Hybrid Differential Software Testing

Disputation

Yannic Noller

Humboldt-Universität zu Berlin
yannic.noller@acm.org

Agenda

Problem

Contribution

Background

Solutions

Validation

Summary

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]

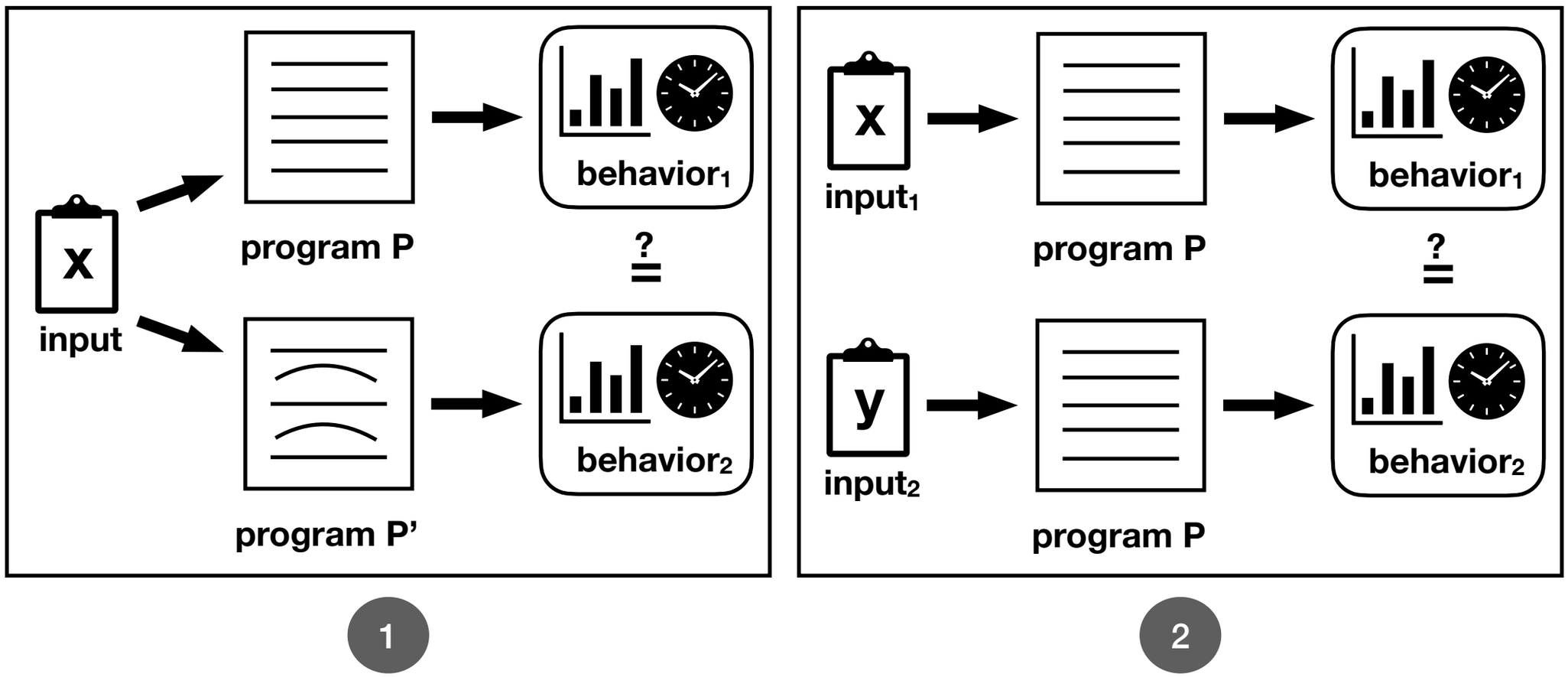
Software Quality Assurance

Software Testing

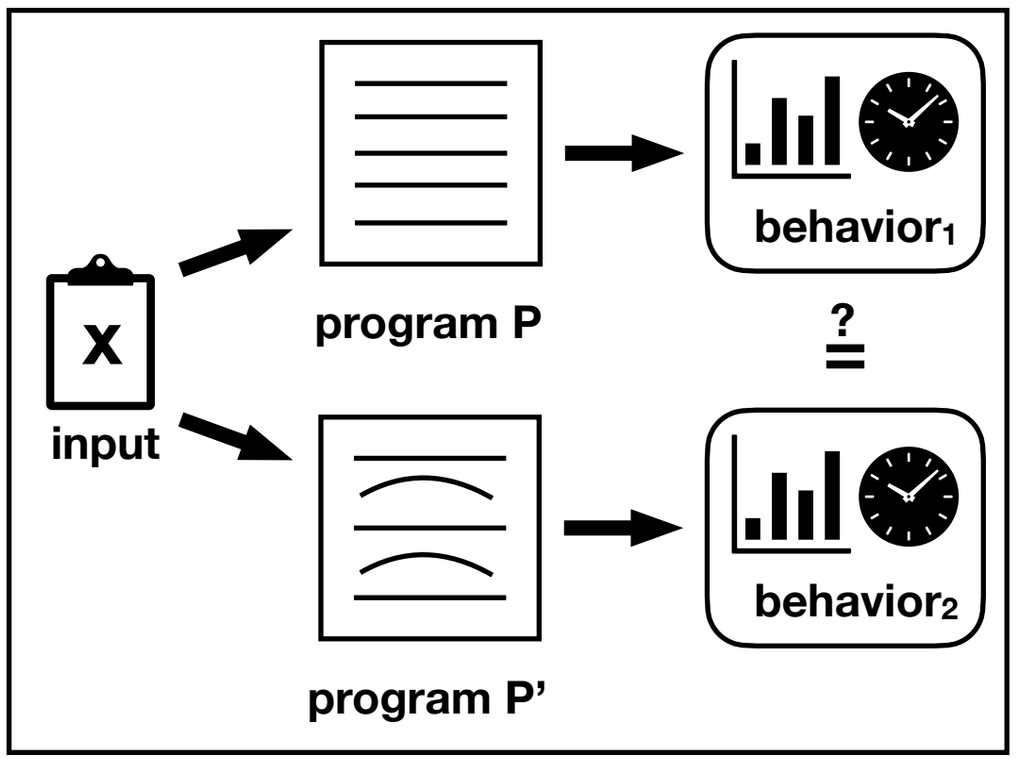
Software Testing

[Orso2014]

Differential Software Testing



Differential Software Testing



1

between **two program versions**
for the **same input**
➥ *software maintenance*

Regression Analysis

Regression Analysis

```
1   int foo (int x) {
2       int y;
3       if (x < 0) {
4-          y = -x;
4+          y = x * x;
5       } else {
6           y = 2 * x;
7       }
8+      y = y + 1;
9       if (y > 1) {
10          return 0;
11      } else {
12          if (y == 1)
13              assert(false);
14      }
15      return 1;
16  }
```

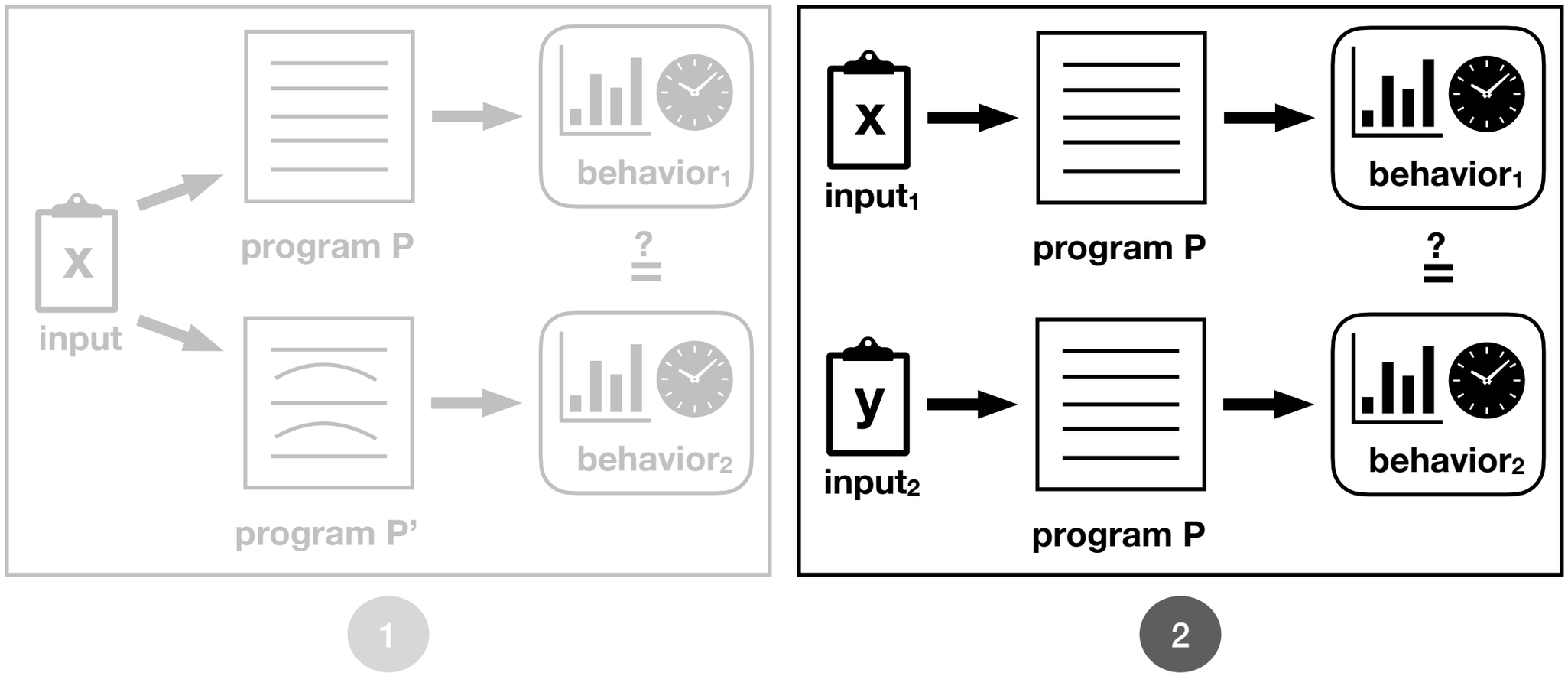
***Fixed** assertion error for **x=-1** (returns 0).*

Are there unintended **behavioral** differences between the two versions?

***introduced** new assertion error for **x=0** (previously returned 1)*

→ ***Regression Bug***

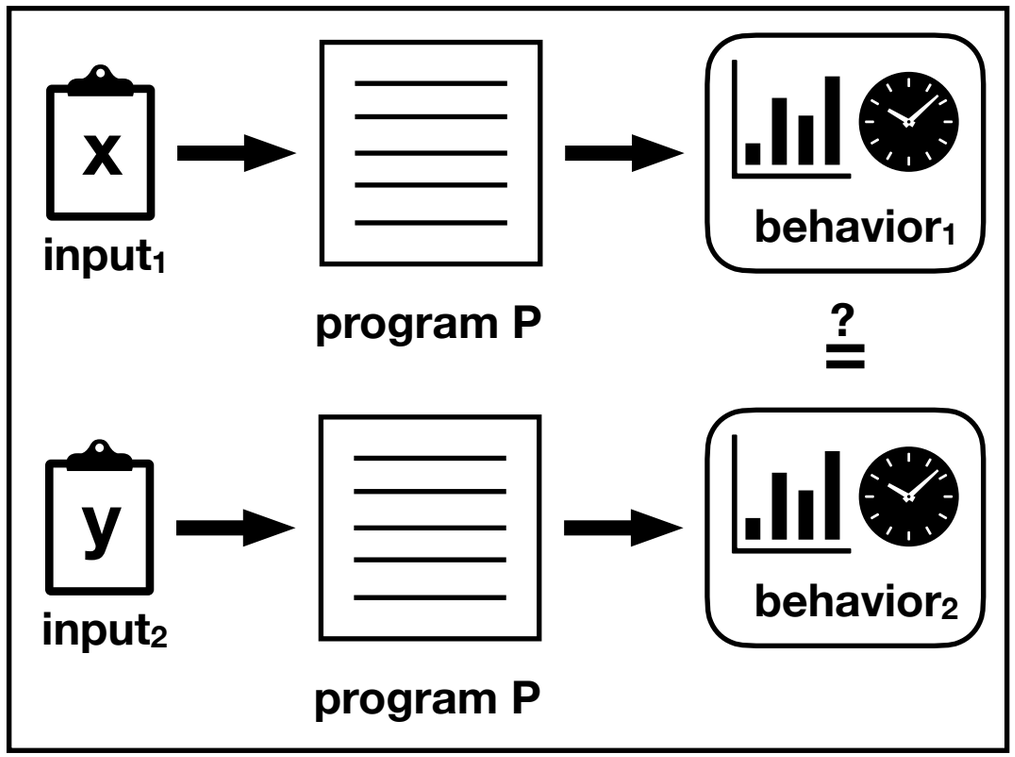
Differential Software Testing



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



2

Worst-Case Complexity Analysis

Goal: discover vulnerabilities related to algorithmic complexity

```
0   public void sort (int[] a) {
1       int N = a.length;
2       for (int i = 1; i < N; i++) {
3           int j = i - 1;
4           int x = a[i];
5           while ((j >= 0) && (a[j] > x)) {
6               a[j + 1] = a[j];
7               j--;
8           }
9           a[j + 1] = x;
10      }
11  }
```

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,
  - …

Example: Side-Channel Vulnerability

```
0   boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1       if (pub.length != sec.length) {
2           return false;
3       }
4       for (int i = 0; i < pub.length; i++) {
5           if (pub[i] != sec[i]) {
6               return false;
7           }
8       }
9       return true;
10  }
```

Unsafe Password Checking

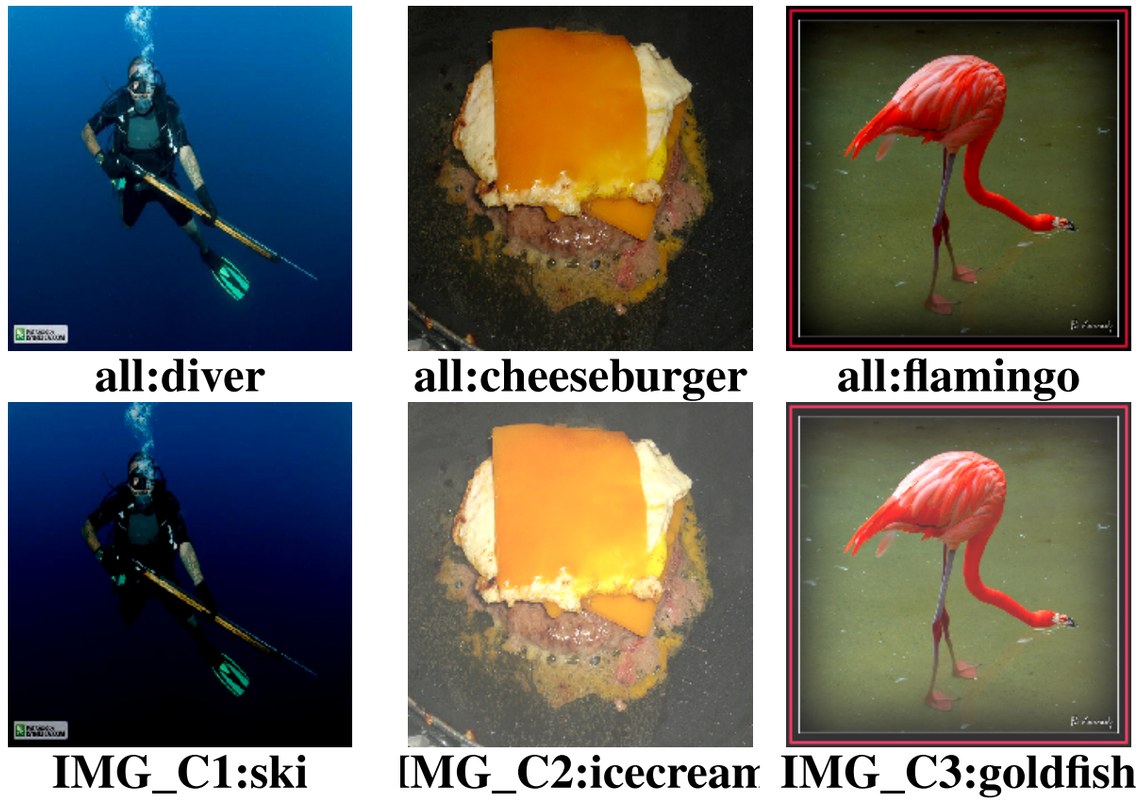
Robustness Analysis of Neural Networks

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

[Pei2017]

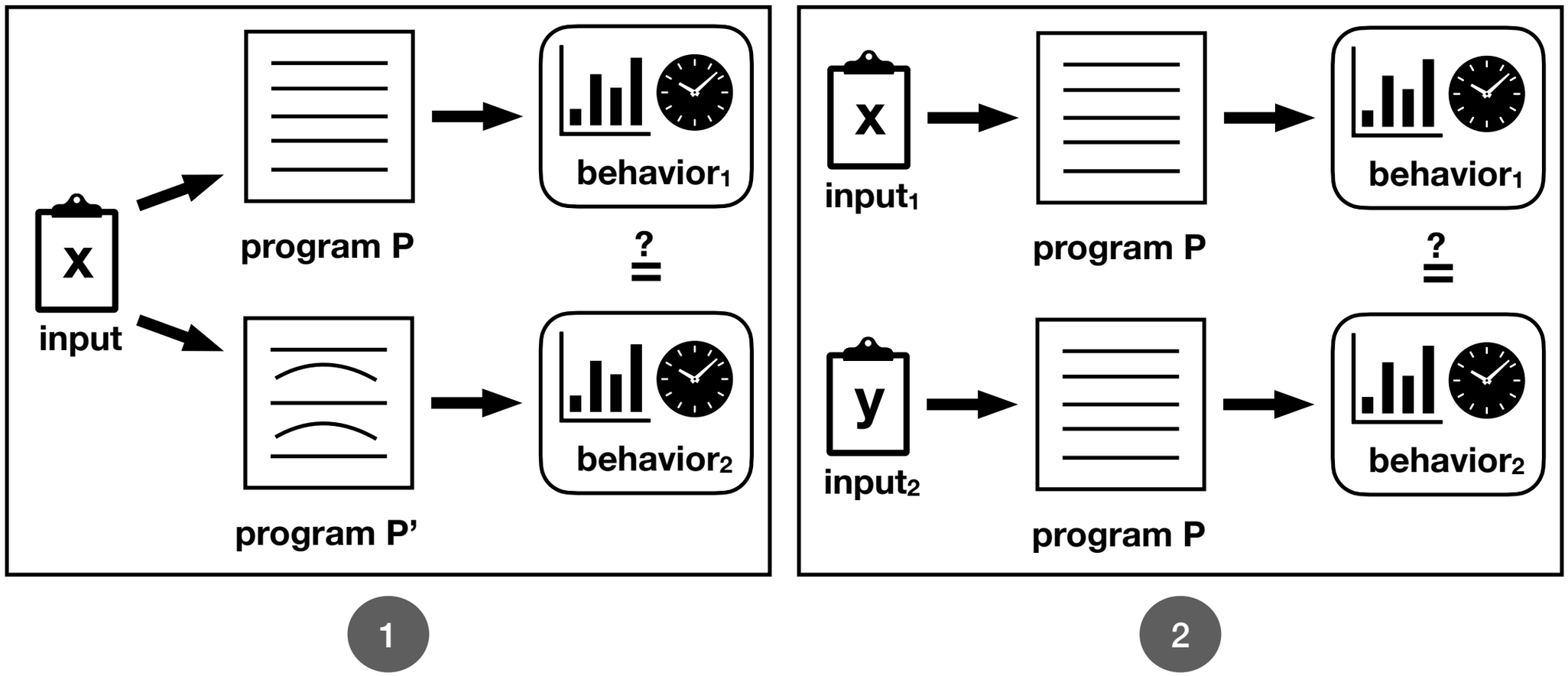
adversarial input

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



(My) Research Problem

➥ identify behavioral differences



Core Contributions

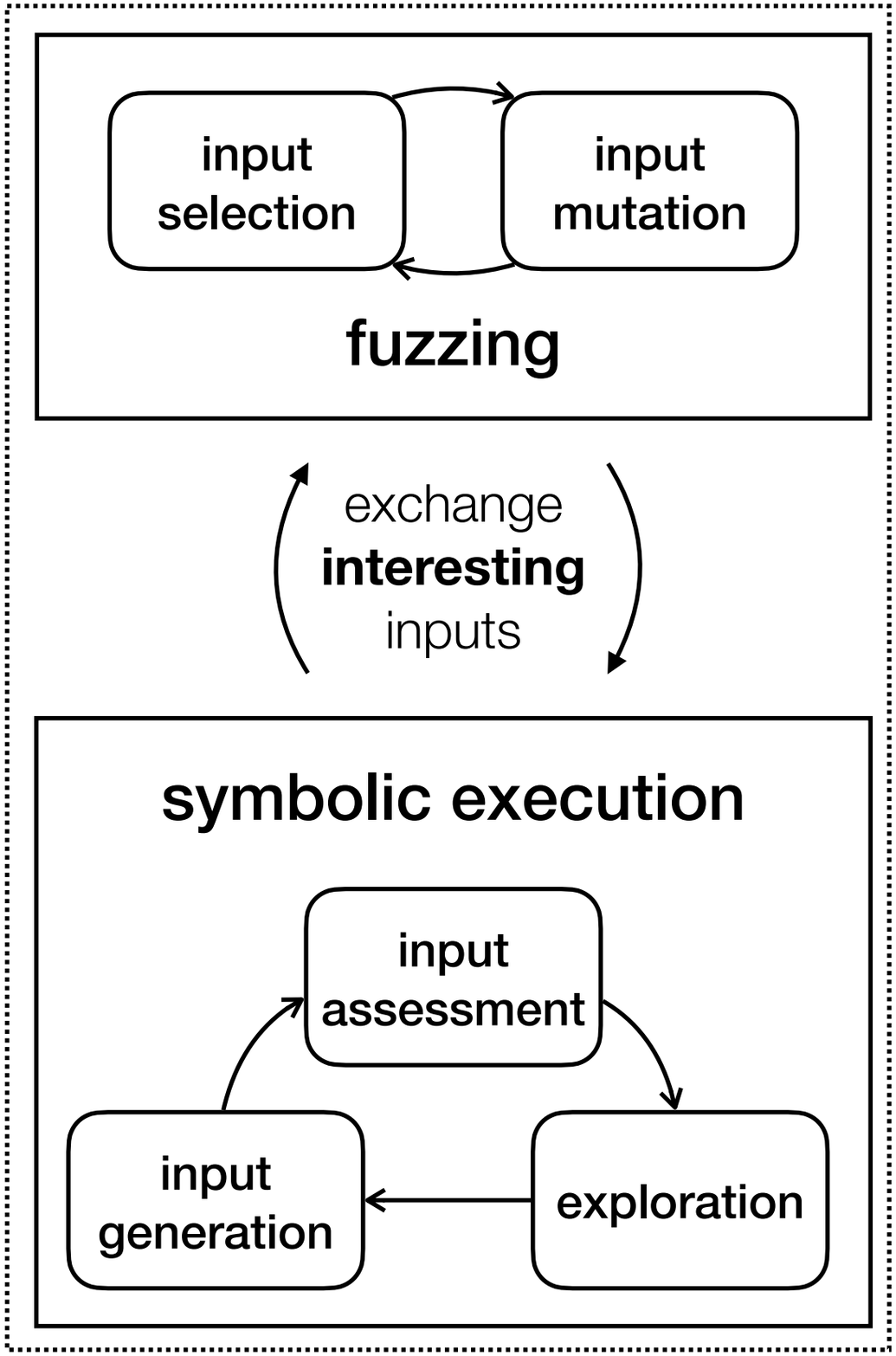
(1) the concept of differential fuzzing

(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

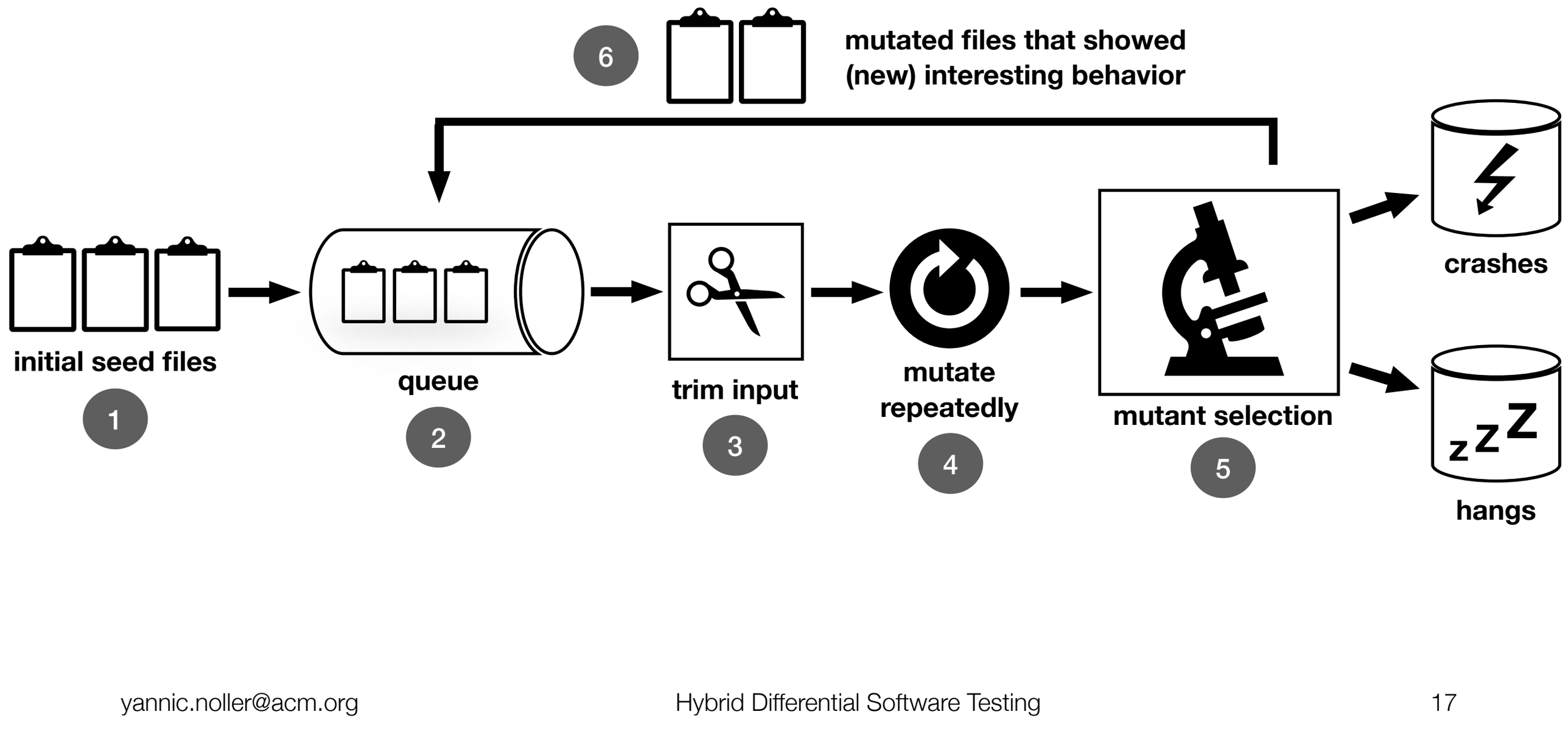
HyDiff



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

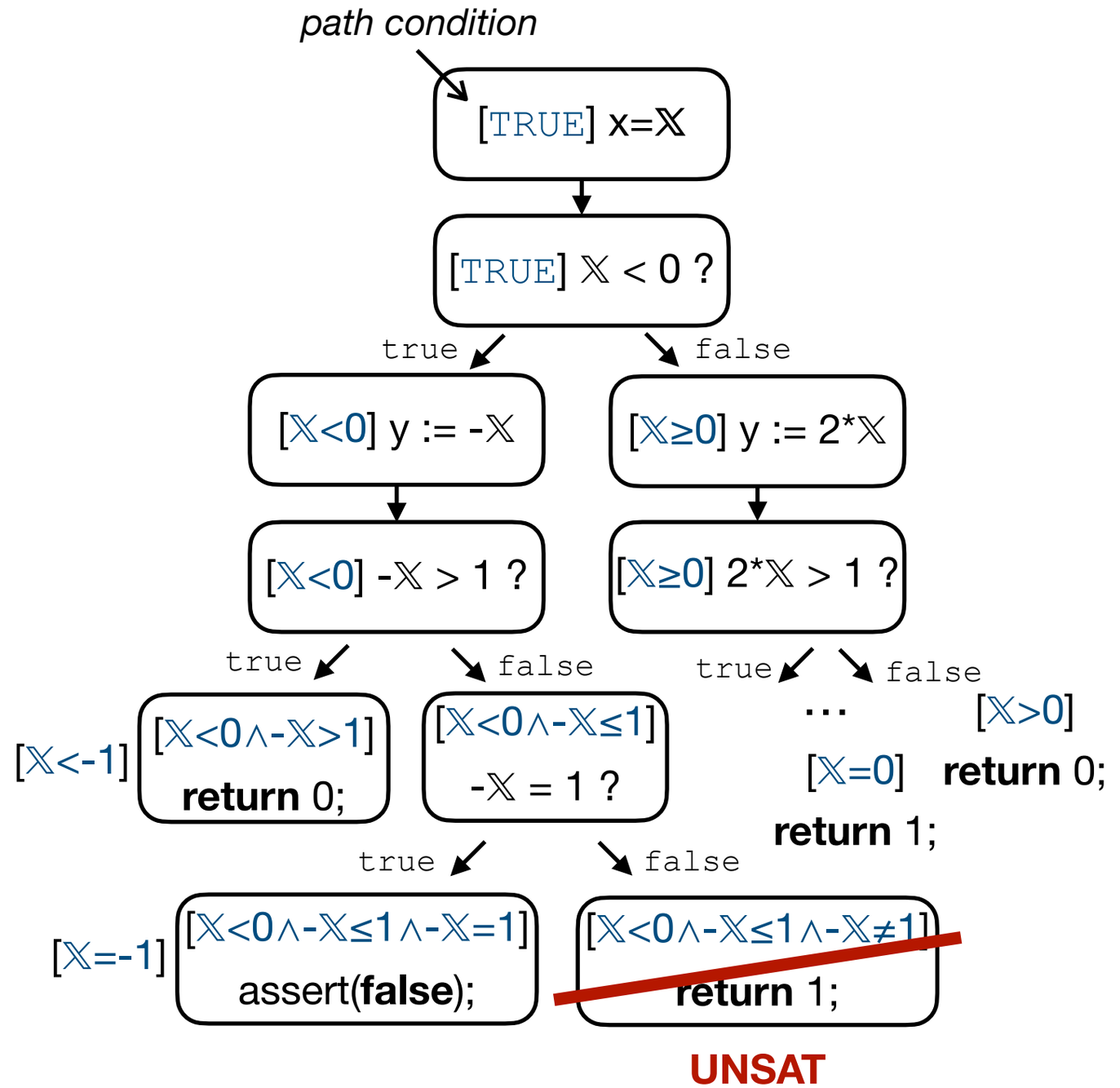


Symbolic Execution

- 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**

Example: Symbolic Execution

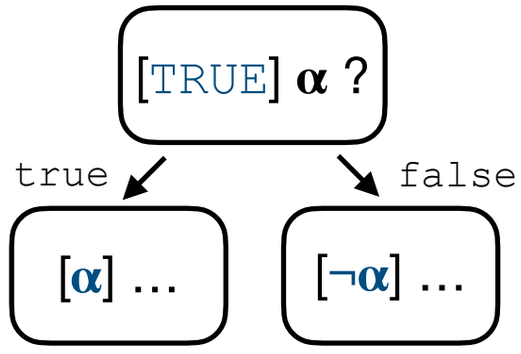
```
1  int foo (int x) {
2      int y;
3      if (x < 0) {
4          y = -x;
5      } else {
6          y = 2 * x;
7      }
8      if (y > 1) {
9          return 0;
10     } else {
11         if (y == 1)
12             assert(false);
13     }
14     return 1;
15 }
```



Shadow Symbolic Execution

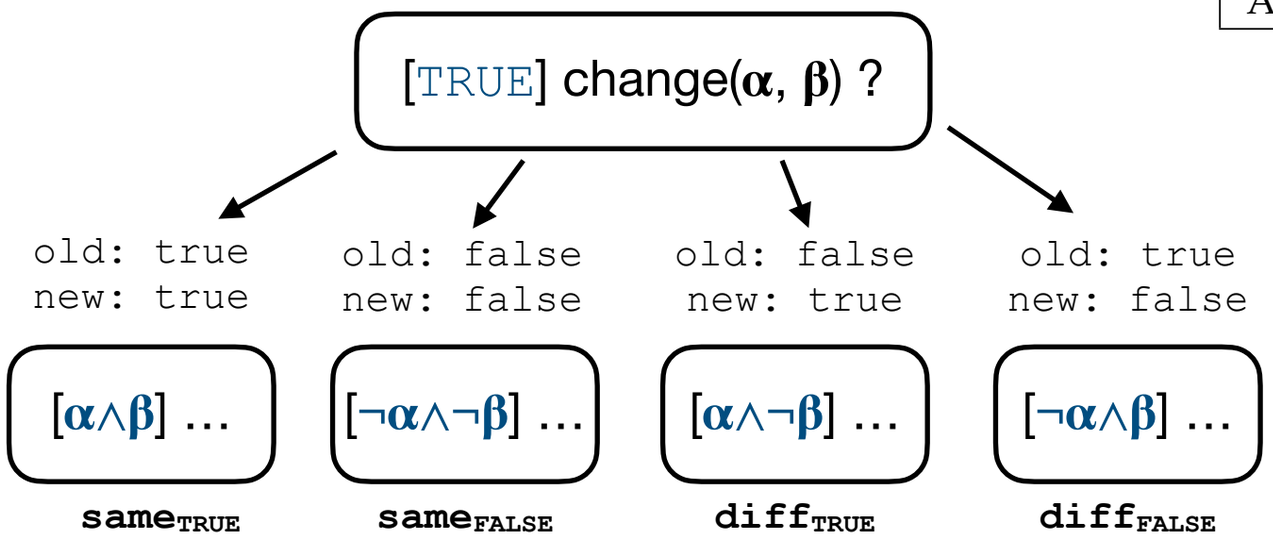
[Palikareva2016]

Two-way Forking



Change Type	Example
Update assignment	`x = x + change(E1, E2);`
Update condition	`if(change(E1, E2)) ...`
Add extra assignment	`x = change(x, E);`
Remove assignment	`x = change(E, x);`
Add conditional	`if(change(false, C)) ...`
Remove conditional	`if(change(C, false)) ...`
Remove code	`if(change(true, false)) ...`
Add code	`if(change(false, true)) ...`

Four-way Forking



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
  [Person2008, Person2011, Yang2012, Orso2008, Taneja2008]
- side-channel analysis
  [Antonopoulos2017, Chen2017, Pasareanu2016, Brennan2018]
- worst-case complexity analysis
  [Petsios2017, Lemieux2018, Burnim2009, Luckow2017]
- robustness analysis of neural networks
  [Ma2018, Pei2017, Sun2018, Goodfellow2014, Tian2018]

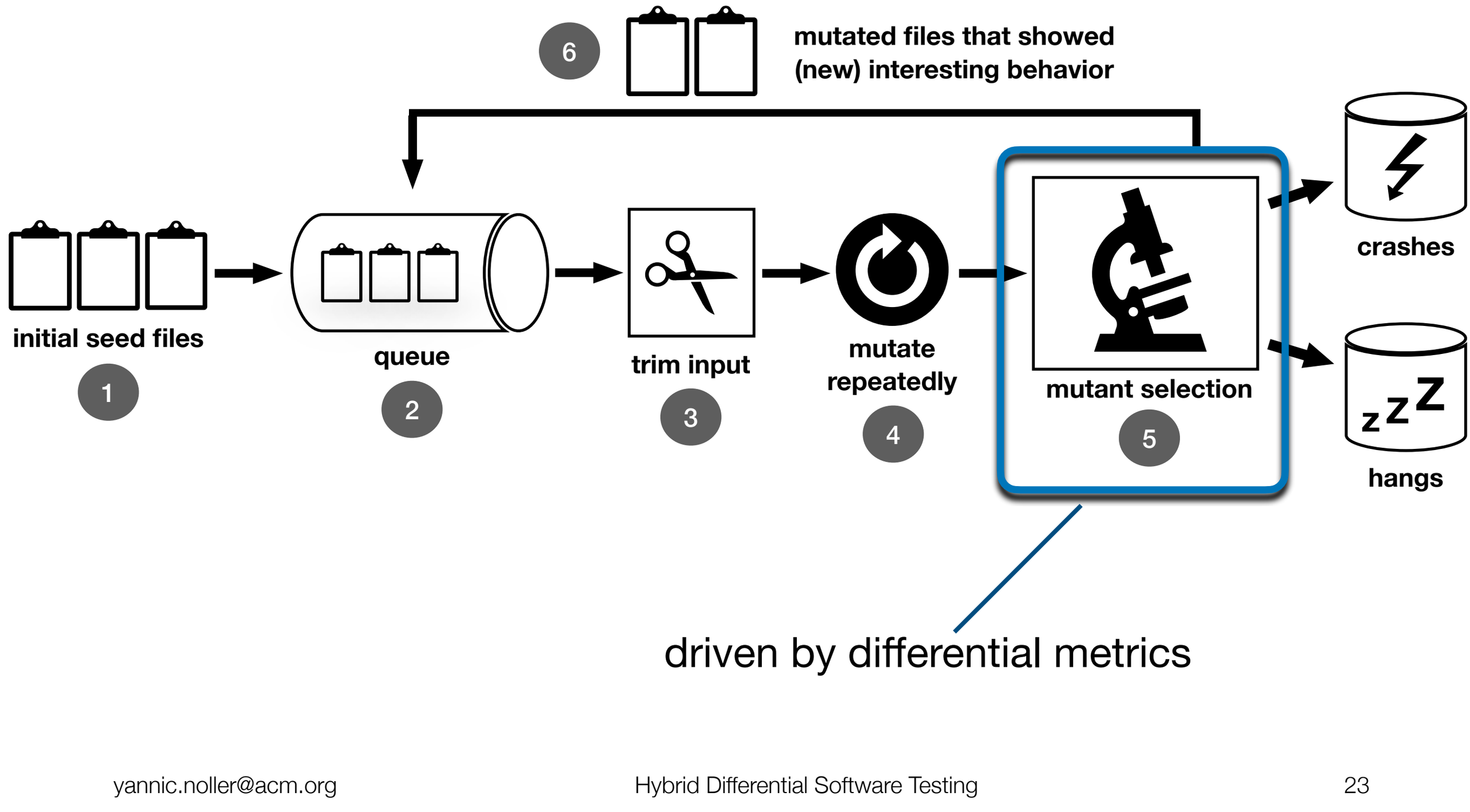
➡ not directed to differential behavior

➡ typical fuzzing problems

➡ exhaustive exploration necessary

➡ abstractions, bounded analysis, depend on models

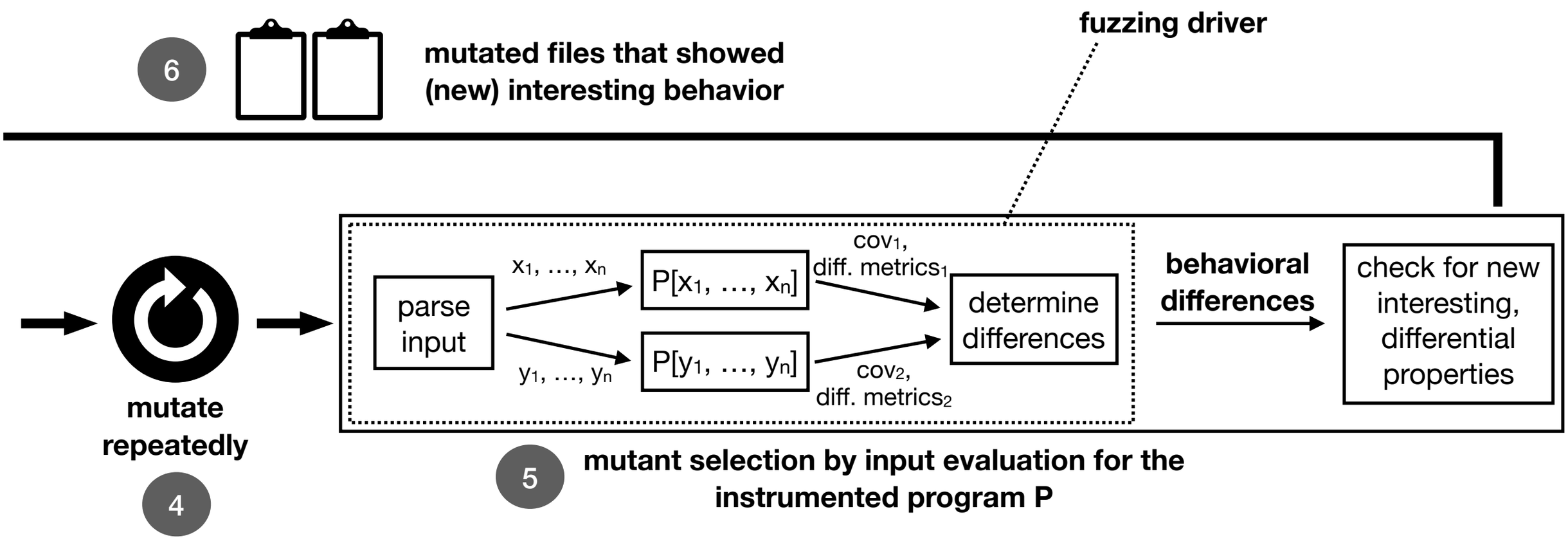
Differential Fuzzing



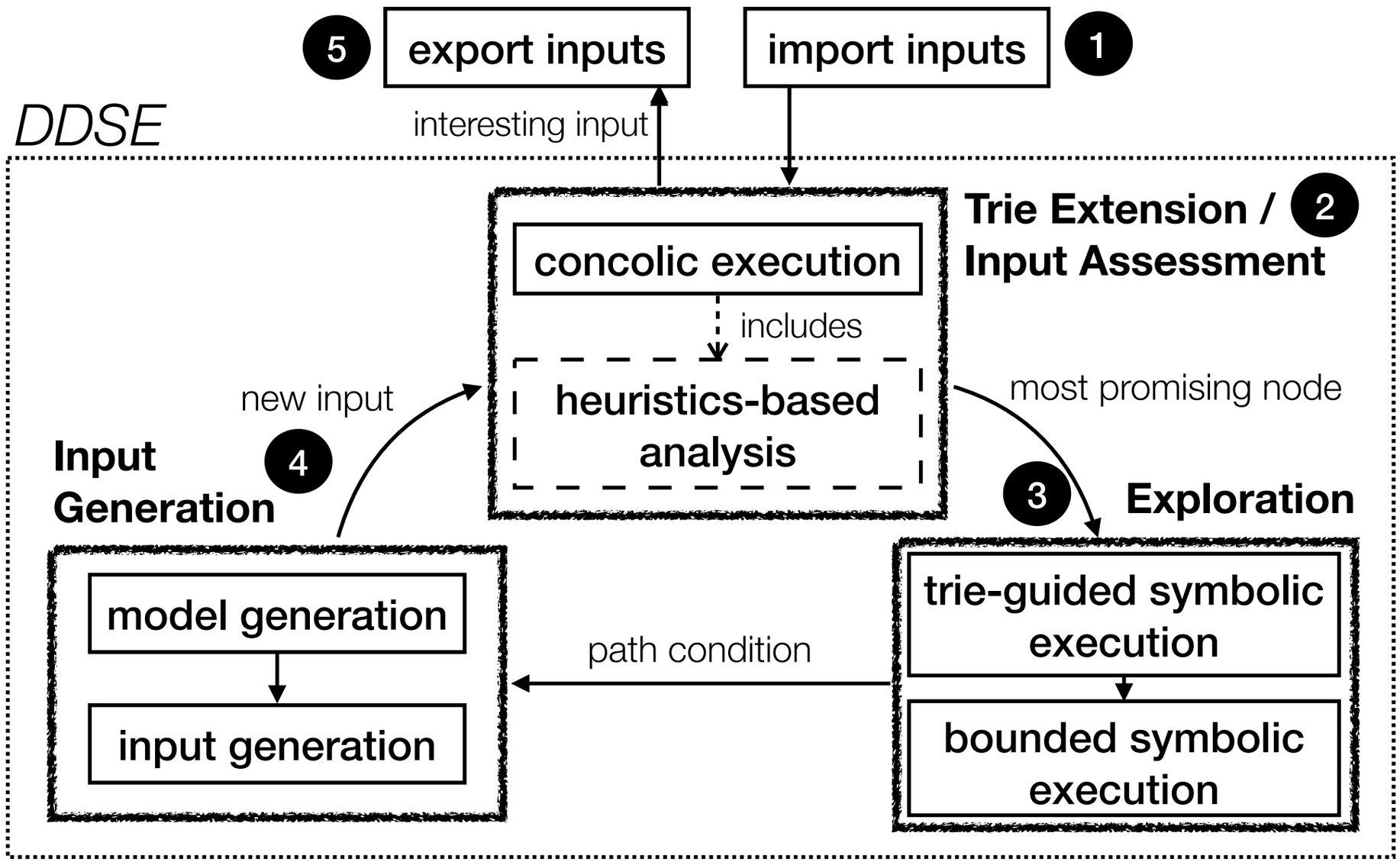
Differential Metrics

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

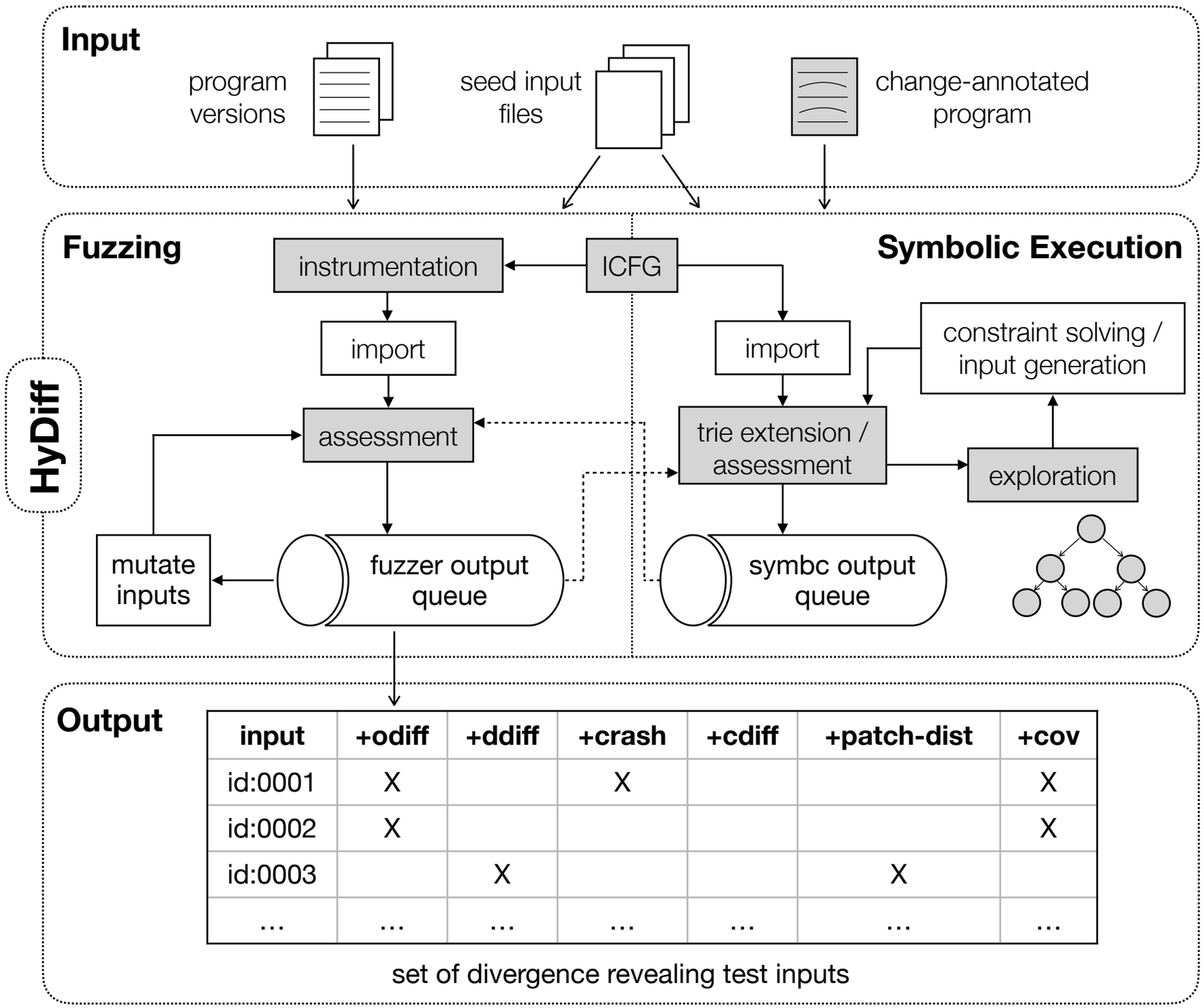
Differential Fuzzing



Differential Dynamic SymExe



HyDiff's overview



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

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

A4 Robustness Analysis of Neural Networks

A2 Worst-Case Complexity Analysis

- average maximum cost
- $cost_{max}$

A3 Side-Channel Analysis

- 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

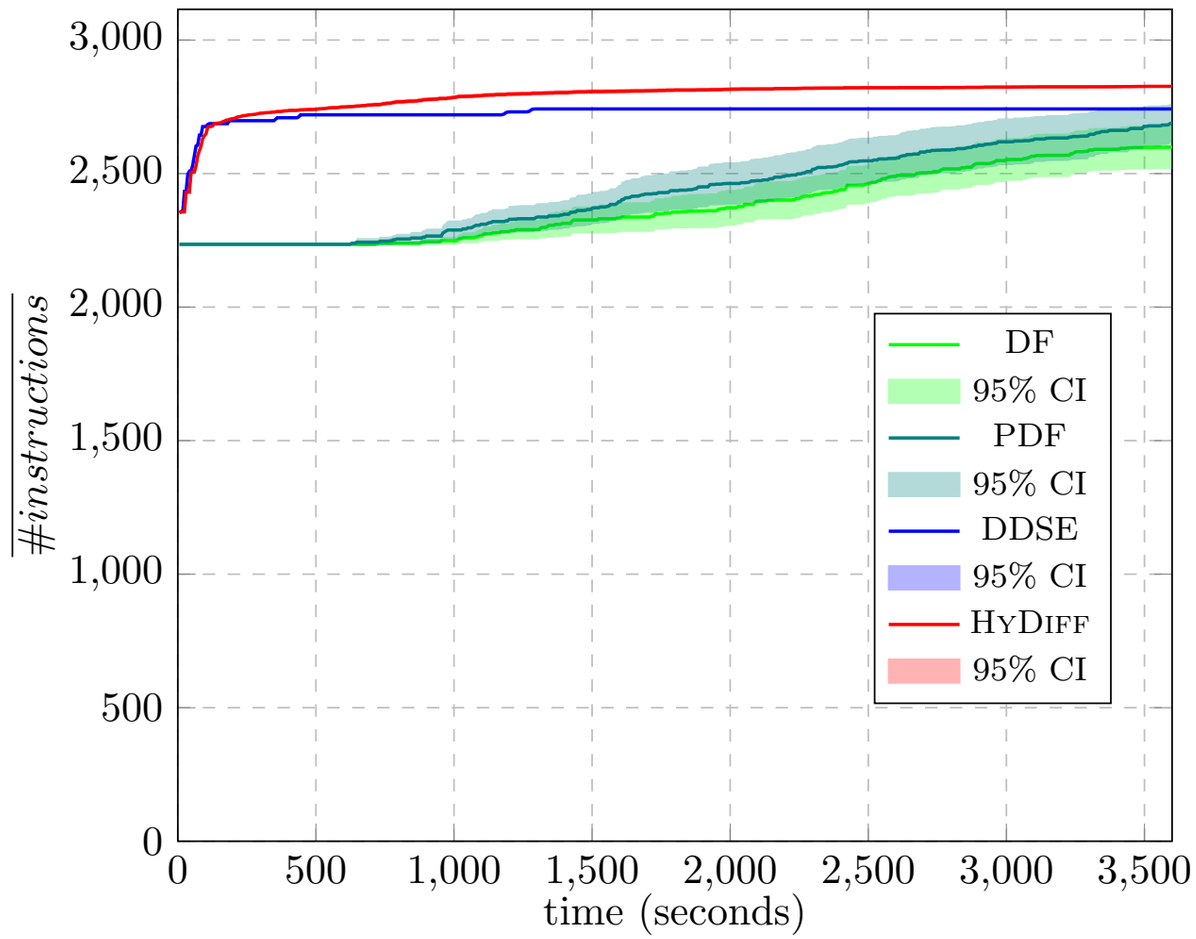
Subject (# changes)	Differential Fuzzing (DF) $\bar{t}$ +odiff	t_{min}	#odiff	#ddiff	Parallel Differential Fuzzing (PDF) $\bar{t}$ +odiff	t_{min}	#odiff	#ddiff	Differential Dynamic Sym. Exec. (DDSE) $\bar{t}$ +odiff	t_{min}	#odiff	#ddiff	DDSE double time budget (DDSEx2T) $\bar{t}$ +odiff	t_{min}	#odiff	#ddiff	HyDiff $\bar{t}$ +odiff	t_{min}	#odiff	#ddiff
TCAS-1 (1)	-	-	0.00 (±0.00)	0.00 (±0.00)	-	-	0.00 (±0.00)	0.00 (±0.00)	**20.10 (±0.14)**	19	1.00 (±0.00)	3.00 (±0.00)	**20.10 (±0.14)**	19	1.00 (±0.00)	3.00 (±0.00)	49.87 (±5.48)	29	1.00 (±0.00)	**4.67 (±0.40)**
TCAS-2 (1)	441.83 (±57.70)	120	0.70 (±0.23)	2.13 (±0.73)	335.93 (±58.24)	16	1.57 (±0.33)	5.40 (±1.29)	**170.07 (±0.32)**	168	1.00 (±0.00)	9.00 (±0.00)	**170.07 (±0.32)**	168	1.00 (±0.00)	9.00 (±0.00)	186.87 (±12.30)	92	1.23 (±0.18)	**13.83 (±0.37)**
TCAS-3 (1)	588.43 (±15.18)	392	0.10 (±0.11)	38.63 (±1.96)	531.87 (±30.90)	295	0.67 (±0.27)	55.53 (±2.18)	**230.37 (±0.52)**	228	2.00 (±0.00)	19.00 (±0.00)	**230.37 (±0.52)**	228	2.00 (±0.00)	19.00 (±0.00)	263.20 (±3.61)	236	2.00 (±0.00)	57.43 (±1.54)
TCAS-4 (1)	28.47 (±10.42)	2	1.00 (±0.00)	18.27 (±1.06)	**9.27 (±3.34)**	1	1.00 (±0.00)	24.10 (±1.24)	-	-	0.00 (±0.00)	3.00 (±0.00)	-	-	0.00 (±0.00)	3.00 (±0.00)	43.70 (±14.01)	3	1.00 (±0.00)	22.53 (±1.01)
TCAS-5 (1)	184.93 (±46.66)	24	2.00 (±0.00)	31.97 (±1.06)	79.77 (±21.40)	3	2.00 (±0.00)	40.00 (±1.73)	173.40 (±0.34)	171	2.00 (±0.00)	23.00 (±0.00)	173.40 (±0.34)	171	2.00 (±0.00)	23.00 (±0.00)	94.60 (±30.72)	1	2.00 (±0.00)	**49.83 (±1.27)**
TCAS-6 (1)	233.63 (±54.48)	4	0.97 (±0.06)	4.13 (±0.83)	114.63 (±37.12)	15	1.00 (±0.00)	9.50 (±0.98)	**4.73 (±0.16)**	4	1.00 (±0.00)	6.00 (±0.00)	**4.73 (±0.16)**	4	1.00 (±0.00)	6.00 (±0.00)	7.57 (±0.26)	6	1.00 (±0.00)	10.37 (±0.70)
TCAS-7 (1)	-	-	0.00 (±0.00)	0.00 (±0.00)	581.60 (±28.73)	164	0.07 (±0.09)	0.27 (±0.36)	73.50 (±0.20)	72	2.00 (±0.00)	6.00 (±0.00)	73.50 (±0.20)	72	2.00 (±0.00)	6.00 (±0.00)	**71.70 (±1.71)**	62	2.00 (±0.00)	**8.93 (±0.39)**
TCAS-8 (1)	-	-	0.00 (±0.00)	0.00 (±0.00)	-	-	0.00 (±0.00)	0.00 (±0.00)	78.73 (±1.24)	75	2.00 (±0.00)	6.00 (±0.00)	78.73 (±1.24)	75	2.00 (±0.00)	6.00 (±0.00)	**65.33 (±0.75)**	61	2.00 (±0.00)	**8.77 (±0.49)**
TCAS-9 (1)	221.73 (±48.83)	10	1.00 (±0.00)	6.13 (±0.85)	**109.73 (±28.35)**	4	1.00 (±0.00)	9.37 (±0.44)	148.57 (±1.76)	143	1.00 (±0.00)	15.00 (±0.00)	148.57 (±1.76)	143	1.00 (±0.00)	15.00 (±0.00)	185.53 (±18.42)	39	1.00 (±0.00)	**22.37 (±0.89)**
TCAS-10 (2)	173.47 (±46.27)	1	1.93 (±0.09)	12.27 (±1.69)	100.53 (±25.20)	3	2.00 (±0.00)	18.07 (±1.07)	**4.87 (±0.52)**	4	2.00 (±0.00)	12.00 (±0.00)	**4.87 (±0.52)**	4	2.00 (±0.00)	12.00 (±0.00)	7.63 (±0.22)	7	2.00 (±0.00)	**21.30 (±0.82)**
Math-10 (1)	221.13 (±56.26)	10	64.50 (±15.98)	15.50 (±2.35)	109.53 (±18.08)	13	**172.37 (±26.21)**	24.03 (±1.33)	**2.97 (±0.17)**	2	7.00 (±0.00)	10.00 (±0.00)	**2.97 (±0.17)**	2	7.00 (±0.00)	10.00 (±0.00)	3.87 (±0.20)	3	44.33 (±5.47)	**32.00 (±1.39)**
Math-46 (1)	377.87 (±63.43)	77	0.80 (±0.14)	36.33 (±1.07)	270.07 (±50.22)	8	1.00 (±0.00)	**43.03 (±0.78)**	**118.93 (±0.90)**	116	1.00 (±0.00)	5.60 (±0.18)	**118.93 (±0.90)**	116	1.00 (±0.00)	8.00 (±0.00)	122.00 (±8.34)	49	1.00 (±0.00)	38.17 (±0.82)
Math-60 (7)	6.93 (±0.63)	4	219.17 (±5.26)	92.90 (±1.64)	5.90 (±0.47)	4	**483.03 (±9.52)**	**138.10 (±3.56)**	**2.27 (±0.16)**	2	2.00 (±0.00)	3.00 (±0.00)	**2.27 (±0.16)**	2	2.00 (±0.00)	3.00 (±0.00)	4.77 (±0.15)	4	234.23 (±5.63)	94.20 (±2.67)
Time-1 (14)	5.17 (±1.20)	2	123.30 (±5.86)	170.63 (±3.43)	3.30 (±0.60)	2	**221.00** (±7.84)	**249.10 (±4.29)**	5.23 (±0.18)	4	33.00 (±0.00)	32.00 (±0.00)	5.23 (±0.18)	4	33.00 (±0.00)	32.00 (±0.00)	3.80 (±0.69)	1	189.73 (±11.94)	225.33 (±5.62)
CLI1-2 (13)	-	-	0.00 (±0.00)	159.53 (±4.05)	-	-	0.00 (±0.00)	**202.17 (±3.48)**	-	-	0.00 (±0.00)	4.00 (±0.00)	-	-	0.00 (±0.00)	4.00 (±0.00)	-	-	0.00 (±0.00)	169.40 (±4.07)
CLI2-3 (13)	10.83 (±3.33)	2	82.30 (±3.98)	176.83 (±3.62)	**4.83 (±1.29)**	1	**161.60 (±6.62)**	242.53 (±6.92)	-	-	0.00 (±0.00)	37.00 (±0.00)	-	-	0.00 (±0.00)	37.00 (±0.00)	13.27 (±3.62)	2	84.63 (±4.24)	242.70 (±3.80)
CLI3-4 (8)	7.43 (±1.60)	1	96.73 (±4.54)	279.13 (±4.51)	7.20 (±1.85)	2	97.87 (±4.02)	467.27 (±5.05)	**4.07 (±0.36)**	3	1.00 (±0.00)	12.00 (±0.00)	**4.07 (±0.36)**	3	1.00 (±0.00)	12.00 (±0.00)	8.93 (±2.13)	2	**113.33 (±4.80)**	471.50 (±8.93)
CLI4-5 (13)	589.57 (±16.05)	358	0.07 (±0.09)	219.30 (±3.74)	-	-	0.00 (±0.00)	**274.43 (±4.22)**	-	-	0.00 (±0.00)	4.00 (±0.00)	-	-	0.00 (±0.00)	4.00 (±0.00)	**551.97 (±45.65)**	125	**0.13 (±0.12)**	235.17 (±5.73)
CLI5-6 (21)	**4.13 (±1.04)**	1	143.87 (±4.99)	182.00 (±5.54)	**3.43 (±0.72)**	1	**277.17 (±6.81)**	**272.17 (±7.32)**	-	-	0.00 (±0.00)	5.00 (±0.00)	-	-	0.00 (±0.00)	5.00 (±0.00)	6.17 (±1.31)	2	177.80 (±4.39)	214.47 (±6.38)

HyDiff classifies all subjects correctly.

Components do benefit from each other.

Worst-Case Complexity Analysis

Hexcolor



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: **changes** in the **program**
- in side-channel analysis: **changes** in the **input**

secret **= change**(secret1**,** secret2)

Side-Channel Analysis

Benchmark	Version	Differential Fuzzing (DF)			Themis		
		$\bar{\delta}$	δ_{max}	$\bar{t} : \delta > 0$	$\epsilon = 64$	$\epsilon = 0$	Time (s)
Spring-Security	Safe	1.00 (±0.00)	1	4.77 (±1.07)	✓	✓	1.70
Spring-Security	Unsafe	149.00 (±0.00)	149	4.17 (±0.90)	✓	✓	1.09
JDK7-MsgDigest	Safe	1.00 (±0.00)	1	10.77 (±2.12)	✓	✓	1.27
JDK6-MsgDigest	Unsafe	140.03 (±20.39)	263	3.20 (±0.81)	✓	✓	1.33
Picketbox	Safe	1.00 (±0.00)	1	16.90 (±3.89)	✓	✗	1.79
Picketbox	Unsafe	363.70 (±562.18)	8,822	5.13 (±1.83)	✓	✓	1.55
Tomcat	Safe	25.07 (±0.36)	26	19.90 (±9.29)	✓	✗	9.93
Tomcat	*Unsafe*	49.00 (±0.36)	50	23.53 (±9.73)	✓	✓	8.64
Jetty	*Safe*	11.77 (±0.60)	15	3.77 (±0.72)	✓	✓	2.50
Jetty	Unsafe	70.87 (±6.12)	105	6.83 (±1.62)	✓	✓	2.07
orientdb	Safe	1.00 (±0.00)	1	16.60 (±5.14)	✓	✗	37.99
orientdb	Unsafe	458.93 (±685.64)	10,776	4.77 (±1.06)	✓	✓	38.09
pac4j	Safe	10.00 (±0.00)	10	1.10 (±0.11)	✓	✗	3.97
pac4j	*Unsafe*	11.00 (±0.00)	11	1.13 (±0.12)	✓	✓	1.85
pac4j	*Unsafe**	39.00 (±0.00)	39	1.10 (±0.11)	-	-	-

DF can find the same vulnerabilities as static analysis

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

Robustness Analysis of Neural Networks

Purpose: stress test proposed technique

- similar to SC analysis: **changes** in the **input**
- similar to regression analysis: **search for output differences**
- idea: allow up to x% changes in the pixels of the input image

a[i][j] = **change**(a[i][j], value);

NN Analysis

Subject (% change)	Differential Fuzzing (DF)				Parallel Differential Fuzzing (PDF)				Differential Dynamic Sym. Exec. (DDSE)				HyDiff			
	$\bar{t}$ +odiff	t_{min}	$\overline{\#odiff}$	$\overline{\#ddiff}$	$\bar{t}$ +odiff	t_{min}	$\overline{\#odiff}$	$\overline{\#ddiff}$	$\bar{t}$ +odiff	t_{min}	$\overline{\#odiff}$	$\overline{\#ddiff}$	$\bar{t}$ +odiff	t_{min}	$\overline{\#odiff}$	$\overline{\#ddiff}$
1	2,725.40 (±341.09)	1,074	0.57 (±0.20)	7.73 (±0.18)	2,928.60 (±289.44)	1,202	1.00 (±0.31)	**12.00 (±0.48)**	296.03 (±1.49)	289	1.00 (±0.00)	1.00 (±0.00)	297.10 (±2.38)	267	**1.20 (±0.14)**	6.10 (±0.11)
2	2,581.47 (±326.21)	1,032	0.93 (±0.28)	7.93 (±0.13)	2.509.20 (±289.37)	1,117	1.23 (±0.33)	**12.63 (±0.48)**	309.77 (±7.04)	293	1.00 (±0.00)	1.00 (±0.00)	**297.93 (±1.29)**	292	1.53 (±0.20)	6.93 (±0.13)
5	2,402.97 (±329.59)	1,189	1.23 (±0.37)	6.47 (±0.18)	2,501.43 (±285.86)	1,429	1.70 (±0.44)	**10.33 (±0.43)**	304.53 (±1.06)	300	1.00 (±0.00)	1.00 (±0.00)	**301.83 (±1.16)**	296	2.07 (±0.29)	6.90 (±0.17)
10	2,155.40 (±343.76)	996	1.57 (±0.34)	8.10 (±0.17)	2,127.70 (±229.21)	1,418	2.20 (±0.33)	**11.23 (±0.40)**	311.90 (±0.74)	308	1.00 (±0.00)	1.00 (±0.00)	311.07 (±1.01)	306	2.37 (±0.31)	7.00 (±0.13)
20	1,695.83 (±228.18)	953	2.70 (±0.37)	9.13 (±0.12)	1,897.67 (±219.97)	1,340	3.30 (±0.49)	**11.57 (±0.50)**	346.87 (±1.98)	339	1.00 (±0.00)	1.00 (±0.00)	**341.83 (±1.27)**	336	3.13 (±0.34)	7.20 (±0.14)
50	1,830.83 (±259.79)	1,220	2.43 (±0.42)	6.33 (±0.21)	1,696.10 (±86.20)	1,423	3.80 (±0.35)	**12.00 (±0.39)**	455.03 (±1.62)	449	1.00 (±0.00)	1.00 (±0.00)	452.63 (±2.06)	434	3.77 (±0.34)	7.27 (±0.16)
100	1,479.17 (±231.25)	960	2.47 (±0.37)	9.37 (±0.20)	1,790.87 (±270.10)	1,109	3.03 (±0.54)	**13.97 (±0.68)**	583.33 (±2.83)	571	1.00 (±0.00)	1.00 (±0.00)	**575.13 (±2.65)**	564	3.10 (±0.35)	7.60 (±0.18)

Shows the limitations of both components.

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

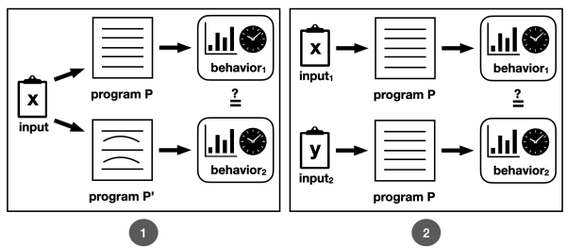
DifFuzz: Differential Fuzzing for Side-Channel Analysis
Shirin Nilizadeh, Yannic Noller*, and Corina S. Păsăreanu (* joint first authors)*
ACM/IEEE International Conference on Software Engineering (ICSE) 2019

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

Differential Software Testing

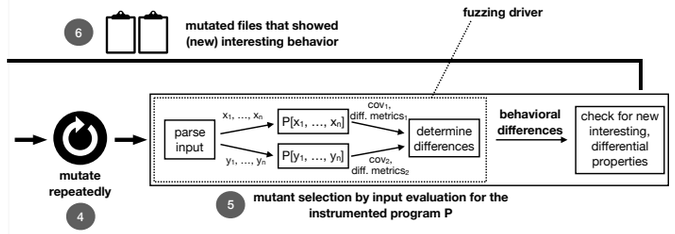


Why combine Fuzzing and Symbolic Execution?

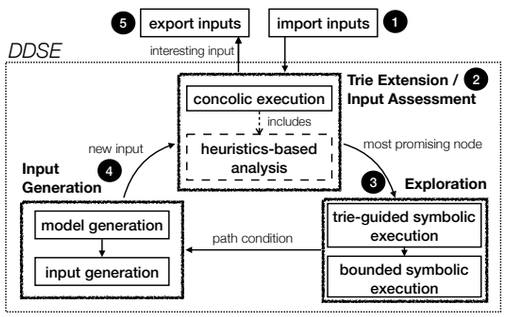
good in finding **shallow** bugs, but **bad** in finding **deep** program paths

input **reasoning ability**, but **path explosion** and **constraint solving**

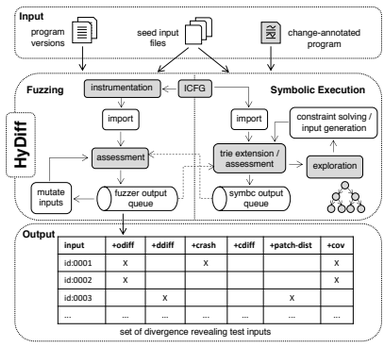
Differential Fuzzing



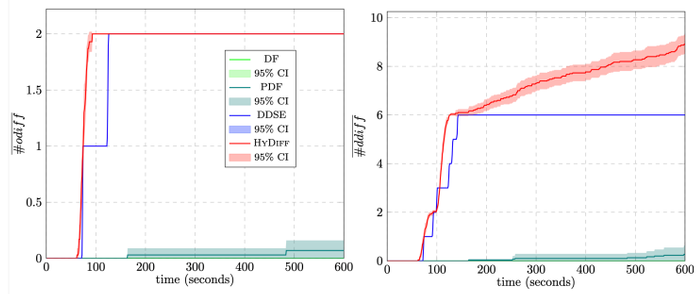
Differential Dynamic SymExe



HyDiff's overview



Regression Analysis *TCAS-7*



Components do benefit from each other.

References

[Antonopoulos2017] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. 2017. Decomposition instead of self-composition for proving the absence of timing channels. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017). ACM, New York, NY, USA, 362-375.

[Barthe2004] G. Barthe, P. R. D'Argenio and T. Rezk, "Secure Information Flow by Self-Composition," Computer Security Foundations Workshop, IEEE(CSFW), Pacific Grove, California, 2004, pp. 100.

[Boyer1975] Boyer, R. S., Elspas, B., & Levitt, K. N. (1975). SELECT - a Formal System for Testing and Debugging Programs by Symbolic Execution. SIGPLAN Not., 10(6), 234–245.

[Brennan2018] Brennan, T., Saha, S., Bultan, T., & Pasareanu, C. S. (2018). Symbolic Path Cost Analysis for Side-channel Detection. Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, 27–37.

[Burnim2009] J. Burnim, S. Juvekar, and K. Sen. 2009. WISE: Automated test generation for worst-case complexity. In 2009 IEEE 31st International Conference on Software Engineering. 463–473.

[Cha2012] Cha, S. K., Avgerinos, T., Rebert, A., & Brumley, D. (2012). Unleashing Mayhem on Binary Code. 2012 IEEE Symposium on Security and Privacy, 380–394.

[Chen2017] Jia Chen, Yu Feng, and Isil Dillig. 2017. Precise Detection of Side-Channel Vulnerabilities using Quantitative Cartesian Hoare Logic. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 875-890.

[Clarke1976] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," in IEEE Transactions on Software Engineering, vol. SE-2, no. 3, pp. 215-222, Sept. 1976.

References (continued)

[Goodfellow2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27 (pp. 2672–2680). Curran Associates, Inc.

[IEEE2017] ISO/IEC/IEEE International Standard - Systems and software engineering - Vocabulary. (2017). ISO/IEC/IEEE 24765:2017(E), 1–541.

[King1976] James C. King. 1976. Symbolic execution and program testing. Commun. ACM 19, 7 (July 1976), 385-394.

[Lemieux2018] Caroline Lemieux, Rohan Padhye, Koushik Sen, and Dawn Song. 2018. PerfFuzz: automatically generating pathological inputs. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018). ACM, New York, NY, USA, 254-265.

[Luckow2017] Kasper Luckow, Rody Kersten, and Corina Pasareanu. 2017. Symbolic Complexity Analysis using Context-preserving Histories. In Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017). 58–68

[Ma2018] Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., & Wang, Y. (2018). DeepMutation: Mutation Testing of Deep Learning Systems. 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), 100–111.

[Miller1990] Barton P. Miller, Louis Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. Commun. ACM 33, 12 (December 1990), 32-44.

References (continued)

[Ognawala2019] Ognawala, S., Kilger, F., & Pretschner, A. (2019). Compositional Fuzzing Aided by Targeted Symbolic Execution. ArXiv Preprint ArXiv:1903.02981

[Orso2008] Orso, A., & Xie, T. (2008). BERT: BEhavioral Regression Testing. Proceedings of the 2008 International Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), 36–42.

[Orso2014] Orso, A., & Rothermel, G. (2014). Software Testing: A Research Travelogue (2000-2014). Proceedings of the on Future of Software Engineering, 117–132.

[Pak2012] Pak, B. S. (2012). Hybrid Fuzz Testing: Discovering Software Bugs via Fuzzing and Symbolic Execution. Carnegie Mellon University.

[Palikareva2016] Hristina Palikareva, Tomasz Kuchta, and Cristian Cadar. 2016. Shadow of a doubt: testing for divergences between software versions. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). ACM, New York, NY, USA, 1181-1192.

[Pasareanu2016] Pasareanu, C. S., Phan, Q.-S., & Malacaria, P. (2016). Multi-run Side-Channel Analysis Using Symbolic Execution and Max-SMT. 2016 IEEE 29th Computer Security Foundations Symposium (CSF), 387–400.

[Petsios2017] Petsios, T., Tang, A., Stolfo, S., Keromytis, A. D., & Jana, S. (2017). NEZHA: Efficient Domain-Independent Differential Testing. 2017 IEEE Symposium on Security and Privacy (SP), 615–632.

[Pei2017] Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). DeepXplore: Automated Whitebox Testing of Deep Learning Systems. Proceedings of the 26th Symposium on Operating Systems Principles, 1–18.

References (continued)

[Person2008] Person, S., Dwyer, M. B., Elbaum, S., & Pasareanu, C. S. (2008). Differential Symbolic Execution. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 226–237.

[Person2011] Suzette Person, Guowei Yang, Neha Rungta, and Sarfraz Khurshid. 2011. Directed incremental symbolic execution. In Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11). ACM, New York, NY, USA, 504-515.

[Stephens2016] Stephens N, Grosen J, Salls C, Dutcher A, Wang R, Corbetta J, Shoshitaishvili Y, Kruegel C, Vigna G. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. InNDSS 2016 Feb (Vol. 16, pp. 1-16).

[Sun2018] Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., & Kroening, D. (2018). Concolic Testing for Deep Neural Networks. Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 109–119.

[Taneja2008] Taneja, K., & Xie, T. (2008). DiffGen: Automated Regression Unit-Test Generation. 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, 407–410.

[Tian2018] Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. Proceedings of the 40th International Conference on Software Engineering, 303–314.

[Yang2012] Yang, G., Pasareanu, C. S., & Khurshid, S. (2012). Memoized Symbolic Execution. Proceedings of the 2012 International Symposium on Software Testing and Analysis, 144–154.

END OF DOCUMENT