Bytecode Mutation for Differential Testing of JVM Implementations

Yuting Chen

Shanghai Jiao Tong University

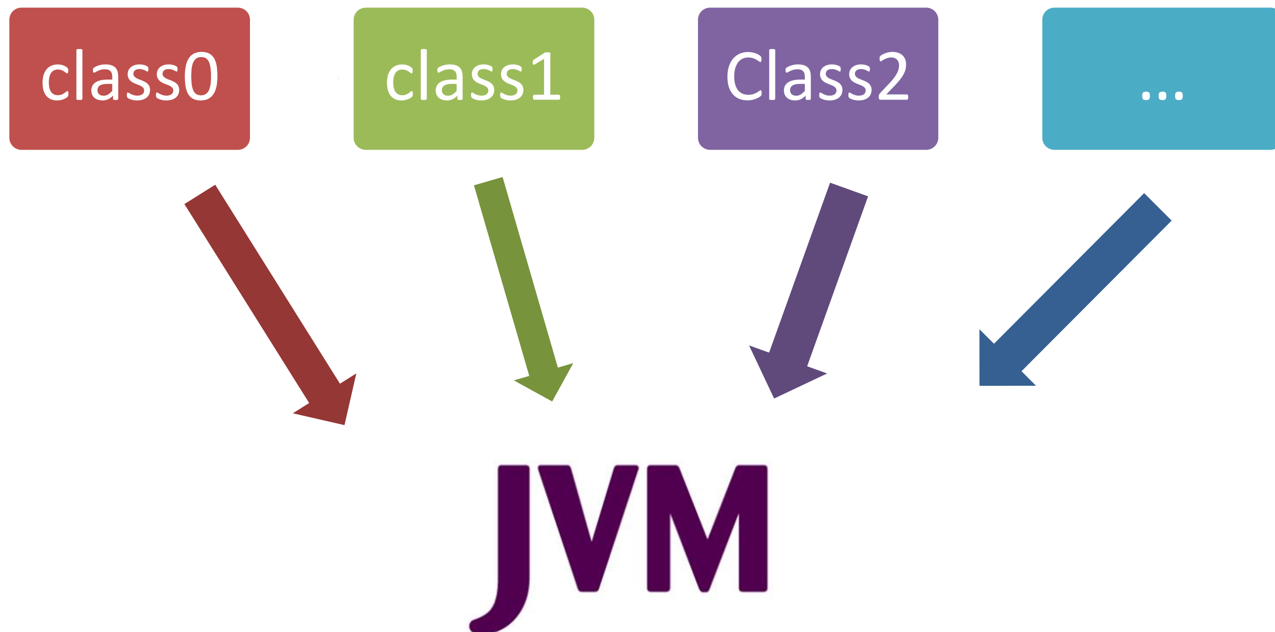
Outline

- Motivation
  - Testing of JVMs
  - Test redundancy
- Design： classfuzz
- Conclusion

Background: JVM

JVM Testing

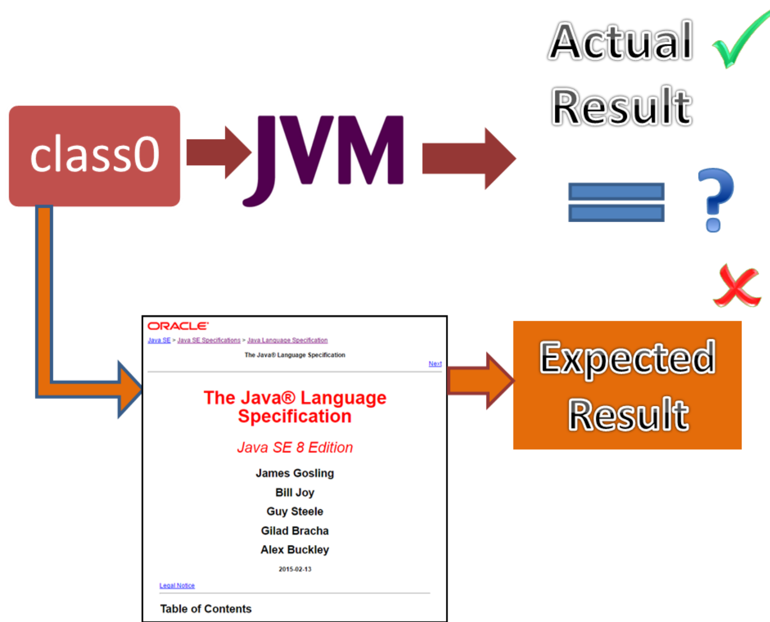
- Testing a JVM using a number of test classfiles



Challenge 1: How to expose a JVM defect?

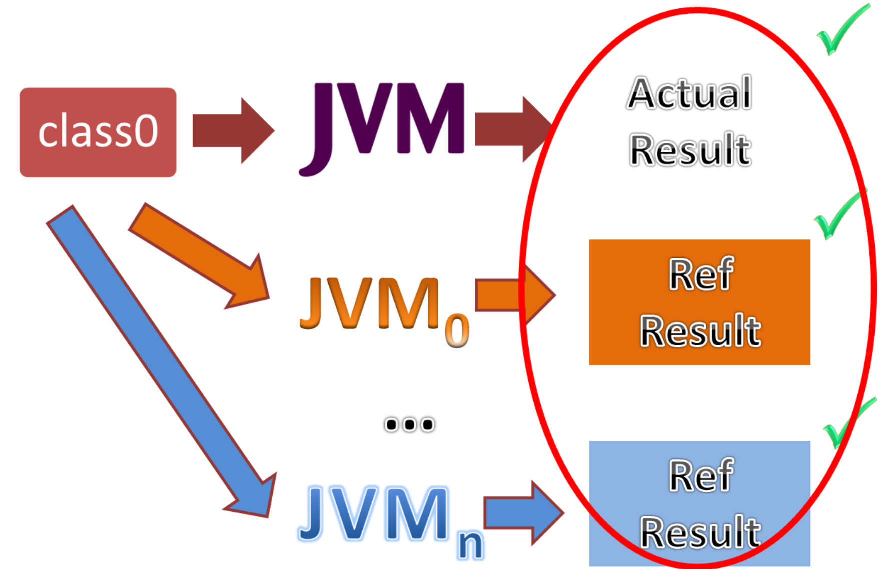
Challenge:

no test oracles



Solution:

differential JVM testing



An Example of JVM Behavior Discrepancy

```
1  ...
2   MD5 checksum 8fb69050bbcb9a83ddd90ae393368c5e
3  ...
4  class M1436188543
5   minor version: 0
6   major version: 51
7   flags: ACC_SUPER
8  Constant pool:
9   ...
10   #7 = Utf8 <clinit>
11   #8 = Utf8 ()V
12   #9 = Class #19 // java/lang/System
13   #10 = Utf8 Code
14   #11 = Utf8 main
15  ...
16 {
17  public abstract {};
18   flags: ACC_PUBLIC, ACC_ABSTRACT
19  public static void main(java.lang.String[]);
20   flags: ACC_PUBLIC, ACC_STATIC
21  Code:
22   stack=2, locals=1, args_size=1
23   0: getstatic #12 // Field java/lang/System.out:
        Ljava/io/PrintStream;
24   3: ldc #4 // String Completed!
25   5: invokevirtual #21 // Method java/io/
        PrintStream.println:(Ljava/lang/String;)V
26   8: return
27 }
```

public abstract {};

- HotSpot takes it as a ordinary method
- J9 reports a format error

Cause: the JVM specification says that "other methods named <clinit> in a class file are of no consequence. They are not class or interface initialization methods. "

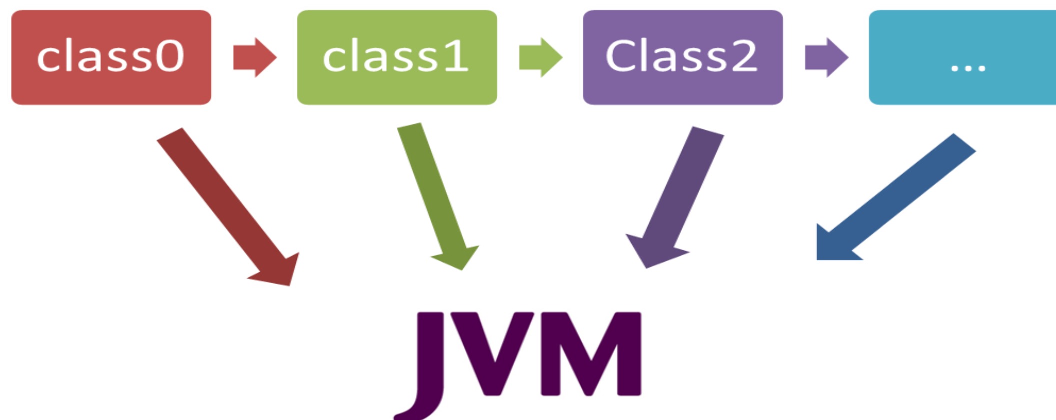
A class method needs to be more strictly defined

Challenge 2: How to obtain test classfiles?

- Option 1: using the real-world classfiles

Some classes can reveal compatibility issues

- Option 2: domain-aware fuzz testing



An Example of JVM Behavior Discrepancy

```
1   ...
2    MD5 checksum 8fb69050bbcb9a83ddd90ae393368c5e
3   ...
4   class M1436188543
5    minor version: 0
6    major version: 51
7    flags: ACC_SUPER
8   Constant pool:
9    ...
10    #7 = Utf8 <clinit>
11    #8 = Utf8 ()V
12    #9 = Class #19 // java/lang/System
13    #10 = Utf8 Code
14    #11 = Utf8 main
15   ...
16  {
17   public abstract {};
18    flags: ACC_PUBLIC, ACC_ABSTRACT
19   public static void main(java.lang.String[]);
20    flags: ACC_PUBLIC, ACC_STATIC
21   Code:
22    stack=2, locals=1, args_size=1
23    0: getstatic #12 // Field java/lang/System.out:
          Ljava/io/PrintStream;
24    3: ldc #4 // String Completed!
25    5: invokevirtual #21 // Method java/io/
          PrintStream.println:(Ljava/lang/String;)V
26    8: return
27  }
```

public abstract ~~mymethod~~{};

clinit

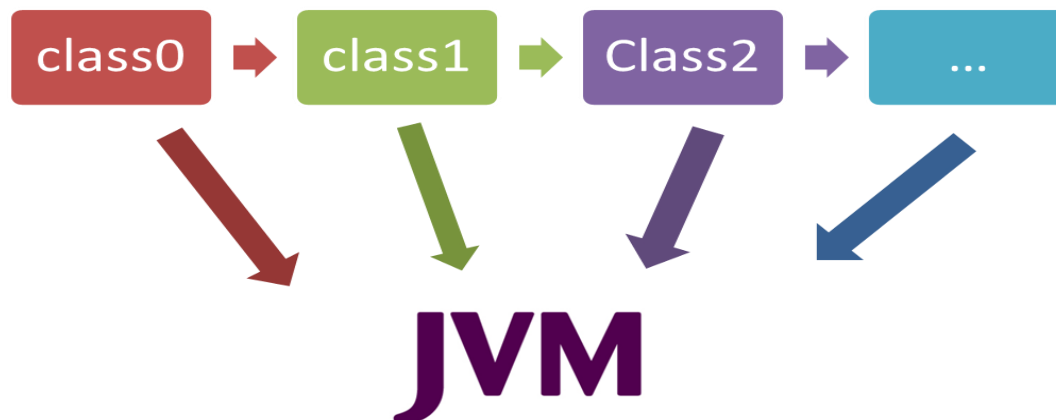
More JVM discrepancies are revealed by domain-aware fuzz testing

Challenge 2: How to obtain test classfiles?

- Option 1: using the real-world classfiles

Some classes can reveal compatibility issues

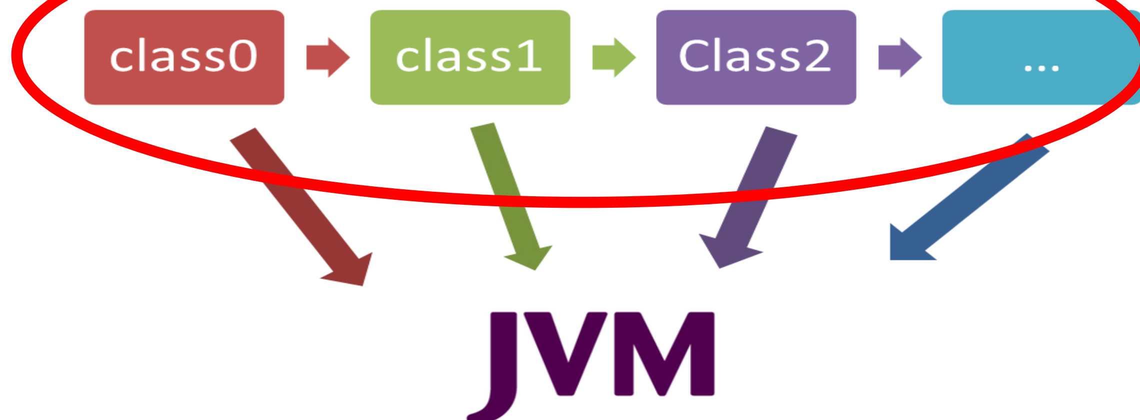
- Option 2: domain-aware fuzz testing



Challenge 2: How to obtain test classfiles?

- **An infinite number of test classfiles can be created**
- **They may reveal a small number of JVM discrepancies**

- Solution 2: domain-aware fuzz testing



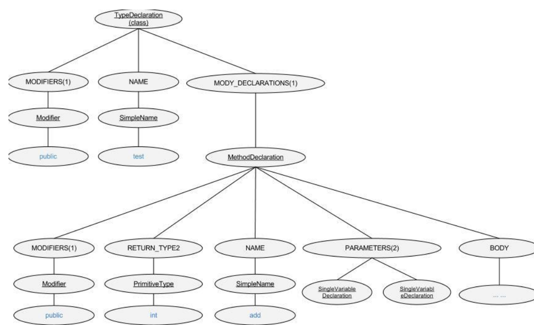
Outline

- Motivation
  - Testing of JVMs
  - Test redundancy
- Design： classfuzz
- Conclusion

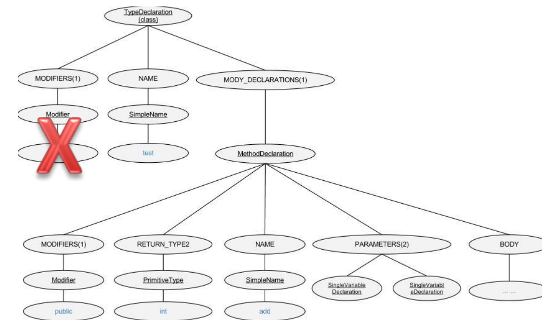
Key Observation (1)

- A classfile can encompass intricate constraints
  - Corner cases can be created through rewriting seeds

legal



Syntax tree

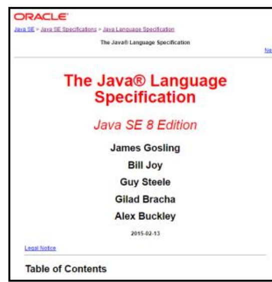


legal

or

illegal

Revised syntax tree



ORACLE

Java SE > Java SE Specifications > Java Language Specification

The Java® Language Specification

Next

The Java® Language Specification

Java SE 8 Edition

James Gosling

Bill Joy

Guy Steele

Gilad Bracha

Alex Buckley

2015-02-13

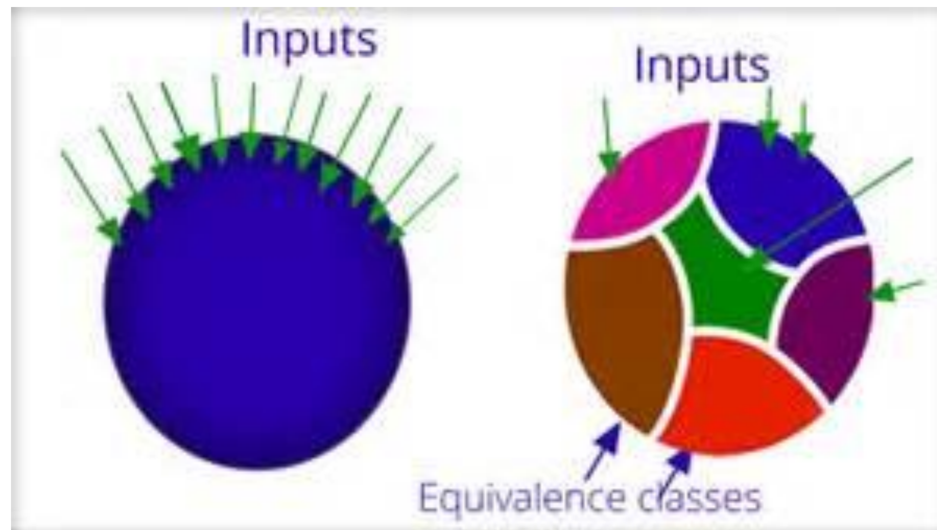
Legal Notice

Table of Contents

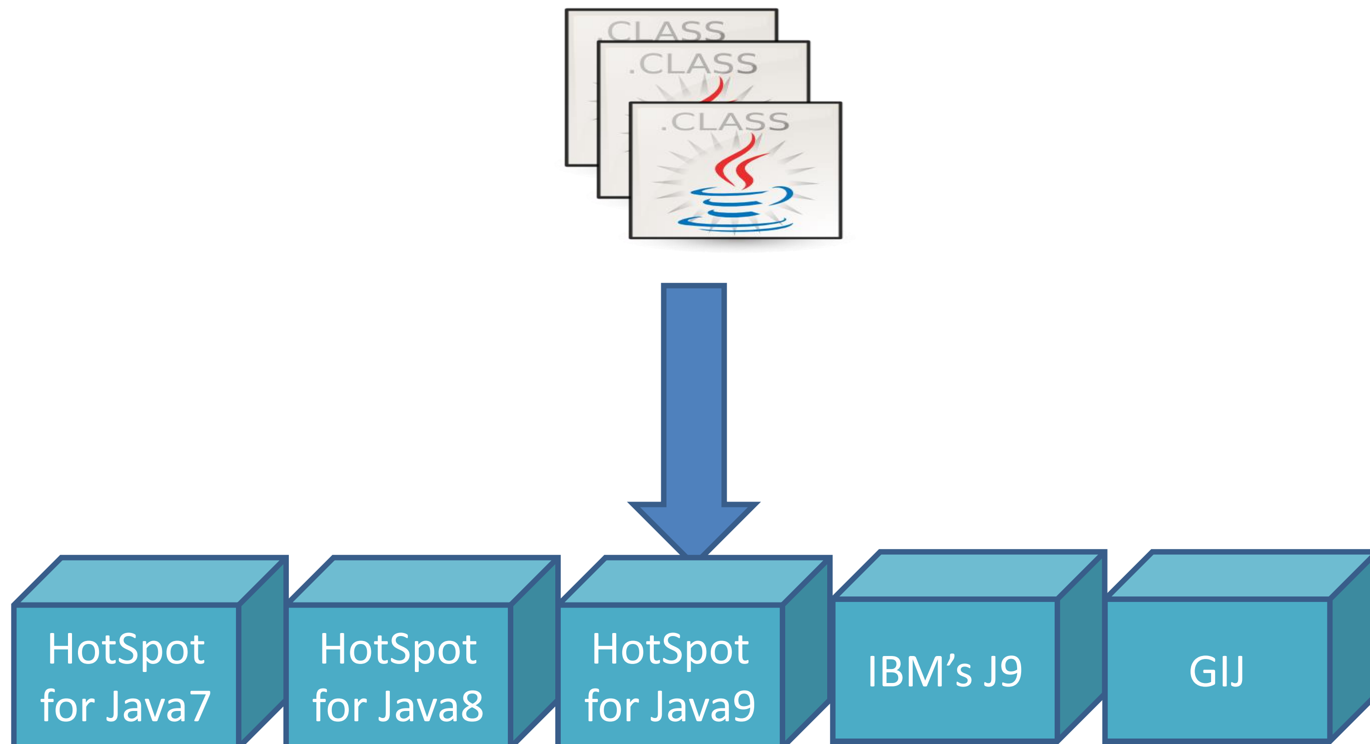
JVM

Key Observation (2)

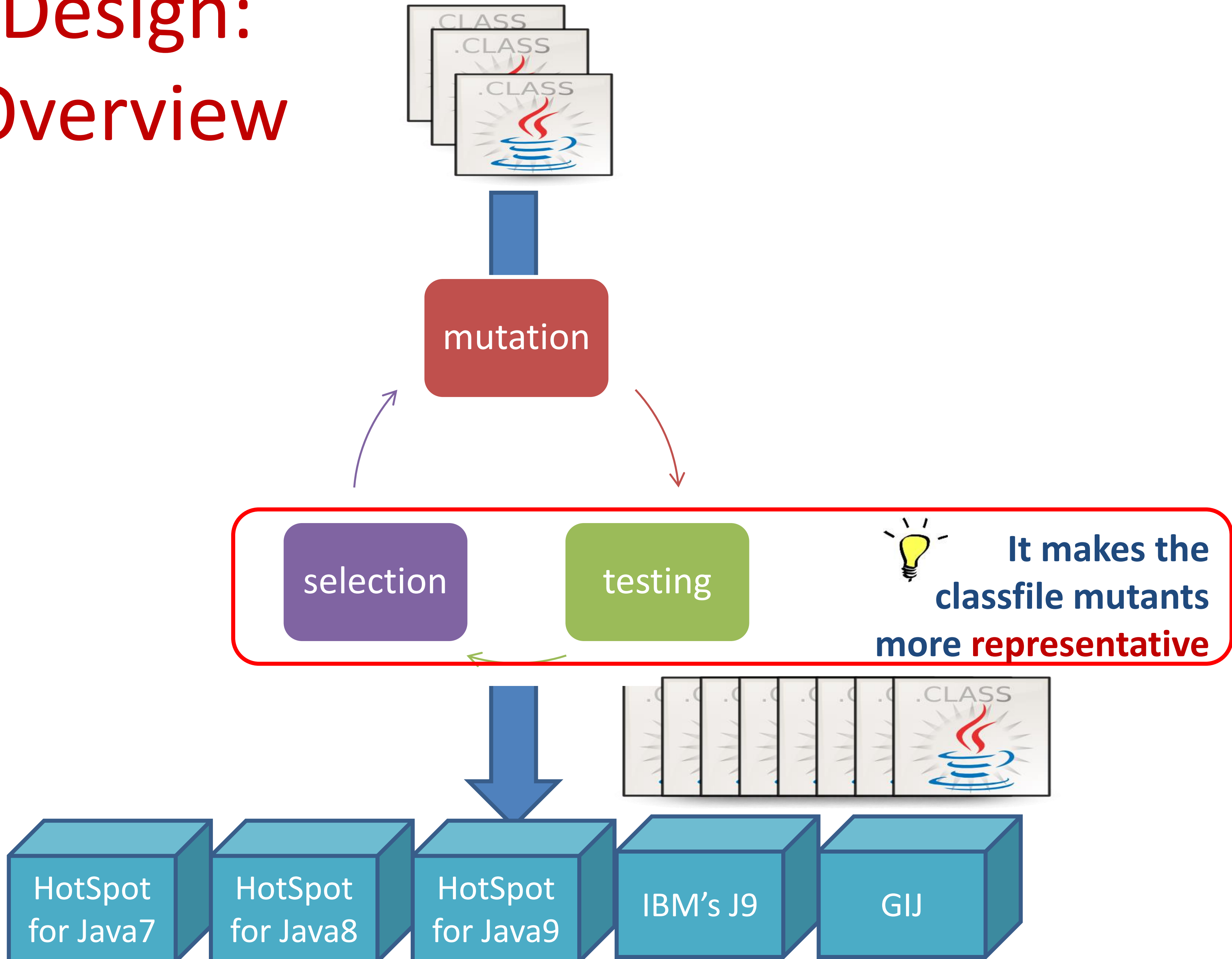
- Equivalence class partition (ECP) saves the testing cost
  - ECP works only if we can decide whether two tests belong to the same partition



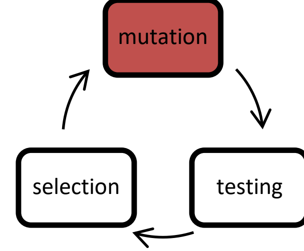
Our Design:
An Overview

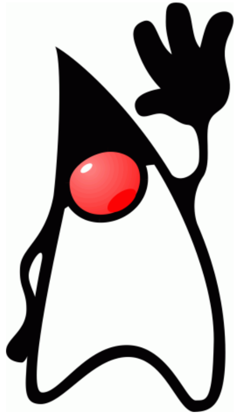


Our Design: An Overview



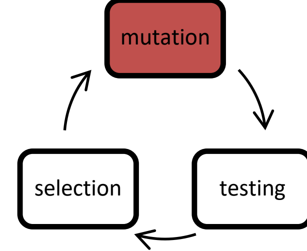
Mutating Classfiles





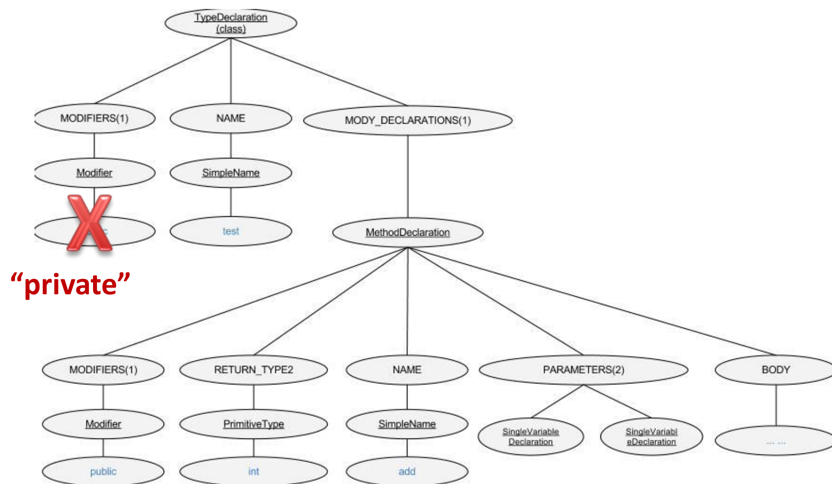
We have designed 129 mutators for mutating classfiles

Mutating Classfiles (2)



123 mutators are designed for rewriting the ASTs of the seeds

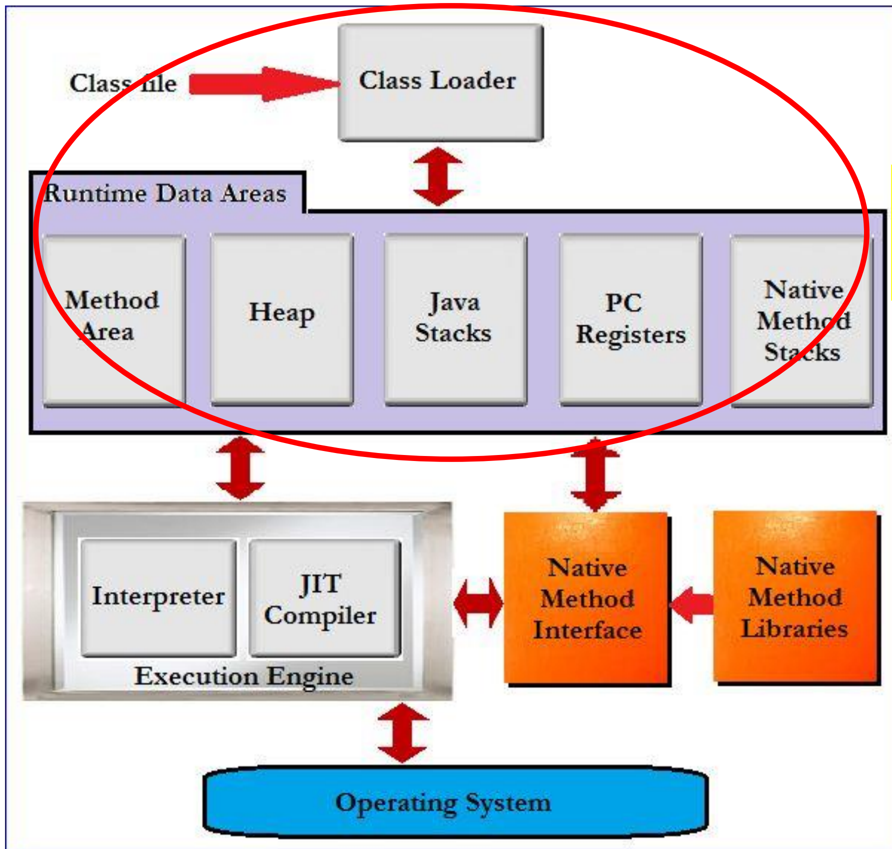
Six mutators are designed for rewriting the Jimple files of the seeds




```
r0:=parameter0: java.lang.String[];

r1:=<java.lang.System: java.io.PrintStream out>;

virtualinvoke $r1.<java.io.PrintStream:void
println(java.lang.String)>("Executed");

...
```

- **Limitation: Only the JVMs' startup processes can be tested**
  - **The mutated program constructs/attributes may be less likely to be activated during execution**

JVM Startup



Errors and exceptions

Creation and loading

Errors and exceptions

Linking

Errors and exceptions

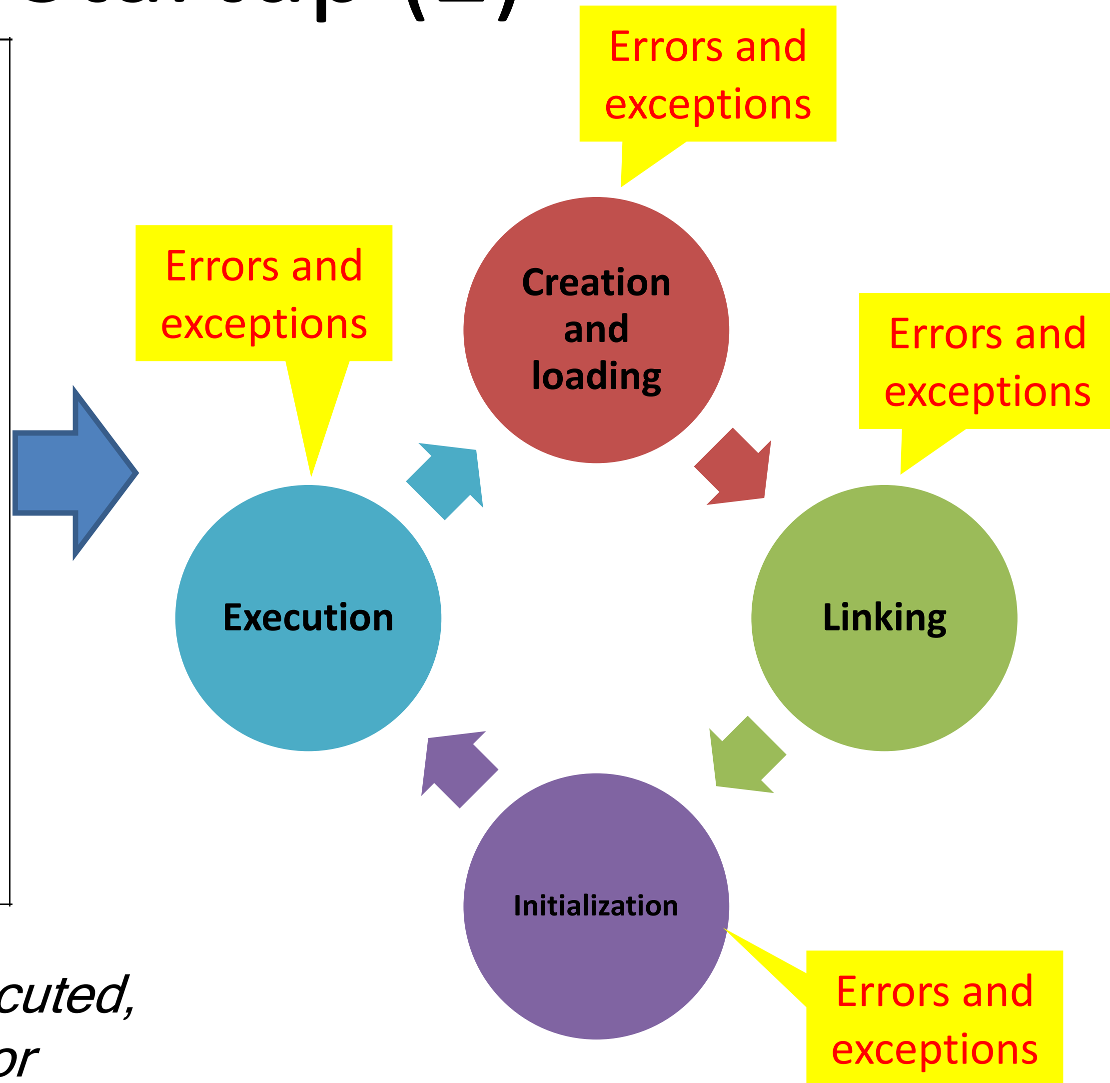
Initialization

Errors and exceptions

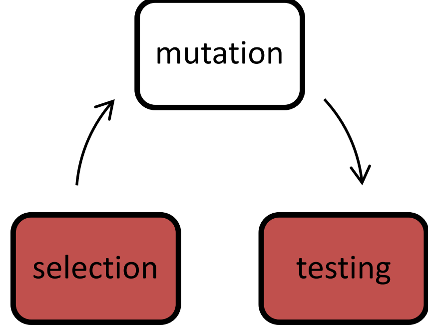
Execution

JVM Startup (2)

```
...
 MD5 checksum 8fb69050bbcb9a83ddd90ae393368c5e
...
class M1436188543
 minor version: 0
 major version: 51
 flags: ACC_SUPER
Constant pool:
 ...
  #7 = Utf8 <clinit>
  #8 = Utf8 ()V
  #9 = Class #19 // java/lang/System
  #10 = Utf8 Code
  #11 = Utf8 main
 ...
{
 public abstract {};
  flags: ACC_PUBLIC, ACC_ABSTRACT
 public static void main(java.lang.String[]);
  flags: ACC_PUBLIC, ACC_STATIC
 Code:
  stack=2, locals=1, args_size=1
  0: getstatic #12 // Field java/lang/System.out:
       Ljava/io/PrintStream;
  3: ldc #4 // String Completed!
  5: invokevirtual #21 // Method java/io/
       PrintStream.println:(Ljava/lang/String;)V
  8: return
}
```

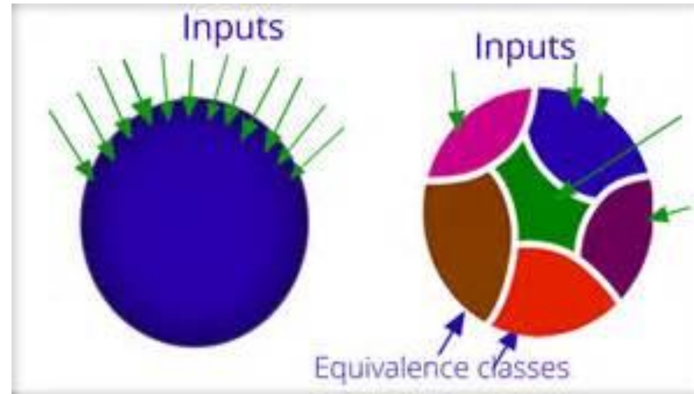


Can this class be normally executed, or at which stage some errors or exceptions can be thrown out?

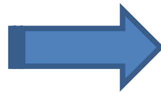


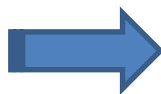
Selecting Representative Classfiles

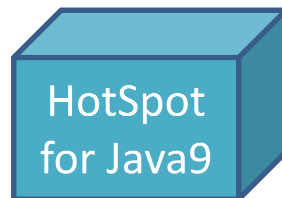
- ECP



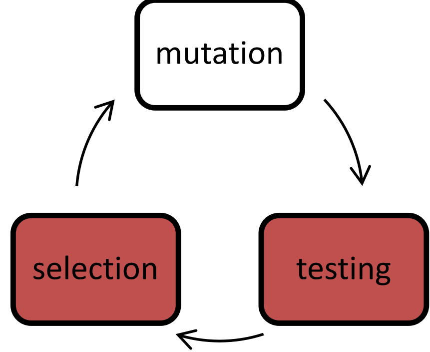
- Do two classfiles belong to the same class?







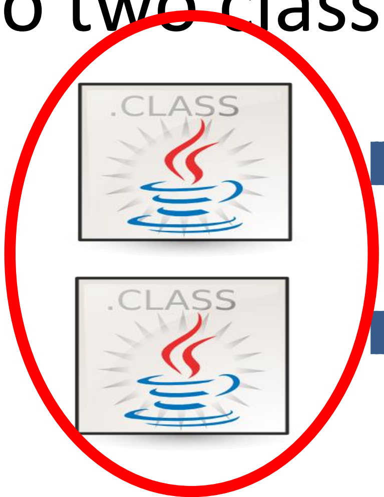
- **Yes, if the reference JVM equally processes them**
- **No, otherwise**

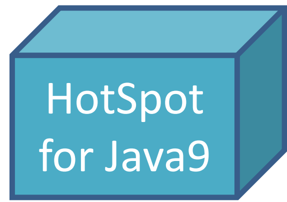


Selecting Representative Classfiles (2)

- Do two classfiles belong to the same class?

Seeds





7000 smts, 2300 branches

7020 stmts, 2340 branches

New mutant

? stmts, ? branches

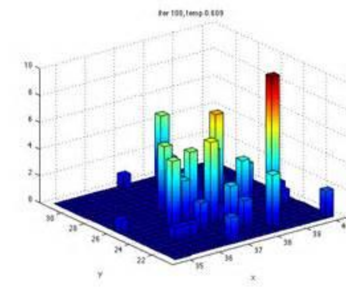
Is it representative?

Several comparison criteria can be given here

Selecting Mutators

- Goal: to create as many representative classfiles as possible
- Fact: mutators are designed arbitrarily; some are effective, while some others are useless
- A naïve solution: to select mutators by learning from prior knowledge

An MCMC Sampling Method



- Which mutator will be selected at each step?
  - A desired distribution: geometric distribution

A has higher success rate than B, and thus it should be easier to be selected than B

The actual distribution

A

B

Proposition: The more number of representative classfiles have been created by a mutator, the more likely the mutator should be selected for further mutations

More Details

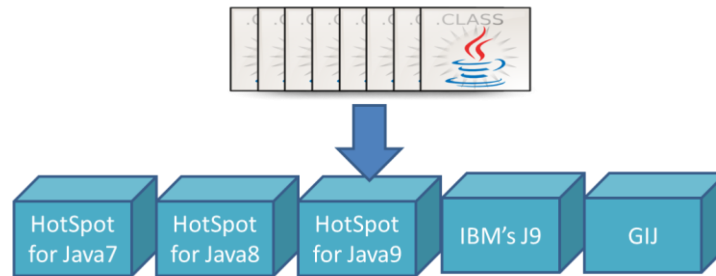
- The desired distribution

$$\Pr(X = k) = (1-p)^{k-1}\,p$$

- Classfuzz picks up mutators at random, and then accepts or rejects the mutators by a Metropolis choice

$$\begin{aligned} A(mu_1 \rightarrow mu_2) &= min(1, \frac{\Pr(mu_2)}{\Pr(mu_1)}) \\ &= min(1, (1-p)^{k_2-k_1}) \end{aligned}$$

Execution Comparison



$result_0$ = jvm_0 (env_0, c, input)

$result_1$ = jvm_1 (env_1, c, input)

$result_2$ = jvm_2 (env_2, c, input)

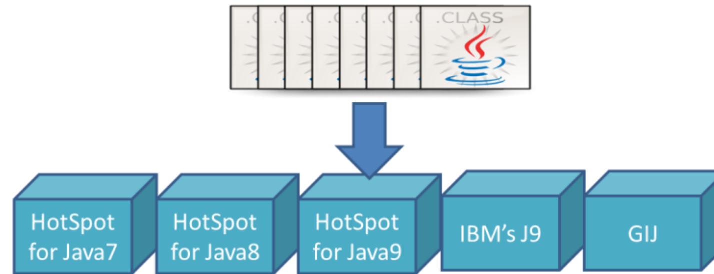
$result_3$ = jvm_3 (env_3, c, input)

$result_4$ = jvm_4 (env_4, c, input)

A *JVM discrepancy* appears when $result_i \neq result_j$

It can either be a JVM defect or a compatibility issue

Execution Comparison (2)



$result_0 = jvm_0 (env_0, c, input)$

$result_1 = jvm_1 (env_0, c, input)$

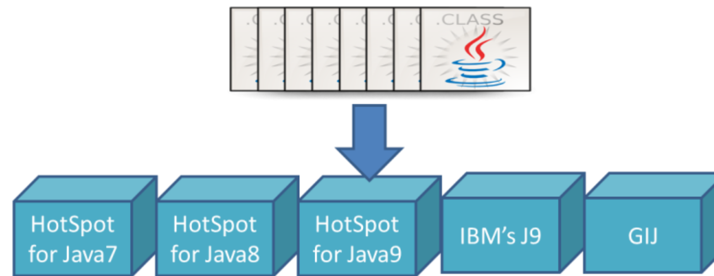
$result_2 = jvm_2 (env_0, c, input)$

$result_3 = jvm_3 (env_0, c, input)$

$result_4 = jvm_4 (env_0, c, input)$

A *JVM defect* appears when $result_i \neq result_j$

Execution Comparison (3)



$result_0 = jvm_0\ (env_0, c, input)$

$result_1 = jvm_1\ (env_0, c, input)$

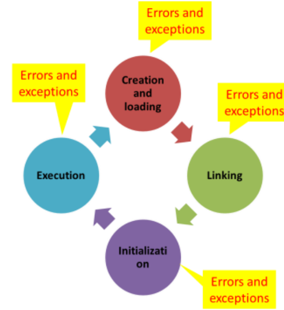
$result_2 = jvm_2\ (env_0, c, input)$

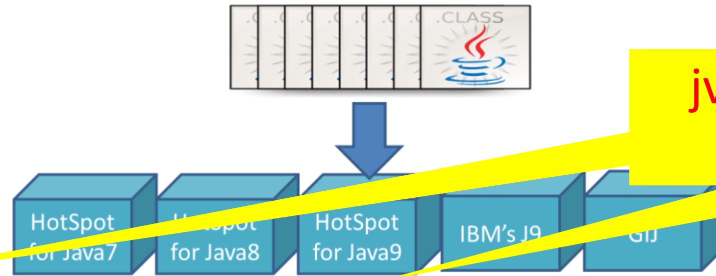
$result_3 = jvm_3\ (env_0, c, input)$

$result_4 = jvm_4\ (env_0, c, input)$

A *JVM defect* appears when $result_i \neq result_j$

Execution Comparison (3)





jvm_0 may miss catching some format errors

0 ⇐ $result_0 = jvm_0 (env_0, c, input)$

1 ⇐ $result_1 = jvm_1 (env_0, c, input)$

1 ⇐ $result_2 = jvm_2 (env_0, c, input)$

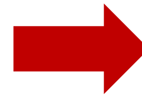
1 ⇐ $result_3 = jvm_3 (env_0, c, input)$

1 ⇐ $result_4 = jvm_4 (env_0, c, input)$

A *JVM defect* appears when $result_i \neq result_j$

Evaluation Setup

- Coverage collection
  - HotSpot for Java9
  - GCOV + LCOV



At each run the coverage can be conveniently collected

HotSpot (260K LOCs)
Cost for cov. analysis: 30+ mins

share/vm/classfile/ (11977 LOCs)
Cost for cov. analysis: 90 secs

- Seeds
  - 1216 classfiles in JRE 7

Evaluated Methods

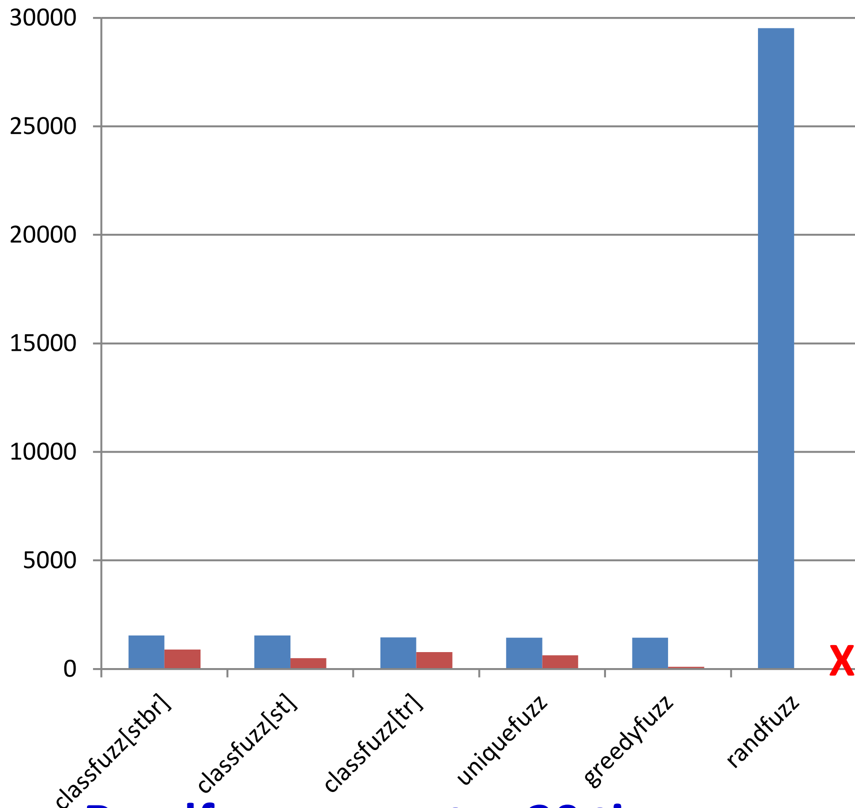
- Classfuzz supplemented with a uniqueness criterion
  - [st], [stbr], [tr] – explained in the paper
- Randfuzz, Greedyfuzz, Uniquefuzz

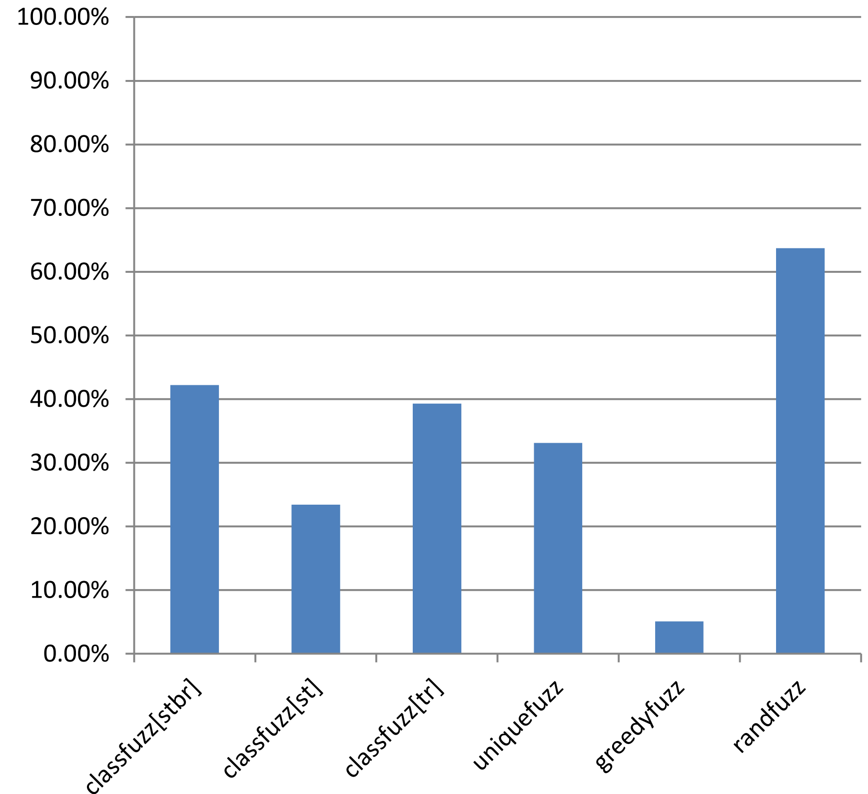
	classfuzz	randfuzz	greedyfuzz	uniquefuzz
Mutation-based	✓	✓	✓	✓
Cov. analysis	✓	×	✓	✓
Uniqueness criterion	[st] [stbr] [tr]	×	[stbr]	[stbr]
Mutator selection	✓	×	×	×

Metrics

- **RQ1**: How many test classfiles can be generated?
  - *#Iterations, |GenClasses|, |TestClasses|*
- **RQ2**: How effective are the test classfiles?
  - *|Discrepancies|, |Distinct Discrepancies|, diff rate*
- **RQ3**: Can the test classfiles find any JVM defects?

Results on Classfile Generation





- **Randfuzz generates 20 times as many classfiles as those generated by any other algorithm**
- **Classfuzz[stbr] generates the most number of representative classfiles**

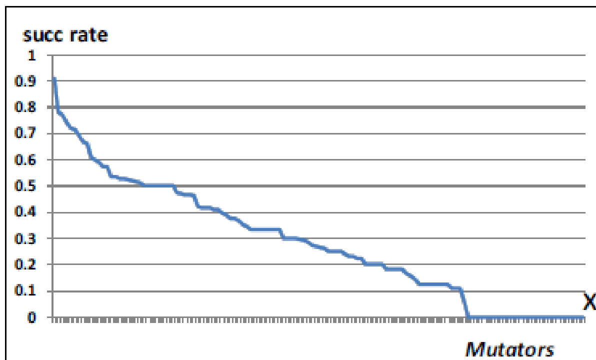
Classfuzz[stbr] achieves the highest success rate among all the coverage-directed algorithms

Results on Classfile Generation (2)

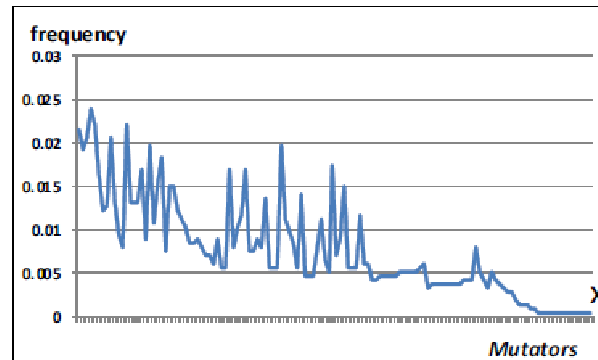
- Classfuzz can utilize the prior knowledge to select mutators

strongly correlated

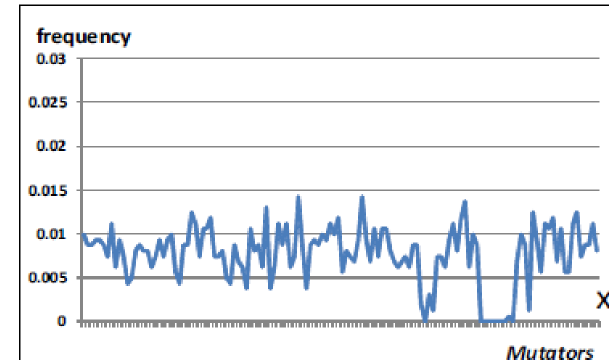
succ rate **classfuzz** **randfuzz**



(a) Success rates of mutators for generating $TestClasses_{classfuzz[stbr]}$.



(b) Frequencies of mutators for generating $TestClasses_{classfuzz[stbr]}$.

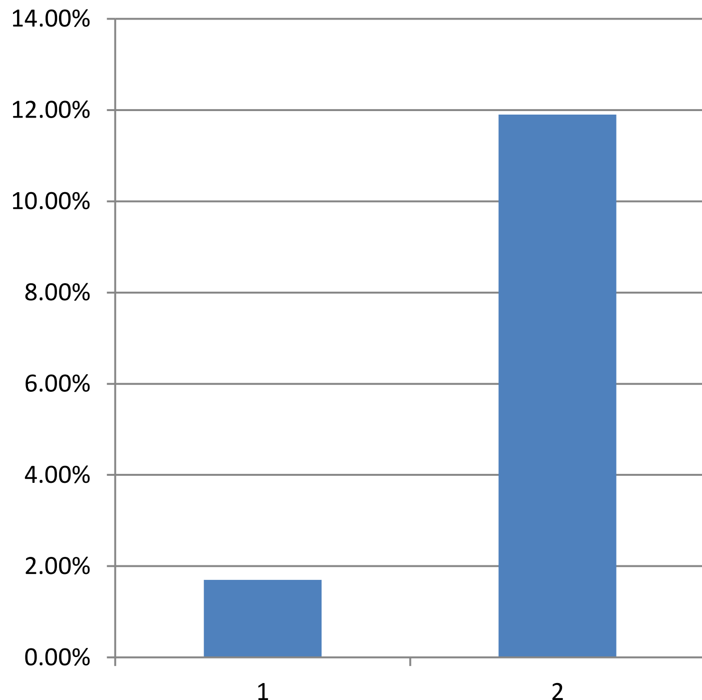


(c) Frequencies of mutators for generating $TestClasses_{uniquefuzz}$.

weakly correlated

Results on Differential JVM Testing

- Classfuzz can enhance the ratio of discrepancy triggering classfiles from 1.7% to 11.9%



- **JVMs are compatible for most of the classfiles, but differ in processing corner cases**
- **We have experienced 898 different execution paths. 107 paths were related to JVM behavior differences**

Discrepancy Analysis (1)

```
1  ...
2   MD5 checksum 8fb69050bbcb9a83ddd90ae393368c5e
3  ...
4  class M1436188543
5   minor version: 0
6   major version: 51
7   flags: ACC_SUPER
8  Constant pool:
9   ...
10   #7 = Utf8 <clinit>
11   #8 = Utf8 ()V
12   #9 = Class #19 // java/lang/System
13   #10 = Utf8 Code
14   #11 = Utf8 main
15  ...
16 {
17  public abstract {};
18   flags: ACC_PUBLIC, ACC_ABSTRACT
19  public static void main(java.lang.String[]);
20   flags: ACC_PUBLIC, ACC_STATIC
21  Code:
22   stack=2, locals=1, args_size=1
23   0: getstatic #12 // Field java/lang/System.out:
        Ljava/io/PrintStream;
24   3: ldc #4 // String Completed!
25   5: invokevirtual #21 // Method java/io/
        PrintStream.println:(Ljava/lang/String;)V
26   8: return
27 }
```

public abstract {};

- HotSpot takes it as a ordinary method
- J9 reports a format error

The JVM specification needs to be clarified

Discrepancy Analysis (2)

```
//The Jimple code of M1433982529
public class M1433982529 extends java.lang.Object
{
  protected void internalTransform(java.lang.
      String)
  {
    java.util.Map r0;
    r0 := @parameter0: java.util.Map;
    staticinvoke <java.lang.Object: boolean
        getBoolean(java.util.Map)>(r0);
    return;
  }
}
```

A type casting needs to be performed

JVMs take their own classfile verification and type checking polices

Discrepancy Analysis (3)

```
//The source code of sun.java2d.pisces.
    PiscesRenderingEngine
public class PiscesRenderingEngine extends
    RenderingEngine {
  ...
  private static enum NormMode {OFF, ON_NO_AA,
      ON_WITH_AA};
  ...
}
//The Jimple code of M1437121261
public class M1437121261 {
  public static void main (String[] r0)
      throws sun.java2d.pisces.
          PiscesRenderingEngine$2{
    ...
  }
}
```

JVMs are not compatible to access some classes

Discrepancy Analysis (4)

- More findings
  - J9 is less strict than HotSpot because J9 only verifies a method when it is invoked, while HotSpot verifies all methods before execution
  - GIJ can execute an interface having a main method
  - GIJ accepts a class with duplicate fields
  - …
- These discrepancies can be found in a package of 62 discrepancy-triggering classes

Conclusions

- JVMs can format check, verify, run a classfile differently
- Classfuzz is designed to generate test classfiles for differently testing JVMs