsoft”
  • “fuzzing at Facebook”
  • “fuzzing at X”
Classes of test generation approaches

• Functional vs. structural test generation

• **Functional test generation** is based on the functionality of the code

• **Structural test generation** is based on the structure of the code
Structural generation granularity

- Projects providing public APIs for external use
  - **Method-level test generation**: consider various method invocation sequences to expose possible faults
- Projects usually used as a whole
  - **Path-level generation**: consider all the execution paths to cover most code elements

**Guided unit test generation** (this lecture and the next)

**Whole-suite test generation** (not covered this semester)
Thought experiment

• How would you go about automatically creating a test suite for class C?

```java
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?
Your thoughts
Recall: the components of a unit test

**Program under test:**
```java
public class Math{
    static int sum(int a, int b){
        return a+b;
    }
}
```

**Example JUnit test:**
```java
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**
How to do random structural test generation?

Program under test

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

Generated test t1

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

Generated test t2

```java
Set s = new HashSet();
s.add("hi");
s.remove(null);
```

Generated test t3

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

• Needed: generate a random sequence of invocations, each of which has
  • A random method
  • Some random parameters
  • A random receiver object
    • Not required for static methods
Your turn...

• What are some limitations of random method-sequence generation?
Random method-sequence generation: limitations

• Does not have test oracles
  • E.g., an ideal test oracle for the test below: `assertEquals(1, s.size())`

• Harder to generate complex tests
  • E.g., the parameters of some method invocations can only be generated by other method invocations

• Can have many redundant or illegal tests

```java
Set s = new HashSet();
s.isEmpty();
s.remove("no");
s.isEmpty();
s.isEmpty();
s.isEmpty();
```
Random method-sequence generation: redundant and illegal tests

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

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

3. Redundant test:
   Set s = new HashSet();
   s.add("hi");

   Should not output

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
We need something more than random

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

Feedback-directed Random Test Generation

Carlos Pacheco\textsuperscript{1}, Shuvendu K. Lahiri\textsuperscript{2}, Michael D. Ernst\textsuperscript{1}, and Thomas Ball\textsuperscript{2}
\textsuperscript{1}MIT CSAIL, \textsuperscript{2}Microsoft Research
\{cpacheco, mernst\}@csail.mit.edu, \{shuvendu, tball\}@microsoft.com
Randoop: feedback-directed random test generation

• Use code contracts as test oracles

• Build tests incrementally
  • new tests extend previous ones
  • in this context, a test 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: inputs and output

• **Input**: Classes under test, time limit, set of contracts
  • Method contracts (e.g. “o.hashCode() throws no exception”)
  • Object invariants (e.g. “o.equals(o) == true”)
  • User-written contracts

• **Output**: contract-violating or contract-preserving unit tests

```java
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
Some contracts that Randoop uses

- `o.equals(o)==true`
- `o.equals(o)` throws no exception
- `o.hashCode()` throws no exception
- `o.toString()` throws no exception
- No null inputs and No NPEs
Randoop: algorithm

1. Seed value pool for various types
   - pool = \{ 0, 1, true, false, “hi”, null ... \}

2. Do until time limit expires:
   a. Create a new sequence
      i. Randomly pick a method call \( m(T_1...T_k)/T_{ret} \)
      ii. For each input parameter of type \( T_i \), randomly pick a sequence \( S_i \)
          from the value pool that constructs an object \( v_i \) of type \( T_i \)
      iii. Create new sequence \( S_{new} = S_1; ... ; S_k; T_{ret} v_{new} = m(v_1...v_k); \)
      iv. If \( S_{new} \) was previously created (lexically), go to step i
   b. Classify new sequence \( S_{new} \): discard, output, or add to pool
Randoop: example

Program under test:
```java
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:
```java
B b1=new B(0);
```

Value pool:
```java
S1: B b1=new B(0);

{0, 1, null, “hi”, …}
```

[0, 1, null, “hi”, …]
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:
S1: B b1=new B(0); {0, 1, null, “hi”,...}
S2: A a1=new A();
Randoop: example

Program under test:

```java
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();
```

Test3:
```
A a1=new A(); //reused from s2
B b2=a1.m1(a1);
```

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”, …}
```
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();

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

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

Value pool:
{0, 1, null, “hi”, ...}
Classifying a sequence

Start

Execute and check contracts

Contract violated?

Yes

Minimize sequence

No

Sequence redundant?

Yes

Discard sequence

No

Value pool

Contract violating tests
Redundant sequences

1. During generation, maintain a set of all objects created

2. A sequence is redundant if all the objects created during its execution are members of the set in 1 (using equals to compare)

• One can also use more sophisticated state equivalence methods to compare, e.g., heap canonicalization
Randoop outputs oracles

• Oracle for contract-violating tests:

```java
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)); // assertion fails
```

Find current bugs

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

```java
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
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
Tool demo