Program Testing and Analysis: Random and Fuzz Testing (Part 2)

Dr. Michael Pradel
Software Lab, TU Darmstadt
What does the following code print?

```javascript
function f(a,b) {
    var x;
    for (var i = 0; i < arguments.length; i++) {
        x += arguments[i];
    }
    console.log(x);
}

f(1,2,3);
```

3  6  NaN  Nothing
What does the following code print?

```javascript
function f(a,b) {
    var x;
    for (var i = 0; i < arguments.length; i++) {
        x += arguments[i];
    }
    console.log(x);
}

f(1,2,3);
```

3 6 NaN Nothing
What does the following code print?

```javascript
function f(a,b) {
    var x;
    for (var i = 0; i < arguments.length; i++) {
        x += arguments[i];
    }
    console.log(x);
}
f(1,2,3);
```

Array-like object that contains all three arguments

3 6 NaN Nothing
What does the following code print?

```javascript
function f(a, b) {
    var x; /* Initialized to undefined */
    for (var i = 0; i < arguments.length; i++) {
        x += arguments[i]; /* undefined + some number yields NaN */
    }
    console.log(x);
}

f(1, 2, 3);
```

3 6  NaN  Nothing
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
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
Algorithm

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

2. Do until time limit expires:
   - Create a new sequence
     - Randomly pick a method \( T_0.m(T_1, ..., T_k)/T_{ret} \)
     - 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
       \[
       S_{new} = S_1; \ldots; S_k; T_{ret} \ v_{new} = m(v_1, ..., v_k);
       \]
     - If \( S_{new} \) was previously created (lexically), go to
   - Classify the sequence \( S_{new} \)
     - May discard, output as test case, or add to components
Classes under test: `java.util.*`

1) Pick a method
   → No values needed
   → New sequence

2) Classify sequence
   → no contract violated
   → not redundant
   → add to components

```java
new HashMap
```

```
HashMap h = new HashMap();
```
3) Pick method new HashMap

→ HashMap h2 = new HashMap()

4) Classify sequence
   → no contract violated
   → redundant

→ discard sequence
5) Pick method \textit{HashMap} values
   \begin{itemize}
   \item Need sequence that constructs value of type \textit{HashMap}
   \item Use sequence from step 2)
   \end{itemize}

6) Create sequence
   \begin{verbatim}
   HashMap h = new HashMap()
   Collection c = h.values()
   \end{verbatim}
   \begin{itemize}
   \item Classify: no contract violated not redundant
   \item Add to component
   \end{itemize}
Test Oracles

- Testing only useful if there is an oracle
- Randoop outputs tests with two kinds of oracles
  - Oracle for contract-violating test cases:
    ```java
    assertTrue(u.equals(u));
    ```
  - Oracle for normal-behavior test cases:
    ```java
    assertEquals(2, l.size());
    assertEquals(false, l.isEmpty());
    ```
Quiz

Which of these tests may be created by Randoop?

Test 1:
```
LinkedList l = new LinkedList();
l.add(23);
```

Test 2:
```
LinkedList l = new LinkedList();
l.get(-5);
```

Test 3:
```
LinkedList l = new LinkedList();
l.add(7);
assertEquals(l.getFirst(), 7);
```
Quiz

Which of these tests may be created by Randoop?

Test 1: \[
\text{LinkedList } l = \text{new LinkedList();}
l.\text{add(23); (oracle missing)}
\]

Test 2: \[
\text{LinkedList } l = \text{new LinkedList();}
l.\text{get(-5); (crashes)}
\]

Test 3: \[
\text{LinkedList } l = \text{new LinkedList();}
l.\text{add(7);}
\text{assertEquals(l.getFirst(), 7);}
\]
Results

- Applied to data structure implementations and popular library classes
- Achieves 80-100% basic block coverage
- Finds various bugs in JDK collections, classes from the .NET framework, and Apache libraries

Read Pacheco et al.’s paper for details
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
Adaptive Random Testing

Idea: Testing is more effective when inputs are spread evenly over the input domain

- Generate candidate inputs randomly
- At every step, select input that is furthest away from already tested inputs
Initially proposed for **numeric values**

- Distance between two values: **Euclidean distance**

**Example:** `f(int x)`

- Suppose to have tested with `Integer.MAX_VALUE` and `Integer.MIN_VALUE`
- Next test: 0
Spread Out Evenly?

- Initially proposed for numeric values
  - Distance between two values: Euclidean distance

- Example: f(int x)
  - Suppose to have tested with `Integer.MAX_VALUE` and `Integer.MIN_VALUE`
  - Next test: 0

Challenge:
How to compute distance of objects?
Adaptive random testing

Space of possible inputs

O... already tested
Object Distance

- Measure **how different two objects are**
- Object: Primitive values, dynamic type, and non-primitive values recursively referred to

\[
\text{dist}(p, q) = \text{combination}( \\
\text{elementaryDistance}(p, q), \\
\text{typeDistance}(\text{type}(p), \text{type}(q)), \\
\text{fieldDistance}(\{\text{dist}(p.a, q.a) \mid a \in \text{fields}(\text{type}(p) \cap \text{fields}(\text{type}(q)))\}))
\]
Object Distance

- Measure how different two objects are
- Object: Primitive values, dynamic type, and non-primitive values recursively referred to

\[ \text{dist}(p, q) = \text{combination}( \]

- \[ \text{elementaryDistance}(p, q), \]
- \[ \text{typeDistance}(\text{type}(p), \text{type}(q)), \]
- \[ \text{fieldDistance}(\{\text{dist}(p.a, q.a) \mid a \in \text{fields}(\text{type}(p) \cap \text{fields}(\text{type}(q)))\}) \]

Does not require traversing the object
Object Distance

- **Measure** how different two objects are
- **Object**: Primitive values, dynamic type, and non-primitive values recursively referred to

\[
\text{dist}(p, q) = \text{combination}(
\text{elementaryDistance}(p, q), \text{typeDistance(type}(p), type(q)), \text{fieldDistance}\left(\{ \text{dist}(p.a, q.a) \mid a \in \text{fields(type}(p) \cap \text{fields(type}(q))) \}\right))
\]
Fixed functions for each possible type:

- For **numbers**: \( F(|p - q|) \), where \( F \) is a monotonically non-decreasing function with \( F(0) = 0 \)
- For **characters**: 0 if identical, \( C \) otherwise
- For **booleans**: 0 if identical, \( B \) otherwise
- For **strings**: the Levenshtein distance
- For **references**: 0 if identical, \( R \) if different but none is null, \( V \) if only one of them is null

\[
C, B, R, V \in \mathbb{N}
\]
Examples: Elementary Distance

- int i=3, j=9 → dist (13-91) = 6
- char c='a', d='a' → dist (c, d) = 0
- String s="foo", t="too" → dist(s, t) = 1
- Object o = null, p = new ArrayList() → dist (o, p) = R
Type Distance

Distance between two types

\[
\text{typeDistance}(t, u) = \\
\lambda \ast \text{pathLength}(t, u) \\
+ \nu \ast \sum_{a \in \text{nonShared}(t, u)} \text{weight}_a
\]

- \( \text{pathLength}(t, u) \) is the minimal distance to a common ancestor in class hierarchy
- \( \text{nonShared}(t, u) \) is the set of non-shared fields
- \( \text{weight}_a \) is the weight for a specific field

\( \lambda, \nu \in \mathbb{N} \)
Examples:  

Type Distance

\[
\text{dist} (B, C) = \lambda \cdot 1 + \nu \cdot (n+1) \\
\text{dist} (A, B) = \lambda \cdot 0 + \nu (1)
\]
Field Distance

Recursively compute distance of all shared fields

\[ \text{fieldDistance}(p, q) = \sum_a \text{weight}_a \times (\text{dist}(p.a, q.a)) \]

Arithmetic mean: Avoid giving too much weight to objects with many fields
Algorithm for Selecting Inputs

- Global sets \textit{usedObjects} and \textit{candidateObjects}
- Choose object for next test:
  - Initialize $\textit{bestDistSum} = 0$ and $\textit{bestObj} = \textit{null}$
  - for each $c \in \textit{candidateObjects}$:
    - for each $u \in \textit{usedObjects}$:
      - $\textit{distSum} += \text{dist}(c, u)$
    - if $\textit{distSum} > \textit{bestDistSum}$:
      - $\textit{bestDistSum} = \textit{distSum}; \textit{bestObj} = c$
  - Remove $\textit{bestObj}$ from $\textit{candidateObjects}$, add to $\textit{usedObjects}$ instead, and run test with $\textit{bestObj}$
Example

Method under test:

Account.transfer(Account dst, int amount)

Pool of candidates:

- Accounts
  - a1: owner=’A’ and balance=6782832
  - a2: owner=’B’ and balance=10
  - a3: owner=’O’ and balance=99
  - a4: null

- Integers:
  - i1: 100, i2: 287391, i3: 0, i4: -50
Example (cont.)

First call:
\texttt{a3.transfer(a1, i2)}
Example (cont.)

First call:
\[ a_3 \text{.transfer}(a_1, i_2) \]

Second call:
\[ a_1 \text{.transfer}(a_4, i_4) \]
Results

- Implemented for Eiffel
- Use randomly generated objects as candidates
- Use Eiffel’s contracts (pre- and post-conditions, class invariants) as test oracle

Comparison with random testing:
- Find bugs with 5x fewer tests
- But: Takes 1.6x the time of random testing
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
Fuzz Testing

Generate random inputs

- **Generative**: Create new random input, possibly based on constraints and rules
- **Mutative**: Derive new inputs from existing input by randomly modifying it
Grammar-based Language Fuzzing

Idea: Combine generative and mutative approach to test JavaScript interpreter

- Create random JavaScript programs based on language grammar
- Use and re-combine fragments of code from existing corpus of programs
  - Corpus: Programs that have exposed bugs before
Overview of LangFuzz

Phase I
Learning code fragments from sample code and test suite

Phase II
LangFuzz generated (mutated) test cases

Phase III
Feed test case into interpreter, check for crashes and assertions
Learning Code Fragments

- Parse existing programs into ASTs
- Extract code fragments
  - Examples for non-terminals of grammar
Corpus:

expressions $E$

command $C$

program $P$
Mutation of Code

- Randomly pick and parse an existing program
- Randomly pick some fragments and replace with fragments from phase 1 of same type
if \( x > 4 \) then \( y := !x \) else ship

\[
\begin{cases}
\text{if } x > 4 \text{ then } y := !x \text{ else } x := !y
\end{cases}
\]

command from corpus
Generation of Code

**Breadth-first application of grammar rules**

- Set current expansion $e_{cur}$ to start symbol $P$
- Loop $k$ iterations:
  - Choose a random non-terminal $n$ in $e_{cur}$
  - Pick one of the rules, $r$, that can be applied to $n$
  - Replace occurrence of $n$ in $e_{cur}$ by $r(n)$

*After $k$ iterations:* Replace remaining non-terminals with fragments
Code Generation: Example

\[
\begin{align*}
\text{l} & = 3 \\
\text{e}_{\text{cur}} & = P \\
\text{e}_{\text{cur}} & = C \\
\text{e}_{\text{cur}} & = \text{while } B \text{ do } C \\
\text{e}_{\text{cur}} & = \text{while } B \text{ do } C; C \\
\text{p} & ::= C 1 \\
\text{c} & ::= \ldots | \text{while } B \text{ do } C 1 \\
\text{c} & ::= \ldots | C; C 1 \\
\end{align*}
\]

Replace with fragments learned from corpus
Quiz

Which of the following SIMP programs could have been generated by LangFuzz?

`if B then C; C`

`if !x > 3 then skip; y := 1`

`if !x > 3 then while; while`
Quiz

Which of the following SIMP programs could have been generated by LangFuzz?

- `if B then C; C` (has unexpanded non-terminals)
- `if !x > 3 then skip; y := 1` (syntactically incorrect)
- `if !x > 3 then while; while`
Results

- Used to test Mozilla’s and Chrome’s JavaScript engines
- Found various bugs
  Mostly crashes of engine due to memory issues
- Rewarded with $50,000 bug bounties
- First author now works at Mozilla
Summary

Random and fuzz testing

- Fully automated and unbiased
- Non-naive approaches can be very effective
- Trade-off: Cost of generating inputs vs. effectiveness in exposing bugs
  - Quickly generated, less effective tests may be better than slowly generated, more effective tests