Hybrid Differential Software Testing

Disputation

Yannic Noller

Humboldt-Universität zu Berlin
yanic.noller@acm.org
1. Problem + Motivation: Differential Testing
2. Contribution Summary
3. Foundations: Fuzzing + Symbolic execution
4. Related Work: Differential Testing
5. Solutions:
   - Differential Fuzzing
   - Differential Dynamic Symbolic Execution
   - HyDiff (Hybrid Differential Software Testing)
6. Validation
7. Conclusion
Software Engineering

"systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"

[IEEE2017]

Software Quality Assurance

Software Testing
Mary Jean Harrold’s roadmap, in particular, began with the following sentence: “A report by the Workshop on Strategic Directions in Software Quality posits that software quality will become the dominant success criterion in the software industry [143].” Few would argue that this prediction was inaccurate. Testing remains one of the most widely practiced approaches for assessing (and ultimately improving) the quality of software, and it remains one of the most extensively researched software engineering topics. In this paper, as requested by the FOSE chairs, we provide an accounting of the research in software testing over the past 14 years, focusing on the areas in which the greatest progress has been made and the highest impact has been achieved. We also comment on significant challenges and opportunities for future researchers in these areas.

While we, the authors of this paper, have a certain amount of experience in various areas of testing research, and follow much of the testing research to the best of our capabilities, we wished to go beyond our personal views, opinions, and knowledge of the area in preparing this Travelogue. We therefore began by reaching out to many of our colleagues, in an attempt to obtain some larger consensus as to the work that the testing research community views as having been the most important and promising. Specifically, we identified over 50 colleagues who are currently active in testing research, and sent them email asking two questions:

1. What do you think are the most significant contributions to testing since 2000, whether from you or from other researchers?
2. What do you think are the biggest open challenges and opportunities for future research in this area?

Primarily due to lack of forethought on our part, we gave our typically very busy colleagues a fairly short deadline for sending responses. We were heartened, therefore, when about 30 of them were able to get back to us with comments and, in many cases, extensive input. To provide a quick overview of the most common topics mentioned in the responses we received, Figure 1 contains a word cloud that we generated using all such responses and filtering out obvious keywords, such as “test” and “testing”.

We used our colleagues’ responses to compile lists of contributions, challenges, and opportunities, and we prioritized these based on the frequency with which they appeared in such responses. We classified most of the identified contributions as “research contributions”, while classifying two as “practical contributions”. These latter are contributions that, in our view, were driven more by industrial effort than by research activities, and yet have had a dramatic effect on both practice and research. As might be expected, in most of these areas of contributions there remain challenges and opportunities for future research, many of which were mentioned by our colleagues, and that we note in the remainder of this paper. We also selected several areas that were seen not as areas in which substantial contribution had yet been made, but rather, as areas that pose new (or continuing) challenges and opportunities for researchers.

In presenting our thoughts on contributions and opportunities, we attempted to cite relevant papers in the areas discussed. We make no pretense, however, of having cited all such papers, a task better left to survey papers, several of which we cite to guide readers further. Similarly, while our colleagues’ input helped guide our choice of topics to cover, we do not claim that our paper represents all the relevant and noteworthy research performed in the area of software testing in the time period considered. Such a task would require considerably more space and time than we have available. Nevertheless, we truly hope that the approach we followed helped this paper better reflect the views of researchers in the software testing community and provide an unbiased view on this challenging and exciting research field.

We structure the rest of this paper as follows. Section 2 describes research contributions and additional opportunities in these areas. Section 3 describes practical contributions and additional opportunities in these areas. Section 4 describes additional areas in which opportunities and challenges exist. Finally, Section 5 concludes.

2. RESEARCH CONTRIBUTIONS

We classified the nine research contributions that we identified from our colleagues’ responses into four categories: (1) automated test input generation, (2) testing strategies, (3) regression testing, and (4) support for empirical studies. The following subsections present contributions in each of these categories.

[Orso2014]
Differential Software Testing

Problem

Validation

Background

Solutions

Contribution

Summary

Differential Software Testing

input

program P

behavior1

input

program P'

behavior2

input1

program P

behavior1

input2

program P

behavior2

yannic.noller@acm.org

Hybrid Differential Software Testing

5
Differential Software Testing

between two program versions for the same input → software maintenance

Regression Analysis
Regression Analysis

Are there unintended behavioral differences between the two versions?

Fixed assertion error for x=-1 (returns 0).

int foo (int x) {
    int y;
    if (x < 0) {
        y = -x;
    } else {
        y = x * x;
    }
    y = y + 1;
    if (y > 1) {
        return 0;
    } else {
        if (y == 1)
            assert(false);
    }
    return 1;
}

introduced new assertion error for x=0 (previously returned 1) → Regression Bug
Differential Software Testing

1. Program P with input X yields behavior 1.
2. Program P' with input y yields behavior 2.

Equivalence is determined between behavior 1 and behavior 2.
Differential Software Testing

for the **same program** with **two different inputs** ➔ **security, reliability**

- Worst-Case Complexity Analysis
- Side-Channel Analysis
- Robustness Analysis of Neural Networks

![Diagram showing program P with inputs x and y leading to behaviors behavior₁ and behavior₂](image)
Worst-Case Complexity Analysis

**Goal:** discover vulnerabilities related to algorithmic complexity

```java
public void sort (int[] a) {
    int N = a.length;
    for (int i = 1; i < N; i++) {
        int j = i - 1;
        int x = a[i];
        while ((j >= 0) && (a[j] > x)) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = x;
    }
}
```

- Insertion Sort

  → find worst-case input: automated + fast + concrete

  - worst-case complexity: $O(n^2)$
  - e.g. $a=[8, 7, 6]$ ($n=3$)
Side-Channel Analysis

- leakage of secret information
- software side-channels
- observables:
  - execution time,
  - memory consumption,
  - response size,
  - ...

yannic.noller@acm.org

Hybrid Differential Software Testing
Example: Side-Channel Vulnerability

```java
boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
    if (pub.length != sec.length) {
        return false;
    }
    for (int i = 0; i < pub.length; i++) {
        if (pub[i] != sec[i]) {
            return false;
        }
    }
    return true;
}
```

Unsafe Password Checking

yannic.noller@acm.org
Robustness Analysis of Neural Networks

**Goal:** identify adversarial inputs or check how amenable the network is for adversarial inputs

adversarial input

- **hardly** perceptible perturbations
- **large** impact on network’s output

---

[Pei2017]
(My) Research Problem

⇒ identify behavioral differences

1. Program P
   → Behavior 1
   → Behavior 2

2. Program P'
   → Behavior 1
   → Behavior 2

x

y

input

input

input

input

program P

program P'

program P

program P

behavior 1

behavior 2

behavior 1

behavior 2

x

y

input

input

yannic.noller@acm.org

Hybrid Differential Software Testing
Core Contributions

(1) the concept of differential fuzzing

(2) the concept of differential dynamic symbolic execution

(3) the concept of hybrid analysis in differential program analysis

(4) the concept of a hybrid setup for applying fuzzing and symbolic execution in parallel

HyDiff
Fuzzing

• term **fuzzing** was coined by Miller et al. in 1990, when they used a random testing tool to investigate the reliability of UNIX tools [Miller1990]

• classification based on degree of program analysis
  • blackbox / greybox / whitebox fuzzing

• classification based on generation technique
  • search-based fuzzing
  • generative fuzzing

• **state-of-the-art** in vulnerability detection: coverage-based, mutational fuzzing
Coverage-Based Mutational Fuzzing

**Initial Seed Files**

1. Initial seed files
2. Queue
3. Trim input
4. Mutate repeatedly
5. Mutant selection
6. Mutated files that showed (new) interesting behavior

- Crashes
- Hangs

**Problem Validation**

- Hybrid Differential Software Testing

**Summary**
Symbolic Execution

- introduced by King, Clarke, and Boyer et al. [King1976] [Clarke1976] [Boyer1975]

- analysis of programs with **unspecified inputs**, i.e. execute a program with **symbolic** inputs

- for each path, build a **path condition**
```c
int foo (int x) {
    int y;
    if (x < 0) {
        y = -x;
    } else {
        y = 2 * x;
    }
    if (y > 1) {
        return 0;
    } else {
        if (y == 1)
            assert(false);
        return 1;
    }
}
```
Shadow Symbolic Execution

[Palikareva2016]

Two-way Forking

\[ [\text{TRUE}] \alpha ? \]

true \quad \text{false}

\[ [\alpha] \ldots \quad [\neg \alpha] \ldots \]

Four-way Forking

\[ [\text{TRUE}] \text{change}(\alpha, \beta) ? \]

old: true
new: true

old: false
new: false

old: false
new: true

old: true
new: false

\[ [\alpha \land \beta] \ldots \quad [\neg \alpha \land \neg \beta] \ldots \quad [\alpha \land \neg \beta] \ldots \quad [\neg \alpha \land \beta] \ldots \]

same\text{TRUE} \quad \text{same}\text{FALSE} \quad \text{diff}\text{TRUE} \quad \text{diff}\text{FALSE}

Change Type | Example
--- | ---
Update assignment | \( x = x + \text{change}(E1, E2); \)
Update condition | \( \text{if} (\text{change}(E1, E2)) \ldots \)
Add extra assignment | \( x = \text{change}(x, E); \)
Remove assignment | \( x = \text{change}(E, x); \)
Add conditional | \( \text{if} (\text{change}(\text{false}, C)) \ldots \)
Remove conditional | \( \text{if} (\text{change}(C, \text{false})) \ldots \)
Remove code | \( \text{if} (\text{change}(\text{true}, \text{false})) \ldots \)
Add code | \( \text{if} (\text{change}(\text{false}, \text{true})) \ldots \)
Why combine Fuzzing and Symbolic Execution?

- **good** in finding **shallow** bugs, but **bad** in finding **deep** program paths
- input **reasoning ability**, but **path explosion** and **constraint solving**
Related Work

- regression analysis
- side-channel analysis
- worst-case complexity analysis
- robustness analysis of neural networks
Differential Fuzzing

Initial seed files

Queue

Trim input

Mutate repeatedly

Mutant selection

Drives by differential metrics

Mutated files that showed (new) interesting behavior

Crashes

Hangs

[yannic.noller@acm.org]

Hybrid Differential Software Testing
Differential Metrics

- output difference (odiff)
- decision difference (ddiff)
- cost difference (cdiff)
- patch distance (only for regression testing)
Differential Fuzzing

mutate repeatedly

mutated files that showed (new) interesting behavior

mutant selection by input evaluation for the instrumented program $P$

fuzzing driver

check for new interesting, differential properties

problem

background

solutions

validation

summary

mutate repeatedly

mutated files that showed (new) interesting behavior

mutant selection by input evaluation for the instrumented program $P$

fuzzing driver

check for new interesting, differential properties
Differential Dynamic SymExe

DDSE

1. import inputs
2. Trie Extension / Input Assessment
3. Exploration
4. Input Generation
5. export inputs

model generation
input generation

concolic execution
includes
heuristics-based analysis

trie-guided symbolic execution
bounded symbolic execution

interesting input
new input
most promising node
path condition

import inputs -> export inputs

Hybrid Differential Software Testing
HyDiff’s overview

Problem

Validation

Background

Solutions

Contribution

Summary

HyDiff's overview

**Input**
- program versions
- seed input files
- change-annotated program

**Fuzzing**
- instrumentation
  - import
  - assessment
  - mutate inputs
  - fuzzer output queue

**Symbolic Execution**
- ICFG
  - import
  - trie extension / assessment
  - exploration
  - constraint solving / input generation

**Output**

<table>
<thead>
<tr>
<th></th>
<th>input</th>
<th>+odiff</th>
<th>+ddiff</th>
<th>+crash</th>
<th>+cdiff</th>
<th>+patch-dist</th>
<th>+cov</th>
</tr>
</thead>
<tbody>
<tr>
<td>id:0001</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>id:0002</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>id:0003</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

set of divergence revealing test inputs
Research Questions

**RQ1:** How good is *differential fuzzing* and what are the limitations?

**RQ2:** How good is *differential dynamic symbolic execution* and what are the limitations?

**RQ3:** Can the *hybrid* approach outperform the single techniques?

**RQ4:** Can the hybrid approach *not only combine* the results of fuzzing and symbolic execution, but also *amplify* the search itself and generate even better results than each approach on its own?

**RQ5:** Can the proposed hybrid differential software testing approach *reveal behavioral differences* in software?
Evaluation Strategy

Quantitative analysis based on benchmarks in the specific application areas in differential analysis:

A1  Regression Analysis
A2  Worst-Case Complexity Analysis
A3  Side-Channel Analysis
A4  Robustness Analysis of Neural Networks
Evaluation Metrics

A1
Regression Analysis

A4
Robustness Analysis of Neural Networks

A2
Worst-Case Complexity Analysis

A3
Side-Channel Analysis

- average time to first output difference $(t +odiff)$
- $t_{min}$
- average output differences (#odiff)
- average decision differences (#ddiff)

- average maximum cost
- $cost_{max}$
- time to first cost improvement
Evaluation Infrastructure

What to compare?

- Differential Fuzzing (DF)
- Parallel Differential Fuzzing (PDF)
- Differential Dynamic Symbolic Execution (DDSE)
  DDSE with double time budget (DDSEx2T)
- Hybrid Differential Software Testing (HyDiff)
## Regression Analysis

<table>
<thead>
<tr>
<th>Subject (§ changes)</th>
<th>Differential Fuzzing (DF)</th>
<th>Parallel Differential Fuzzing (PDF)</th>
<th>Differential Dynamic Sym. Exec. (DDSE)</th>
<th>DDSE double time budget (DDSE×2T)</th>
<th>HyDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCAS-1 (1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-2 (1)</td>
<td>441.83 (57.70)</td>
<td>120 0.70 (-0.23)</td>
<td>213.93 (0.73)</td>
<td>213.93 (0.73)</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-3 (1)</td>
<td>588.43 (15.18)</td>
<td>392 0.10 (-0.11)</td>
<td>388.63 (5.16)</td>
<td>388.63 (5.16)</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-4 (1)</td>
<td>28.47 (10.42)</td>
<td>2 1.00 (0.00)</td>
<td>182.87 (1.06)</td>
<td>182.87 (1.06)</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-5 (1)</td>
<td>384.95 (46.66)</td>
<td>24 2.00 (0.00)</td>
<td>317.91 (1.16)</td>
<td>317.91 (1.16)</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-6 (1)</td>
<td>233.63 (54.46)</td>
<td>4 0.97 (0.00)</td>
<td>413.89 (0.83)</td>
<td>413.89 (0.83)</td>
<td>20.10 (0.14)</td>
</tr>
<tr>
<td>TCAS-7 (1)</td>
<td>0</td>
<td>-</td>
<td>0.00 (0.00)</td>
<td>0.00 (0.00)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>TCAS-8 (1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>TCAS-9 (1)</td>
<td>221.73 (-48.83)</td>
<td>10 1.00 (0.00)</td>
<td>221.73 (-48.83)</td>
<td>221.73 (-48.83)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>TCAS-10 (2)</td>
<td>173.47 (-46.27)</td>
<td>1 1.93 (0.09)</td>
<td>173.47 (-46.27)</td>
<td>173.47 (-46.27)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>Math-10 (1)</td>
<td>221.13 (-56.20)</td>
<td>10 64.50 (-15.98)</td>
<td>15.50 (-2.35)</td>
<td>15.50 (-2.35)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>Math-46 (1)</td>
<td>377.87 (-63.43)</td>
<td>77 0.80 (-0.14)</td>
<td>363.63 (-1.07)</td>
<td>363.63 (-1.07)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>Math-60 (7)</td>
<td>6.93 (-0.63)</td>
<td>4 219.17 (-5.26)</td>
<td>92.90 (-1.64)</td>
<td>92.90 (-1.64)</td>
<td>0.00 (0.00)</td>
</tr>
<tr>
<td>Time-1 (14)</td>
<td>5.17 (1.20)</td>
<td>2 123.30 (5.86)</td>
<td>170.63 (3.43)</td>
<td>170.63 (3.43)</td>
<td>221.00 (7.84)</td>
</tr>
<tr>
<td>CLT-2 (13)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.23 (0.18)</td>
</tr>
<tr>
<td>CLT-2 (13)</td>
<td>10.83 (3.33)</td>
<td>2 82.30 (-3.98)</td>
<td>176.83 (-3.62)</td>
<td>176.83 (-3.62)</td>
<td>5.23 (0.18)</td>
</tr>
<tr>
<td>CLT-4 (8)</td>
<td>7.43 (4.60)</td>
<td>1 96.73 (-4.54)</td>
<td>279.13 (-5.41)</td>
<td>279.13 (-5.41)</td>
<td>463.83 (9.52)</td>
</tr>
<tr>
<td>CLT-5 (13)</td>
<td>589.57 (-16.05)</td>
<td>358 0.07 (-0.07)</td>
<td>219.50 (-5.74)</td>
<td>219.50 (-5.74)</td>
<td>463.83 (9.52)</td>
</tr>
<tr>
<td>CLT-6 (21)</td>
<td>4.13 (-3.04)</td>
<td>1 143.87 (-4.99)</td>
<td>182.60 (-5.54)</td>
<td>182.60 (-5.54)</td>
<td>463.83 (9.52)</td>
</tr>
</tbody>
</table>

**HyDiff** classifies all subjects correctly.

Components do benefit from each other.

yannic.noller@acm.org

Hybrid Differential Software Testing
Worst-Case Complexity Analysis

DDSE quickly makes progress, DF continuously improves the score.

HyDiff successfully combines strengths of DDSE and DF.

HyDiff outperforms the components in isolation.
Side-Channel Analysis

- in regression testing: \textit{changes} in the \textit{program}
- in side-channel analysis: \textit{changes} in the \textit{input}

\[
\text{secret} = \text{change}(\text{secret1, secret2})
\]
Side-Channel Analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Version</th>
<th>Differential Fuzzing (DF)</th>
<th>Themis</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \delta ) ( \delta_{\text{max}} ) ( \bar{\delta} : \delta &gt; 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spring-Security</td>
<td>Safe</td>
<td>1.00 (±0.00)</td>
<td>1</td>
<td>4.77 (±1.07)</td>
</tr>
<tr>
<td>Spring-Security</td>
<td>Unsafe</td>
<td>149.00 (±0.00)</td>
<td>149</td>
<td>4.17 (±0.90)</td>
</tr>
<tr>
<td>JDK7-MsgDigest</td>
<td>Safe</td>
<td>1.00 (±0.00)</td>
<td>1</td>
<td>10.77 (±2.12)</td>
</tr>
<tr>
<td>JDK6-MsgDigest</td>
<td>Unsafe</td>
<td>140.03 (±20.39)</td>
<td>263</td>
<td>3.20 (±0.81)</td>
</tr>
<tr>
<td>Picketbox</td>
<td>Safe</td>
<td>1.00 (±0.00)</td>
<td>1</td>
<td>16.90 (±3.89)</td>
</tr>
<tr>
<td>Picketbox</td>
<td>Unsafe</td>
<td>363.70 (±562.18)</td>
<td>8,822</td>
<td>5.13 (±1.83)</td>
</tr>
<tr>
<td>Tomcat</td>
<td>Safe</td>
<td>25.07 (±0.36)</td>
<td>26</td>
<td>19.90 (±9.29)</td>
</tr>
<tr>
<td>Tomcat</td>
<td>Unsafe</td>
<td>49.00 (±0.36)</td>
<td>50</td>
<td>23.53 (±9.73)</td>
</tr>
<tr>
<td>jetty</td>
<td>Safe</td>
<td>11.77 (±0.60)</td>
<td>15</td>
<td>3.77 (±0.72)</td>
</tr>
<tr>
<td>Jetty</td>
<td>Unsafe</td>
<td>70.87 (±6.12)</td>
<td>105</td>
<td>6.83 (±1.62)</td>
</tr>
<tr>
<td>orientdb</td>
<td>Safe</td>
<td>1.00 (±0.00)</td>
<td>1</td>
<td>16.60 (±5.14)</td>
</tr>
<tr>
<td>orientdb</td>
<td>Unsafe</td>
<td>458.93 (±685.64)</td>
<td>10,776</td>
<td>4.77 (±1.06)</td>
</tr>
<tr>
<td>pac4j</td>
<td>Safe</td>
<td>10.00 (±0.00)</td>
<td>10</td>
<td>1.10 (±0.11)</td>
</tr>
<tr>
<td>pac4j</td>
<td>Unsafe</td>
<td>11.00 (±0.00)</td>
<td>11</td>
<td>1.13 (±0.12)</td>
</tr>
<tr>
<td>pac4j*</td>
<td>Unsafe*</td>
<td>39.00 (±0.00)</td>
<td>39</td>
<td>1.10 (±0.11)</td>
</tr>
</tbody>
</table>

DF can find the same vulnerabilities as static analysis

well-balanced combination: fast and high delta (important to assess the severity of vulnerability)

yannic.noller@acm.org

Hybrid Differential Software Testing

35
Robustness Analysis of Neural Networks

Purpose: stress test proposed technique

- similar to SC analysis: \texttt{changes} in the input
- similar to regression analysis: \texttt{search for output differences}
- idea: allow up to $x\%$ changes in the pixels of the input image

\[
\text{a}[i][j] = \text{change}(\text{a}[i][j], \text{value});
\]
## NN Analysis

### Problem

- **Subject**: 50
- **Contribution**: 10
- **Validation**: 2
- **Solutions**: 1

### Results for robustness analysis of neural networks (t=)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T +odiff</td>
<td>t_{min}</td>
<td>#odiff</td>
<td>ddiff</td>
</tr>
<tr>
<td>1</td>
<td>2,725.40 (±341.09)</td>
<td>1,074</td>
<td>0.57 (±0.20)</td>
<td>7.73 (±0.18)</td>
</tr>
<tr>
<td>2</td>
<td>2,581.47 (±326.21)</td>
<td>1,032</td>
<td>0.93 (±0.28)</td>
<td>7.93 (±0.13)</td>
</tr>
<tr>
<td>5</td>
<td>2,402.97 (±329.59)</td>
<td>1,189</td>
<td>1.23 (±0.37)</td>
<td>6.47 (±0.18)</td>
</tr>
<tr>
<td>10</td>
<td>2,155.40 (±343.76)</td>
<td>996</td>
<td>1.57 (±0.34)</td>
<td>8.10 (±0.17)</td>
</tr>
<tr>
<td>20</td>
<td>1,695.83 (±228.18)</td>
<td>953</td>
<td>2.70 (±0.37)</td>
<td>9.13 (±0.12)</td>
</tr>
<tr>
<td>50</td>
<td>1,830.83 (±259.79)</td>
<td>1,220</td>
<td>2.43 (±0.42)</td>
<td>6.33 (±0.21)</td>
</tr>
<tr>
<td>100</td>
<td>1,479.17 (±231.25)</td>
<td>960</td>
<td>2.47 (±0.37)</td>
<td>9.37 (±0.20)</td>
</tr>
</tbody>
</table>

### Background

- HyDiff can combine them so that both can benefit from each other.
RQ 1: Differential Fuzzing

- **Regression**: performs quite reasonable, but not all subject correctly classified (parallel DF did not help)
- **WCA**: improves cost continuously over time
- **SC**: outperforms Blazer and Themis
- **NN**: effective, but very slow (gets better with more x%)

Differential Fuzzing **continuously improves** its differential analysis over time

Parallel Differential Fuzzing **even better**, sometimes outperformed hybrid combination
RQ 2: Differential Dynamic Symbolic Execution

- **Regression**: fast in finding output differences, but not all subject correctly classified
- **WCA**: often stays in plateaus without improvement, but good in finding some first slowdown
- **SC**: slow in the beginning, but eventually high delta
- **NN**: very fast for first output difference, but limited by heavy constraint solving

DDSE develops in jumps and only rarely in continuous improvement

**Effective** technique due to constraint solving

DDSE with twice the time budget does not improve the result
RQ 3+4: Hybrid combination

- **Regression**: HyDiff finds all output differences and often generates higher values in a shorter time period
- **WCA**: clearly outperforms components
- **SC**: no clear improvement, but well balanced combination
- **NN**: good combination, finds output differences and is fast

HyDiff does not only combine results of components but also amplifies them
RQ 5: HyDiff for Differential Testing

- **Regression**: crashes not present, but inputs for behavioral differences
- **WCA**: AC vulnerabilities identified
- **SC**: all vulnerabilities identified
- **NN**: limits of HyDiff, however found adversarial inputs

HyDiff is **effective** for differential testing
Publications

Shadow Symbolic Execution with Java PathFinder
Yannic Noller, Hoang Lam Nguyen, Minxing Tang, and Timo Kehrer
Java Pathfinder Workshop 2017, SIGSOFT Software Engineering Notes 42 (January 2018)

Badger: Complexity Analysis with Fuzzing and Symbolic Execution
Yannic Noller, Rody Kersten, and Corina S. Păsăreanu
ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA) 2018

Differential Program Analysis with Fuzzing and Symbolic execution
Yannic Noller (Doctoral Symposium Paper)
ACM/IEEE International Conference on Automated Software Engineering (ASE) 2018

DifFuzz: Differential Fuzzing for Side-Channel Analysis
Shirin Nilizadeh*, Yannic Noller*, and Corina S. Păsăreanu (* joint first authors)
ACM/IEEE International Conference on Software Engineering (ICSE) 2019

Complete Shadow Symbolic Execution with Java PathFinder
Yannic Noller, Hoang Lam Nguyen, Minxing Tang, Timo Kehrer and Lars Grunske
Java Pathfinder Workshop 2019, SIGSOFT Software Engineering Notes 44 (December 2019)

HyDiff: Hybrid Differential Software Analysis
Yannic Noller, Corina S. Păsăreanu, Marcel Böhme, Youcheng Sun, Hoang Lam Nguyen, and Lars Grunske
ACM/IEEE International Conference on Software Engineering (ICSE) 2020
Hybrid Differential Software Testing

Problem: Why combine Fuzzing and Symbolic Execution?

- Good in finding shallow bugs, but bad in finding deep program paths
- Input reasoning ability, but path explosion and constraint solving

Background: What is Hybrid Differential Software Testing?

- Differential Fuzzing
- Differential Dynamic SymExe

Solutions: How does HyDiff's overview work?

- HyDiff: Trie Extension/Input Assessment
- DOSE: Model generation

Validation: Regression Analysis

Summary: Contribution

- mutant selection by input evaluation for the instrumented program
- exploration and constraint solving
References


yannic.noller@acm.org

Hybrid Differential Software Testing
References (continued)


References (continued)


References (continued)


END OF DOCUMENT