Program Testing and Analysis: Random and Fuzz Testing

Prof. Dr. Michael Pradel

Software Lab, TU Darmstadt

Warm-up Quiz

What does the following code print?

```
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**

Warm-up Quiz

What does the following code print?

```
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**

Warm-up Quiz

What does the following code print?

```
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**

Warm-up Quiz

What does the following code print?

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

Initialized to `undefined`

`undefined` + some number yields `NaN`

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

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

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`

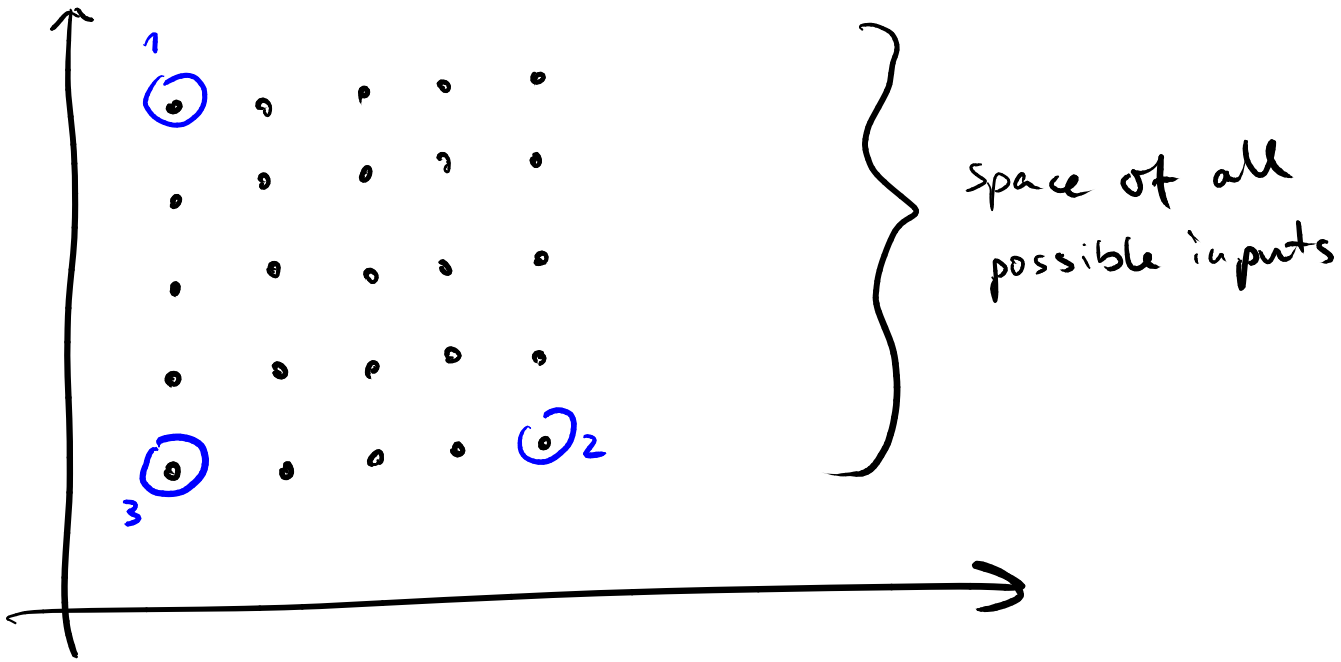
Spread Out Evenly?

- **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



Object Distance

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

$$
\begin{aligned}
dist(p,q) = & combination(\\
& elementaryDistance(p,q), \\
& typeDistance(type(p), type(q)), \\
& fieldDistance(\{dist(p.a, q.a) \mid \\
& a \in fields(type(p) \cap fields(type(q)))\})
\end{aligned}
$$

Object Distance

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

Does not require traversing the object

$$
\begin{aligned}
dist(p,q) = & combination(\\
& \boxed{elementaryDistance(p,q),} \\
& typeDistance(type(p), type(q)), \\
& fieldDistance(\{dist(p.a, q.a) \mid \\
& a \in fields(type(p) \cap fields(type(q)))\})
\end{aligned}
$$

Object Distance

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

$$dist(p,q) = combination($$
$$elementaryDistance(p,q),$$
$$typeDistance(type(p), type(q)),$$
$$fieldDistance(\{dist(p.a, q.a) \mid$$
$$a \in fields(type(p) \cap fields(type(q)))\})$$

Recursively defined

Elementary Distance

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 $i = 3$, $j = 9$ $\longrightarrow$ $dist(i, j) = F(|3-9|) = 6$
  ↑ if F is identity fct.
- char $c = 'a'$, $d = 'a'$ $\longrightarrow$ $dist(c, d) = 0$
- String $s =$ "foo", $t =$ "too" $\longrightarrow$ $dist(s, t) = 1$
- Object $o =$ null, $p =$ new ArrayList() $\longrightarrow$ $dist(o, p) = V$

Type Distance

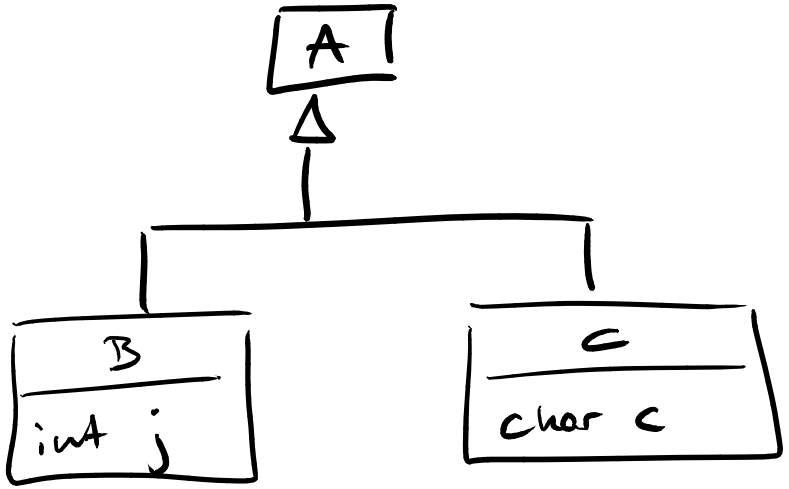
Distance between two types

$$typeDistance(t,u) = \lambda * pathLength(t,u) + \nu * \sum_{a \in nonShared(t,u)} weight_a$$

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

$\lambda, \nu \in \mathbb{N}$

Examples: Type Distance



$dist(B, C) = \lambda \cdot 1 + \nu \cdot (1 + 1)$

$dist(A, B) = \lambda \cdot 0 + \nu \cdot (1)$

Field Distance

Recursively compute distance of all shared fields

$$fieldDistance(p, q)$$

$$= \overline{\sum_a weight_a * (dist(p.a, q.a))}$$

Arithmetic mean: Avoid giving too much weight to objects with many fields

Algorithm for Selecting Inputs

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

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:

```
a3.transfer(a1, i2)
```

Second call:

```
a1.transfer(a4, i4)
```

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

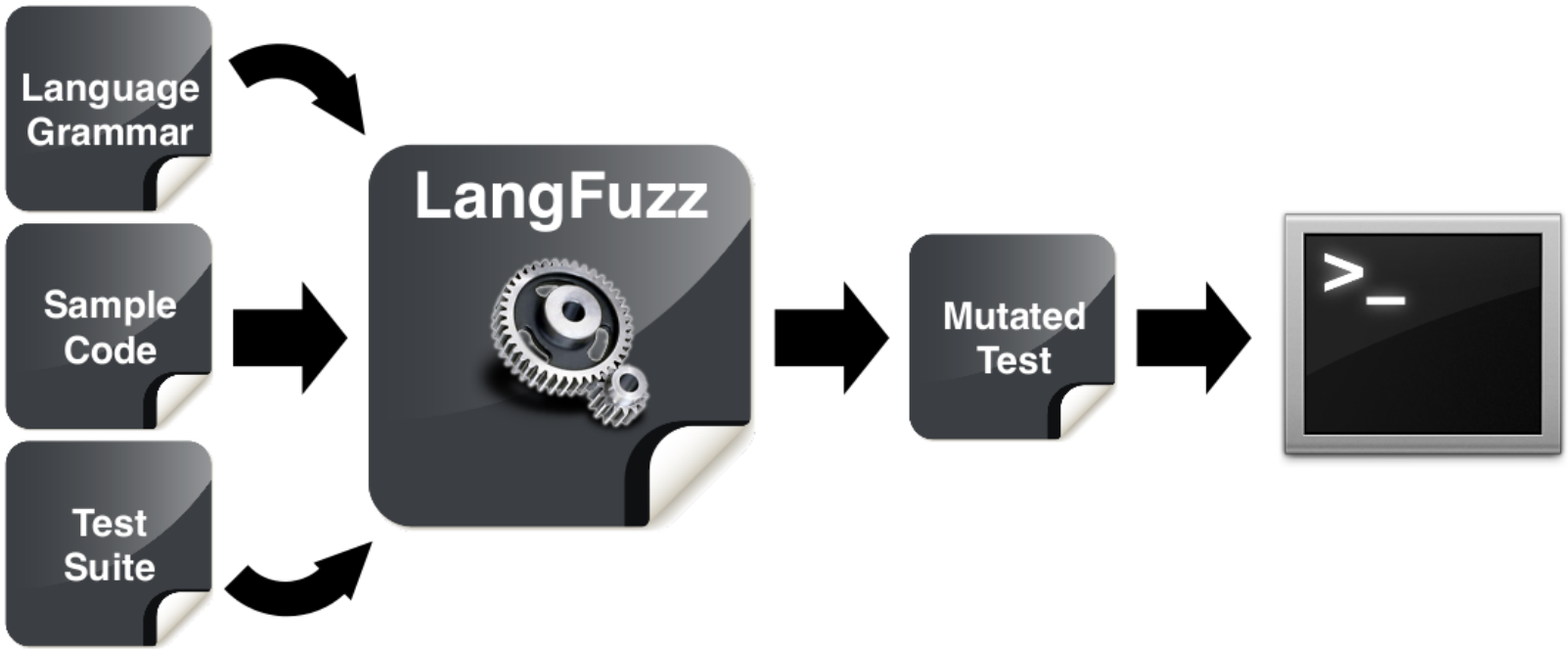
- Generative: Create new random input, possibly based on constraints and rules
- Mutative: Derive new inputs from existing input by randomly modifying it

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

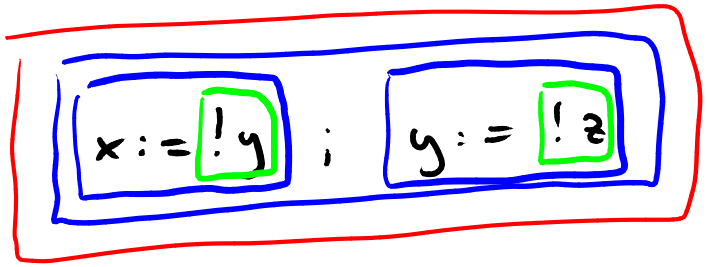
Phase III

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 :



expression 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 := !z else skip

if !x > 4 then y := !z else x := !y

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

$h = 3$

$e_{cur} = P$

$\downarrow$

$e_{cur} = C$

$\downarrow$

$e_{cur} = \text{while } B \text{ do } C$

$\downarrow$

$e_{cur} = \text{while } B \text{ do } C;C$

$P ::= C \mid \ldots$

$C ::= \ldots \mid \text{while } B \text{ do } C \mid \ldots$

$C ::= \ldots \mid C;C \mid \ldots$

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
```

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

(syntactically incorrect)

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