Program Testing and Analysis: Random and Fuzz Testing

Dr. Michael Pradel
Software Lab, TU Darmstadt
Outline

- **Feedback-directed random test generation**
  Based on *Feedback-Directed Random Test Generation*, Pacheco et al., ICSE 2007

- **Adaptive random testing**
  Based on *ARTOO: Adaptive Random Testing for Object-oriented Software*, Ciupa et al., ICSE 2008

- **Fuzz testing**
  Based on *Fuzzing with Code Fragments*, Holler et al., USENIX Security 2012
Motivating Examples

Two randomly generated tests:

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

Set s = new HashSet();
s.add("hi");
s.isEmpty();
s.isEmpty();
assertTrue(s.equals(s));
Motivating Examples

Two randomly generated tests:

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

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

s.isEmpty();
assertTrue(s.equals(s));

Only difference
Motivating Examples

Two randomly generated tests:

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

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

Redundant test
Motivating Examples (2)

Three randomly generated tests:

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

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

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

Three randomly generated tests:

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

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

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

Violates pre-condition
Motivating Examples (2)

Three randomly generated tests:

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

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

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

Illegal tests
Feedback-directed Test Generation

Idea: **Guide** randomized **creation** of new test inputs by **feedback** about **execution** of previous inputs

- Avoid redundant inputs
- Avoid illegal inputs

- Test input here means **sequence of method calls**
- Software under test: Classes in Java-like language
Approach

■ Build test inputs **incrementally**
  □ New test inputs extend previous ones

■ **As soon as test input is created, execute it**

■ Use execution results to guide generation
  □ away from redundant or illegal method sequences
  □ toward sequences that create **new object states**
Randoop: Implementation of feedback-directed random test generation

- **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:** Test cases with assertions
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));
Example

```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 when executed
Example

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));

No contracts violated up to last method call

Fails when executed
Algorithm

1. Initialize seed components: \( i=0; \ b=false; \ldots \)

2. Do until time limit expires:
   - Create a new sequence
     - Randomly pick a method \( T_0.m(T_1, \ldots, T_k)/T_{ret} \)
     - For each \( T_i \), randomly pick a sequence \( S_i \) from the components that constructs a value \( v_i \) of type \( T_i \)
     - Create new sequence
       \[ S_{new} = S_1; \ldots; S_k; T_{ret} v_{new} = m(v_1, \ldots, v_k); \]
     - If \( S_{new} \) was previously created (lexically), go to
   - Classify the sequence \( S_{new} \)
     - May discard, output as test case, or add to components
Classifying a Sequence

Image source: Slides by Pacheco et al.
Redundant Sequences

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

- **Sequence is redundant** if all objects created during its execution are in the above set (using `equals()` to compare).

- Could also use more sophisticated state equivalence methods
  - E.g., heap canonicalization