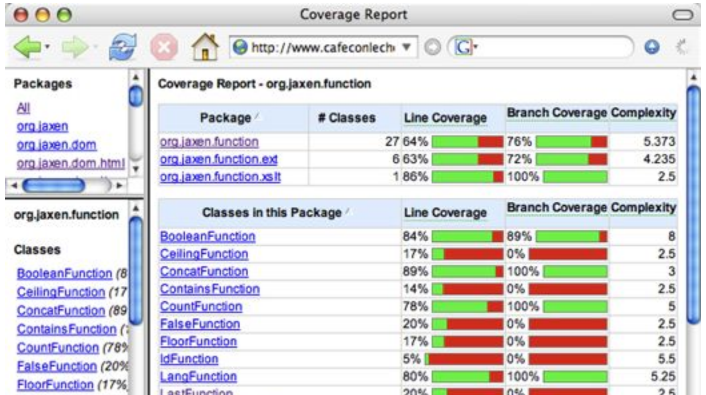
# Software Testing Basics

Spring 2025

Lingming Zhang

# 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



Packages

All
org.jaxen
org.jaxen.dom
org.jaxen.dom.html

org.jaxen.function

Classes

BooleanFunction (8
CeilingFunction (17
ConcatFunction (89
ContainsFunction (
CountFunction (78%
FalseFunction (20%
FloorFunction (17%

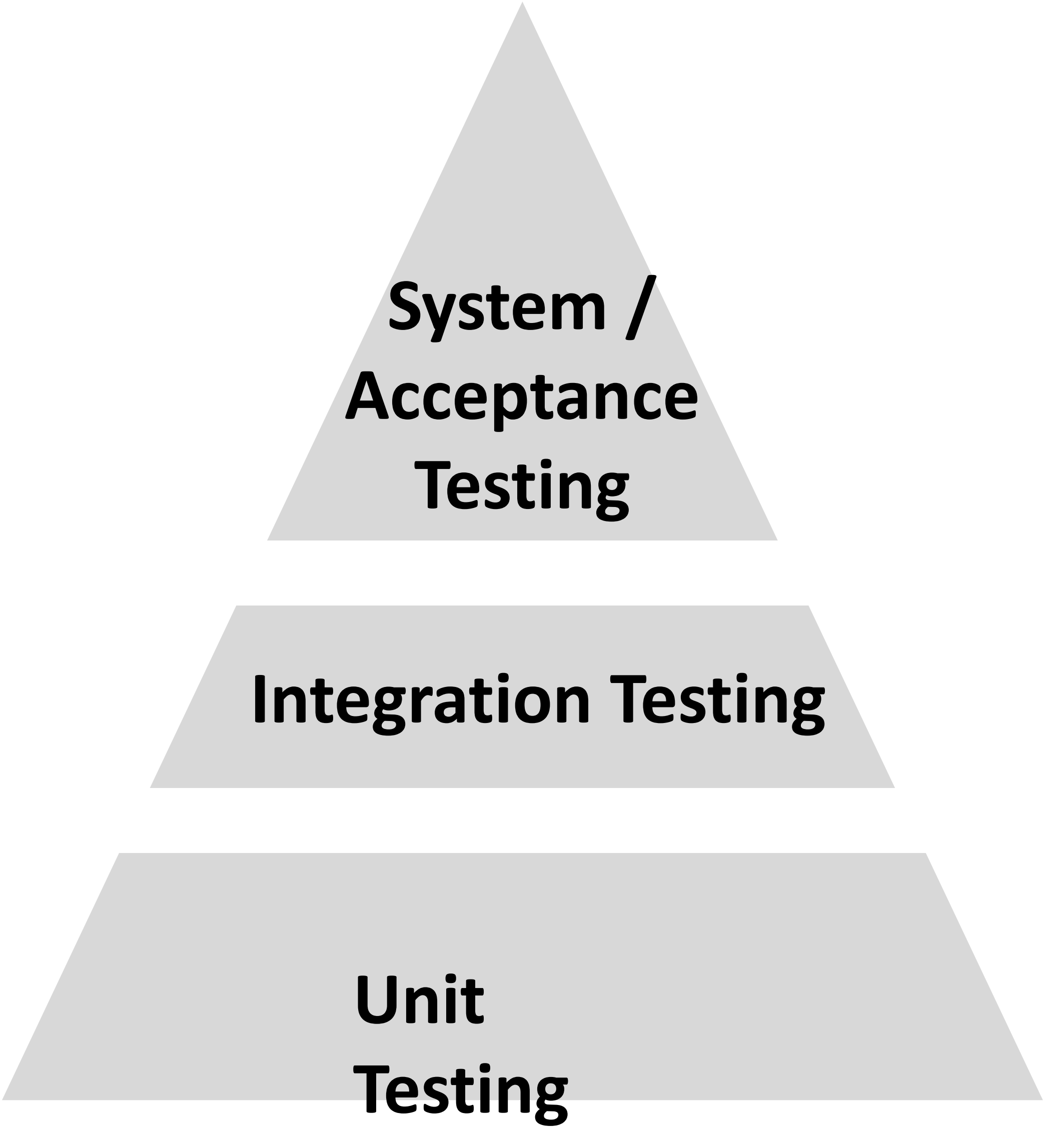
Coverage Report - org.jaxen.function

| Package | # Classes | Line Coverage | Branch Coverage | Complexity |
|---|---|---|---|---|
| org.jaxen.function | 27 | 64% | 76% | 5.373 |
| org.jaxen.function.ext | 6 | 63% | 72% | 4.235 |
| org.jaxen.function.xslt | 1 | 86% | 100% | 2.5 |

| Classes in this Package | Line Coverage | Branch Coverage | Complexity |
|---|---|---|---|
| BooleanFunction | 84% | 89% | 8 |
| CeilingFunction | 17% | 0% | 2.5 |
| ConcatFunction | 89% | 100% | 3 |
| ContainsFunction | 14% | 0% | 2.5 |
| CountFunction | 78% | 100% | 5 |
| FalseFunction | 20% | 0% | 2.5 |
| FloorFunction | 17% | 0% | 2.5 |
| IdFunction | 5% | 0% | 5.5 |
| LangFunction | 80% | 100% | 5.25 |
| LastFunction | 20% | 0% | 2.5 |

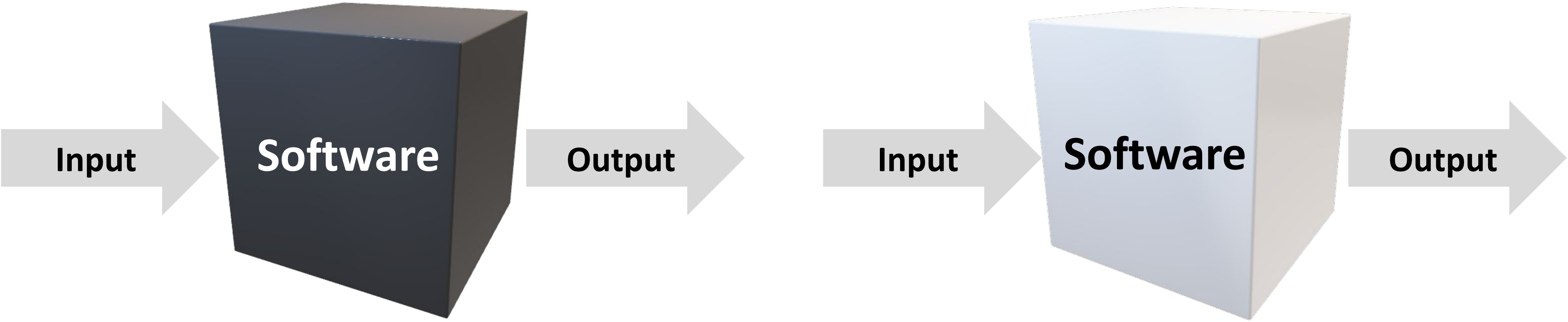
# Testing: levels

- Unit Testing
  - Test each single module in isolation
- Integration Testing
  - Test the interaction between modules
- System Testing
  - Test the system as a whole, by developers
- Acceptance Testing
  - Validate the system against user requirements, by customers with no formal test  cases



# Types of test generation

- Black-box (functional) vs. white-box (structural) testing



- **Black-box test generation**: generates tests based on the functionality of the program
- **White-box test generation**: generates tests based on the source-code structure of the program

# Today's focus

- **Unit testing**: involves testing individual units (e.g., methods or classes) of a software to ensure that each part is correct, typically
  - Unit level
  - White-box
  - Deterministic
  - …

- **Fuzz testing** (fuzzing): involves providing invalid, unexpected, or random data as inputs to a software, typically
  - System level
  - Black-box
  - Non-deterministic
  - …

# This class

- Unit Testing
  - Feedback-directed Random Test Generation (ICSE'07)
- Fuzz Testing
  - Finding and Understanding Bugs in C Compilers (PLDI’11)
  - Fuzzing with Code Fragments (SEC'12)
  - Compiler Validation via Equivalence Modulo Inputs (PLDI’14)
  - AFL: American Fuzzy Lop (https://github.com/google/AFL)
- LLM-based Fuzz Testing
  - Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models (ISSTA’23)

# 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);
  }
  …
}
```

Input values

Execution steps

Test oracle

## Is this an important problem?

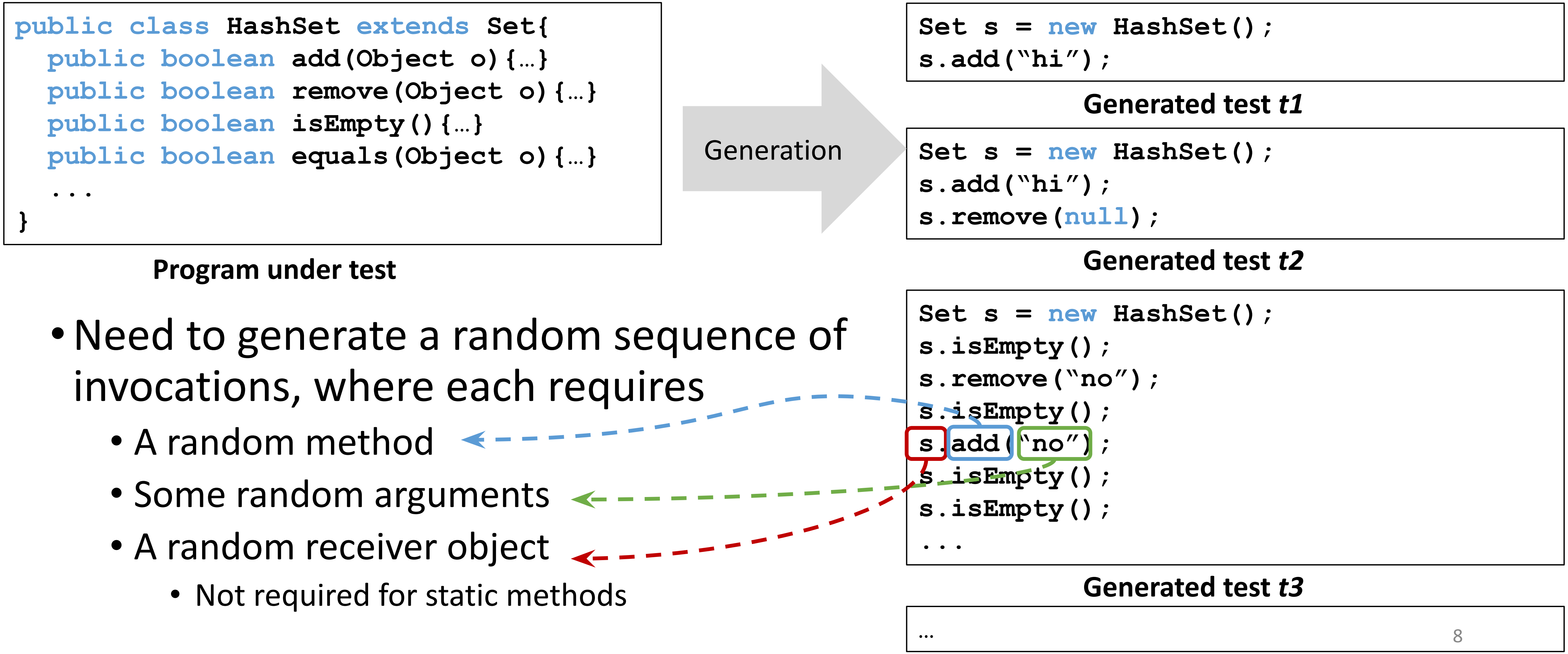
THE APACHE SOFTWARE FOUNDATION

**Commons-Math**

84,377 lines of **source code**

86,924 lines of **unit-test code**

# How to perform random white-box test generation?



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

**Program under test**

```
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();
s.isEmpty();
...
```

**Generated test *t3***

```
…
```

- Need to generate a random sequence of invocations, where each requires
  - A random method
  - Some random arguments
  - A random receiver object
    - Not required for static methods

# 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 arguments of some method invocations can be generated by other method invocations
- Can have many redundant&illegal tests

```
Set s = new HashSet();
s.isEmpty();
s.remove("no");
s.isEmpty();
s.add("no");
s.isEmpty();
s.isEmpty();
```

**A random test**

# Random method-sequence generation: redundant&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); // pre: argument >= 0
```

Should not output

```
5. Illegal test:
Date d = new Date(2006, 2, 14);
d.setMonth(-1); // pre: argument >= 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

```
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));
```

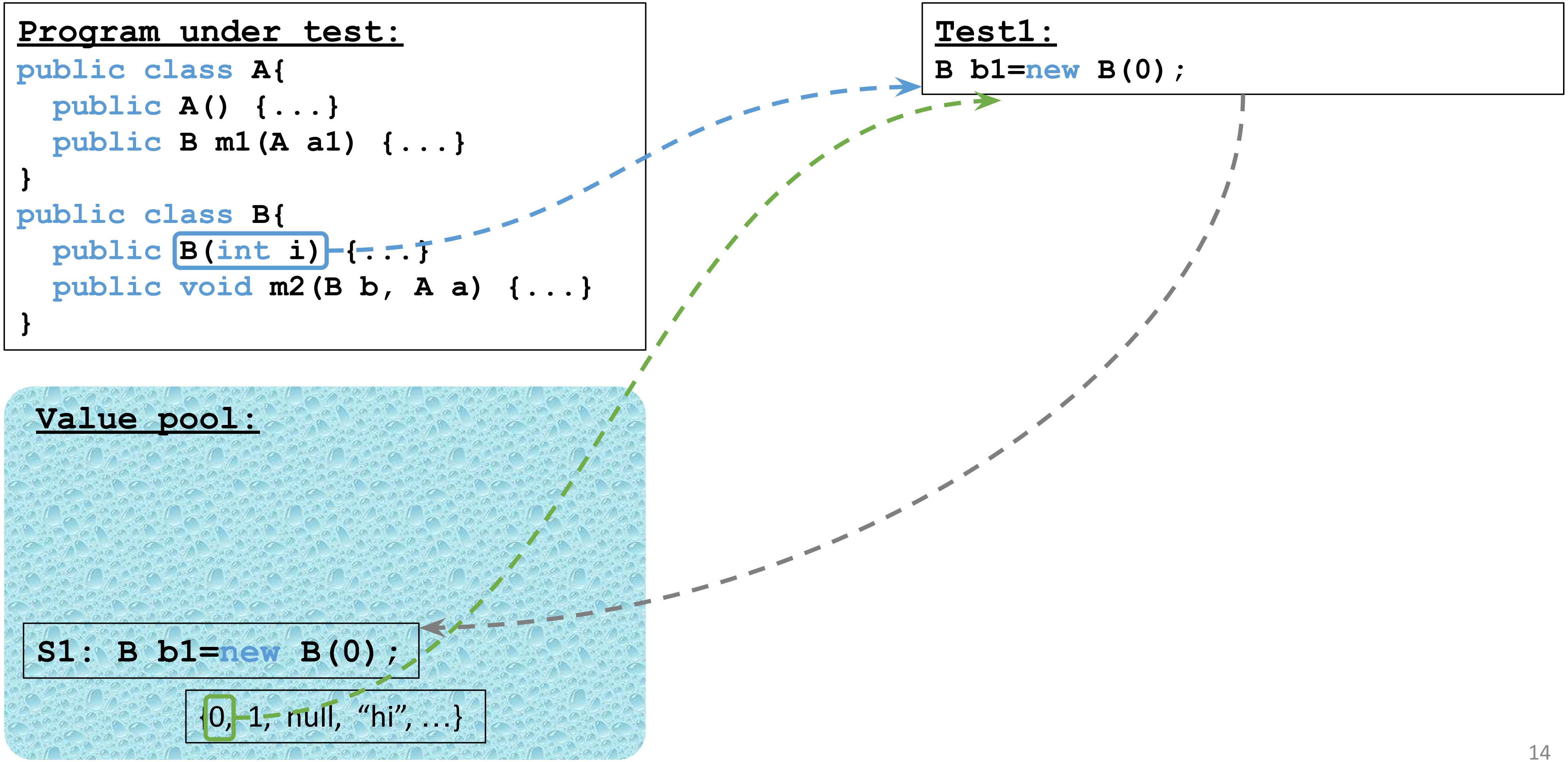
fails on Sun’s JDK 1.5/1.6 when executed

# 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 $\mathbf{m}(\mathbf{T_1}...\mathbf{T_k})/\mathbf{T_{ret}}$
        - For each input parameter of type $\mathbf{T_i}$, randomly pick a sequence $\mathbf{S_i}$ from the value pool that constructs an object $\mathbf{v_i}$ of type $\mathbf{T_i}$
        - Create new sequence $\mathbf{S_{new}} = \mathbf{S_1}; ... ; \mathbf{S_k}; \mathbf{T_{ret}}\ \mathbf{v_{new}} = \mathbf{m}(\mathbf{v_1}...\mathbf{v_k});$
        - if $\mathbf{S_{new}}$ was previously created (lexically), go to first step
    - Classify the new sequence $\mathbf{S_{new}}$
        - May discard, output as test case, or add to pool

Method
Parameter
Receiver object

# 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);
```

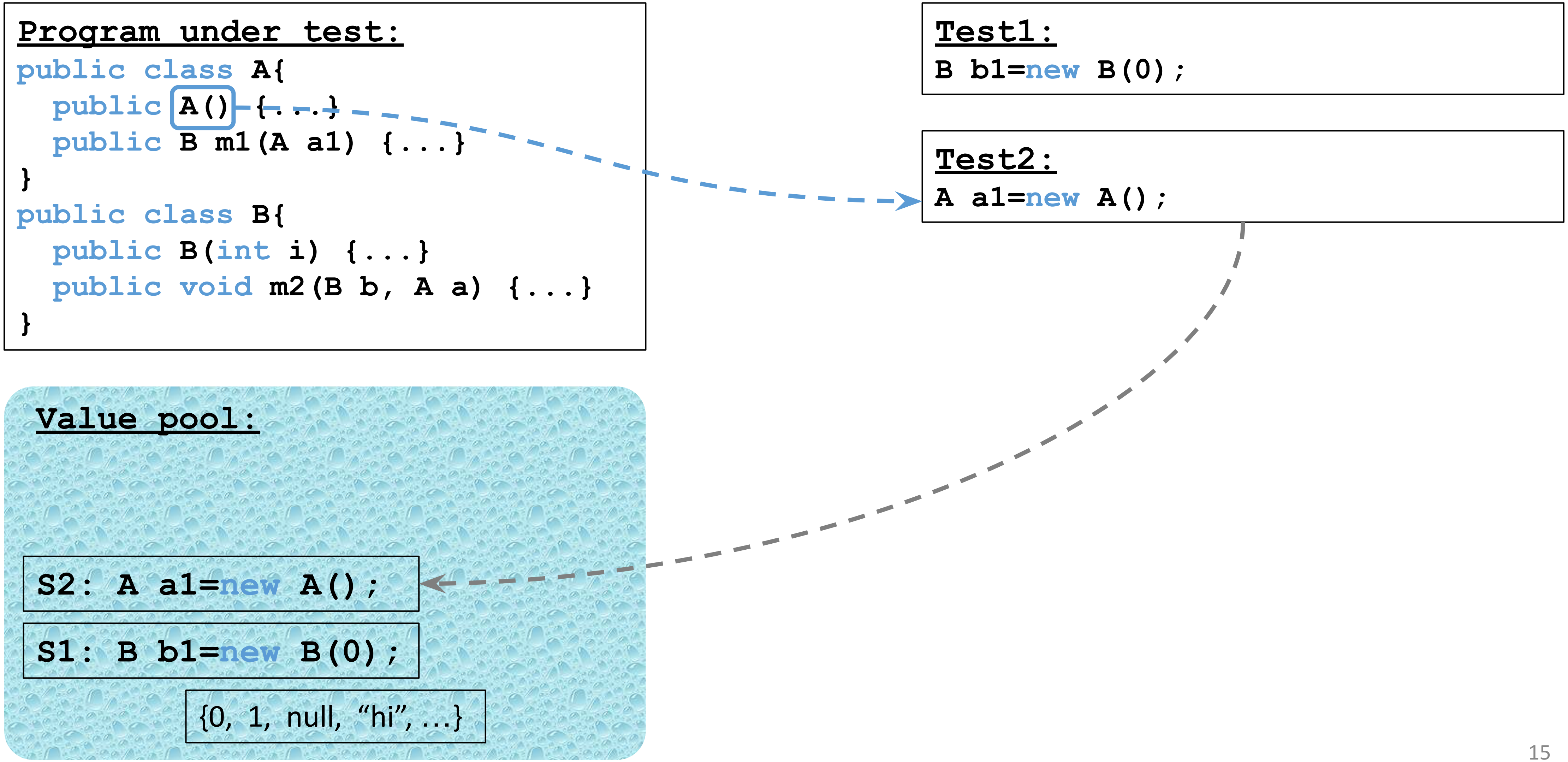
**Value pool:**

```
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

- - - ➤ Method
- - - ➤ Parameter
- - - ➤ Receiver object

# 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);
```

**Test2:**

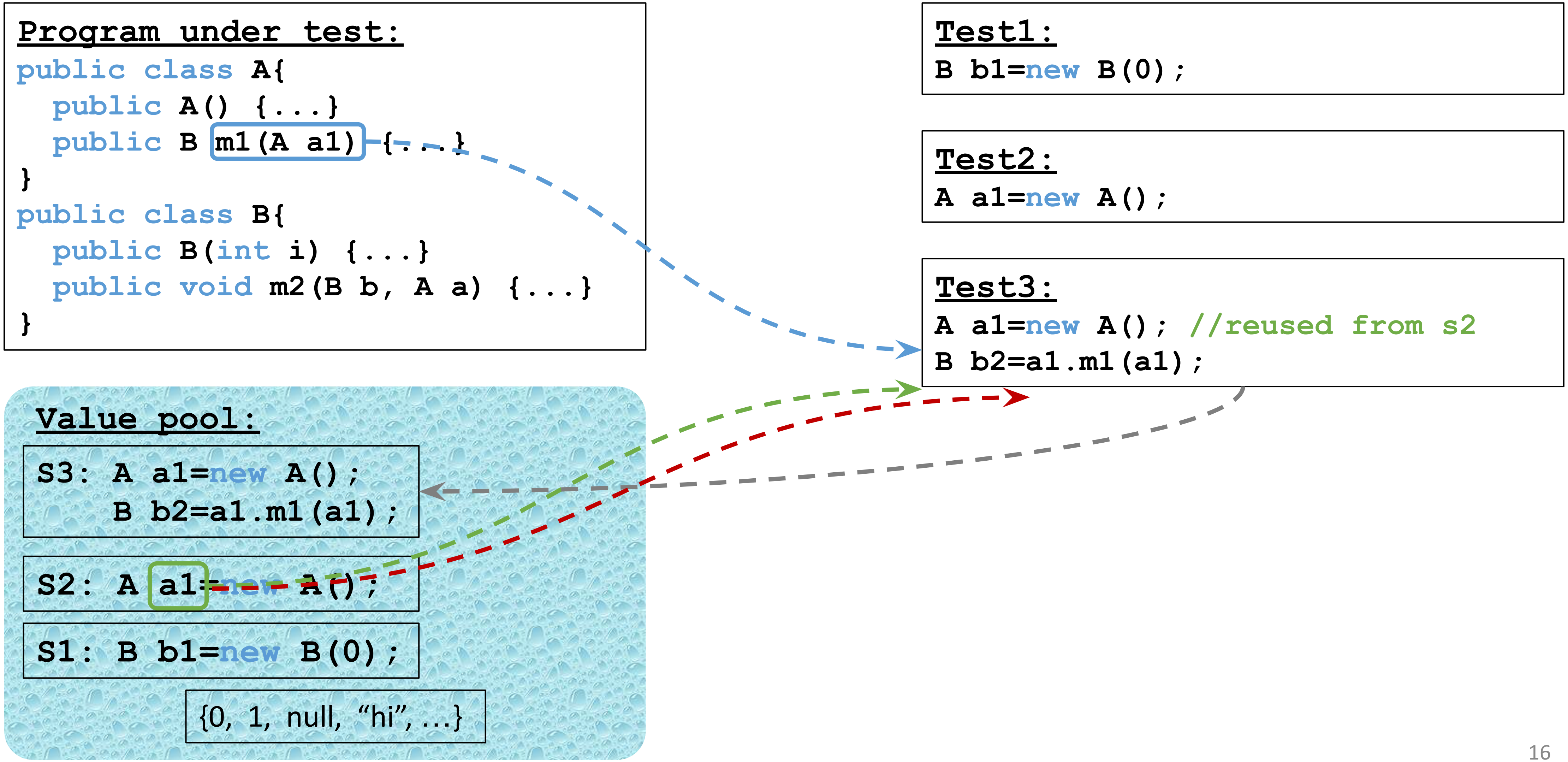
```
A a1=new A();
```

**Value pool:**

```
S2: A a1=new A();
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

# 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) {...}
}
```

**Value pool:**

```
S3: A a1=new A();
    B b2=a1.m1(a1);
```

```
S2: A a1=new A();
```

```
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

**Test1:**

```
B b1=new B(0);
```

**Test2:**

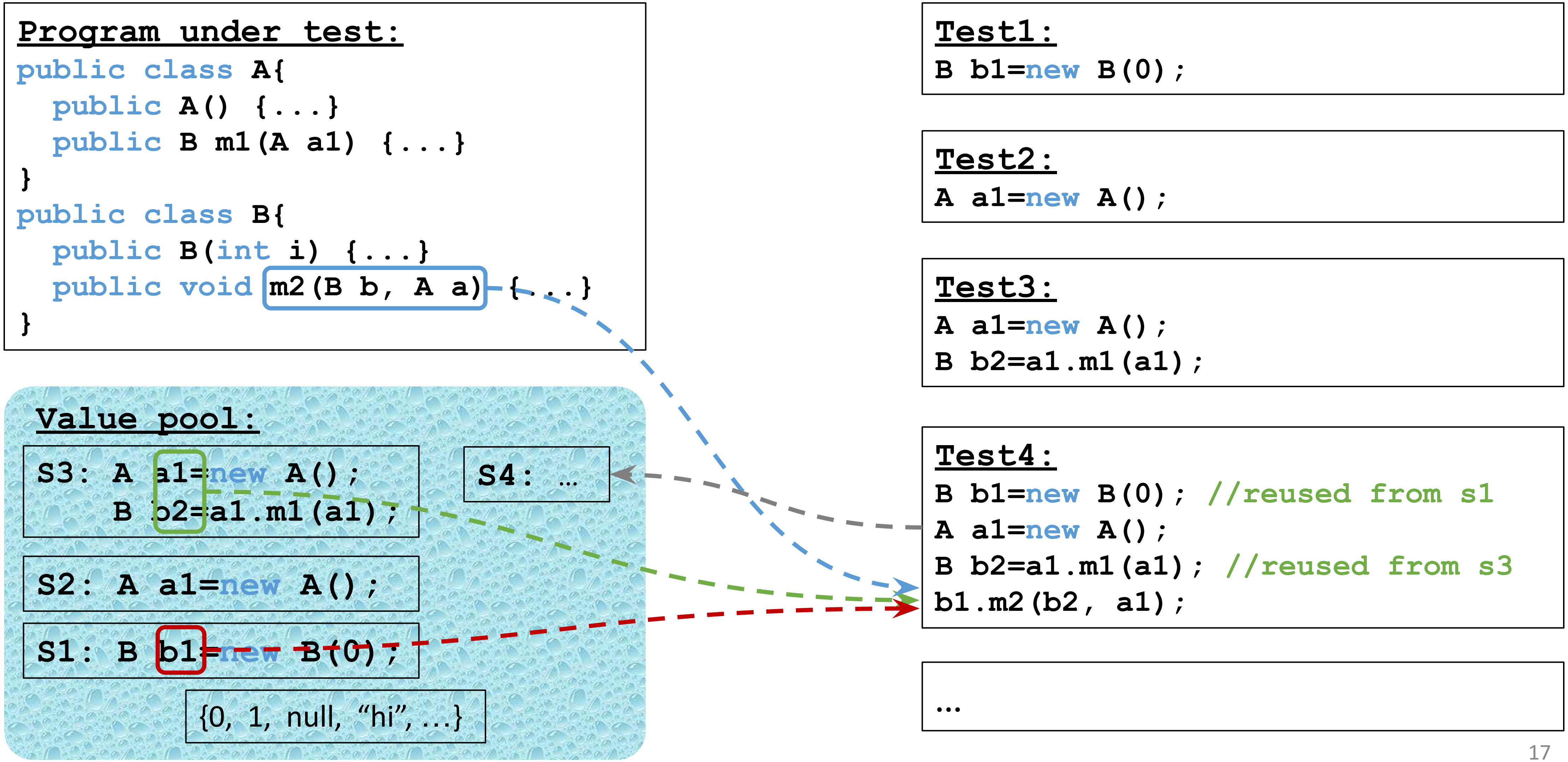
```
A a1=new A();
```

**Test3:**

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

- 🟦 Method
- 🟩 Parameter
- 🟥 Receiver object

# 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) {...}
}
```

**Value pool:**

```
S3: A a1=new A();
    B b2=a1.m1(a1);
```

```
S4: …
```

```
S2: A a1=new A();
```

```
S1: B b1=new B(0);
```

{0, 1, null, “hi”, …}

**Test1:**

```
B b1=new B(0);
```

**Test2:**

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

**Test3:**

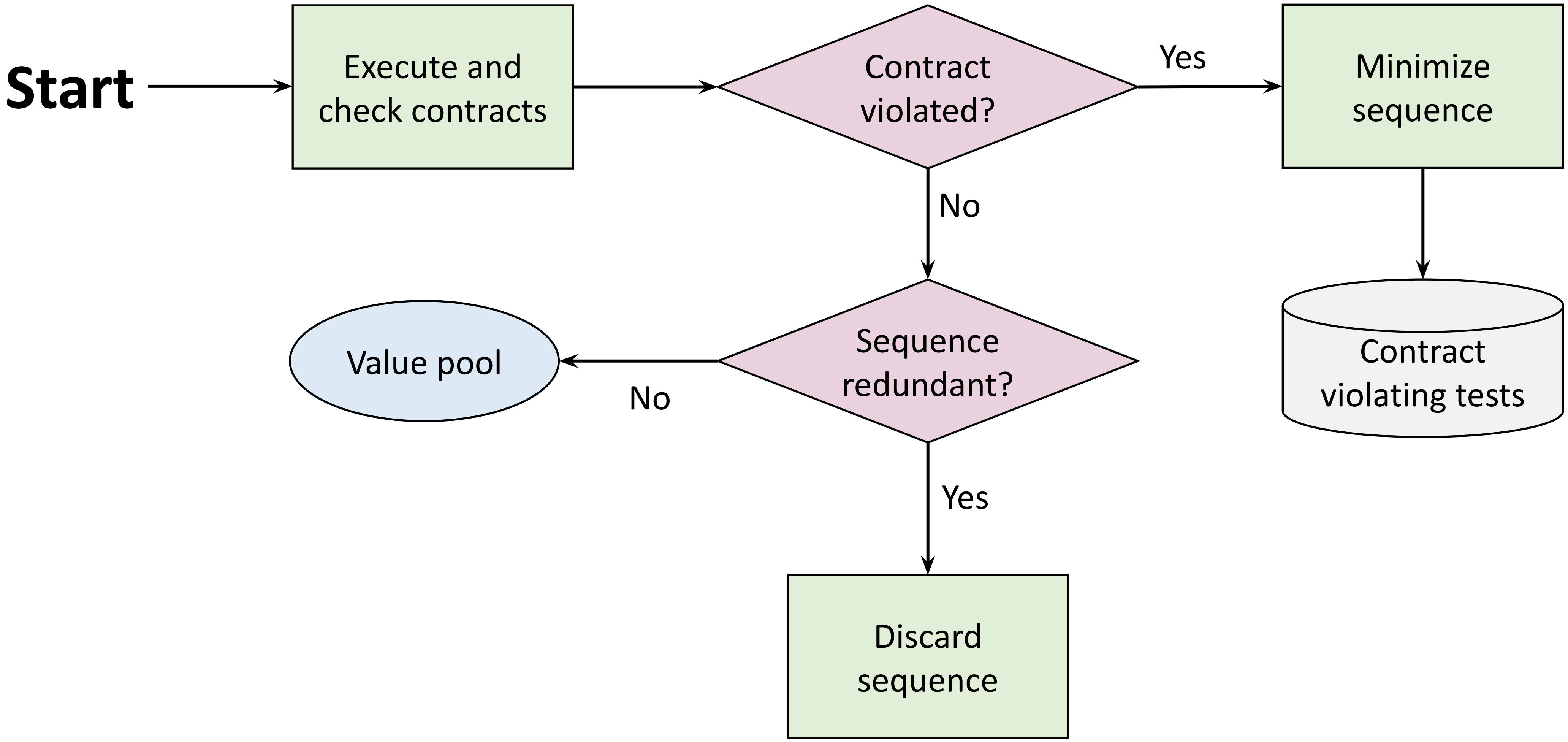
```
A a1=new A();
B b2=a1.m1(a1);
```

**Test4:**

```
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



# 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 *equals* to compare)
- Could also use more sophisticated state equivalence methods
    - E.g. heap canonicalization used in model checkers

# 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
Automatic unit test generation

# Randoop outputs oracles

- Oracle for contract-violating tests:

```
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):

```
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

```
Statically:
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

| **Data structure programs** | **Time (s)** | **Branch cov.** |
|---|---|---|
| Bounded stack (30 LOC) | 1 | 100% |
| Unbounded stack (59 LOC) | 1 | 100% |
| BS Tree (91 LOC) | 1 | 96% |
| Binomial heap (309 LOC) | 1 | 84% |
| Linked list (253 LOC) | 1 | 100% |
| Tree map (370 LOC) | 1 | 81% |
| Heap array (71 LOC) | 1 | 100% |

# Bug detection by Randoop: subjects

| Subjects | LOC | Classes |
|---|---|---|
| JDK (2 libraries)<br>(java.util, javax.xml) | 53K | 272 |
| Apache commons (6 libraries)<br>(logging, primitives, chain, jelly, math, collections) | 114K | 974 |
| .Net libraries (6 libraries) | 615K | 3455 |

# Bug detection by Randoop: methodology

- Ran Randoop on each library
  - Used default time limit (2 minutes)
- Contracts:
  - **o.equals(o)==true**
  - **o.equals(o)** throws no exception
  - **o.hashCode()** throws no exception
  - **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

| Subjects | Failed tests | Unique failed tests | Error-revealing tests | Distinct errors |
|---|---|---|---|---|
| JDK | 613 | 32 | 29 | 8 |
| Apache commons | 3,044 | 187 | 29 | 6 |
| .Net framework | 543 | 205 | 196 | 196 |
| Total | 4,200 | 424 | 254 | 210 |

# 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

# 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

```
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

Microsoft

THE APACHE® SOFTWARE FOUNDATION

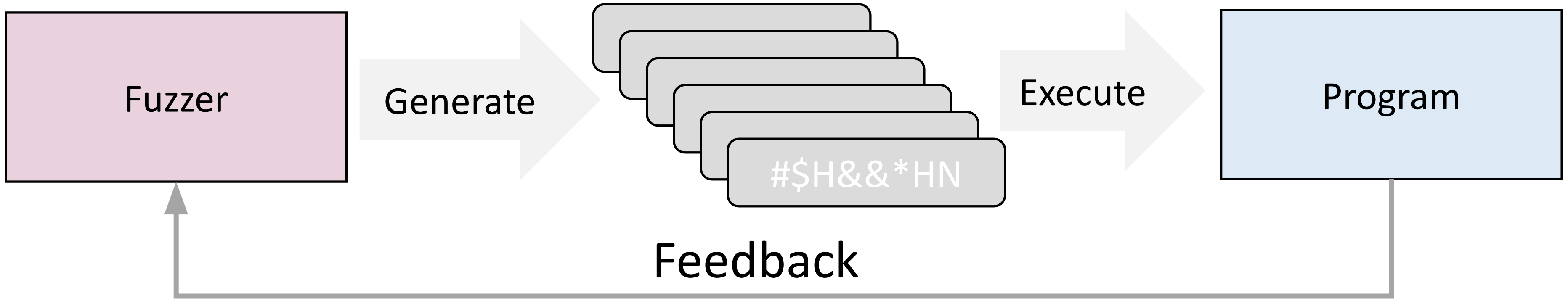
ABB Power and productivity for a better world™

# This class

- Unit Testing
  - Feedback-directed Random Test Generation (ICSE'07)
- Fuzz Testing
  - Finding and Understanding Bugs in C Compilers (PLDI’11)
  - Fuzzing with Code Fragments (SEC'12)
  - Compiler Validation via Equivalence Modulo Inputs (PLDI’14)
  - AFL: American Fuzzy Lop (https://github.com/google/AFL)
- LLM-based Fuzz Testing
  - Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models (ISSTA’23)

# Fuzz testing

**./Program < /dev/random**



- Fuzzing strategies
  - Mutation-based
  - Generation-based
  - Learning-based

- Feedback guide
  - New coverage?
  - Shorter execution?
  - Valid input?

- Targeted programs
  - Binaries
  - Compilers
  - Browsers
  - DB systems
  - ML systems
  - …

# Generation-based fuzzing

- Create test inputs based on predefined structure/grammar

```
<start> ::= <expr>
<expr> ::= <term> + <expr> | <term> - <expr> | <term>
<term> ::= <term> * <factor> | <term> / <factor> | <factor>
<factor> ::= +<factor> | -<factor> | (<expr>) | <int> | <int>.<int>
<int> ::= <digit><int> | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

**Arithmetic expression grammar**

Generate

```
(-+++((1 + (+7 - (-1 * (++-+7.7 - -+-4.0)))))) * +--4 - -(6) + 64) 8.2 - 27 -
-9 / +((+9 * --2 + --+-+-((-1 * +(8 - 5 - 6)) * (-((-+(((+(4)))))) - ++4) /
+(-+---((5.6 - --(3 * -1.8 * +(6 * +-(((-(-6) * ---+6)) / +--(+-+-7 * (-0 *
(+(((((2)) + 8 - 3 - ++9.0 + ---(--+7 / (1 / +++6.37) + (1) / 482) /
+++-+0)))) * -+5 + 7.513)))) - …
```

**Arithmetic expression**

**C grammar**

Csmith

```
void foo (void) {
 int x;
 for (x =0; x < 5; x++){
  if (x) continue;
  if (x) break;
 }
 printf("%d", x);
}
```

**C program triggering an LLVM bug**

**Javascript grammar**

LangFuzz

```
var haystack = "foo";
var re_text = "^foo";
haystack += "x";
re_text += "(x)";
var re = new RegExp(re_text);
re.test(haystack);
RegExp.input = Number();
print(RegExp.$1);
```

**JS program crashing Mozilla**

# Generation-based fuzzing: examples

- Finding and Understanding Bugs in C Compilers (PLDI’11)
  - Targeting C compilers
  - Cited for **1,000+** times
  - **400+** GCC/LLVM bugs found
- Fuzzing with Code Fragments (SEC’12)
  - Targeting JS browsers/engines
  - Cited for **400+** times
  - USD **50,000+** bug bounties in the first month
  - **2,000+** bugs found for Mozilla Firefox, Google Chrome, and Microsoft Edge to date

# Mutation-based fuzzing

- Apply small mutations on high-quality seed inputs to generate more test inputs

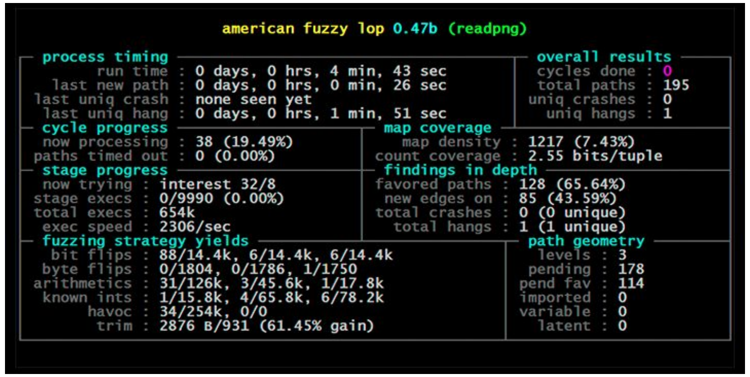
```
int a, b, c, d, e;
int main() {
  for (b = 4; b > -30; b--)
    for (; c;)
      for (;;) {
        b++;
        e = a > 2147483647 - b;
        if (d) break;
      }
  return 0;
}
```

▨ **: not executed**

EMI

```
int a, b, c, d, e;
int main() {
  for (b = 4; b > -30; b--)
    for (; c;)
      for (;;) {
        ~~b++;~~
        e = a > 2147483647 - b;
        if (d) break;
      }
  return 0;
}
```

- Structured mutation not generalizable?
- Mutation at the binary level!



```
                        american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec        │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec        │  total paths : 195   │
│ last uniq crash : none seen yet                       │ uniq crashes : 0     │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec        │   uniq hangs : 1     │
├─ cycle progress ───────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 38 (19.49%)      │    map density : 1217 (7.43%)         │
│ paths timed out : 0 (0.00%)        │ count coverage : 2.55 bits/tuple      │
├─ stage progress ───────────────────┼─ findings in depth ────────────────────┤
│  now trying : interest 32/8        │ favored paths : 128 (65.64%)          │
│ stage execs : 0/9990 (0.00%)       │  new edges on : 85 (43.59%)           │
│ total execs : 654k                 │ total crashes : 0 (0 unique)          │
│  exec speed : 2306/sec             │   total hangs : 1 (1 unique)          │
├─ fuzzing strategy yields ──────────┴─────────────┬─ path geometry ────────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k          │    levels : 3          │
│  byte flips : 0/1804, 0/1786, 1/1750              │   pending : 178        │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k           │  pend fav : 114        │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k           │  imported : 0          │
│       havoc : 34/254k, 0/0                        │  variable : 0          │
│        trim : 2876 B/931 (61.45% gain)            │    latent : 0          │
└───────────────────────────────────────────────────┴────────────────────────┘
```

**AFL**

# Mutation-based fuzzing: examples

- Compiler Validation via Equivalence Modulo Inputs (PLDI’14)
  - Insight: EMI takes existing input programs and generates equivalent variants on a particular set of inputs (by removing unexecuted statements) for compiler fuzzing
  - **147** confirmed bugs found in the paper
  - Found **1,000+** LLVM/GCC bugs together with follow-up work
- AFL: American Fuzzy Lop ([https://github.com/google/AFL](https://github.com/google/AFL))
  - The pioneer binary fuzzing tool leveraging coverage feedback
  - Highly scalable and generalizable due to the practical design
  - Found numerous bugs in real-world software systems

LLVM Maven™ LibreOffice MySQL™ hadoop

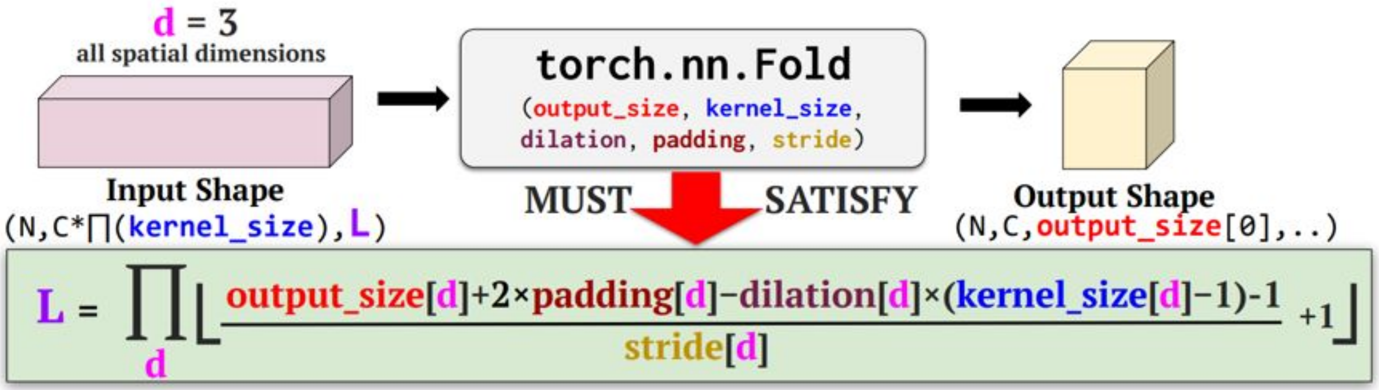
# This class

- Unit Testing
  - Feedback-directed Random Test Generation (ICSE'07)
- Fuzz Testing
  - Finding and Understanding Bugs in C Compilers (PLDI’11)
  - Fuzzing with Code Fragments (SEC'12)
  - Compiler Validation via Equivalence Modulo Inputs (PLDI’14)
  - AFL: American Fuzzy Lop (https://github.com/google/AFL)
- LLM-based Fuzz Testing
  - Large Language Models are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via Large Language Models (ISSTA’23)

# How can we fuzz test Deep Learning libraries?

Deep Learning (DL) libraries serve as the fundamental building block for all DL pipelines

```
fold = nn.Fold(output_size=(4,
5), kernel_size=(2, 2))
input = torch.randn(1, 3 * 2 *
2, 12)
output = fold(input)
```



d = 3
all spatial dimensions
Input Shape
(N,C*∏(kernel_size),L)
torch.nn.Fold
(output_size, kernel_size,
dilation, padding, stride)
MUST
SATISFY
Output Shape
(N,C,output_size[0],..)
$L = \prod_{d} \left\lfloor \frac{output\_size[d]+2\times padding[d]-dilation[d]\times(kernel\_size[d]-1)-1}{stride[d]} +1 \right\rfloor$

**Challenges:**

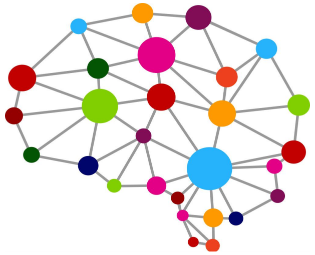
○ Python syntax/semantics

○ Complex tensor computation constraints

○ Other implicit/explicit API constraints

# Large Language Models (LLMs) for fuzzing!

💡 They are trained on **trillions** of open-source code tokens and can autoregressively generate **human-like** code!



**Traditional** fuzzing techniques (**generation-** & **mutation**-based)

***Explicit* Modeling** → ✗

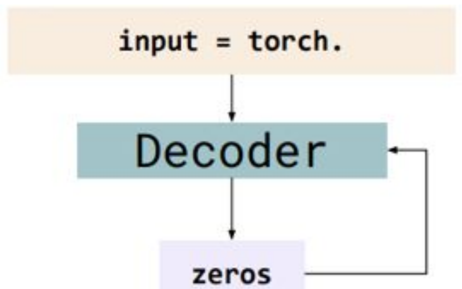
- ○ Language constraints
- ○ Tensor shape constraints
- ○ Other API constraints

← ***Implicit* Learning** ✓

Trained on **>400K** Tensorflow/PyTorch projects

Directly use LLMs to generate programs satisfying *heterogeneous constraints* for fuzzing DL libraries **and beyond**!

# LLMs for fuzzing: design

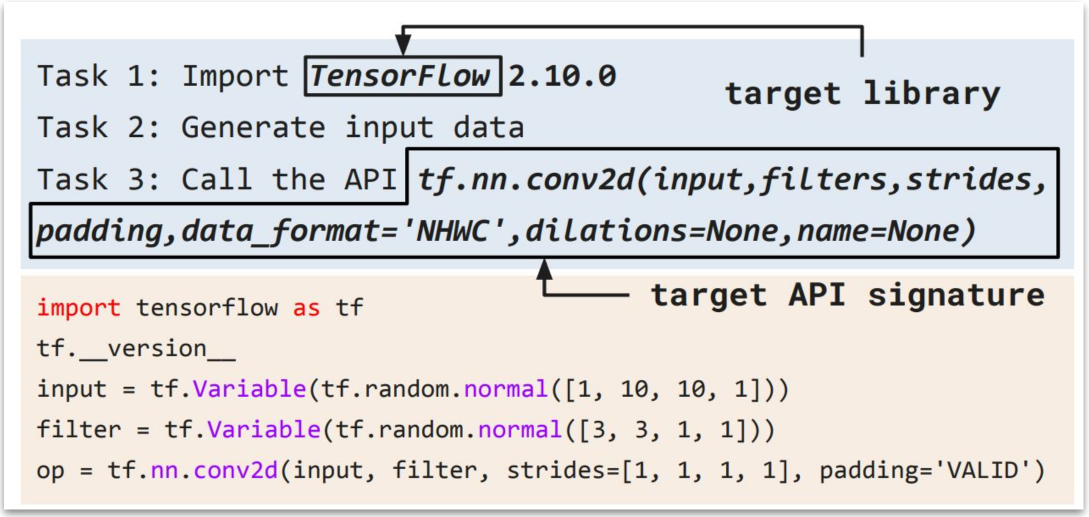


**Left-to-right generation**

🧑‍💻**Prompt**

**Generation-based Fuzzing**

🤖**Codex**

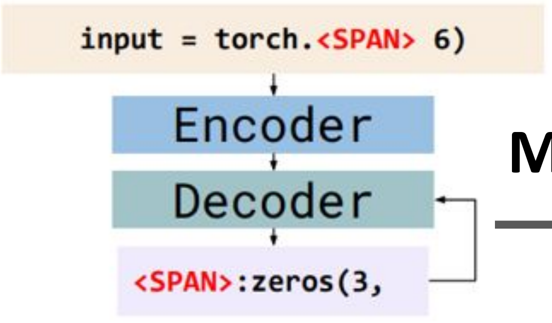


```
Task 1: Import TensorFlow 2.10.0
Task 2: Generate input data
Task 3: Call the API tf.nn.conv2d(input,filters,strides,
padding,data_format='NHWC',dilations=None,name=None)
```

target library

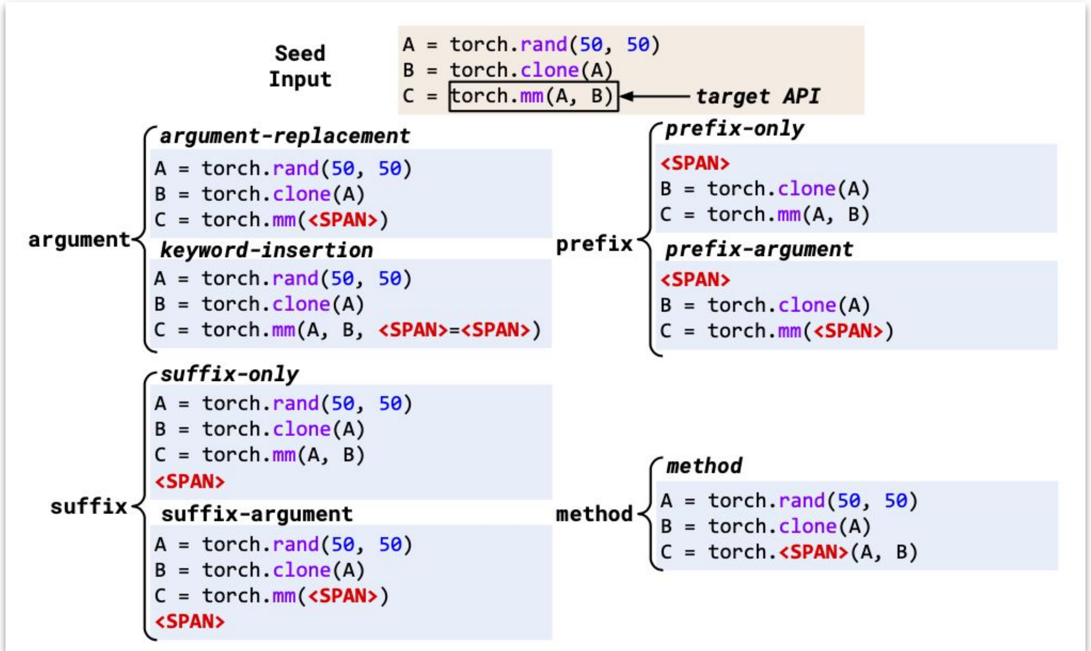
target API signature

```
import tensorflow as tf
tf.__version__
input = tf.Variable(tf.random.normal([1, 10, 10, 1]))
filter = tf.Variable(tf.random.normal([3, 3, 1, 1]))
op = tf.nn.conv2d(input, filter, strides=[1, 1, 1, 1], padding='VALID')
```



**Infilling**

**Mutation-based Fuzzing**



Seed Input

```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.mm(A, B)
```

target API

argument

*argument-replacement*

```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.mm(<SPAN>)
```

*keyword-insertion*

```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.mm(A, B, <SPAN>=<SPAN>)
```

prefix

*prefix-only*

```
<SPAN>
B = torch.clone(A)
C = torch.mm(A, B)
```

*prefix-argument*

```
<SPAN>
B = torch.clone(A)
C = torch.mm(<SPAN>)
```

suffix

*suffix-only*

```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.mm(A, B)
<SPAN>
```

suffix-argument

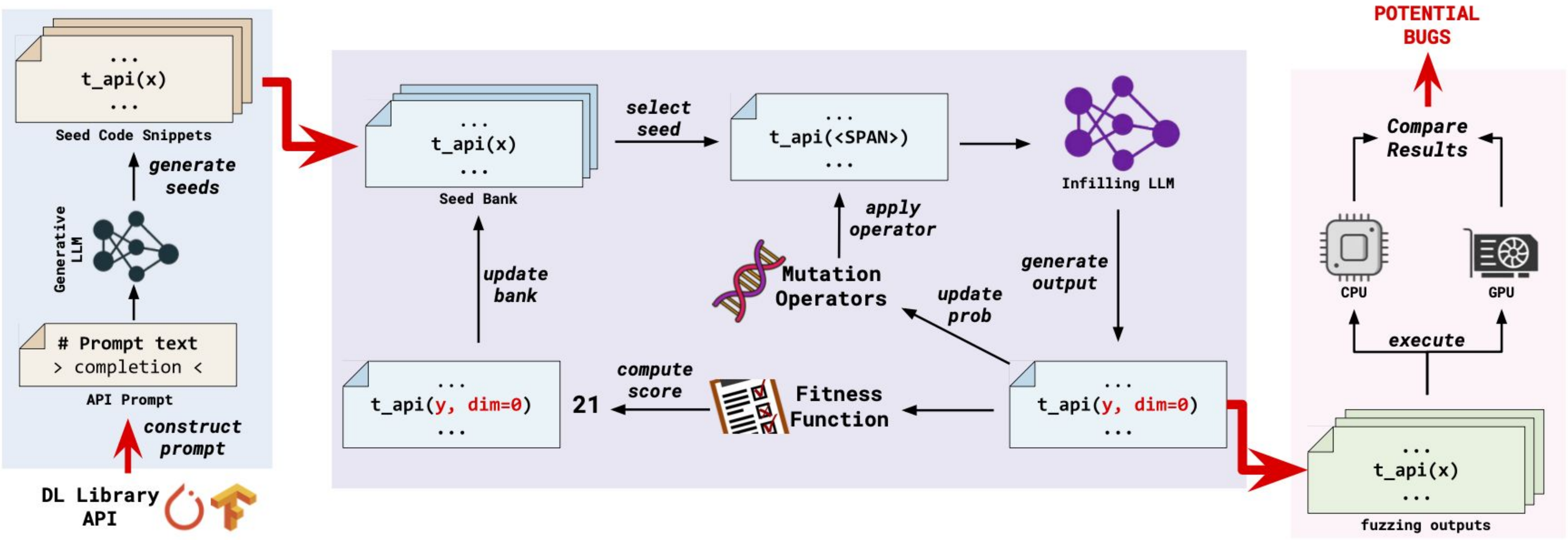
```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.mm(<SPAN>)
<SPAN>
```

method

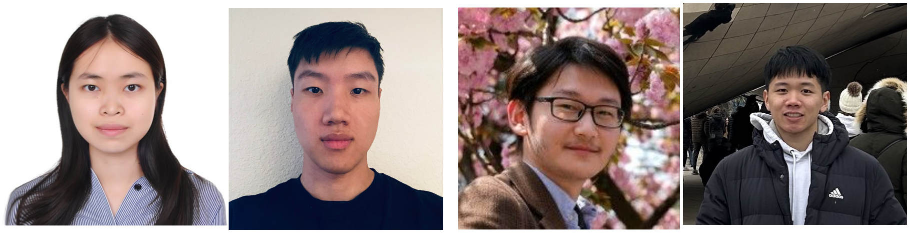
*method*

```
A = torch.rand(50, 50)
B = torch.clone(A)
C = torch.<SPAN>(A, B)
```

# TitanFuzz



1. Seed program *generation* using **LLM-based generation**
2. Evolutionary program *mutation* via **LLM-based infilling**
3. **Differential testing** oracle



# TitanFuzz: Summary

- The first LLM-based approach for fuzzing (DL libraries and beyond)
  - Seed generation/mutation using generative/infilling LLMs
  - Up to **50.84%** higher coverage than traditional fuzzers
  - Detect **65** bugs, with **44** confirmed
- Implications for LLM-based fuzzing/testing
  - **LLMs can directly perform generation- and mutation-based fuzzing** (with minimal engineering efforts)!
  - Applicable to challenging domains with **heterogeneous constraints**
  - Easily generalizable to **other system domains**!

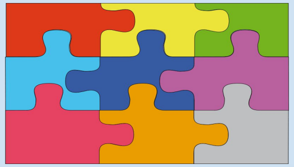
Our recent studies:
**400+** bugs found
**300+** confirmed

❏ Deng et al., *LLMs are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries via LLMs.* **ISSTA 2023.**

# Recent trends for LLM-based fuzzing

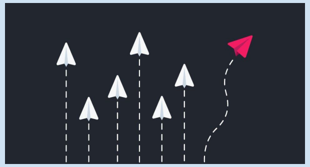
## More application domains

- LaST [ASE’23]
- BusyBoxFuzzer [Security’24]
- Fuzz4All [ICSE’24]
- …



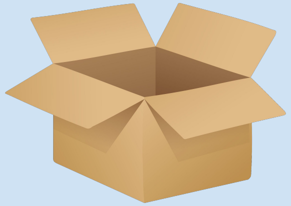
## Edge-case test generation

- FuzzGPT [ICSE’24]
- InputBlaster [ICSE’24]
- Yanhui [ISSTA’24]
- …



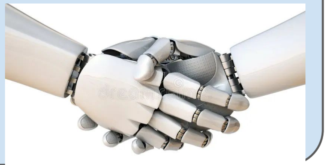
## Opening the blackbox

- ChatAFL [NDSS’24]
- WhiteFox [OOPSLA’24]
- CovRL [ISSTA’24]
- LLM4Fuzz [arXiv’24]
- …



## Fuzzer (not input) generation

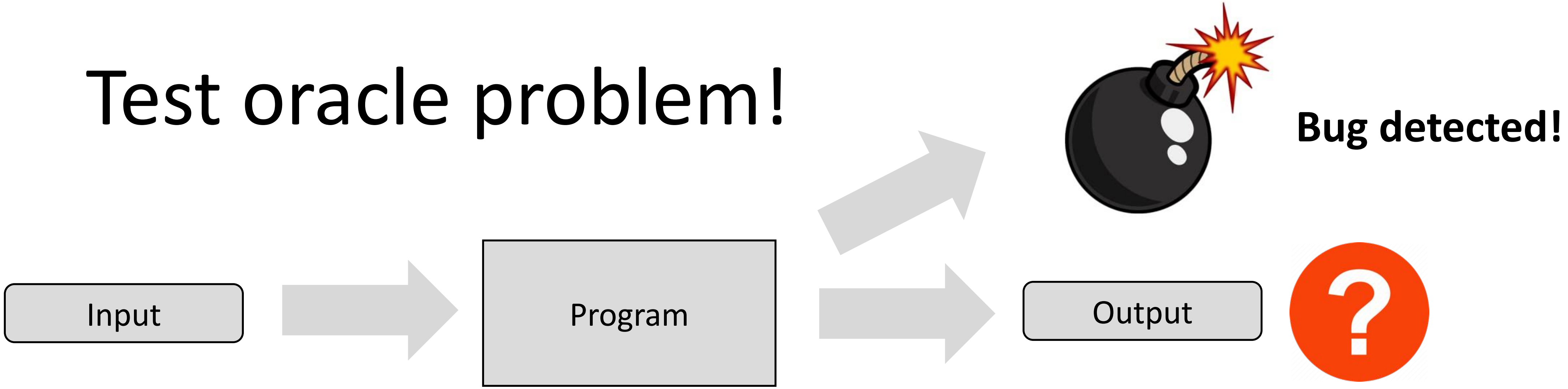
- MetaMut [ASPLOS’24]
- PromptFuzz [CCS’24]
- KernelGPT [ASPLOS’25]
- …



Thanks!

# Backup slides

# Test oracle problem!



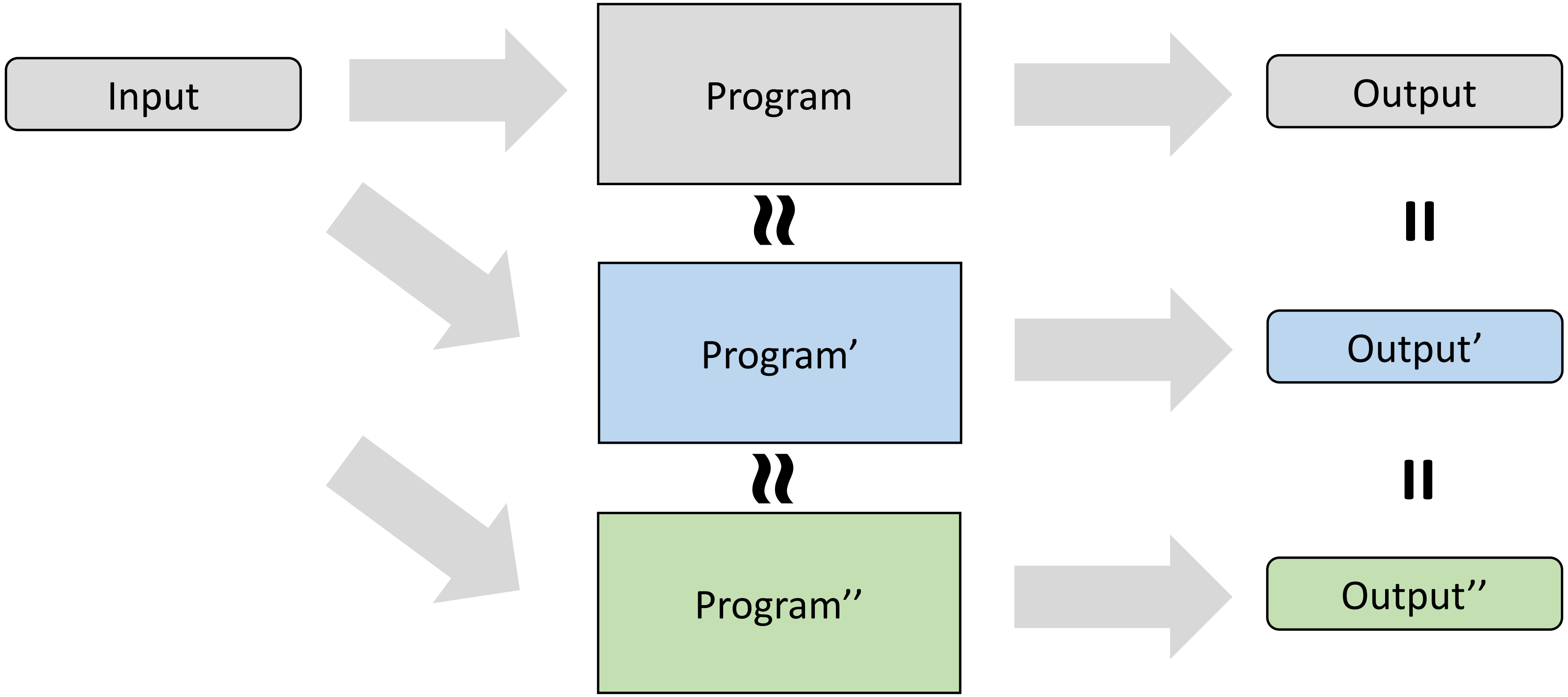
**Test oracle:** a mechanism for determining whether software executed correctly for a test[1].

**One of the hardest problem in Software Engineering!**

**How to mitigate it?**

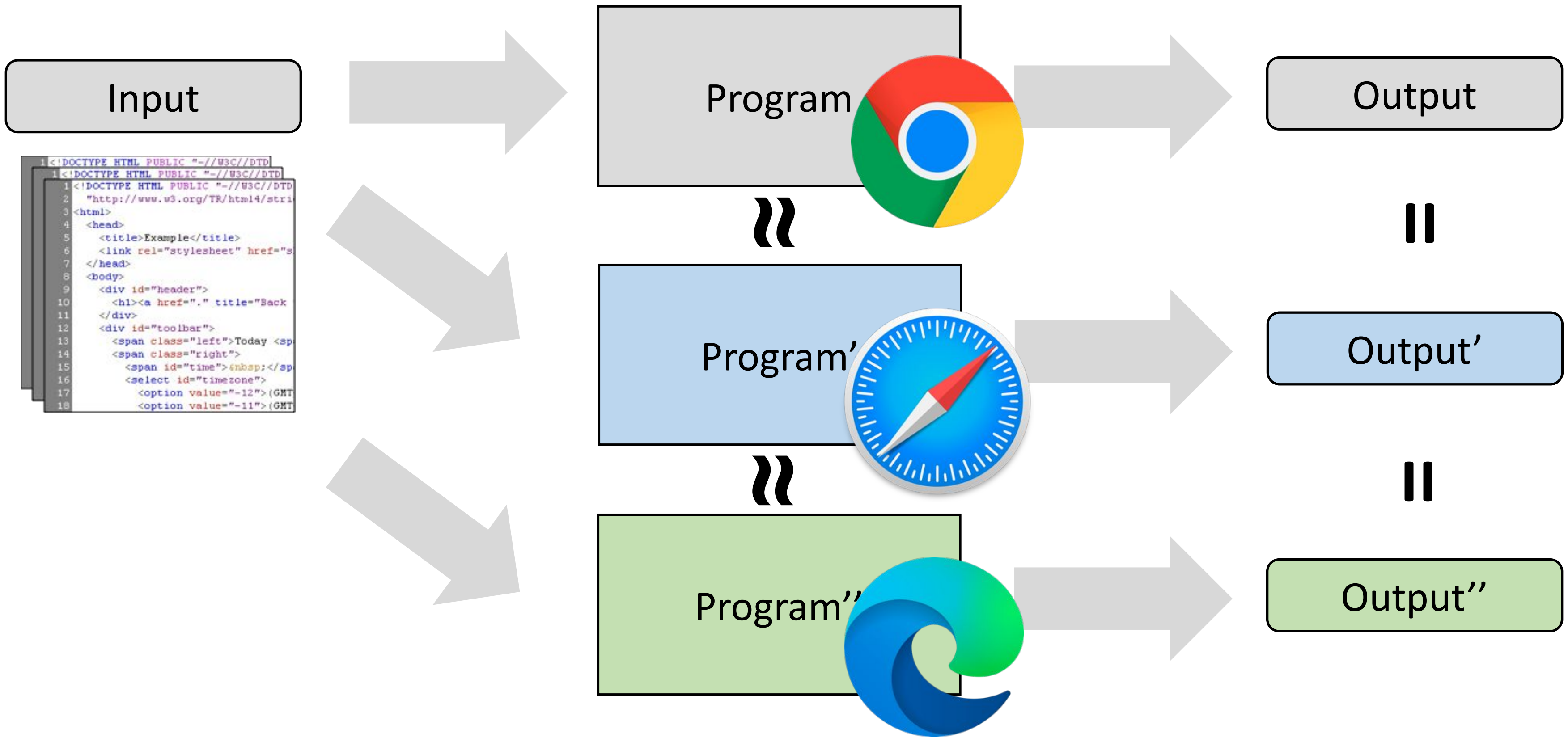
[1] https://en.wikipedia.org/wiki/Test_oracle

# Differential testing

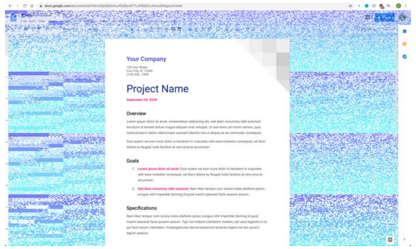


Provide the same input to **similar** applications, and observe output **differences**

# Differential testing: browsers



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
  "http://www.w3.org/TR/html4/stri
<html>
  <head>
    <title>Example</title>
    <link rel="stylesheet" href="s
  </head>
  <body>
    <div id="header">
      <h1><a href="." title="Back
    </div>
    <div id="toolbar">
      <span class="left">Today <sp
      <span class="right">
        <span id="time">&nbsp;</sp
        <select id="timezone">
          <option value="-12">(GMT
          <option value="-11">(GMT
```



Your Company

Project Name

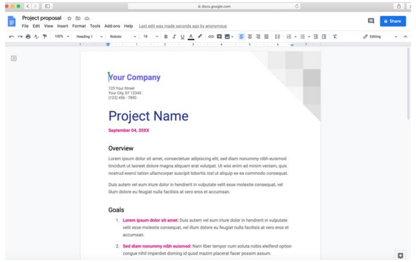
September 04, 20XX

Overview

Goals

Specifications

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.



docs.google.com

Project proposal

File Edit View Insert Format Tools Add-ons Help Last edit was made seconds ago by anonymous

Share

100% Heading 1 Roboto 16 Editing

Your Company

123 Your Street
Your City, ST 12345
(123) 456 - 7890

Project Name

September 04, 20XX

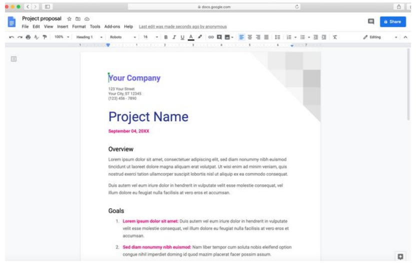
Overview

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.

Goals

1. **Lorem ipsum dolor sit amet:** Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.
2. **Sed diam nonummy nibh euismod:** Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.



docs.google.com

Project proposal

File Edit View Insert Format Tools Add-ons Help Last edit was made seconds ago by anonymous

Share

100% Heading 1 Roboto 16 Editing

Your Company

123 Your Street
Your City, ST 12345
(123) 456 - 7890

Project Name

September 04, 20XX

Overview

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

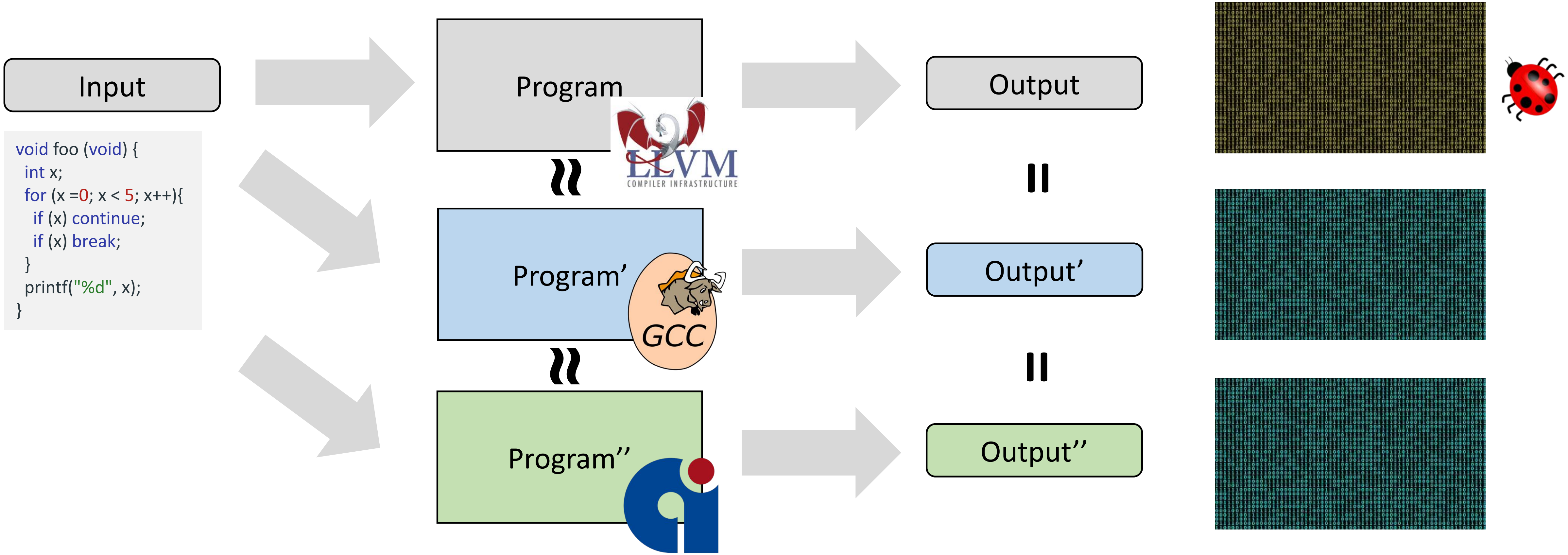
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.

Goals

1. **Lorem ipsum dolor sit amet:** Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.
2. **Sed diam nonummy nibh euismod:** Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

Provide the same input to **similar** applications, and observe output **differences**

# Differential testing: compilers (Csmith)



Input

```
void foo (void) {
  int x;
  for (x =0; x < 5; x++){
    if (x) continue;
    if (x) break;
  }
  printf("%d", x);
}
```

Program

Program’

Program’’

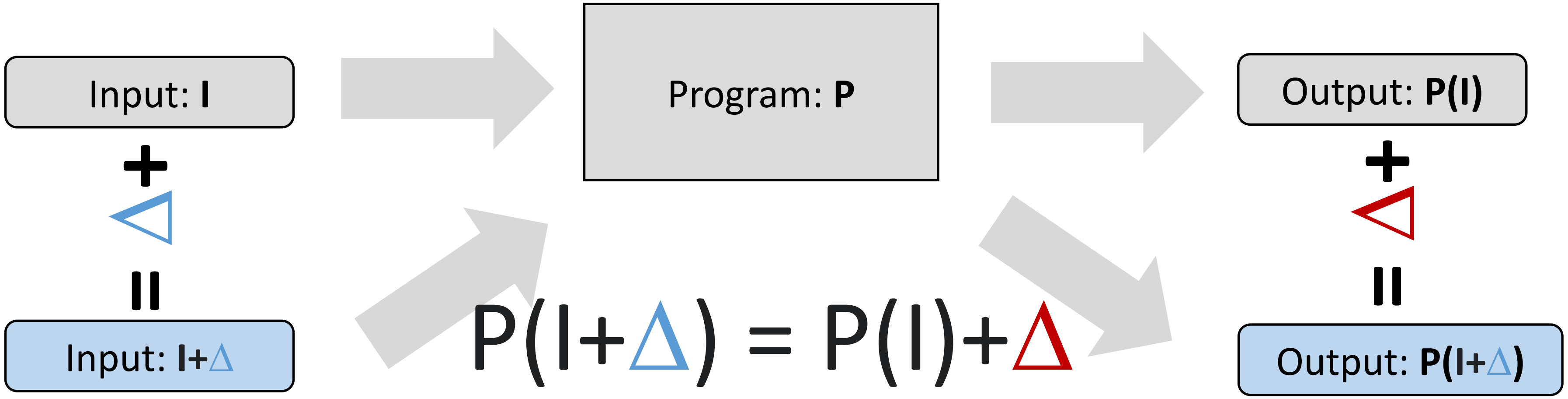
Output

Output’

Output’’

Provide the same input to **similar** applications, and observe output **differences**

# Metamorphic testing



Input: **I**

**+**

Δ

**=**

Input: **I+**Δ

Program: **P**

Output: **P(I)**

**+**

Δ

**=**

Output: **P(I+**Δ**)**

$$P(I+\Delta) = P(I)+\Delta$$

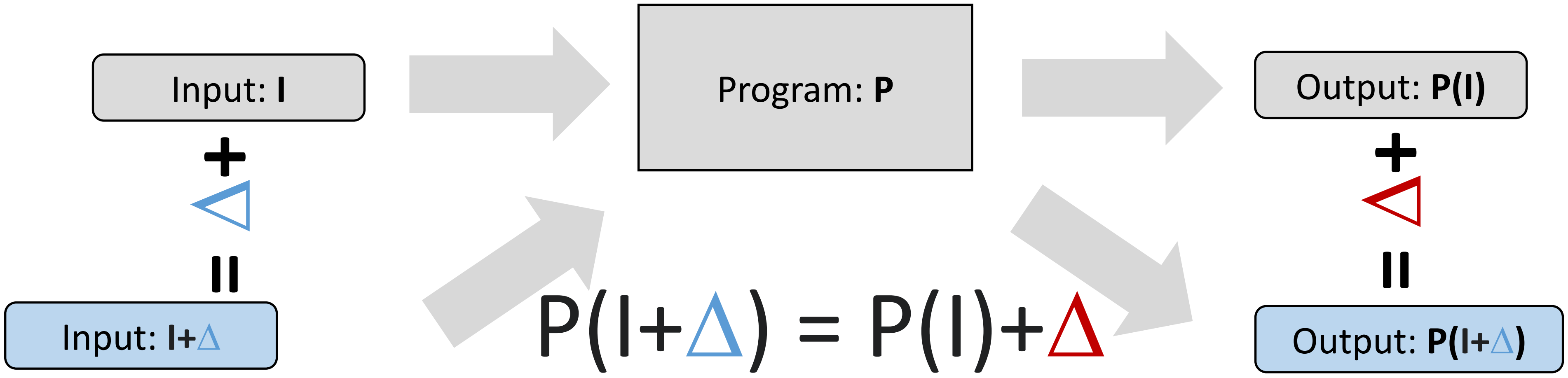
**For example:**

$$\mathrm{Sin}(x+2\pi) = \mathrm{Sin}(x)$$

$$\mathrm{Sin}(-x) = -\mathrm{Sin}(x)$$

Provide the manipulated inputs to **same** application, and observe if output **differences** are as expected

# Metamorphic testing: compilers (EMI)



GCC(

```
int a, b, c, d, e;
int main() {
 for (b = 4; b > -30; b--)
  for (; c;)
   for (;;) {
    ~~b++;~~
    e = a > 2147483647 - b;
    if (d) break;
   }
 return 0;
}
```

) = GCC(

```
int a, b, c, d, e;
int main() {
 for (b = 4; b > -30; b--)
  for (; c;)
   for (;;) {
    b++;
    e = a > 2147483647 - b;
    if (d) break;
   }
 return 0;
}
```

)+∅

Provide the manipulated inputs to **same** application, and observe if output **differences** are as expected