Expanding the Reach of Fuzzing

Caroline Lemieux

*University of British Columbia*
*(currently postdoc @ Microsoft Research NYC)*

Talk at CSER

Nov 21st, 2021
My Work

Program Synthesis

Specification Mining

Test-Input Generation

Input
My Work

Test-Input Generation (Fuzzing)

Program Synthesis

Specification Mining

Input
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
1990s: Random Fuzzing

When we use basic operating system facilities to test each new software program, we often do a poor job of reliability. These gaps in the system are due to the frequent use of an inferior set of testing procedures. To make a system more reliable, we need to expand the reach of fuzzing.

An Empirical Study of the Reliability of Unix Utilities

The conventional wisdom of the 1990s was that fuzzing was required to achieve high reliability. We found, however, that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach. We found that the reach of fuzzing could be expanded by using a novel approach.
1990s: Random Fuzzing

1990s: Random Fuzzing

1990s: Random Fuzzing

1990s: Random Fuzzing

Random Source → $Y&Ó<83>ýø → $ bc

1990s: Random Fuzzing

1990s: Random Fuzzing

1990s: Random Fuzzing

Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz
Coverage-Guided Fuzzing
AFL, libFuzzer, honggfuzz

Initial

pick

mutate

execute

save

Interesting Feedback?

Input_n'

Execution Feedback_n

Interesting Feedback?

Input_n'

Interesting Feedback?
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz

Initial \(<a>b</a>\) → pick \(<a>b</a>\) → mutate \(<a>b</a>\) → execute $\text{xml}$ lint

save

Interesting Feedback?

Input

Interesting Feedback?
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz

Initial → pick → mutate → execute

$xmllint

tags_match(input)

Tag = “a”

Tag = “b”

Branches Covered

Interesting Feedback?

Initial Input

\(<a>b</a>\)

\(<a>b</a>\)

\(<a>a</a>b</a>\)

11/21/21

Caroline Lemieux --- Expanding the Reach of Fuzzing
Coverage-Guided Fuzzing
AFL, libFuzzer, honggfuzz

- **Initial** input:
  - `<a>b</a>`
- **Pick** input:
  - `<a>b</a>`
- **Mutate** input:
  - `<a>g</a>b</a>`
- **Execute** with `$ xmllint`:
  - `tags_match(input)`
  - `tag == “a”`
  - `tag == “b”`
  - `is_recoverable(err)`
  - `T` or `F`

**Interesting Feedback?**
- `T` or `F`
Coverage-Guided Fuzzing
AFL, libFuzzer, honggfuzz

Initial

<\textit{a}\textgreater \textit{b}</\textit{a}>

pick

\textit{<a}\textit{b}</\textit{a}>

mutate

\textit{<\textit{a}}\textit{g}\textit{b}</\textit{a}>

execute

\$\textit{xml}ll\textit{int}$

$tags\_match(input)$

$tag == \textit{a}$

$T$

$F$

$tags\_match(input)$

$\textit{is\_recoverable(err)}$

...$

$T$

$F$

$\textit{tag} == \textit{b}\$

...$

$T$

$F$

New Branch Covered?

Caroline Lemieux --- Expanding the Reach of Fuzzing
Coverage-Guided Fuzzing
AFL, libFuzzer, honggfuzz

Initial

\(<a>b</a>\) pick \(<a>b</a>\) mutate \(<a>b</a>\) execute

save

New Branch Covered?

\(<a>b</a>\)

$\text{xmlint}$

tags_match(input)

tag == “a”

tags_match(input)

tag == “b”

is_recoverable(err)

T

F

F

T

T

F

T

F

T

F

T
Coverage-Guided Fuzzing

Initial → Input → pick → Input → mutate → Input_n' → execute

save

Interesting Feedback?

Input_n'

Execution Feedback_n
Modern Fuzzing

Coverage-Guided Fuzzing

Input \rightarrow \text{pick} \rightarrow \text{Input} \rightarrow \text{mutate} \rightarrow \text{Input}_m \rightarrow \text{execute} \rightarrow \text{Interesting Feedback?} \rightarrow \text{Execution Feedback} \rightarrow \text{save}
Modern Fuzzing

Coverage-Guided Fuzzing

Generator-Based Fuzzing
Generator-Based Fuzzing
Generator-Based Fuzzing

Input Generator $\rightarrow$ Input $\rightarrow$ $\texttt{xmlint}$
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
Generator-Based Fuzzing

def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node

$ xmllint

<a>bb</a>
Generator-Based Fuzzing

```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
```

```
$ xmllint
<go>x</go>
```
Generator-Based Fuzzing

def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node

$ xmllint
Generator-Based Fuzzing

def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node

$ xmllint <bar>f</bar>
Generator-Based Fuzzing

def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
um_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node

$ xmllint <go><b></b><x>spm</x></go>$
Generator-Based Fuzzing

Input Generator

written by developer conducting testing (or reuse a suitable one)

Input
Modern Fuzzing

Coverage-Guided Fuzzing

Generator-Based Fuzzing

Coverage-Guided Fuzzing:
- Input
- pick
- Input
- mutate
- Input
- execute
- Execution Feedback
- Interesting Feedback?
- save
- Input

Generator-Based Fuzzing:
- Input Generator
- Input
- Execution Feedback
- save
- Input
Modern Fuzzing

Coverage-Guided Fuzzing
- Drawback: malformed inputs
- Exploring Core Logic
- Drawback: fixed testing goal (coverage)
- Performance Bugs

Generator-Based Fuzzing
- Drawback: user effort to make effective
- Smart Generators
- Input Generator → Input
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

**PerfFuzz**
*Lemieux*, Padhye, Sen & Song. ISSTA ’18

**FuzzFactory**
Padhye, *Lemieux*, Sen, Laurent & Vijayakumar. OOPSLA ‘19

Exploring Core Logic

Smart Generators

Future Directions
Performance Bugs
Performance Bugs
Example Program: Word Frequency (wf)

• Count frequency of words in string

input:

the quick brown the dog

output:

brown: 1
dog: 1
quick: 1
the: 2
Example Program: Word Frequency (wf)

• Count frequency of words in string

input:
the quick brown the dog

output:
brown: 1
dog: 1
quick: 1
the: 2

```python
for word in words:
    id = hash(word)
    entry = table[id]

    if entry.word == word:
        entry.count += 1
        entry = entry.next
        break

    entry = entry.next

    table[id] = new entry(word=word, count=1, next=table[id])
```
**wf** Performance Response

- Usual case:

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

```
for word in words
    id = hash(word)
    entry = table[id]
while entry != None
    if entry.word == word
        entry.count += 1
        break
    entry = entry.next
    entry = entry.next
table[id] = new entry(word=word, count=1, next=table[id])
```
**wf Performance Response**

- **Usual case:**

  ```
  the quick brown the dog
  ```

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

  ```
  for word in words
      id = hash(word)
      entry = table[id]
  ```

  ```
  while entry != None
      if entry.word == word
          entry = entry.next
          entry.count += 1
          break
  ```

  ```
  table[id] = new entry(word=word, count=1, next=table[id])
  ```
**wf Performance Response**

- Usual case:

  the quick brown the dog

---

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

```python
for word in words:
    id = hash(word)
    entry = table[id]

while entry != None:
    if entry.word == word:
        entry = entry.next
        entry.count += 1
        break

    table[id] = new entry(word=word, count=1, next=table[id])
```
• Usual case:

the quick brown the dog

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

```
for word in words
    id = hash(word)
    entry = table[id]
while entry != None
    if entry.word == word
        entry.count += 1
        break
    entry = entry.next
```

```
table[id] = new entry(word=word, count=1, next=table[id])
```
**Performance Response**

### Usual case:

- **the quick brown the dog**

### Hash collisions:

- **t ?t xt at$ #a ))t Qwaa**

---

**Code Snippet**

```python
for word in words:
    id = hash(word)
    entry = table[id]
    while entry != None:
        if entry.word == word:
            entry.count += 1
            break
    table[id] = new entry(word=word, count=1, next=table[id])
```

---

**Edge # Hits**

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>21</td>
</tr>
<tr>
<td>C</td>
<td>21</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

**Hits per Edge**

- **A**: 7
- **B**: 21
- **C**: 21
- **D**: 0
**wf Performance Response**

- **Usual case:**
  
  ```
  the quick brown the dog
  ```

- **Hash collisions:**
  
  ```
  t ?t xt at$ #a )t Qwaa
  ```

- **Small words:**
  
  ```
  the quick brown
  ```

```python
for word in words:
    id = hash(word)
    entry = table[id]
    while entry != None:
        if entry.word == word:
            entry.count += 1
            break
    else:
        entry = new entry(word=word, count=1, next=table[id])
```
**Performance Response**

- **Usual case:**
  
  ```
  the quick brown the dog
  ```

- **Hash collisions:**
  
  ```
  t ?t xt at$ #a ))t Qwaa
  ```

- **Small words:**
  
  ```
  the quick brown
  ```

```python
for word in words:
    id = hash(word)
    entry = table[id]
    if entry.word == word:
        entry.count += 1
        break
    entry = entry.next
    table[id] = new entry(word=word, count=1, next=table[id])
```
**Performance Response**

- **Usual case:**
  - `the quick brown the dog`

- **Hash collisions:**
  - `t ?t xt at$ #a ))t Qwaa`

- **Small words:**
  - `t h e q u i c k b r o w`

---

```python
for word in words:
    entry = table[hash(word)]
    if entry.word == word:
        entry.count += 1
        break
    entry = entry.next
    entry.count = 1
    break
```

- **Pathological performance behavior** characterized by a few CFG edges.
- **Idea:** maximize CFG edge hit count independently.

---

Caroline Lemieux --- Expanding the Reach of Fuzzing
Coverage-Guided Fuzzing

Initial

Input

pick

Input

mutate

Input\_n'

execute

Interesting Feedback?

Input\_n'

save

Feedback\_n

Execution Feedback\_n
PerfFuzz

Initial → Input → pick → Input → mutate → Input_′ → execute → save → Interesting Feedback? → Input_′ → ... → Edge # Hits → ... → Interesting Feedback? → Input_′ → ... → Edge # Hits → ...
PerfFuzz

Maximizes # hits for some edge?
PerfFuzz

pick input maximizing # hits for some edge

Maximizes # hits for some edge?

Input

Initial

save

mutate

execute

Input

Input

Input

Input

... 

Edge | # Hits
---|---
... | ...
... | ...
... | ...
... | ...
... | ...

Input

Input

Input

Input

... 

Edge | # Hits
---|---
... | ...
... | ...
... | ...
... | ...
... | ...

11/21/21

Caroline Lemieux --- Expanding the Reach of Fuzzing
wf results: PerfFuzz Finds True Worst Cases
wf results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

```
trtstfoÖertstfortxxtstfortxx
```
wf results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

```
t r t t s f o Öe r t s f o r t x x t s f o r t x x
```

PerfFuzz worst case:

```
t <81>v ^?@t <80>!^?@t <80>!t t^Rn t t t t t t t t t t t t
wf results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

PerfFuzz worst case:

```
t <81>v ^?@t <80>!^?@t <80>!t t^Rn t t t t t t t t t
```
PerfFuzz

Initial → Input

pick input maximizing # hits for some edge

Input → Input

mutate

Inputₙ → execute

Maximizes # hits for some edge?

save

Inputₙ → ...

Edge | # Hits
--- | ---
... | ...
... | ...
... | ...
... | ...
... | ...

11/21/21

Caroline Lemieux --- Expanding the Reach of Fuzzing
Observation: Algorithm is More General

pick input maximizing # hits for some edge

Maximizes # hits for some edge?

Input → Input → Input' → execute

save

Initial

<table>
<thead>
<tr>
<th>Edge</th>
<th># Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Caroline Lemieux --- Expanding the Reach of Fuzzing
Observation: Algorithm is More General

pick input maximizing value for some key

Initial -> Input -> Input -> Input_1' -> execute

save

Maximizes value for some key?

Key | Value
--- | ---
... | ...
... | ...
... | ...
... | ...
FuzzFactory

pick input maximizing value for some key

Initial

Input

Input

Input

Input

Input

Input

execute

mutate

Maximizes value for some key?

save

... ... ...

... ... ...

... ... ...

... ... ...
With Fuzzfactory, we could build
- SlowFuzz
- PerfFuzz
- Validity Fuzzing
- Maximizing Memory Usages
- “Hard Comparison” Fuzzer
- Fuzzer targeting diffs
Background on Fuzzing

**PerfFuzz**
Lemieux, Padhye, Sen & Song. ISSTA ’18

**FuzzFactory**
Padhye, Lemieux, Sen, Laurent & Vijayakumar. OOPSLA ’19

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

**PerfFuzz**
Lemieux, Padhye, Sen & Song. ISSTA '18

**FuzzFactory**
Padhye, Lemieux, Sen, Laurent & Vijayakumar. OOPSLA '19

Generalize “feedback” with map-based abstraction
→ new applications for CGF algorithm
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Performance Bugs

Background on Fuzzing

FairFuzz
Lemieux & Sen. ASE ’18

Zest
Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ’19

Smart Generators

Future Directions
Where Are the Fuzzer-Found Bugs?

Input Validation

Core Logic
Where Are the Fuzzer-Found Bugs?
Problem: Random Mutations Ruin Structure
Problem: Random Mutations Ruin Structure
Problem: Random Mutations Ruin Structure

Initial → Input → pick → Input → mutate → Input_n' → execute

<foo>ez</foo> → mutate → <foo>>>foo

Interesting Feedback? → Input_n' → save

Execution Feedback_n
How to Retain Important Structure?
FairFuzz: Filter Mutations Likely to Ruin Structure

![Diagram of FairFuzz process]

- **Initial**
  - Seeds

- **Input**
  - Pick input hitting rare branch
  - Compute + mutate with branch mask
  - Execute

- **Input’**
  - Execution Feedback
  - Interesting Feedback?

- **Input’**
  - Save
Can we get higher-level mutations?
with more information about input structure?
Generators as Input Structure Specification
How to Get Mutations?

```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
```
Generator: Source of Randomness $\rightarrow$ Input

```python
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
```

Generator: Source of Randomness → Input

```xml
<foo>
<bar/>
</foo>
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node

<foo>
  <bar/>
  <baz>xyz</baz>
</foo>
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
um_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
Bit Mutations $\rightarrow$ Structured Input Mutations

pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ...

def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node

---

Caroline Lemieux --- Expanding the Reach of Fuzzing
Bit Mutations $\rightarrow$ Structured Input Mutations

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

def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
Zest: Integrate Generator + CGF

Initial seeds → Input → pick → Input → mutate → Input_{n}' → execute → Execution Feedback_{n} → Interesting Feedback? → Input_{n}' → save → Initial seeds

Input

mutate

execute

Feedback

Generator

Params

Interesting Feedback?
Zest: Integrate Generator + CGF

Initial → seeds

Params → pick → Params → mutate → Params_n' → Generator

Generator → execute → input

save

Interesting Feedback?

Params_n' → Execution Feedback_n

n

n
Zest: Integrate Generator + CGF

Initial seeds → pick Params → mutate Params → Generator → execute input

Higher-level mutations via generator

Params → save → Params

Interesting Feedback?

Execution Feedback

n

n
Zest finds complex semantic bugs

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

Unreachable statement... but not dead code!

IllegalStateException in VarCheck during optimization
Performance Bugs

Background on Fuzzing

FairFuzz
Lemieux & Sen. ASE ’18

Zest
Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ’19

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

**FairFuzz**
Lemieux & Sen. ASE ’18

**Zest**
Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ’19

Structure-aware mutations
→ New depth of program exploration
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

RLCheck

AutoPandas
Bavishi, Lemieux, Sen & Stoica. OOPSLA ’19

Future Directions
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
RLCheck: Directly Control the Choices
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node

What value to return to maximize the chance of generating a valid input?
Directly Control the *Choices*

```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN):
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
return node
```

What value to return to maximize the chance of generating a valid input?

Depends on *context*
Different Context ➔ Different "Good" Choices

```python
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```
Different Context → Different “Good” Choices

```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean( ):
        node.addText(random.nextString( )
    return node
Step 1: Add Context to Generator

```python
def genXML(random):
tag = random.choice(tags)
node = XMLElement(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
for i in range(0, num_child):
    node.addChild(genXML(random))
if random.nextBoolean():
    node.addText(random.nextString())
return node
```
Step 1: Add Context to Generator

def genXML(random, context):
    tag = random.choice(tags)
    node = Element(tag)

    num_child = random.nextInt(0, MAX_CHILDREN)

    for i in range(0, num_child):
        node.addChild(genXML(random, context))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
Step 1: Add Context to Generator

```python
def genXML(random, context):
tag = random.choice(tags)
node = XMLElement(tag)
context.append(tag)
num_child = random.nextInt(0, MAX_CHILDREN)
context.append(num_child)
for i in range(0, num_child):
    node.addChild(genXML(random, context))
if random.nextBoolean():
    context.append("text")
    node.addText(random.nextString())
return node
```
Step 2: Make Choices Based on Context

def genXML(random, context):
tag = random.choice(tags)
node = XMLElement(tag)
context.append(tag)
um_child = random.nextInt(0, MAX_CHILDREN)
context.append(num_child)
for i in range(0, num_child):
    node.addChild(genXML(random, context))
if random.nextBoolean():
    context.append("text")
    node.addText(random.nextString( ))
return node
Step 2: Make Choices Based on Context

```python
def genXML(random, context):
    tag = random.choice(tags, context)
    node = XMLElement(tag)
    context.append(tag)
    num_child = random.nextInt(0, MAX_CHILDREN, context)
    context.append(num_child)
    for i in range(0, num_child):
        node.addChild(genXML(random, context))
    if random.nextBoolean(context):
        context.append("text")
        node.addText(random.nextString(context))
    return node
```
Our Problem Setting

context

[“project”, “2”, “dependencies”, ...]

choice space

tag = random.choice(tags, context)
Our Problem Setting

State

Action
Sounds Like Reinforcement Learning
def genXML(random, context):
    tag = random.choice(tags, context)
    node = XMLElement(tag)
    context.append(tag)
    num_child = random.nextInt(0, MAX_CHILDREN, context)
    context.append(num_child)
    for i in range(0, num_child):
        node.addChild(genXML(random, context))
    if random.nextBoolean(context):
        context.append("text")
        node.addText(random.nextString(context))
    return node
def genXML(guide, context):
tag = guide.choice(tags, context)
node = XMLElement(tag)
context.append(tag)
num_child = guide.nextInt(0, MAX_CHILDREN, context)
context.append(num_child)
for i in range(0, num_child):
    node.addChild(genXML(random, context))
if guide.nextBoolean(context):
    context.append("text")
    node.addText(guide.nextString(context))
return node
RLCheck: Many More Unique Valid Inputs

Closure Compiler (JS)
RLCheck: Many More Unique Valid Inputs

- Ant (XML)
- Maven (XML)
- Rhino Compiler (JS)
- Closure Compiler (JS)
Background on Fuzzing

Performance Bugs

Exploring Core Logic

**RLCheck**

**AutoPandas**
Bavishi, Lemieux, Sen & Stoica. OOPSLA ’19

Future Directions
Separate distribution from user-facing generator → faster fuzzing, new synthesis paradigm

**RLCheck**  

**AutoPandas**  
Bavishi, **Lemieux**, Sen & Stoica. OOPSLA ’19

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions
So Far: Innovations in Test-Input Generation
Problems Around Test-Input Generation

How do I write a good test driver?
Problems Around Test-Input Generation

How do I specify the structure of my inputs?

How do I write a good test driver?
Problems Around Test-Input Generation

How do I specify the structure of my inputs?

How do I write a good test driver?

Do I really care about this bug?
Test-Input Generation
Test-Input Generation
Automating Fuzzing Infrastructure
Automating Fuzzing Infrastructure

- Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching
  ...

Test-Input Generation
Automating Fuzzing Infrastructure

Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching

FUDGE
Babic, Bucur, Chen, Ivancic, King, Kusano, Lemieux, Szekeres, Wang
ESEC/FSE’19 (Industry Track)

Test-Input Generation
Automating Fuzzing Infrastructure

- Test-Input Generation
- Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching

FUDGE
Babic, Bucur, Chen, Ivancic, King, Kusano, Lemieux, Szekeres, Wang
ESEC/FSE’19 (Industry Track)

200 drivers integrated into open-source projects ➔ 150 security-improving fixes
Automating Fuzzing Infrastructure

- Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching

...
Automating Fuzzing Infrastructure

- Test Input Generation
- Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching

5x higher recall than SOTA 😊
1.27x slowdown 😞
Generalize “feedback” with map-based abstraction → new bug domains

Structure-aware mutations → New depth of program exploration

Separate distribution from user-facing generator → faster fuzzing, new synthesis paradigm