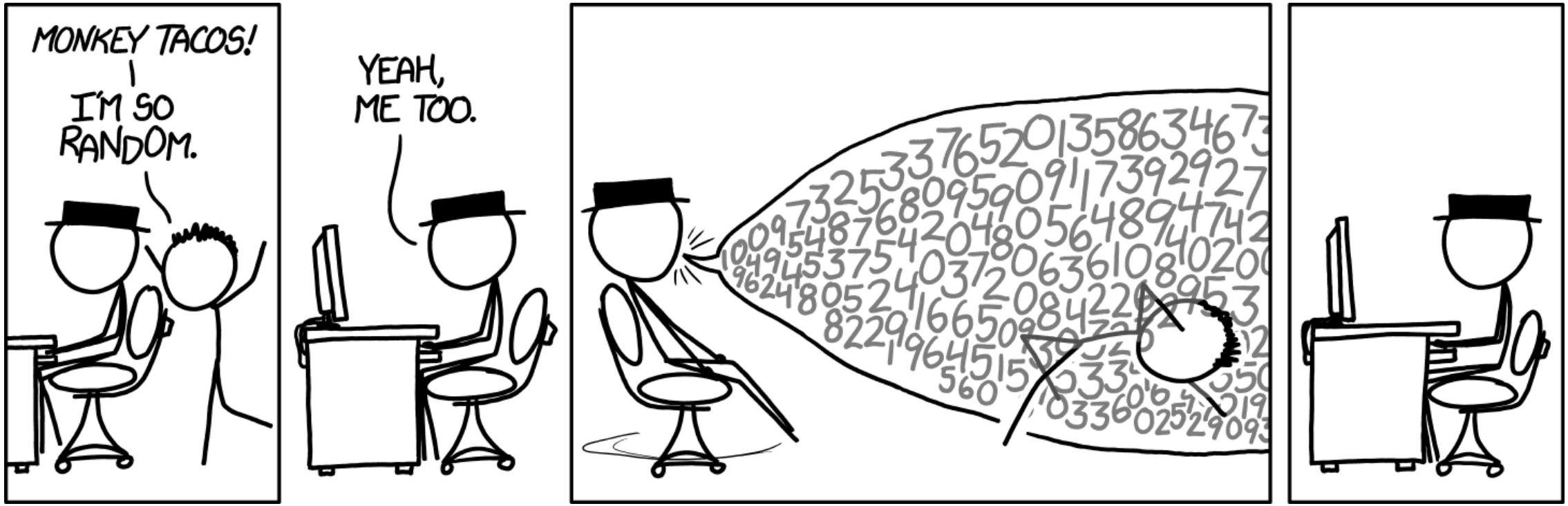
Semantic Fuzzing *with Zest*

Rohan Padhye, Caroline Lemieux, Koushik Sen, Mike Papadakis, Yves Le Traon

Berkeley
UNIVERSITY OF CALIFORNIA

UNIVERSITÉ DU
LUXEMBOURG

Fuzz Testing



Fuzz Testing is Extremely *Popular* and *Effective*

Releasing jsfunfuzz and DOMFuzz
Tuesday, July 28th, 2015

Today I'm releasing two fuzzers: jsfunfuzz, which tests JavaScript engines, and DOMFuzz, which tests layout and DOM APIs.

Over the last 11 years, these fuzzers have found 6450 Firefox bugs, including 790 bugs that were rated as security-critical.

What is Microsoft Security Risk Detection?

Security Risk Detection is Microsoft's unique fuzz testing service for finding security critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

Google Testing Blog

Announcing OSS-Fuzz: Continuous Fuzzing for Open Source Software

Thursday, December 01, 2016

Linux 4.14-rc5

From: Linus Torvalds
Date: Sun Oct 15 2017 - 21:48:40 EST

The other thing perhaps worth mentioning is how much random fuzzing people are doing, and it's finding things. We've always done fuzzing (who remembers the old "crashme" program that just generated random code and jumped to it? We used to do that quite actively very early on), but people have been doing some nice targeted fuzzing of driver subsystems etc, and there's been various fixes (not just this last week either) coming out of those efforts. Very nice to see.

CVE-2014-6277: “ShellShock” bug in Bash

CVE-2014-0160: “Heartbleed” bug in OpenSSL

CVE-2015-1606
CVE-2015-1607
CVE-2014-9087
CVE-2014-6355
CVE-2015-0061
CVE-2015-7855
CVE-2016-7434
CVE-2015-7941
CVE-2015-8035
CVE-2015-8241
CVE-2015-8242
CVE-2015-8317
CVE-2016-4658
CVE-2016-5131
CVE-2015-5309
CVE-2015-5311
CVE-2015-0232
CVE-2017-5340
CVE-2015-2158
CVE-2015-0860
CVE-2015-8380
CVE-2016-1925
CVE-2014-9771
CVE-2016-3994

Fuzz Testing is Extremely *Popular* and *Effective*

Releasing jsfunfuzz and DOMFuzz
Tuesday, July 28th, 2015

Today I'm releasing
and DOMFuzz, which

Over the last 11 y
including 790 bugs t

Security Ri

Buffer overflows

Memory leaks

Use-after-free

…

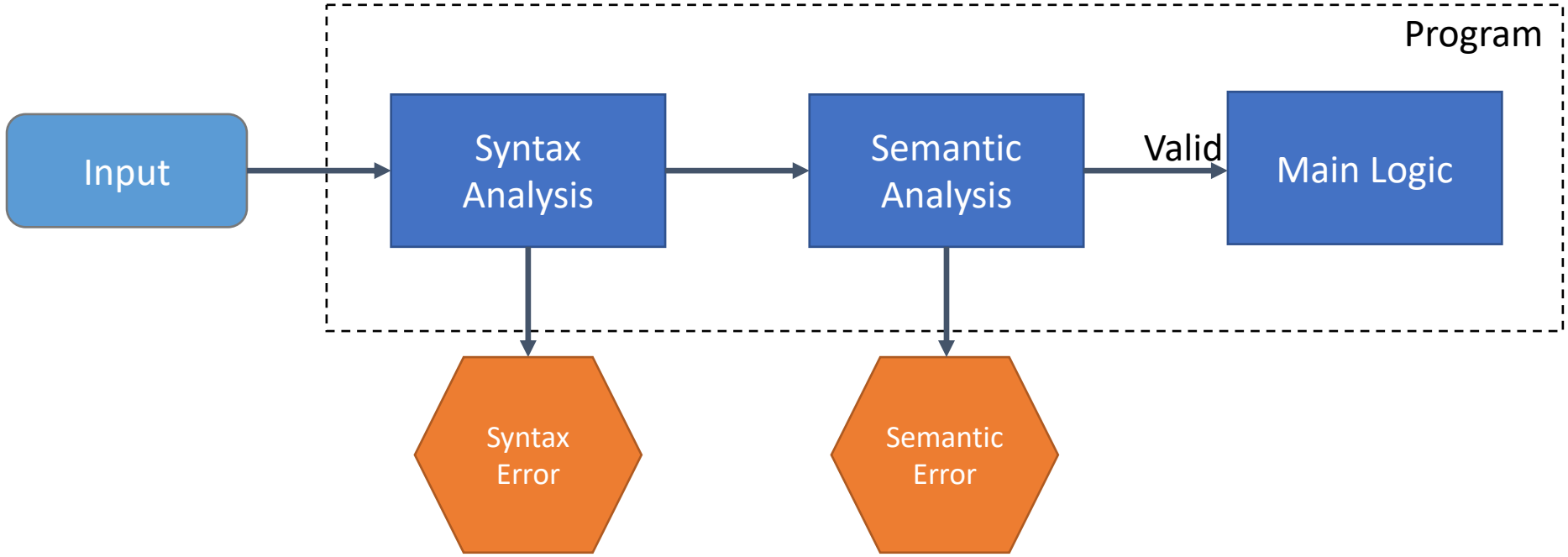
CVE-2014-

CVE-2014-0160: “Heartbleed” bug in OpenSSL

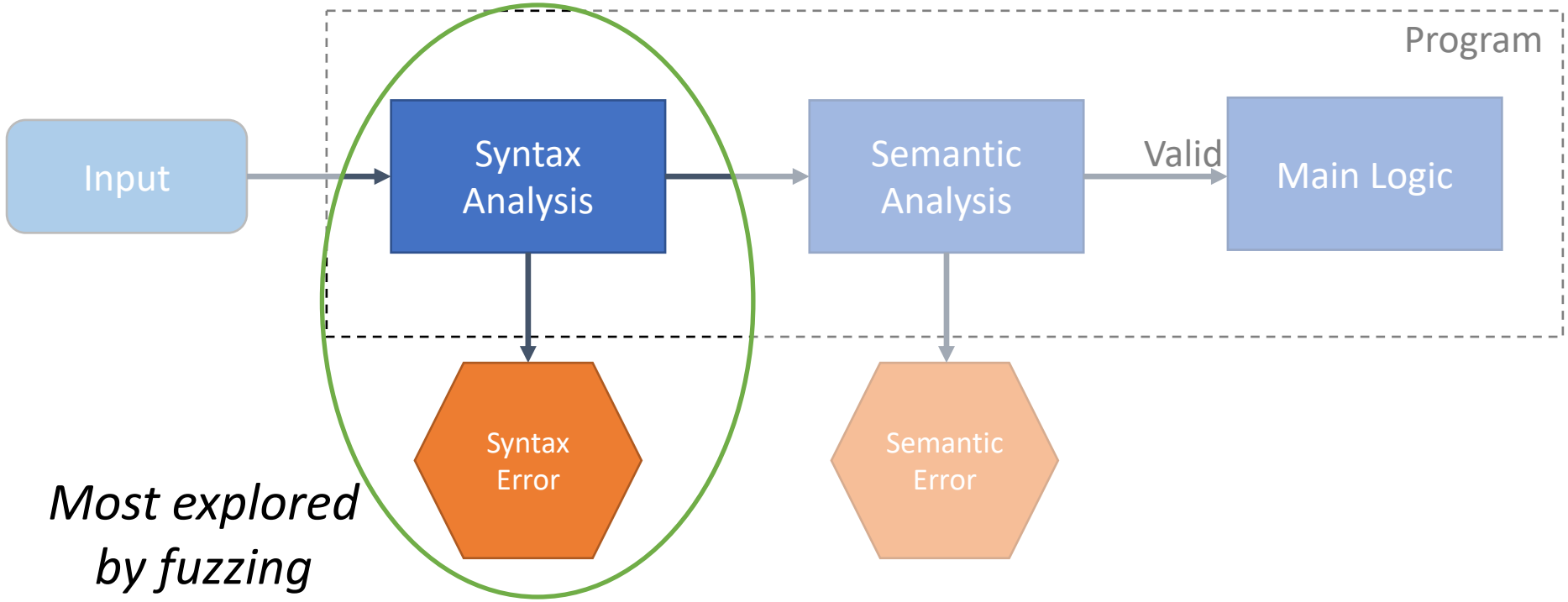
code and jumped to it? We used to do that quite actively very early on), but people have been doing some nice targeted fuzzing of driver subsystems etc, and there's been various fixes (not just this last week either) coming out of those efforts. Very nice to see.

CVE-2015-1606
CVE-2015-1607
CVE-2014-9087
CVE-2014-6355
CVE-2015-0061
CVE-2015-7855
CVE-2016-7434
CVE-2015-7941
CVE-2015-8035
CVE-2015-8241
CVE-2015-8242
CVE-2015-8317
CVE-2016-4658
CVE-2016-5131
CVE-2015-5309
CVE-2015-5311
CVE-2015-0232
CVE-2017-5340
CVE-2015-2158
CVE-2015-0860
CVE-2015-8380
CVE-2016-1925
CVE-2014-9771
CVE-2016-3994

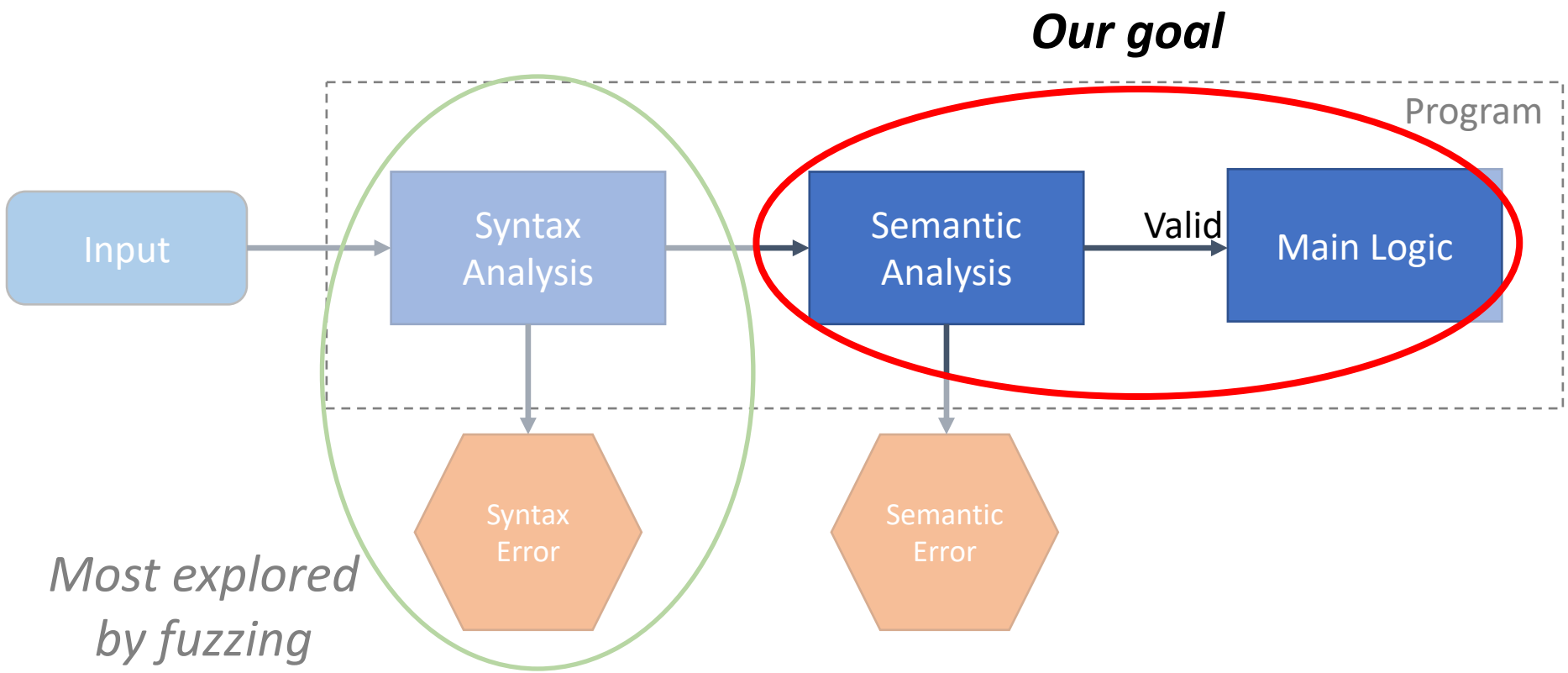
Many test programs look like this:



Many test programs look like this:

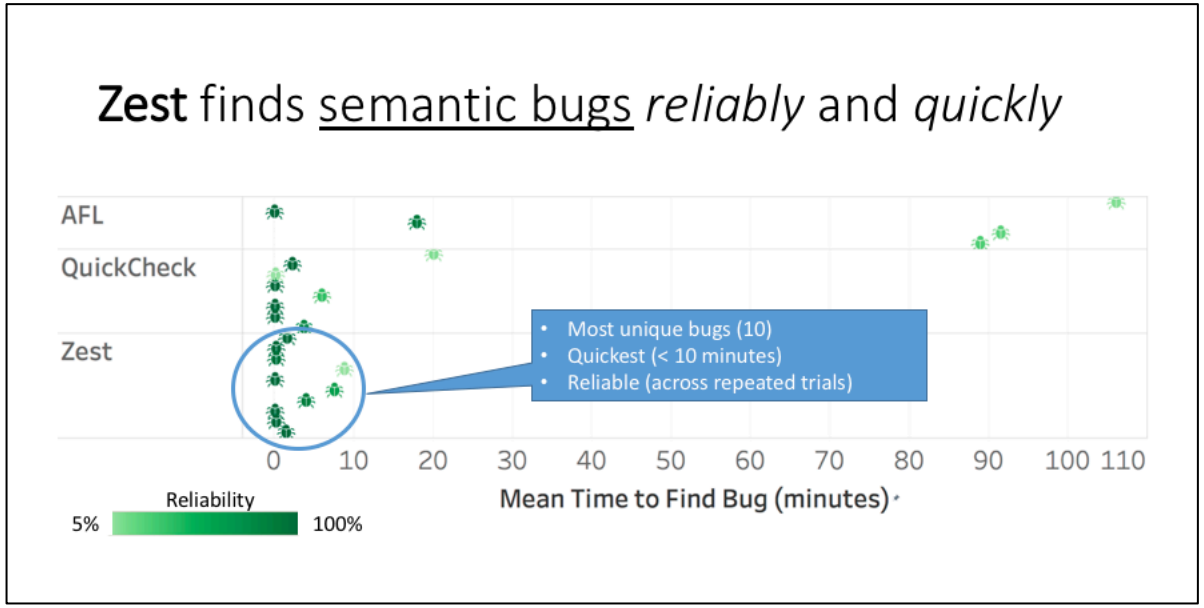


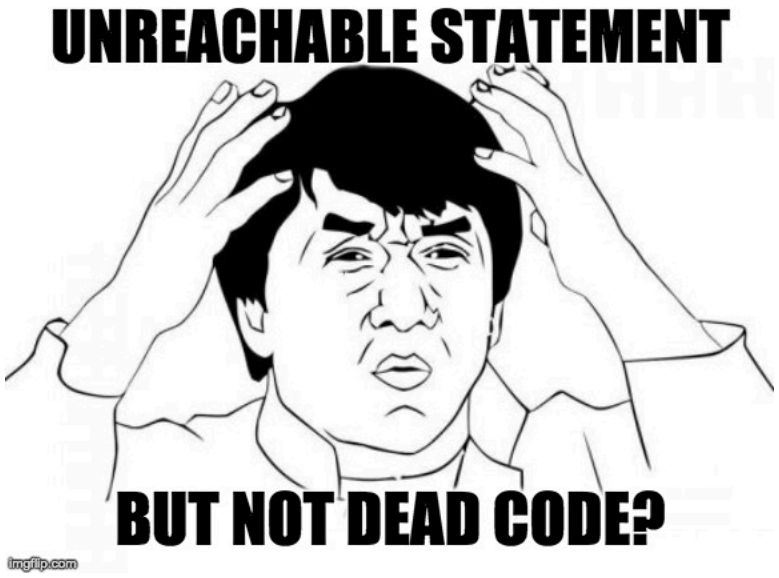
Many test programs look like this:



This Talk: Sneak Peak

Fuzzing Apache Ant	Syntax Error	Semantic Error	Semantically Valid
Baseline 1 (AFL)	99.63 %	0.37 %	0 %
Baseline 2 (QuickCheck)	0 %	99.99%	0.0000005%
Semantic Fuzzing with Zest	**0 %**	**80.12 %**	**19.88 %**





1. Coverage-guided Fuzzing
(prior work)

2. Generator-based Fuzzing
(prior work)

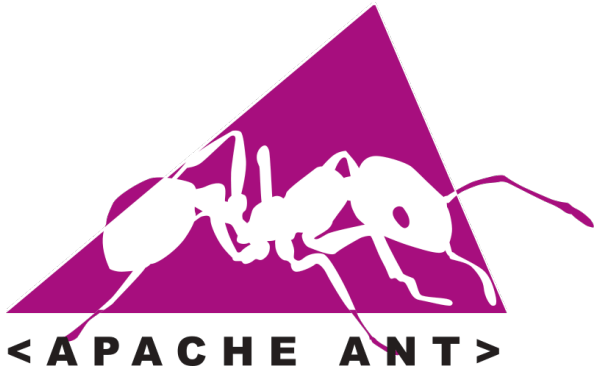
3. Semantic Fuzzing with Zest
(our work)

1. Coverage-guided Fuzzing
(prior work)

2. Generator-based Fuzzing
(prior work)

3. Semantic Fuzzing with Zest
(our work)

Case Study




```
$ ant -f build.xml
```

Mutation-based Fuzzing

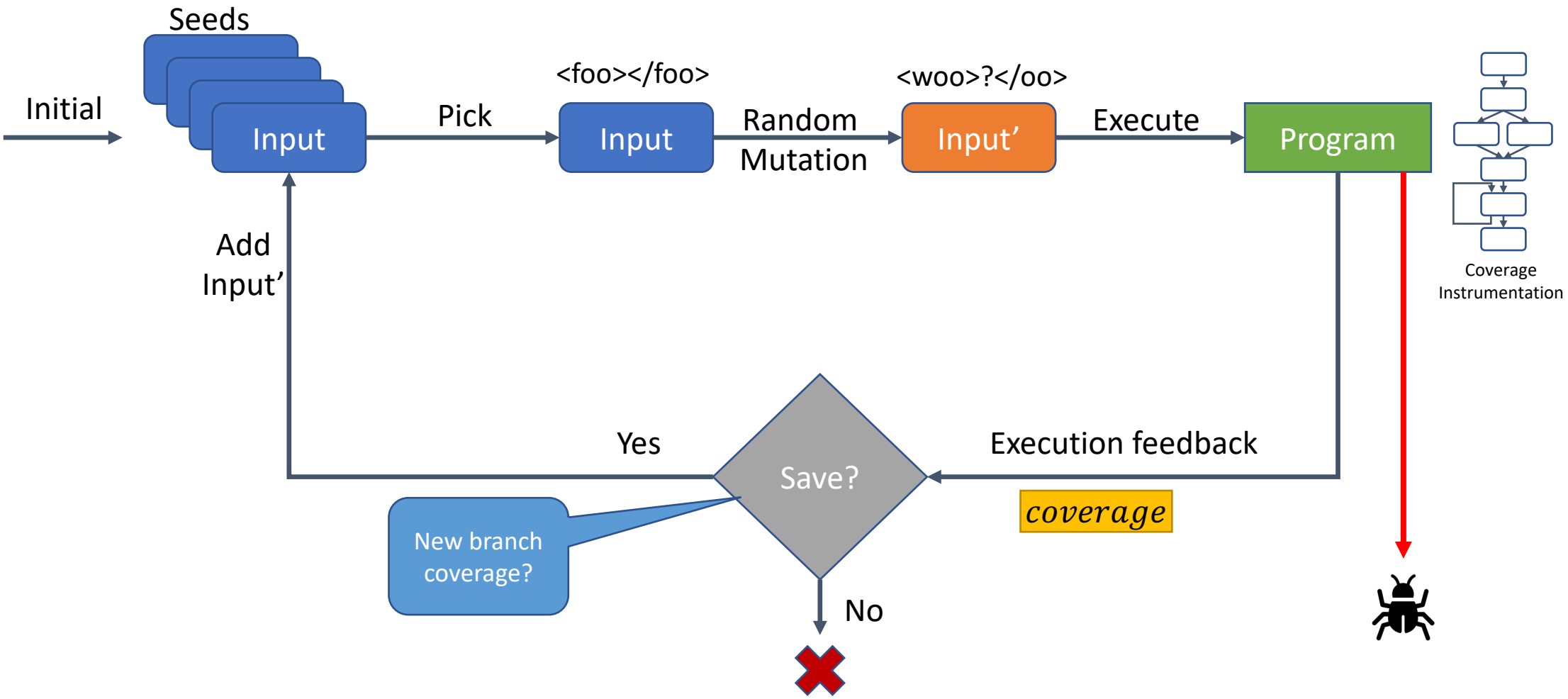
Valid Seed Input (build.xml)

```
<project default="dist">
 <target name="init">
  <mkdir dir="${build}"/>
 </target>
 …
```

New Input (Mutated from Seed)

```
<project default="dist">
 <taWget name="init">
  <madir dir="2{build}"/@
 </tar?get>
 …
```

Coverage-guided fuzzing

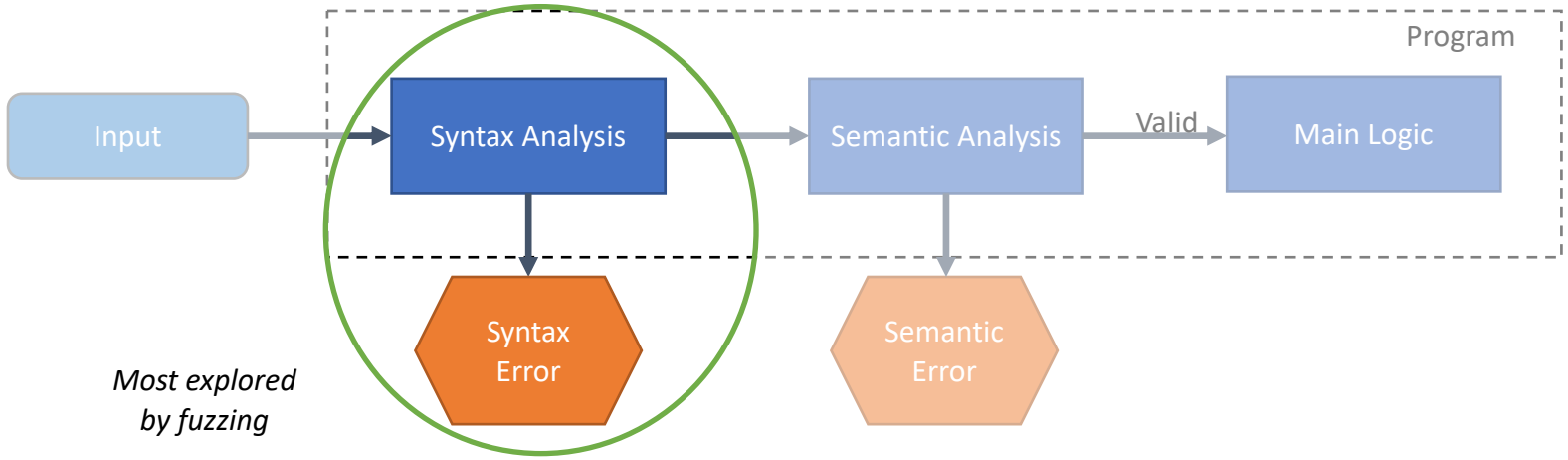


Fuzzing Apache Ant

AFL generates ~500,000 new build.xml files in 1 hour

	Syntax Error	Semantic Error	Semanically Valid
AFL (Coverage-guided fuzzing)	**99.63 %**	**0.37 %**	**0 %**

Example: `... <taWget name="init"><madir dir="2{build}"/@</tar?get> ...`



1. Coverage-guided Fuzzing
(prior work)

2. Generator-based Fuzzing
(prior work)

3. Semantic Fuzzing with Zest
(our work)

Let’s generate *syntactically valid* inputs

- **<u>QuickCheck Generator Functions</u>**
- Context-Free Grammars
- Peach Pits
- Protocol Buffers
- etc.

A Simple XML Generator

```
public XMLElement genXML(Random random)
```

A Simple XML Generator

```
public XMLElement genXML(Random random) {
```

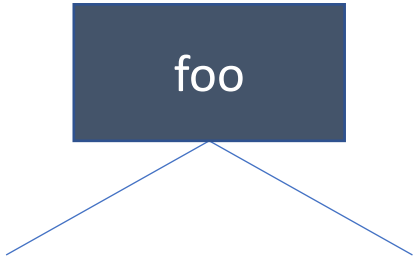
foo

```
XMLElement node = new XMLElement(random.nextString());
```

"foo"

A Simple XML Generator

```
public XMLElement genXML(Random random) {
```

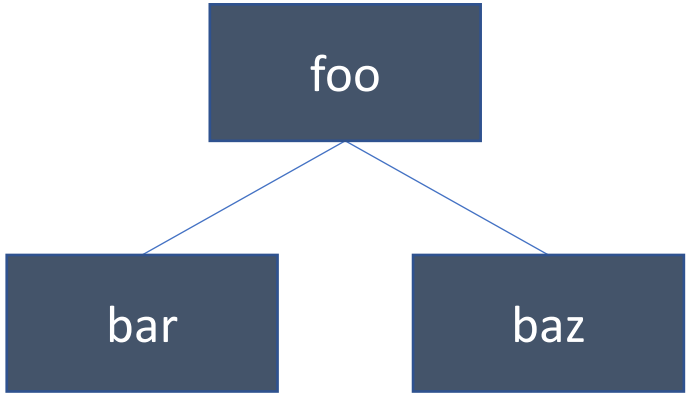



```
    XMLElement node = new XMLElement(random.nextString());
    int children = random.nextInt(0, MAX_CHILDREN);
```

2

A Simple XML Generator

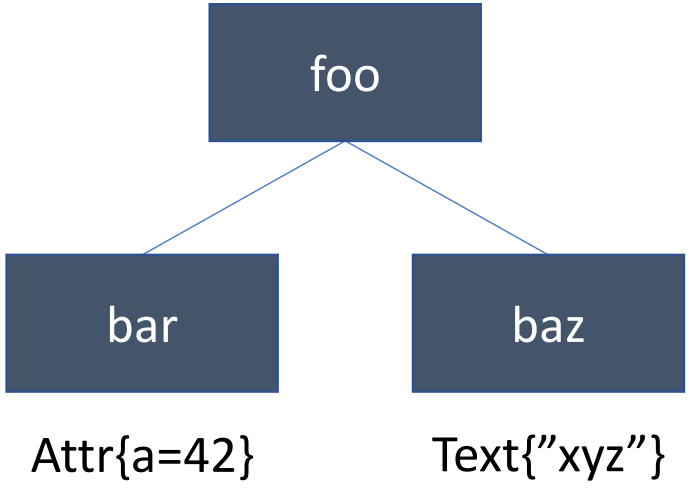
```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }
```

A Simple XML Generator

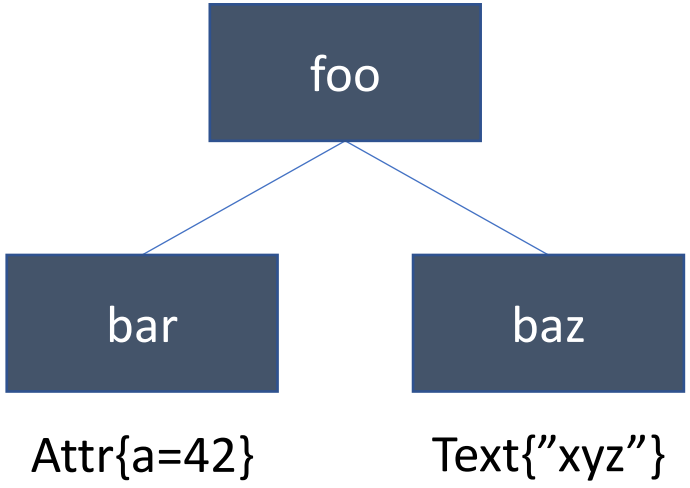
```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }

    /* ... Maybe add attributes ... */
    return node;
}
```

A Simple XML Generator

```
public XMLElement genXML(Random random) {
```



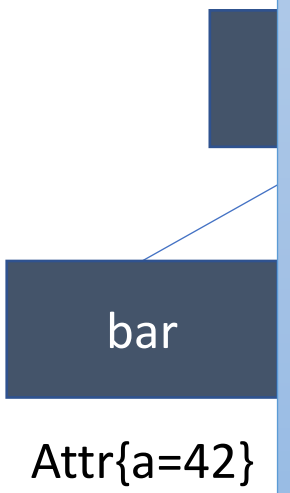

```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
```

```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

```
}
```

A Simple XML Generator

```
public XMLElement genXML(Random random) {
```



Observations:

1. Generators are easy to write

2. Every execution produces a *syntactically valid* input (not necessary *semantically* valid)

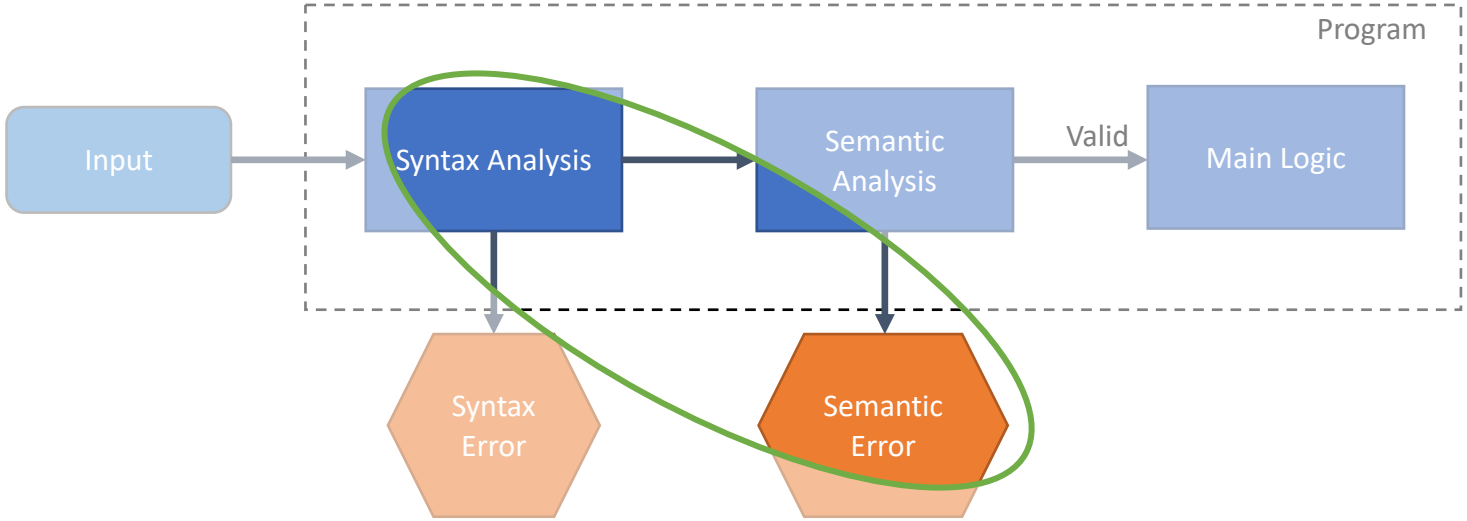
```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

```
}
```

Fuzzing Apache Ant

	Syntax Error	Semantic Error	Semantically Valid
AFL	99.63 %	0.37 %	0 %
QuickCheck (Generator-based fuzzing)	**0 %**	**99.99%**	**0.0000005%**

Example: `<sleep><delete copy="propertyhelper” /></sleep>`



1. Coverage-guided Fuzzing
(prior work)

+

2. Generator-based Fuzzing
(prior work)

???

1. Coverage-guided Fuzzing
(prior work)

+

2. Generator-based Fuzzing
(prior work)

+

=

3. Semantic Fuzzing with Zest
(our work)

Idea: Parametric Generators

```
public XMLElement genXML(Random random)
```

Idea: Parametric Generators

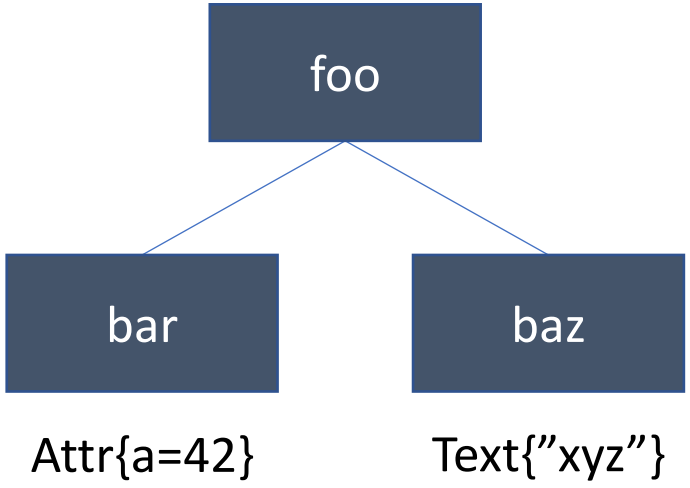
Pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ….

```
public XMLElement genXML(Random random)
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ….

```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
```

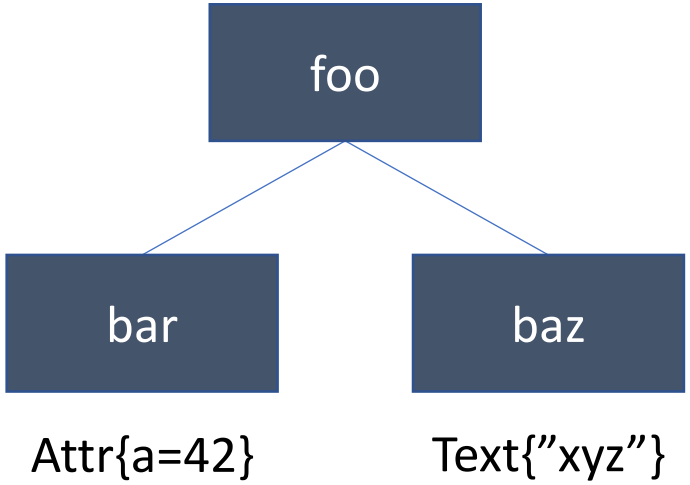
```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

```
}
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ….

```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children; i
        node.addChild(genXML(random
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
```

"foo"

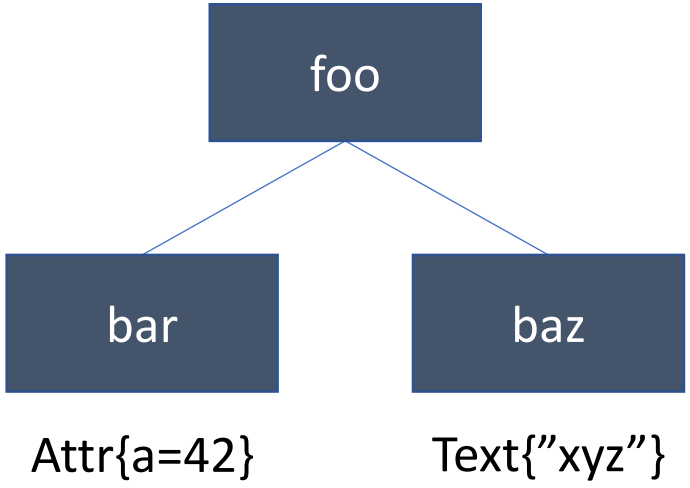
```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

```
}
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ….

```
public XMLElement genXML(Random random)
```




```
XMLElement node = new XMLElement(random.nextString());

int children = random.nextInt(0, MAX_CHILDREN);
for (int i = 0; i < children; i
    node.addChild(genXML(random
}
if (random.nextBoolean()) {
    node.addText(random.nextString());
}
```

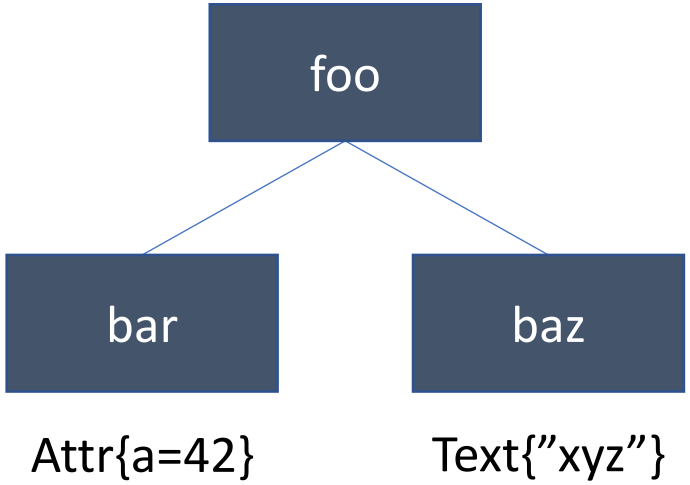
"foo"

```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0010 ....

```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children
        node.addChild(genXML(ran
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
```

"foo" => "woo"

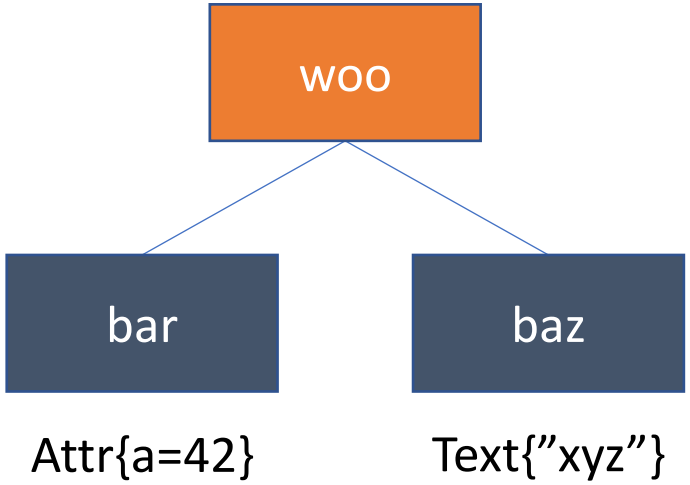
```
}
```

```
<foo><bar a="42" /><baz>xyz</baz></foo>
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0010 ....

```
public XMLElement genXML(Random random) {
```




```
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);

    for (int i = 0; i < children
        node.addChild(genXML(ran
    }

    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
```

"foo" => "woo"

```
<woo><bar a="42" /><baz>xyz</baz></woo>
```

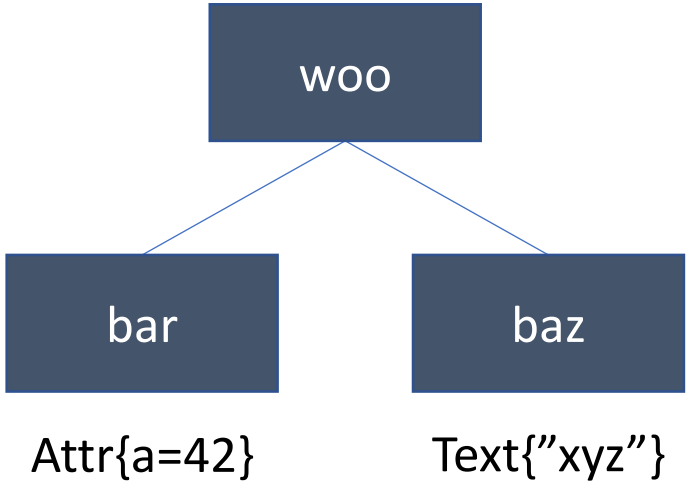
```
}
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0010 ….

```
public XMLElement genXML(Random random) {
    XMLElement node = new XMLElement(random.nextString());

    int children = random.nextInt(0, MAX_CHILDREN);
    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }
    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
}
```

2



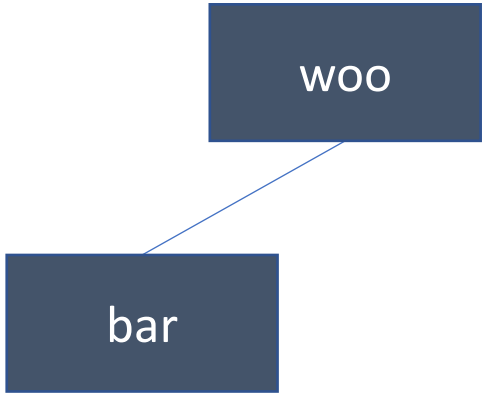

```
<woo><bar a="42" /><baz>xyz</baz></woo>
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0001 ....

```
public XMLElement genXML(Random random) {
    XMLElement node = new XMLElement(random.nextString());
    int children = random.nextInt(0, MAX_CHILDREN);
    for (int i = 0; i < children; i++) {
        node.addChild(genXML(random));
    }
    if (random.nextBoolean()) {
        node.addText(random.nextString());
    }
}
```

2 => 1



Attr{a=42}

```
<woo><bar a="42" /></woo>
```

Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0001 ….

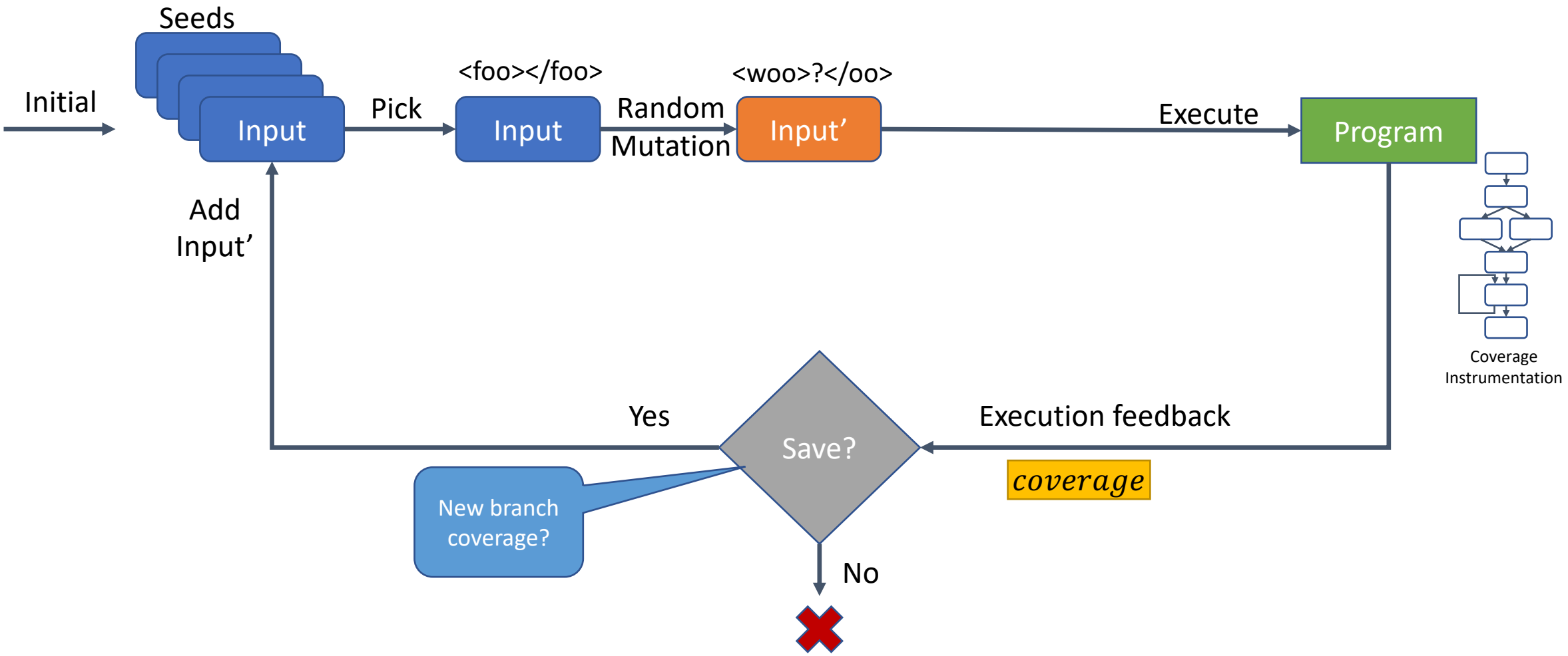
Key takeaways:

1. Mutations in “parameter” bits = *structural mutations* in input

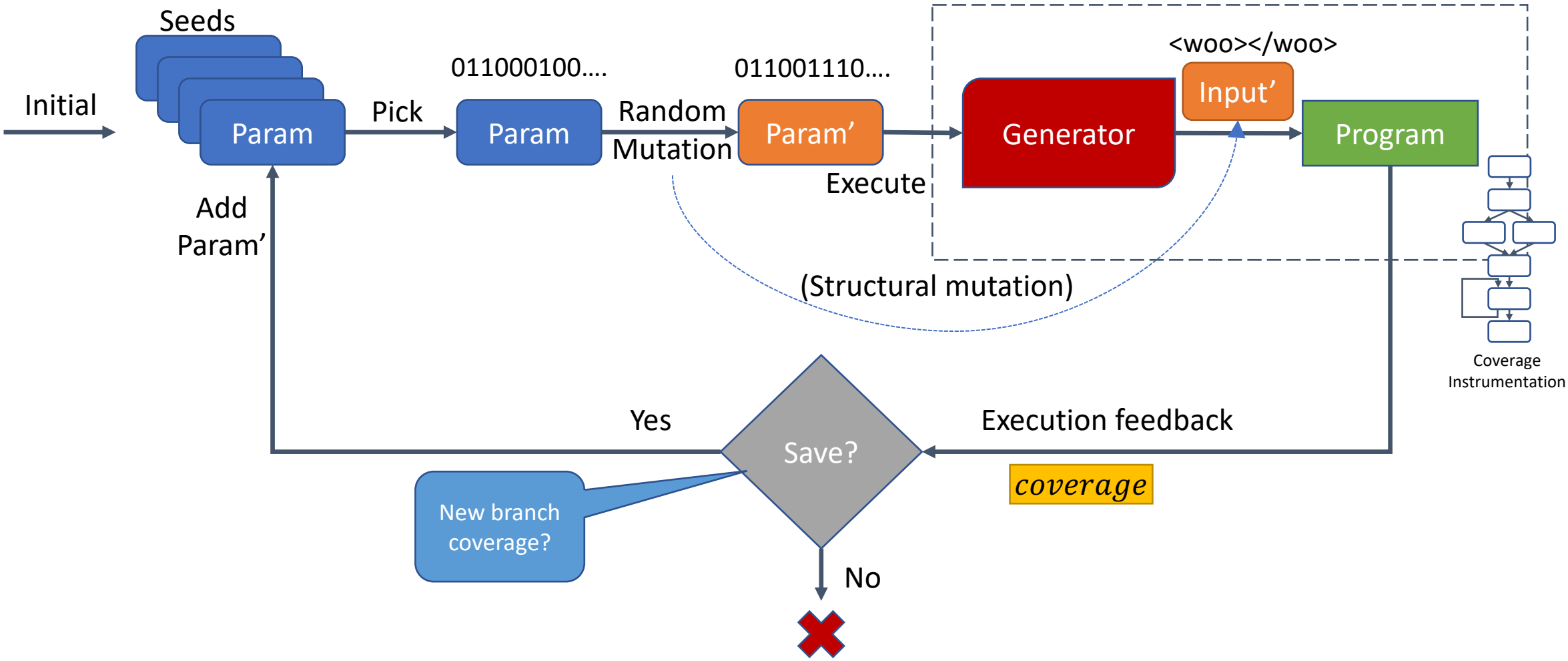
2. Every execution produces a *syntactically valid* input

```
<woo><bar a=“42” /></woo>
```

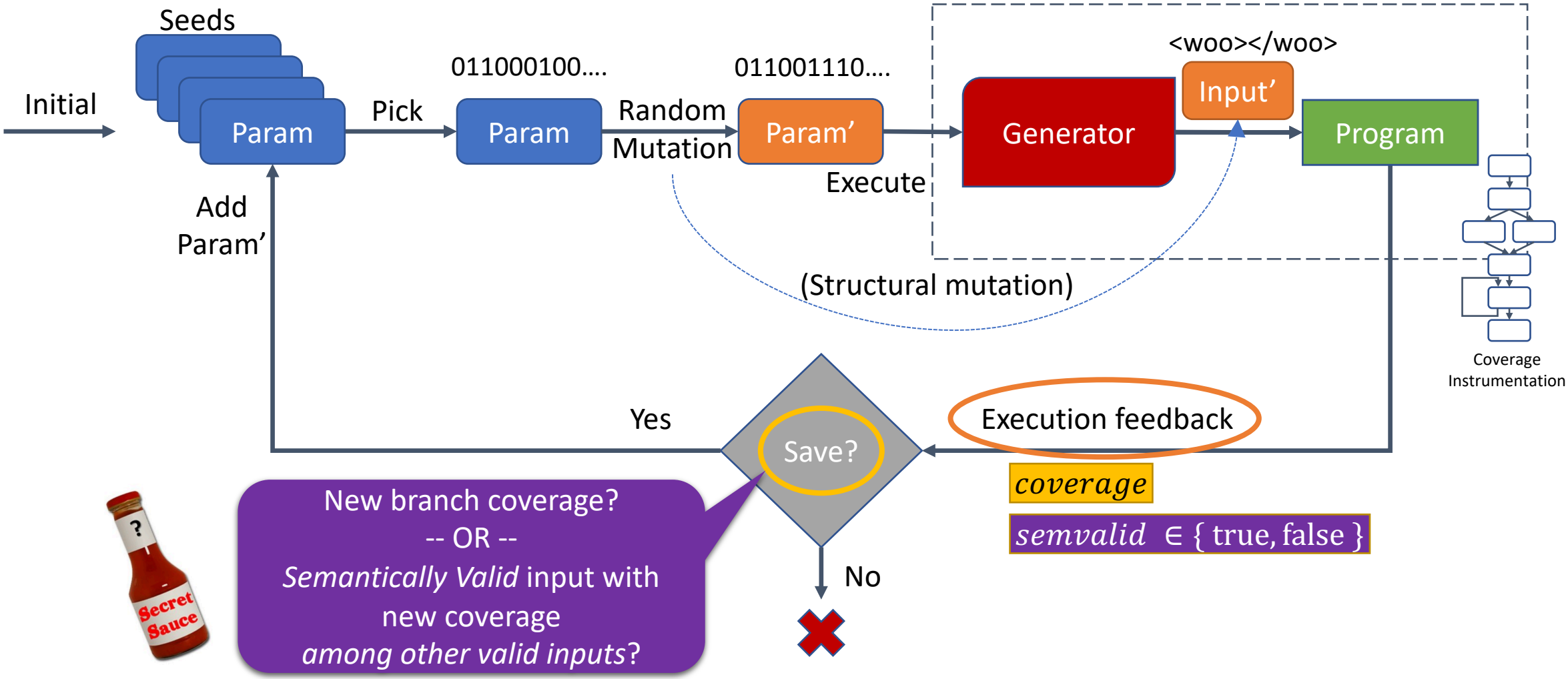
Coverage-guided fuzzing



Coverage-guided fuzzing with Parametric Generators



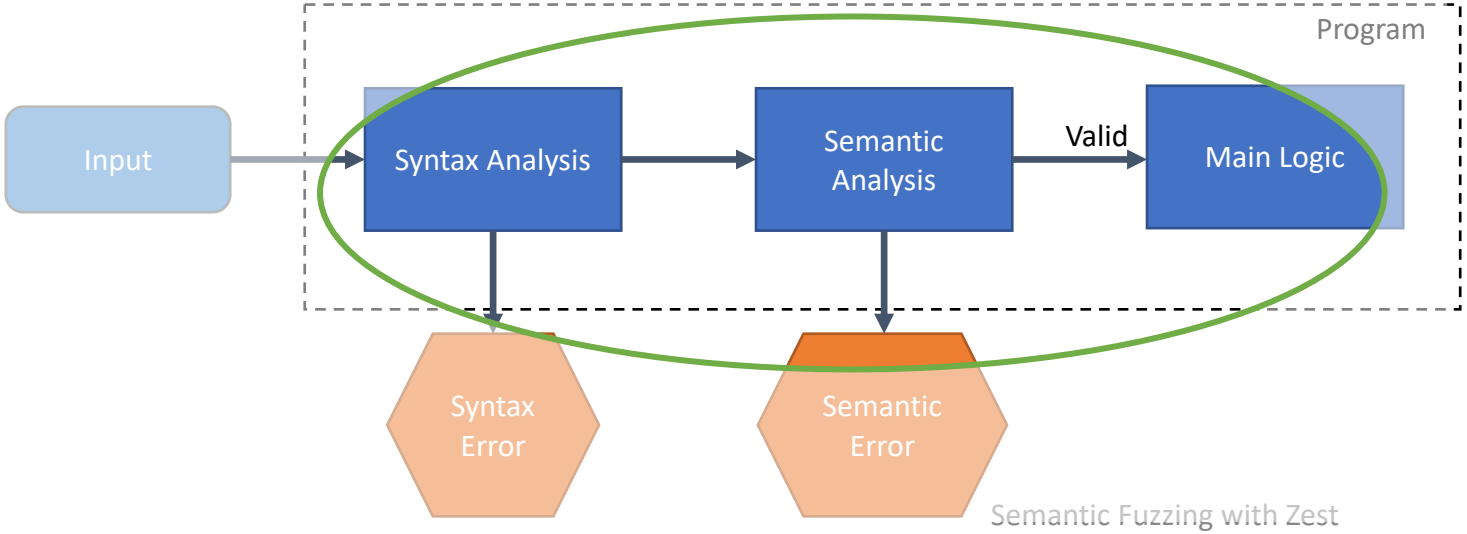
Zest: Validity Fuzzing + Parametric Generators



Fuzzing Apache Ant

	Syntax Error	Semantic Error	Semantically Valid
AFL	99.63 %	0.37 %	0 %
QuickCheck	0 %	99.99%	0.0000005%
Semantic Fuzzing with Zest	**0 %**	**80.12 %**	**19.88 %**

Example: `<project><augment></augment><target name="init">></target></project>`



Ant Bug #62655: Uncaught Exception when augmenting task

Evaluation of **Zest**

Benchmark	Generator	Semantic Validity
Apache Ant - *Process build.xml*	XML Generator (75 LOC)	Ant Build Schema
Apache Maven - *Process pom.xml*		Maven POM Schema

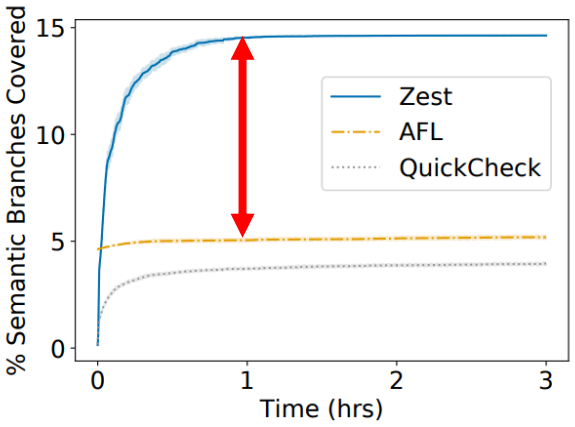
Evaluation of **Zest**

Benchmark	Generator	Semantic Validity
Apache Ant - *Process build.xml*	XML Generator (75 LOC)	Ant Build Schema
Apache Maven - *Process pom.xml*		Maven POM Schema
Google Closure Compiler - *Optimize JavaScript*	JavaScript AST Generator (300 LoC)	Valid ES6
Mozilla Rhino - *Translate JavaScript*		Can be translated to JVM bytecode

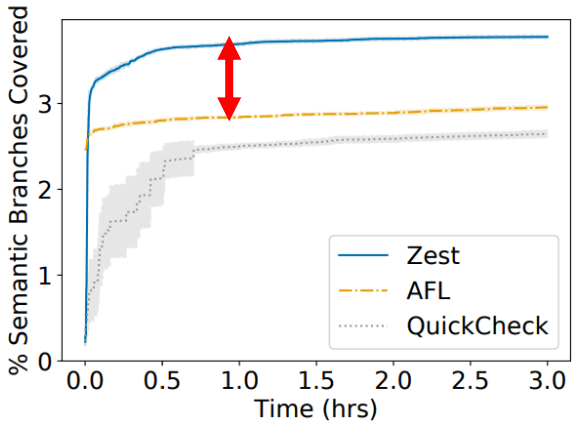
Evaluation of **Zest**

Benchmark	Generator	Semantic Validity
Apache Ant - *Process build.xml*	XML Generator (75 LOC)	Ant Build Schema
Apache Maven - *Process pom.xml*		Maven POM Schema
Google Closure Compiler - *Optimize JavaScript*	JavaScript AST Generator (300 LoC)	Valid ES6
Mozilla Rhino - *Translate JavaScript*		Can be translated to JVM bytecode
Apache BCEL - *Verify .class files*	Java Class Generator (500 LoC)	Passes bytecode verification

Zest attains *significantly* higher semantic <u>coverage</u>

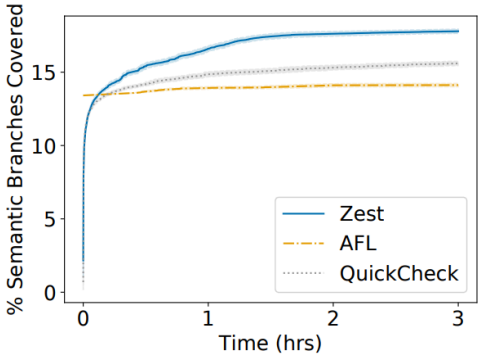


(a) maven

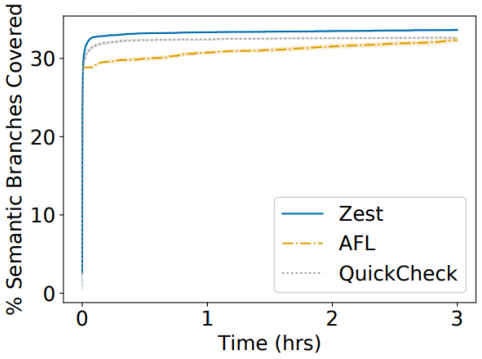


(b) ant

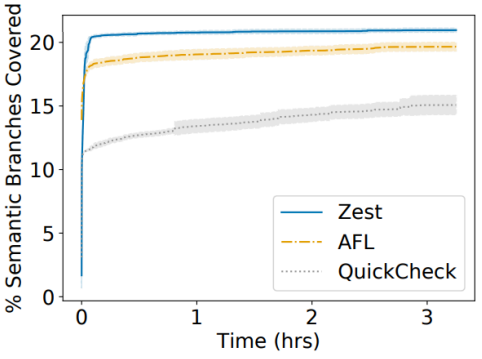
Higher is better



(c) closure

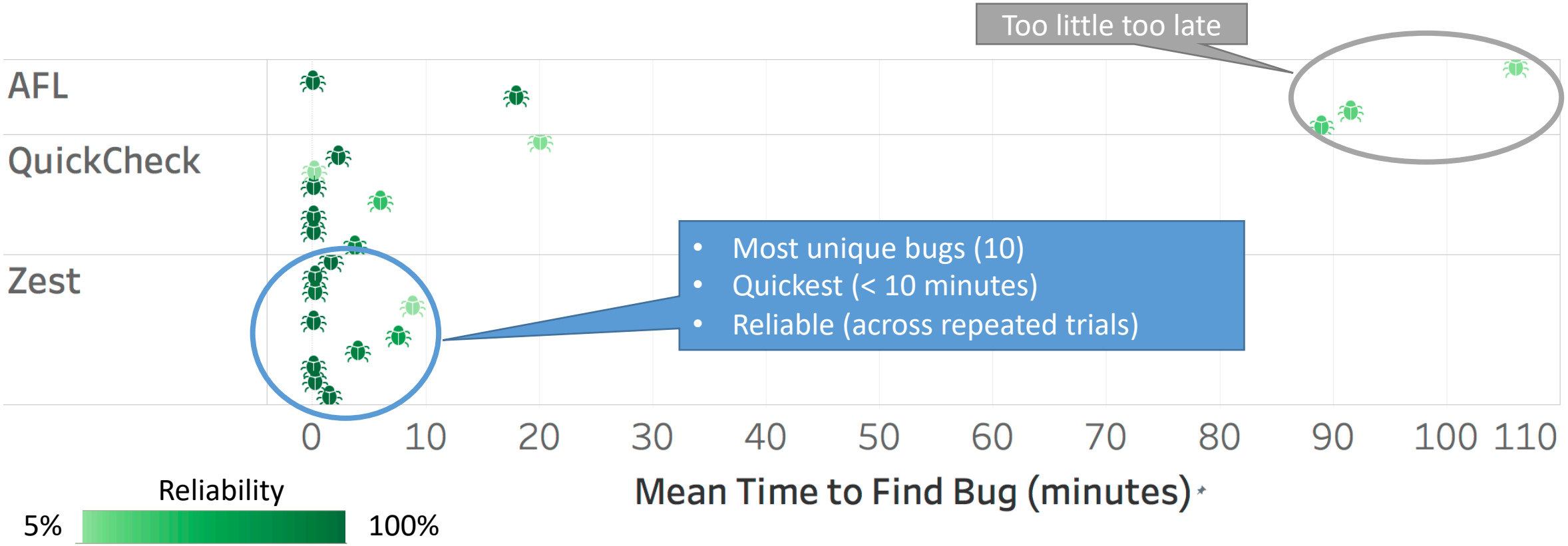


(d) rhino



(e) bcel

Zest finds semantic bugs *reliably* and *quickly*

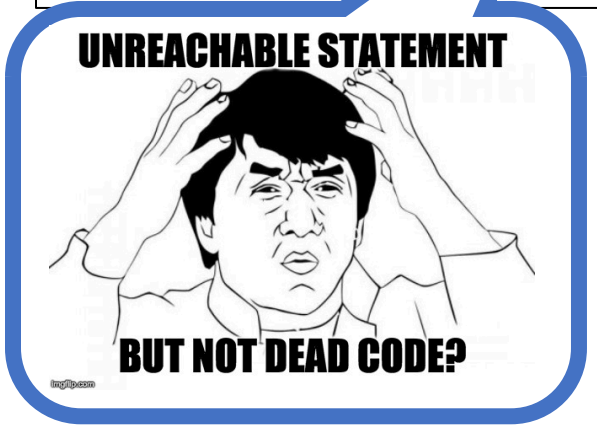


- Most unique bugs (10)
- Quickest (< 10 minutes)
- Reliable (across repeated trials)

Zest finds complex semantic bugs

```
while ((l_0)){
  while ((l_0)){
    if ((l_0))
    { break;var l_0;continue }
    { break;var l_0 }
  }
}
```

Zest-generated JavaScript input



Google Closure Compiler

IllegalStateException in VarCheck
during optimization

More semantic bugs…

- **Mozilla Rhino**: Compiler output fails bytecode verification
- **Google Closure Compiler**: Function inlining fails during decomposition
- **Apache BCEL**: Assertion violation when invoking unresolved method

Zest is open-source!

[ISSTA '19 Distinguished Artifact]

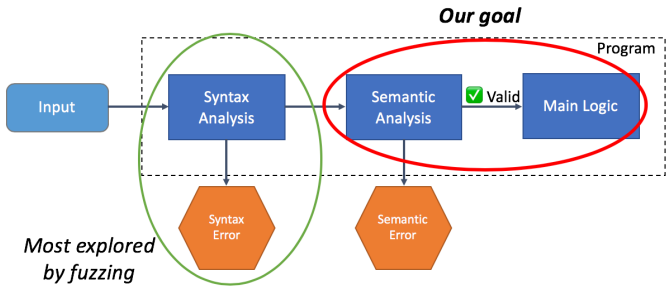
https://github.com/rohanpadhye/jqf

Integrated into **JQF**, our Java fuzzing framework [ISSTA '19 Best Tool Demo]

- Used by OSS-community + industry to find 40+ new bugs / CVEs

Summary

Many test programs look like this:

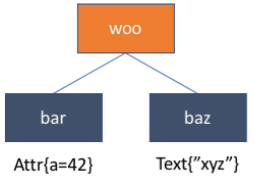


Idea: Parametric Generators

Pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0010 ....

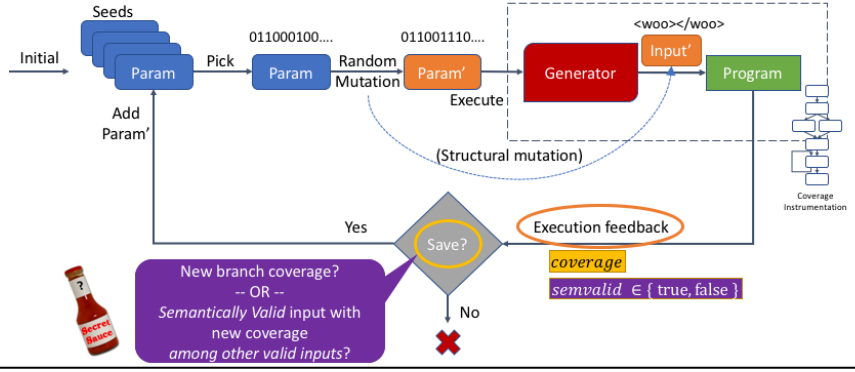
```
public XMLElement genXML(Random random) {
  XMLElement node = new XMLElement(random.nextString());
  int children = random.nextInt(0, MAX_CHILDREN);
  for (int i = 0; i < children
      node.addChild(genXML(ran
  }
  if (random.nextBoolean()) {
      node.addText(random.nextString());
  }
}
```

"foo" => "woo"




```
<woo><bar a="42" /><baz>xyz</baz></woo>
```

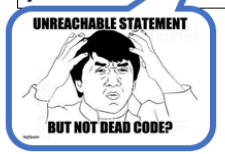
Zest: Validity Fuzzing + Parametric Generators



Zest finds complex semantic bugs

```
while ((l_0)){
  while ((l_0)){
    if ((l_0))
    { break; var l_0; continue }
    { break; var l_0 }
  }
}
```

Zest-generated JavaScript input

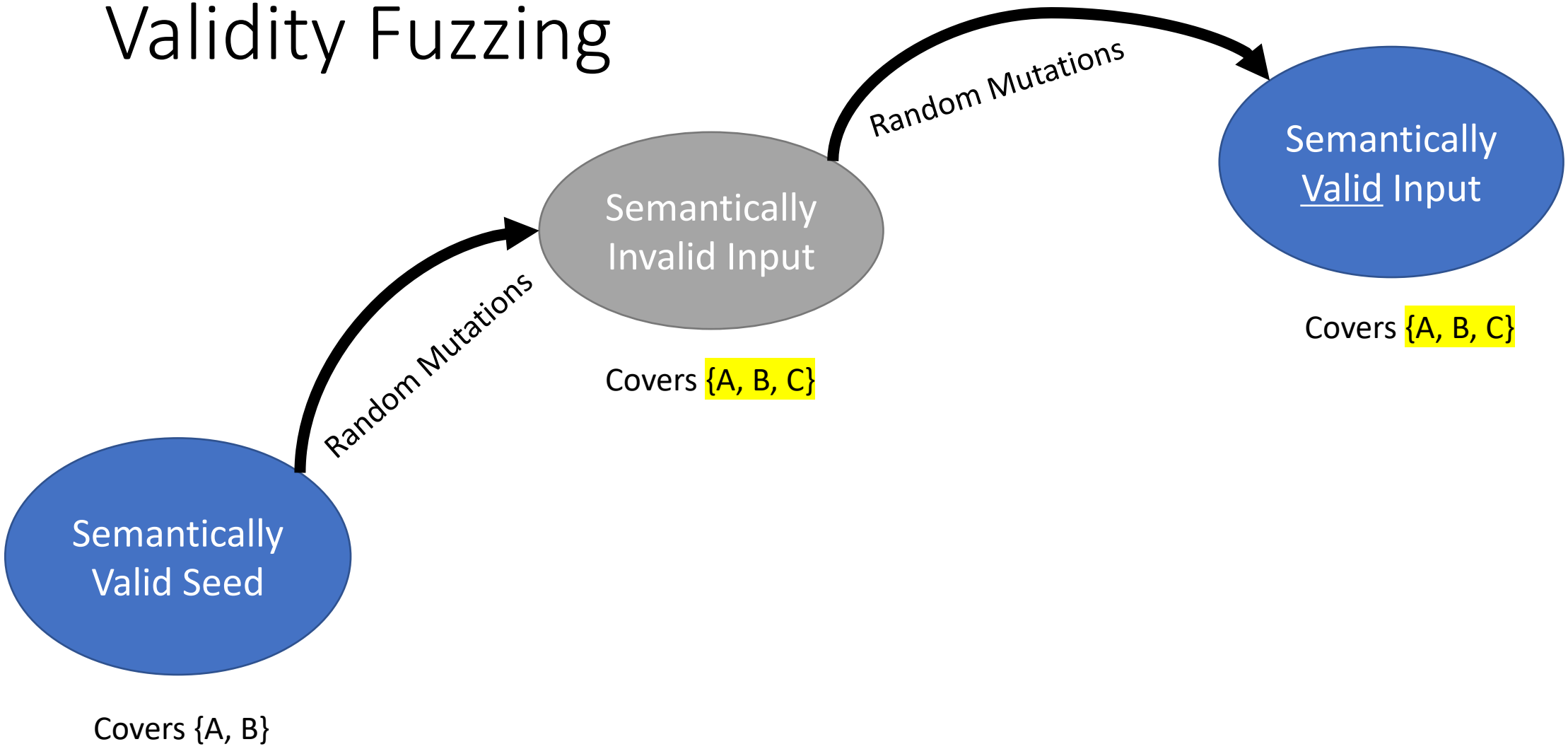


Google Closure Compiler

IllegalStateException in VarCheck during optimization

Backup slides

Validity Fuzzing



Backup slides

Generators in Practice

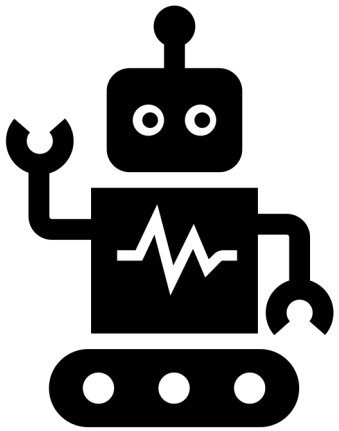
- Erdős–Rényi model: Generator for random graphs (1959)
  - https://en.wikipedia.org/wiki/Erdos-Renyi_model
- DartFuzz: Generator for Dart programs written in Dart
  - https://github.com/dart-lang/sdk/blob/master/runtime/tools/dartfuzz/dartfuzz.dart
- Csmith: Generator for C programs written in C++
  - https://github.com/csmith-project/csmith [PLDI '11]
- StringFuzz: Generator for SMT-lib formulas written in Python
  - https://github.com/dblotsky/stringfuzz [ASE '18]
- … and many more!

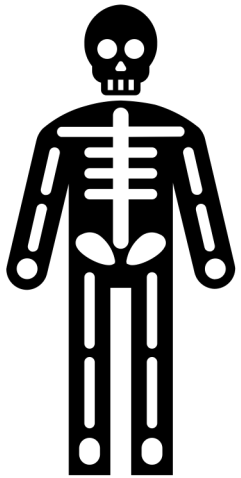
Backup slides

Benchmarks: Identification of Stages

Name	Syntax Analysis Classes	Semantic Analysis Classes
Apache Ant *- Process build.xml*	com.sun.org.apache.**xerces**	org.apache.tools.**ant**
Apache Maven *- Process pom.xml*	org.codehaus.plexus.util.**xml**	org.apache.maven.**model**
Google Closure Compiler *- Optimize JavaScript*	com.google.javascript.jscomp.**parsing**	com.google.javascript.**jscomp**.[A-Z]
Mozilla Rhino *- Translate JavaScript*	org.mozilla.javascript.**Parser**	org.mozilla.javascript.**optimizer** org.mozilla.javascript.**CodeGen**
Apache BCEL *- Verify .class files*	org.apache.bcel.**classfile**	org.apache.bcel.**verifier**

Backup slides





Structure-aware greybox fuzzing

- **Zest [ISSTA '19]**
- libFuzzer + protobuf [LLVM '18]
- Nautilus [NDSS'19]
- Superion [ICSE'19]
- Greybox fuzzing with grammars [fuzzingbook.org]
- AFLSmart
- CGPT [OOPSLA 2019]

Backup slides