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.e., expected output)

- **Test oracle:**
  - Decides if a test passes
  - Expected output of a test

- **Test suite:**
  - Tests what output of test should be
  - 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
Thought experiment

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

specifications (what's expected)
Types of test generation

- Functional vs. structural testing

  - **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\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

\texttt{\{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);
    }
    ...
}

Is this an important problem?

84,377 lines of source code
86,924 lines of unit-test code
How to do random structural test generation?

- 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

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

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();

**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

```java
Set s = new HashSet();
s.isEmpty();
s.remove("no");
s.isEmpty();
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); \(\text{pre: argument} \geq 0\)
   Should not output

5. Illegal test:
   Date d = new Date(2006, 2, 14);
   d.setMonth(-1); \(\text{pre: argument} \geq 0\)
   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:**
  - 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”)

• **Output:** contract-violating test cases

```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
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 \( m(T_1...T_k)/T_{ret} \)
    • 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 \( T_i \)
    • Create new sequence \( S_{new} = S_1; ... ; S_k ; T_{ret} v_{new} = m(v_1...v_k); \)
    • if \( S_{new} \) was previously created (lexically), go to first step
  • Classify the new sequence \( S_{new} \)
    • May discard, output as test case, or add to pool
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:
{0, 1, null, "hi", ...}
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);
```

Test2:

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

Value pool:

```java
{0, 1, null, "hi",...}
```
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);
```

Test2:

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

Test3:

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

Value pool:

```java
{0, 1, null, "hi",...}
```
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);
```

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

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

Test4:
```java
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

Start
- Execute and check contracts

Contract violated?
- Yes: Minimize sequence
- No: Value pool

Sequence redundant?
- No: Contract violating tests
- Yes: Discard 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 \textit{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 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)); // expected to fail
```

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
Some Randoop options

• Avoid use of null

  Specially:
  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

<table>
<thead>
<tr>
<th>Data structure programs</th>
<th>Time (s)</th>
<th>Branch cov.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounded stack (30 LOC)</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>Unbounded stack (59 LOC)</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>BS Tree (91 LOC)</td>
<td>1</td>
<td>96%</td>
</tr>
<tr>
<td>Binomial heap (309 LOC)</td>
<td>1</td>
<td>84%</td>
</tr>
<tr>
<td>Linked list (253 LOC)</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>Tree map (370 LOC)</td>
<td>1</td>
<td>81%</td>
</tr>
<tr>
<td>Heap array (71 LOC)</td>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>
## Bug detection by Randoop: subjects

<table>
<thead>
<tr>
<th>Subjects</th>
<th>LOC</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK (2 libraries) (java.util, javax.xml)</td>
<td>53K</td>
<td>272</td>
</tr>
<tr>
<td>Apache commons (6 libraries) (logging, primitives, chain, jelly, math, collections)</td>
<td>114K</td>
<td>974</td>
</tr>
<tr>
<td>.Net libraries (6 libraries)</td>
<td>615K</td>
<td>3455</td>
</tr>
</tbody>
</table>
Bug detection by Randoop: methodology

• Ran Randoop on each library
  • Used default time limit (2 minutes)

• Contracts:
  • \texttt{o.equals(o)==true}
  • \texttt{o.equals(o)} throws no exception
  • \texttt{o.hashCode()} throws no exception
  • \texttt{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

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Failed tests</th>
<th>Unique failed tests</th>
<th>Error-revealing tests</th>
<th>Distinct errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>613</td>
<td>32</td>
<td>29</td>
<td>8</td>
</tr>
<tr>
<td>Apache commons</td>
<td>3,044</td>
<td>187</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>.Net framework</td>
<td>543</td>
<td>205</td>
<td>196</td>
<td>196</td>
</tr>
<tr>
<td>Total</td>
<td>4,200</td>
<td>424</td>
<td>254</td>
<td>210</td>
</tr>
</tbody>
</table>
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

```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
```
Randoop: applications
Discussion

• Strengths
• Limitations
• Future work