Randomized Testing

Announcements

Lab3 Due Today

HW2 released

- Assertion Inference
- Optionally can work with a partner
- Due in 2 weeks from Wednesday (March 5th)

Outline

- Randomized Testing
- Randoop
- Benchmarks for evaluation

What have we learned so far?

- Program analysis is impossible
- Program transformations

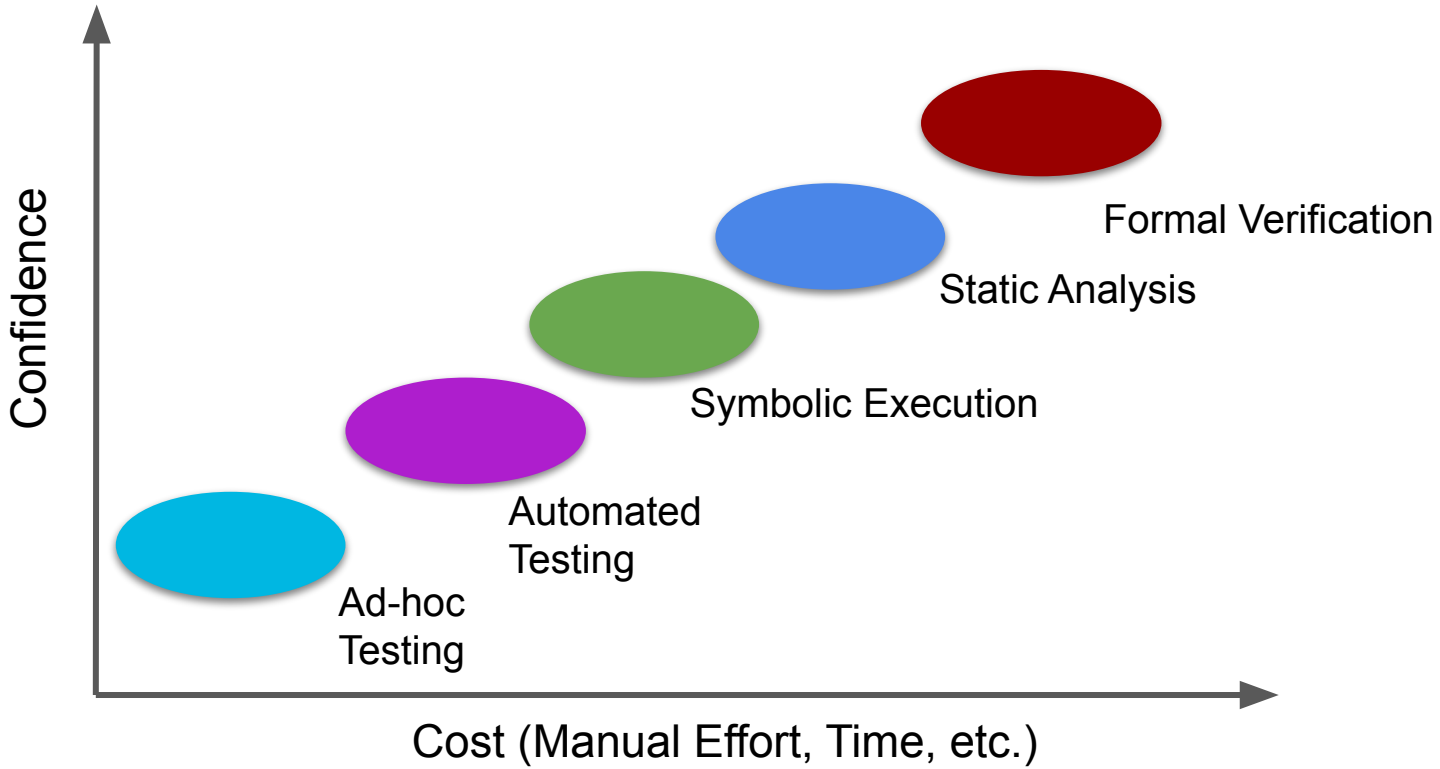
- How to evaluate a test suite
  - Coverage
  - Mutation testing

- Given a prefix, approaches to generate an assertion

- **Today we will learn how to generate the prefix**

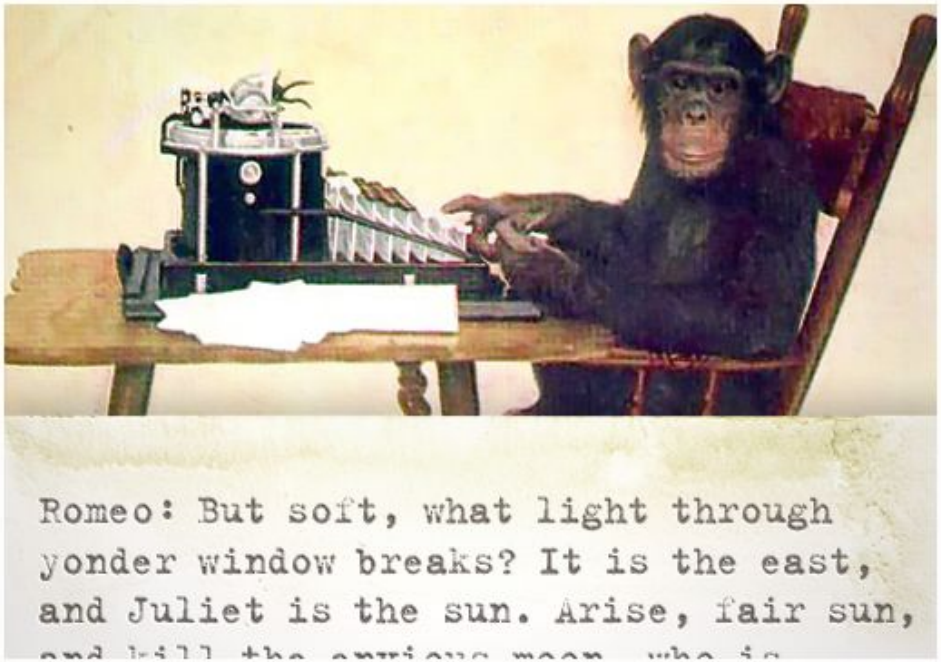
Automated Software Testing

Landscape of Program Analysis Techniques



The Infinite Monkey Theorem

"A monkey hitting keys at random on a typewriter keyboard will produce any given text, such as the complete works of Shakespeare, with probability approaching 1 as time increases."



Randomized Testing

```
public static double divide(int x, int y) {
    return x / y;
}
```

```
class Number {
    private double value;
    public Number(double value) {
        this.value = value;
    }
    public double getValue() {
        return value;
    }
    public Number divide(Number other) {
        return new Number(this.value / other.getValue());
    }
    public String toString() {
        return Double.toString(value);
    }
}
```

Randomized Testing for OOP (Java)

Idea: a test can be built up iteratively by randomly selecting a method or constructor to invoke, using previously computed values as inputs.

Problem with uniform random testing: Creates too many illegal or redundant tests

Random Testing: Pitfalls

1. Useful test

```
Set t = new HashSet();
s.add("hi");
assertTrue(s.equals(s));
```

2. Redundant test

```
Set t = new HashSet();
s.add("hi");
s.isEmpty();
assertTrue(s.equals(s));
```

3. Useful test

```
Date d = new Date(2006, 2, 14);
assertTrue(d.equals(d));
```

4. Illegal test

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

5. Illegal test

```
Date d = new Date(2006, 2, 14);
d.setMonth(-1);
d.setDay(5);
assertTrue(d.equals(d));
```

Randoop

Randoop

Automatic unit test generation for Java

To get around this, Randomly create new test guided by feedback from previously created tests

test == method sequence

Idea:

- Build new sequences incrementally, extending past sequences
- As soon as a sequence is created, execute it
- Use execution results to guide test generation towards sequences that create new object states

Randoop

Input:

- classes under test
- time limit
- set of contracts

e.g. “o.hashCode() throws no exception”
e.g. “o.equals(o) == true”

Output:

- contract-violating test cases

```
LinkedList l1 = new LinkedList();
Object o1 = new Object();
l1.addFirst(o1);
TreeSet t1 = new TreeSet(l1);
Set s1 = Collections.unmodifiableSet(t1);

assert(s1.equals(s1));
```

No contract violated up to here

fails when executed

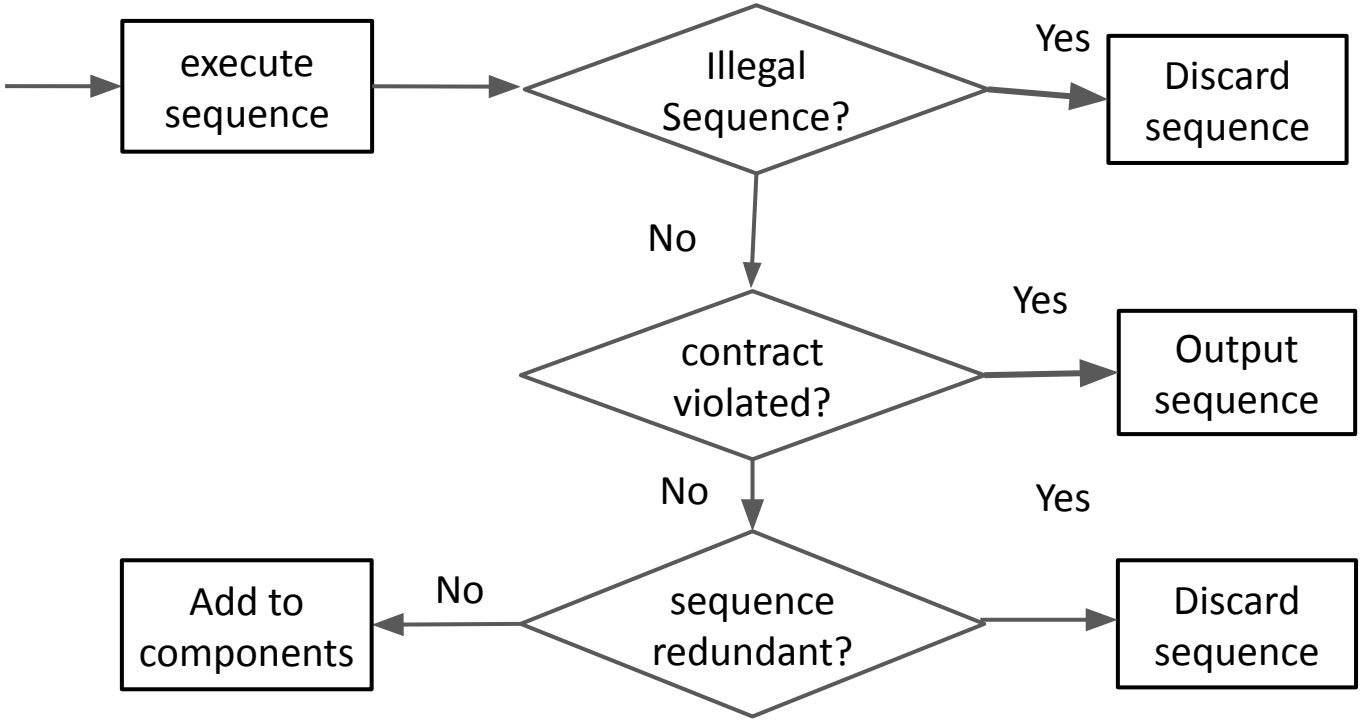
Randoop Algorithm

components = { int i = 0; boolean b = false; . . . }

Repeat until time limit expires:

- Create a new sequence
  - Randomly pick a method call T_{ret} `m(`T_1`,...,`T_n`)`
  - For each argument of type T_i, randomly pick sequence S_i from components that constructs an object v_i of that type
  - Create S_{new} = S_1; ...; S_n; T_{ret} v_{new} = `m(`v_1`,...,`v_n`);`
- Classify new sequence S_{new}:
  - discard / output as test / add to components

Classifying a Sequence



Classifying a Sequence as Illegal

4. Illegal test

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

5. Illegal test

```
Date d = new Date(2006, 2, 14);
d.setMonth(-1);
d.setDay(5);
assertTrue(d.equals(d));
```

A **precondition** is a condition that must be true before method execution. A **precondition violation** usually leads to an exception

Classifying a Sequence as a Contract Violation

```
o.equals(o) == true
o1.equals(o2) == o2.equals(o1)
If (o1.equals(o2)) then o1.hashCode() == o2.hashCode()
   ...
```

Classifying a Sequence as Redundant

- 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)

Exercise:

1. Write the smallest sequence Randoop can generate to create a valid `BinaryTree`
   a. Hint: consider a default value for objects

2. Once generated, how does Randoop classify it?
   a. Discard as illegal
   b. Output as Contract Violating bug
   c. Add to components for future extension

3. Extend the sequence you created in (1) to violate the assertion in removeRoot(). How would the sequence be classified?

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
     root = r;
  }
  public Node removeRoot() {
     assert(root != null);
     ...
  }
}
```

```
class Node {
  Node left;
  Node right;
  public Node(Node l, Node r) {
     left = l; right = r;
  }
}
```

A Note on Assertions

- Major weakness of randoop!
- Can only find safety properties
- Also has a mode for regression oracles
  - How are regression oracles generated?

Benchmarks for Evaluation

How do we evaluate if an approach generates good test suites?

1. Coverage
    a. Branch Coverage

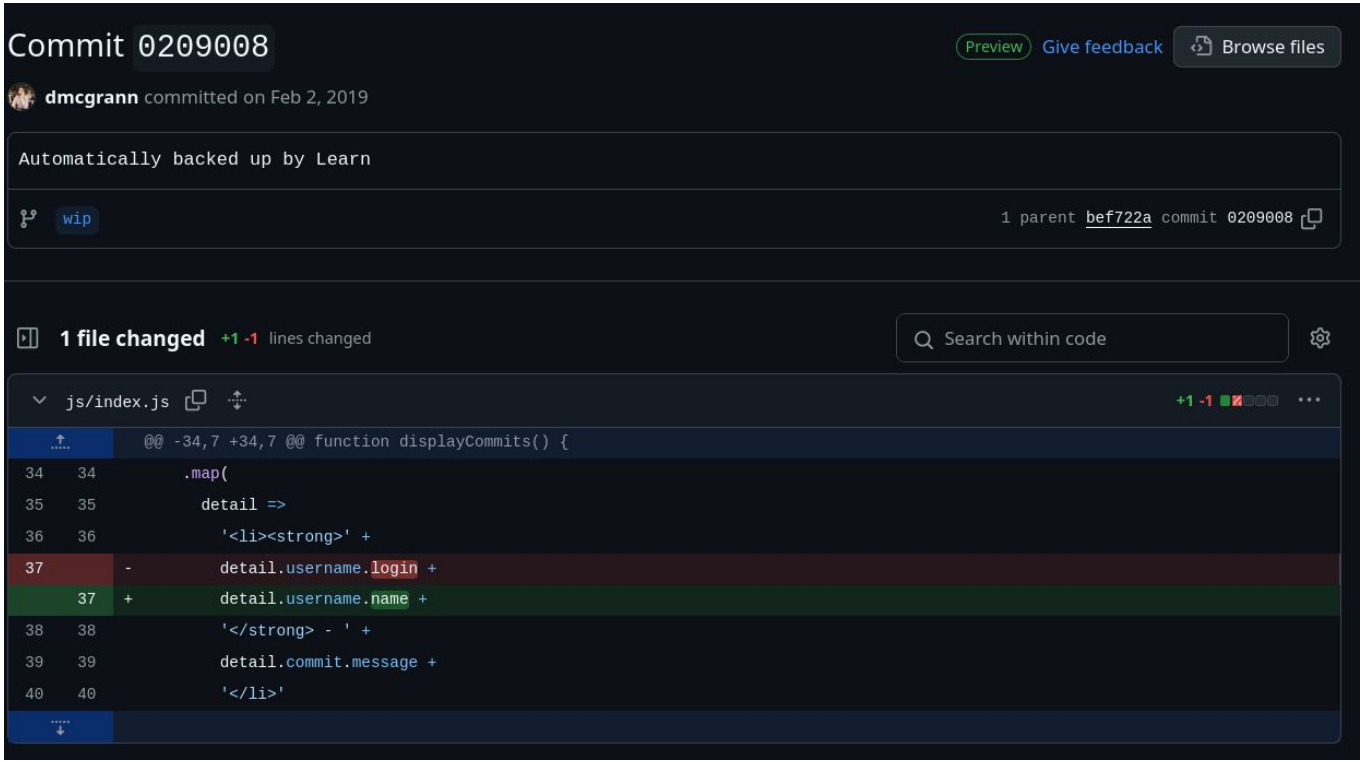
2. Assertion
    a. Mutation Testing

3. Does it find bugs?
    a. To do this we need a *benchmark* of bugs

Defects4J

- Collection of reproducible bugs to evaluate testing
- 854 bugs from open source projects
- Includes a “patch” or fix
- Contains a “triggering test” that fails before the fix and passes after the fix

Source Code Version Control



Source Code Version Control

- **Patch**: a set of changes that moves the code from the “before” version to the “after” version
  - Can involve modifying, adding, or removing lines of code

- **Checkout**: the process of switching between different versions of the code

Summary

- Randomized Testing
  - OOP: randomly select method to call. For arguments, use objects of that type that you have already created.
  - Problem: leads to redundant or illegal sequences
  - Solution: only keep legal and non-redundant sequences to extend

- Defects4J is a benchmark for evaluation of testing tools

- Lab today: running Randoop on Defects4J bugs