Complete Shadow Symbolic Execution with Java PathFinder

Yannic Noller

Hoang Lam Nguyen

Minxing Tang

Timo Kehrer

Lars Grunske

Humboldt-Universität zu Berlin

Regression Testing

```
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  }
```

assertion error for **x=-1**

Regression Testing

```
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  }
```

assertion error
for **x=-1** is **fixed**
(returns 0)

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

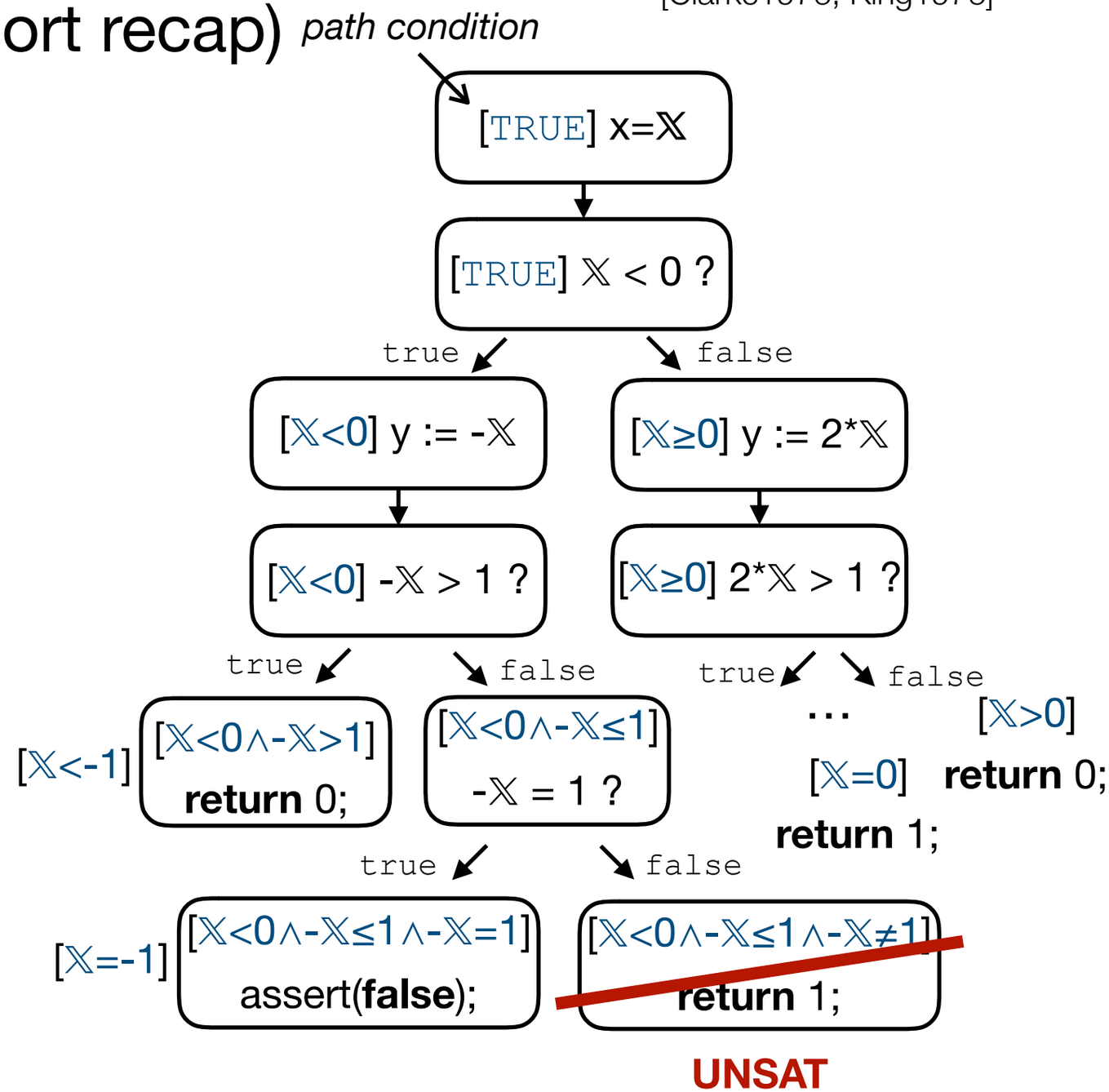
→ **Regression Bug**

Symbolic Execution

[Clarke1976, King1976]

(a short recap)

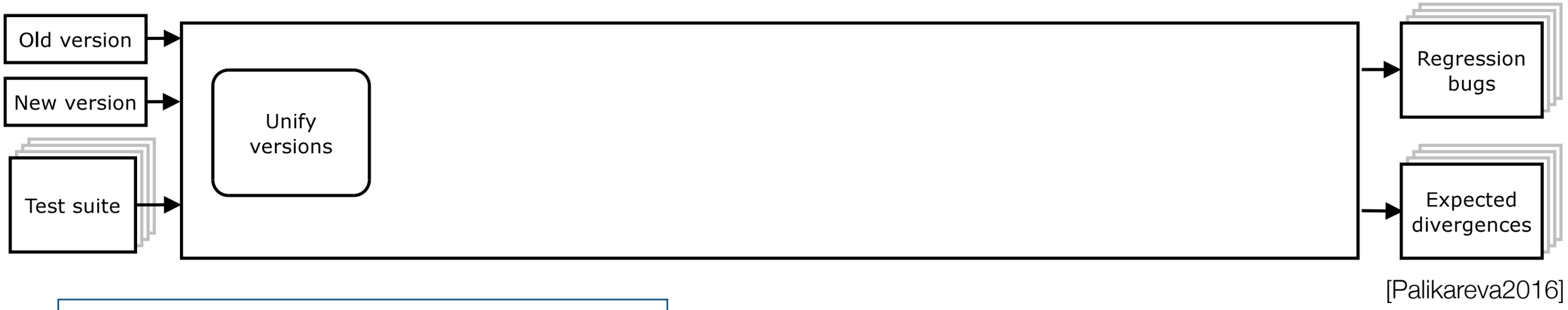
```
1   int foo (int x) {
2       int y;
3       if (x < 0) {
4           y = -x;
5       } else {
6           y = 2 * x;
7       }
8       if (y > 1) {
9           return 0;
10      } else {
11          if (y == 1)
12              assert(false);
13      }
14      return 1;
15  }
```



Shadow Symbolic Execution

(Palikareva, Kuchta, and Cadar; ICSE 2016)

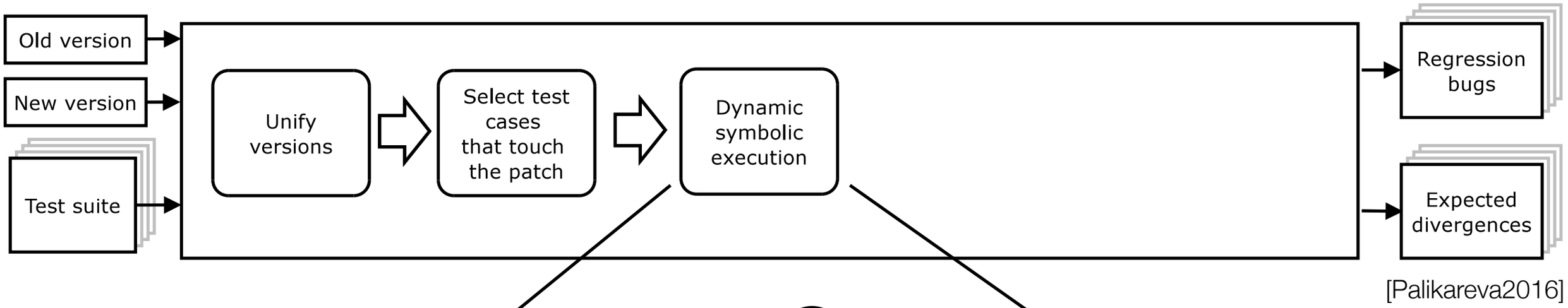
Goal: *generate test cases to expose diverging behavior of two software versions*



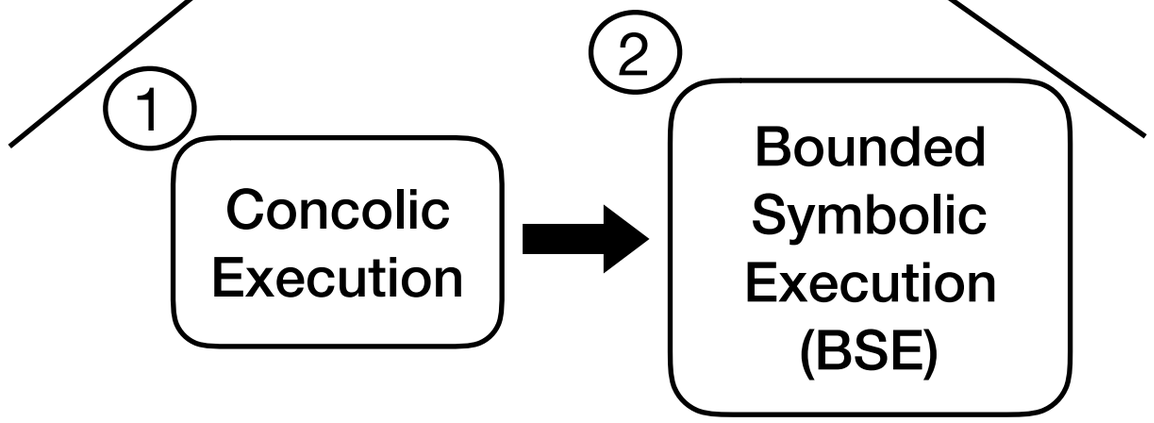
[Palikareva2016]

```
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  }
```

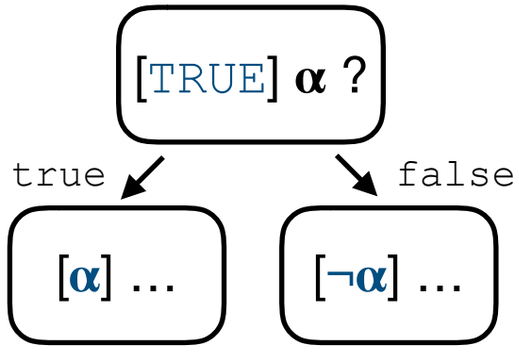
```
1   int foo (int x) {
2       int y;
3       if (x < 0) {
4           y = change(-x, x*x);
5       } else {
6           y = 2 * x;
7       }
8       y = change(y, y + 1);
9       if (y > 1) {
10          return 0;
11      } else {
12          if (y == 1)
13              assert(false);
14      }
15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



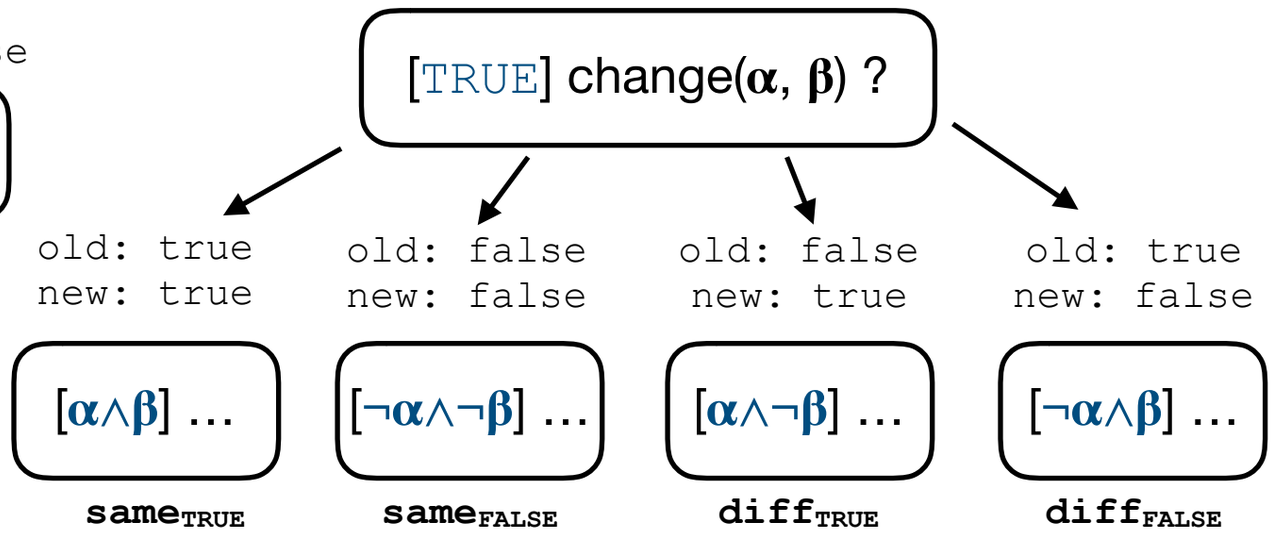
[Palikareva2016]

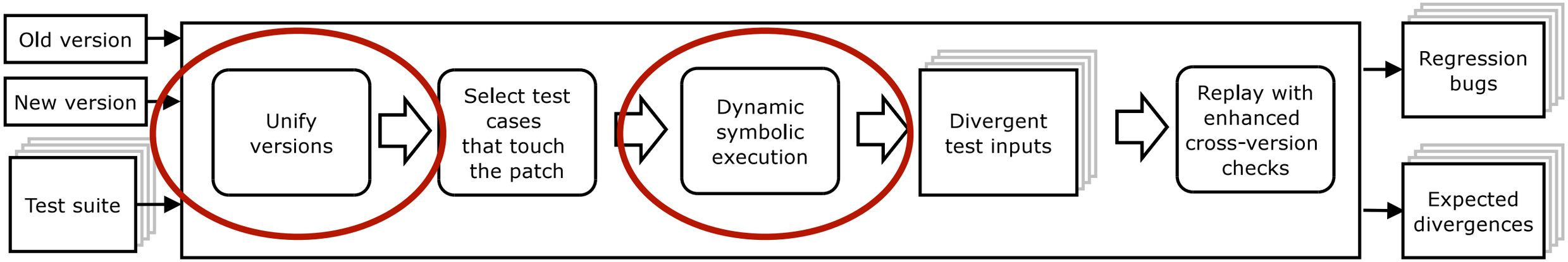


Two-way Forking



Four-way Forking





[Palikareva2016]

ACM SIGSOFT Software Engineering Notes Page 1 October 2017 Volume 42 Number 4

Shadow Symbolic Execution with Java PathFinder

Yannic Noller
Humboldt University of Berlin
yannic.noller@informatik.hu-berlin.de

Hoang Lam Nguyen
Humboldt University of Berlin
nguyenhx@informatik.hu-berlin.de

Minxing Tang
Humboldt University of Berlin
tangminx@informatik.hu-berlin.de

Timo Kehrer
Humboldt University of Berlin
timo.kehrer@informatik.hu-berlin.de

DOI: 10.1145/3149485.3149492
http://doi.acm.org/10.1145/3149485.3149492

ABSTRACT

Regression testing ensures that a software system when it evolves still performs correctly and that the changes introduce no unintended side-effects. However, the creation of regression test cases that show divergent behavior needs a lot of effort. A solution is the idea of *shadow symbolic execution*, originally implemented based on KLEE for programs written in C, which takes a unified version of the old and the new program and performs symbolic execution guided by concrete values to explore the changed behavior. In this work, we apply the idea of shadow symbolic execution

bolic execution-based technique, which they refer to as *shadow symbolic execution*. Their technique is designed to generate test inputs that cover new program behaviors introduced by a patch. Shadow symbolic execution works by executing both the old (buggy) and new (patched) version in the same symbolic execution instance, with the old version *shadowing* the new one. Therefore, it is necessary to manually *merge* both programs into a change-annotated, unified version. Based on such a unified version, the technique detects divergences along the execution path of an input that exercises the patch. Their tool SHADOW, which we refer

(Noller et al.; JPF 2017)

Limitations (1)

Deeper divergences might be missed in the BSE phase due to narrow path conditions based on concrete inputs.

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

x=-1 (fully covers the changes)

path condition up to line 9:
[$\mathbb{X} < 0$]

to reach assertion error BSE
needs to follow **`false`** branch
with condition: [$\mathbb{X}^2 + 1 \leq 1$]

only possible for **x=0**, but [$\mathbb{X} < 0$]

Limitations (2)

The initial input has to cover not only changed locations, but also potential divergence points.

```
1  int bar (int x, int y) {
2      int z = change(x, y);
3      if ((x+y) == 5) {
4          if (z == -100)
5              assert(false);
6      }
7      return 0;
8  }
```

divergence only possible in line 4

collect change and then reach **divergence** (point)

all inputs with **x+y ≠ 5** would miss the divergence

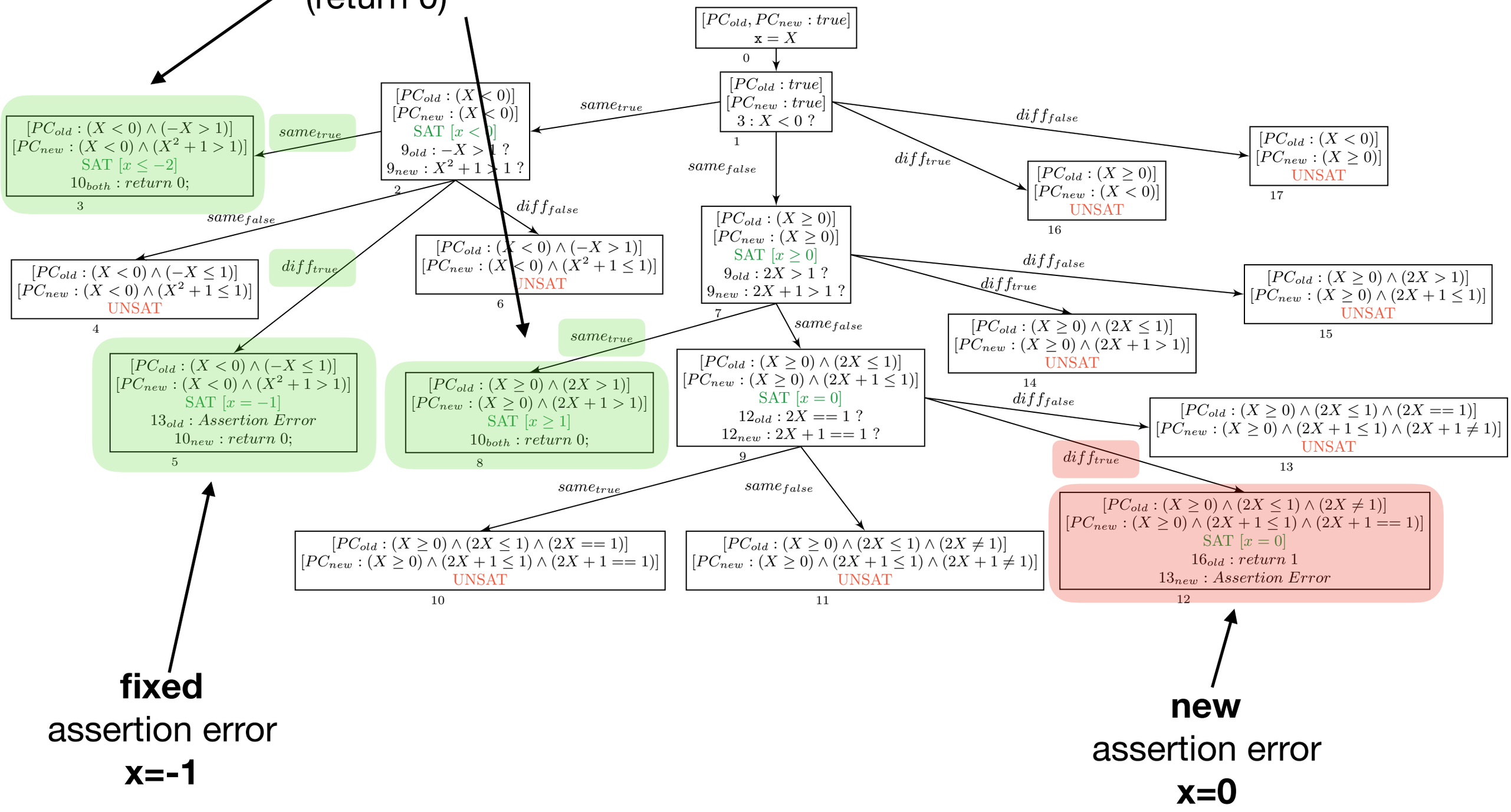
Shadow Symbolic Execution **strongly depends** on **concrete** inputs

Complete Shadow Symbolic Execution

1. combines bounded symbolic execution with four-way forking
2. full exploration of $same_{TRUE/FALSE}$ paths, as long as they can or have reached a change
3. exploration of $diff_{TRUE/FALSE}$ paths only for the new version

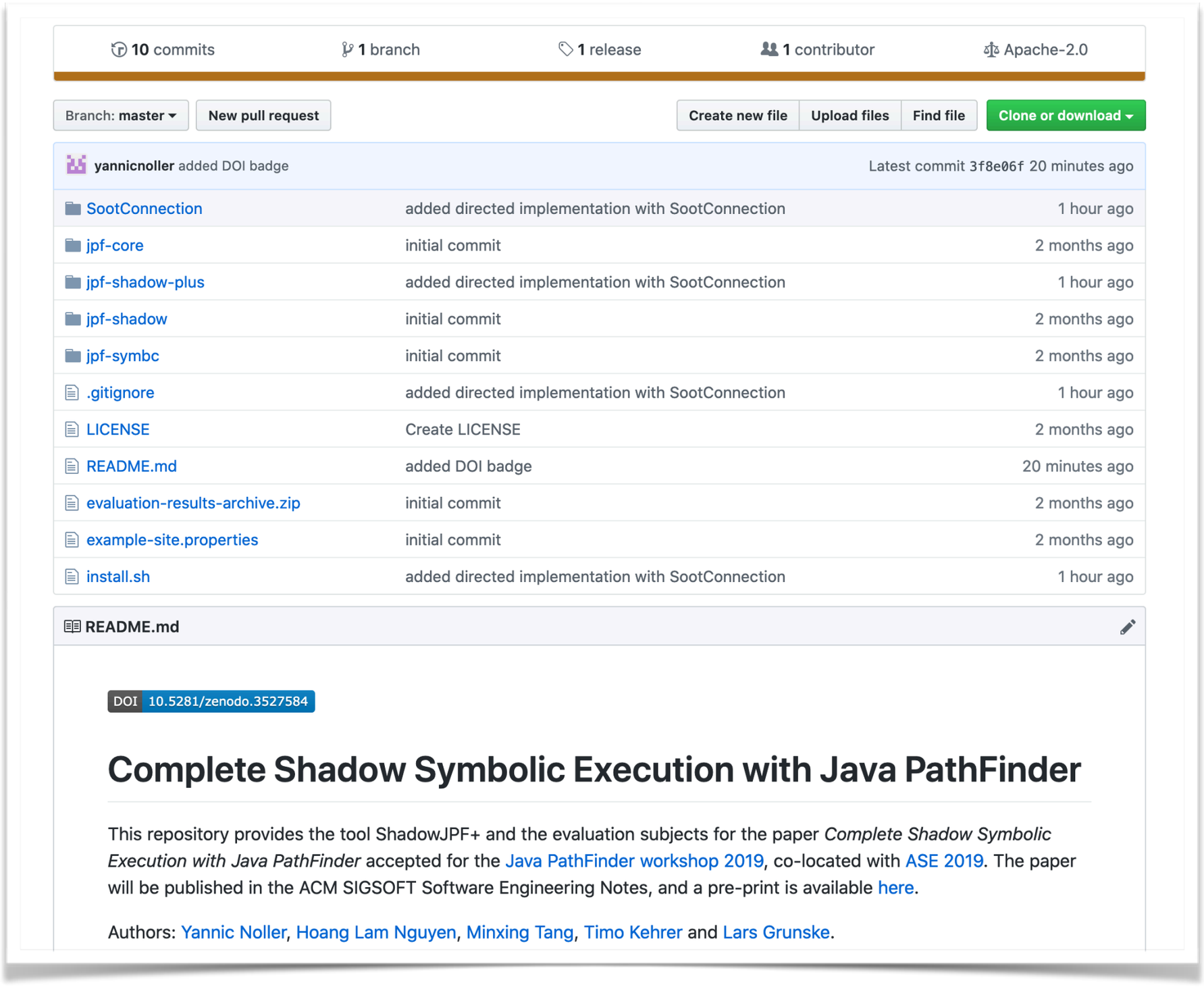
```
1   int foo (int x) {
2       int y;
3       if (x < 0) {
4           y = change(-x, x*x);
5       } else {
6           y = 2 * x;
7       }
8       y = change(y, y + 1);
9       if (y > 1) {
10          return 0;
11      } else {
12          if (y == 1)
13              assert(false);
14      }
15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

same behavior
for **x ≥ 1** and **x ≤ -2**
(return 0)



fixed
assertion error
x=-1

new
assertion error
x=0



10 commits | 1 branch | 1 release | 1 contributor | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

yannicnoller added DOI badge — Latest commit 3f8e06f 20 minutes ago

SootConnection	added directed implementation with SootConnection	1 hour ago
jpf-core	initial commit	2 months ago
jpf-shadow-plus	added directed implementation with SootConnection	1 hour ago
jpf-shadow	initial commit	2 months ago
jpf-symbc	initial commit	2 months ago
.gitignore	added directed implementation with SootConnection	1 hour ago
LICENSE	Create LICENSE	2 months ago
README.md	added DOI badge	20 minutes ago
evaluation-results-archive.zip	initial commit	2 months ago
example-site.properties	initial commit	2 months ago
install.sh	added directed implementation with SootConnection	1 hour ago

README.md

DOI 10.5281/zenodo.3527584

Complete Shadow Symbolic Execution with Java PathFinder

This repository provides the tool ShadowJPF+ and the evaluation subjects for the paper *Complete Shadow Symbolic Execution with Java PathFinder* accepted for the Java PathFinder workshop 2019, co-located with ASE 2019. The paper will be published in the ACM SIGSOFT Software Engineering Notes, and a pre-print is available here.

Authors: Yannic Noller, Hoang Lam Nguyen, Minxing Tang, Timo Kehrer and Lars Grunske.

https://github.com/hub-se/jpf-shadow-plus

Experiments

comparison between **ShadowJPF+** with **ShadowJPF**

RQ1: Effectiveness
Can ShadowJPF+ reveal more divergent behaviors than ShadowJPF?

RQ2: Performance
How does ShadowJPF+ compare to ShadowJPF in terms of performance?

RQ3: Real Regression Bugs
Can ShadowJPF+ expose real-world regression bugs?

Subject	LOC
Rational.abs	30
Rational.gcd	40
Rational.simplify	51
WBS.update	234
WBS.launch	242

generated **79 mutants** with **Major** [Just2011]

RQ1: Effectiveness

Subject	Type	Time [s]		# States		# Paths (diff)	
		SJ	*SJ+*	*SJ*	*SJ+*	*SJ*	*SJ+*
Rational.abs_1	ROR	<1	<1	21	32	1	1
Rational.abs_2	ROR	<1	<1	21	32	1	1
Rational.abs_3	ROR	<1	<1	13	20	1	1
Rational.abs_4	ORU	<1	<1	5	6	0	0
Rational.abs_5	ORU	<1	<1	5	6	0	0
Rational.gcd_1	ROR	<1	<1	42	220	0	0
Rational.gcd_2	ROR	<1	<1	23	48	2	4
Rational.gcd_3	ROR	<1	<1	40	234	3	3
Rational.gcd_4	STD	<1	<1	43	223	3	3
Rational.gcd_5	ROR	<1	<1	27	174	1	2
Rational.gcd_6	ROR	<1	<1	27	610	1	2
Rational.gcd_7	ROR	<1	<1	87	692	1	16
Rational.gcd_8	STD	inf	inf	-	-	-	-
Rational.gcd_9	ROR	<1	<1	45	434	0	0
Rational.gcd_10	ROR	<1	<1	57	626	3	48
Rational.gcd_11	ROR	<1	<1	15	42	1	2
Rational.gcd_12	ROR	<1	<1	104	308	3	6
Rational.gcd_13	ROR	<1	<1	104	642	3	14
Rational.gcd_14	ROR	<1	<1	43	236	1	6
Rational.gcd_15	AOR	<1	<1	43	178	4	10
Rational.gcd_16	AOR	<1	<1	39	170	4	10
Rational.gcd_17	AOR	<1	1	60	342	8	36
Rational.gcd_18	STD	<1	<1	37	166	2	6
Rational.gcd_19	AOR	<1	4	49	198	5	18
Rational.gcd_20	AOR	<1	<1	49	198	5	18
Rational.gcd_21	AOR	1	94	83	386	9	34
Rational.gcd_22	STD	<1	<1	49	198	5	18
Rational.simplify_1	ROR	<1	<1	55	284	4	6
Rational.simplify_2	ROR	<1	<1	63	370	3	3
Rational.simplify_3	ROR	<1	<1	71	252	4	6
Rational.simplify_4	ORU	<1	<1	28	280	2	8
Rational.simplify_5	ROR	<1	<1	42	364	0	1
Rational.simplify_6	ROR	<1	<1	31	96	3	7
Rational.simplify_7	ROR	<1	<1	63	366	4	4
Rational.simplify_8	STD	<1	<1	19	355	1	4
Rational.simplify_9	ROR	<1	<1	31	222	1	3
Rational.simplify_10	ROR	<1	<1	73	770	1	3
Rational.simplify_11	ROR	<1	<1	67	588	1	17
Rational.simplify_12	STD	inf	inf	-	-	-	-
Rational.simplify_13	ROR	<1	1	45	578	0	1
Rational.simplify_14	ROR	<1	<1	61	898	3	49
Rational.simplify_15	ROR	<1	<1	15	74	1	3
Rational.simplify_16	ROR	<1	<1	104	388	3	7
Rational.simplify_17	ROR	<1	<1	104	674	3	15
Rational.simplify_18	ROR	<1	<1	34	280	1	7
Rational.simplify_19	AOR	<1	<1	47	274	4	11
Rational.simplify_20	AOR	<1	<1	43	266	4	11
Rational.simplify_21	AOR	<1	1	72	550	8	37
Rational.simplify_22	STD	<1	<1	37	246	2	7
Rational.simplify_23	AOR	<1	6	49	230	5	19
Rational.simplify_24	AOR	<1	<1	49	230	5	19
Rational.simplify_25	AOR	<1	95	83	418	9	35
Rational.simplify_26	STD	<1	<1	49	230	5	19
Rational.simplify_27	AOR	<1	<1	29	338	0	1
Rational.simplify_2_16	ROR^2	<1	<1	138	420	6	9
Rational.simplify_2_27	ROR,AOR	<1	<1	63	370	3	3
Rational.simplify_3_11	ROR^2	<1	<1	108	368	3	12
Rational.simplify_16_27	ROR,AOR	<1	<1	104	388	3	7
Rational.simplify_2_16_27	ROR^2, AOR	<1	<1	138	420	6	9

Subject	Type	Time [s]		# States		# Paths (diff)	
		SJ	*SJ+*	*SJ*	*SJ+*	*SJ*	*SJ+*
WBS.update_1	ROR^8	<1	1	70	880	2	24
WBS.update_2	ROR^8	<1	<1	73	428	2	12
WBS.update_3	ROR^7, AOR	<1	<1	51	554	2	24
WBS.update_4	ROR^6, AOR, STD	<1	<1	97	618	4	18
WBS.update_5	ROR^7, AOR	<1	<1	109	266	6	12
WBS.update_6	ROR^8	<1	<1	135	632	6	24
WBS.update_7	ROR^6, AOR, STD	<1	<1	123	618	6	28
WBS.update_8	ROR^5, AOR^2, STD	<1	<1	147	232	8	8
WBS.update_9	ROR^5, AOR^2, STD	<1	<1	89	576	4	12
WBS.update_10	ROR^7, AOR	<1	<1	118	914	4	7
WBS.launch_1	ROR^8	4	121	11724	281080	576	13824
WBS.launch_2	ROR^8	<1	2	1083	12944	36	432
WBS.launch_3	ROR^7, AOR	7	120	20701	248354	1152	13824
WBS.launch_4	ROR^6, AOR, STD	3	47	10208	111876	628	5472
WBS.launch_5	ROR^7, AOR	<1	1	1717	3506	111	222
WBS.launch_6	ROR^8	11	76	32508	195176	1600	9600
WBS.launch_7	ROR^6, AOR, STD	7	146	22414	313930	1152	16128
WBS.launch_8	ROR^5, AOR^2, STD	2	14	7313	15232	512	896
WBS.launch_9	ROR^5, AOR^2, STD	3	56	7585	143819	745	7109
WBS.launch_10	ROR^7, AOR	30	193	48460	497118	2404	15204

RQ2: Performance

Subject	Type	Time [s]		# States		# Paths (diff)	
		SJ	*SJ+*	*SJ*	*SJ+*	*SJ*	*SJ+*
Rational.abs_1	ROR	<1	<1	21	32	1	1
Rational.abs_2	ROR	<1	<1	21	32	1	1
Rational.abs_3	ROR	<1	<1	13	20	1	1
Rational.abs_4	ORU	<1	<1	5	6	0	0
Rational.abs_5	ORU	<1	<1	5	6	0	0
Rational.gcd_1	ROR	<1	<1	42	220	0	0
Rational.gcd_2	ROR	<1	<1	23	48	2	4
Rational.gcd_3	ROR	<1	<1	40	234	3	3
Rational.gcd_4	STD	<1	<1	43	223	3	3
Rational.gcd_5	ROR	<1	<1	27	174	1	2
Rational.gcd_6	ROR	<1	<1	27	610	1	2
Rational.gcd_7	ROR	<1	<1	87	692	1	16
Rational.gcd_8	STD	inf	inf	-	-	-	-
Rational.gcd_9	ROR	<1	<1	45	434	0	0
Rational.gcd_10	ROR	<1	<1	57	626	3	48
Rational.gcd_11	ROR	<1	<1	15	42	1	2
Rational.gcd_12	ROR	<1	<1	104	308	3	6
Rational.gcd_13	ROR	<1	<1	104	642	3	14
Rational.gcd_14	ROR	<1	<1	43	236	1	6
Rational.gcd_15	AOR	<1	<1	43	178	4	10
Rational.gcd_16	AOR	<1	<1	39	170	4	10
Rational.gcd_17	AOR	<1	1	60	342	8	36
Rational.gcd_18	STD	<1	<1	37	166	2	6
Rational.gcd_19	AOR	<1	4	49	198	5	18
Rational.gcd_20	AOR	<1	<1	49	198	5	18
Rational.gcd_21	AOR	1	94	83	386	9	34
Rational.gcd_22	STD	<1	<1	49	198	5	18
Rational.simplify_1	ROR	<1	<1	55	284	4	6
Rational.simplify_2	ROR	<1	<1	63	370	3	3
Rational.simplify_3	ROR	<1	<1	71	252	4	6
Rational.simplify_4	ORU	<1	<1	28	280	2	8
Rational.simplify_5	ROR	<1	<1	42	364	0	1
Rational.simplify_6	ROR	<1	<1	31	96	3	7
Rational.simplify_7	ROR	<1	<1	63	366	4	4
Rational.simplify_8	STD	<1	<1	19	355	1	4
Rational.simplify_9	ROR	<1	<1	31	222	1	3
Rational.simplify_10	ROR	<1	<1	73	770	1	3
Rational.simplify_11	ROR	<1	<1	67	588	1	17
Rational.simplify_12	STD	inf	inf	-	-	-	-
Rational.simplify_13	ROR	<1	1	45	578	0	1
Rational.simplify_14	ROR	<1	<1	61	898	3	49
Rational.simplify_15	ROR	<1	<1	15	74	1	3
Rational.simplify_16	ROR	<1	<1	104	388	3	7
Rational.simplify_17	ROR	<1	<1	104	674	3	15
Rational.simplify_18	ROR	<1	<1	34	280	1	7
Rational.simplify_19	AOR	<1	<1	47	274	4	11
Rational.simplify_20	AOR	<1	<1	43	266	4	11
Rational.simplify_21	AOR	<1	1	72	550	8	37
Rational.simplify_22	STD	<1	<1	37	246	2	7
Rational.simplify_23	AOR	<1	6	49	230	5	19
Rational.simplify_24	AOR	<1	<1	49	230	5	19
Rational.simplify_25	AOR	<1	95	83	418	9	35
Rational.simplify_26	STD	<1	<1	49	230	5	19
Rational.simplify_27	AOR	<1	<1	29	338	0	1
Rational.simplify_2_16	ROR^2	<1	<1	138	420	6	9
Rational.simplify_2_27	ROR,AOR	<1	<1	63	370	3	3
Rational.simplify_3_11	ROR^2	<1	<1	108	368	3	12
Rational.simplify_16_27	ROR,AOR	<1	<1	104	388	3	7
Rational.simplify_2_16_27	ROR^2, AOR	<1	<1	138	420	6	9

Subject	Type	Time [s]		# States		# Paths (diff)	
		SJ	*SJ+*	*SJ*	*SJ+*	*SJ*	*SJ+*
WBS.update_1	ROR^8	<1	1	70	880	2	24
WBS.update_2	ROR^8	<1	<1	73	428	2	12
WBS.update_3	ROR^7, AOR	<1	<1	51	554	2	24
WBS.update_4	ROR^6, AOR, STD	<1	<1	97	618	4	18
WBS.update_5	ROR^7, AOR	<1	<1	109	266	6	12
WBS.update_6	ROR^8	<1	<1	135	632	6	24
WBS.update_7	ROR^6, AOR, STD	<1	<1	123	618	6	28
WBS.update_8	ROR^5, AOR^2, STD	<1	<1	147	232	8	8
WBS.update_9	ROR^5, AOR^2, STD	<1	<1	89	576	4	12
WBS.update_10	ROR^7, AOR	<1	<1	118	914	4	7
WBS.launch_1	ROR^8	4	121	11724	281080	576	13824
WBS.launch_2	ROR^8	<1	2	1083	12944	36	432
WBS.launch_3	ROR^7, AOR	7	120	20701	248354	1152	13824
WBS.launch_4	ROR^6, AOR, STD	3	47	10208	111876	628	5472
WBS.launch_5	ROR^7, AOR	<1	1	1717	3506	111	222
WBS.launch_6	ROR^8	11	76	32508	195176	1600	9600
WBS.launch_7	ROR^6, AOR, STD	7	146	22414	313930	1152	16128
WBS.launch_8	ROR^5, AOR^2, STD	2	14	7313	15232	512	896
WBS.launch_9	ROR^5, AOR^2, STD	3	56	7585	143819	745	7109
WBS.launch_10	ROR^7, AOR	30	193	48460	497118	2404	15204

Shadow Symbolic Execution:
+ **scalability**
- strongly **depends** on **concrete** inputs

+

Complete Shadow Symbolic Execution:
+ **no dependence** on concrete inputs
- **scalability issue**

Complete Shadow Symbolic Execution with Java PathFinder

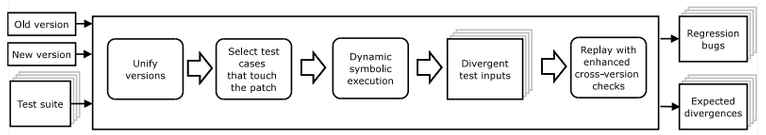
Regression Testing

```
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 }
```

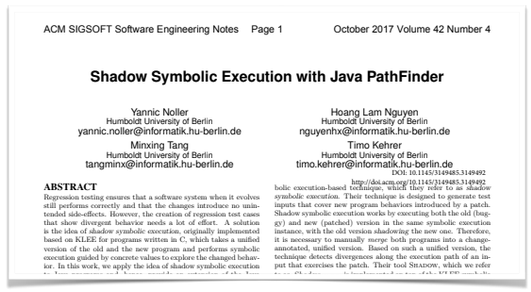
assertion error
for **x=-1** is **fixed**
(returns 0)

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

→ **Regression Bug**



[Palikareva2016]



ACM SIGSOFT Software Engineering Notes Page 1 October 2017 Volume 42 Number 4

Shadow Symbolic Execution with Java PathFinder

Yannic Noller
Humboldt University of Berlin
yannic.noller@informatik.hu-berlin.de

Hoang Lam Nguyen
Humboldt University of Berlin
nguyenhx@informatik.hu-berlin.de

Minxing Tang
Humboldt University of Berlin
tangminx@informatik.hu-berlin.de

Timo Kehrer
Humboldt University of Berlin
timo.kehrer@informatik.hu-berlin.de

DOI: 10.1145/3149485.3149492
http://doi.acm.org/10.1145/3149485.3149492

ABSTRACT

Regression testing ensures that a software system when it evolves still performs correctly and that the changes introduce no unintended side-effects. However, the creation of regression test cases that show divergent behavior needs a lot of effort. A solution is the idea of shadow symbolic execution, originally implemented based on KLEE for programs written in C, which takes a unified version of the old and the new program and performs symbolic execution guided by concrete values to explore the changed behavior. In this work, we apply the idea of shadow symbolic execution

bolic execution-based technique, which they refer to as shadow symbolic execution. Their technique is designed to generate test inputs that cover new program behaviors introduced by a patch. Shadow symbolic execution works by executing both the old (buggy) and new (patched) version in the same symbolic execution instance, with the old version shadowing the new one. Therefore, it is necessary to manually *merge* both programs into a change-annotated, unified version. Based on such a unified version, the technique detects divergences along the execution path of an input that exercises the patch. Their tool SHADOW, which we refer

(Noller et al.; JPF 2017)

Shadow Symbolic Execution **strongly depends** on **concrete** inputs

Complete Shadow Symbolic Execution

① combines bounded symbolic execution with four-way forking

② exploration of $diff_{TRUE/FALSE}$ paths only for the new version

③ full exploration of $same_{TRUE/FALSE}$ paths, as long as they can or have reached a change

Shadow Symbolic Execution:
+ **scalability**
- strongly **depends** on **concrete** inputs

+

Complete Shadow Symbolic Execution:
+ **no dependence** on concrete inputs
- **scalability issue**

`git clone https://github.com/hub-se/jpf-shadow-plus.git`

References

[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. DOI: https://doi.org/10.1109/TSE.1976.233817

[Just2011] Rene Just, Franz Schweiggert, and Gregory M. Kapfhammer. 2011. MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11). IEEE Computer Society, Washington, DC, USA, 612-615. DOI: http://dx.doi.org/10.1109/ASE.2011.6100138

[King1976] James C. King. 1976. Symbolic execution and program testing. Commun. ACM 19, 7 (July 1976), 385-394. DOI: http://dx.doi.org/10.1145/360248.360252

[Noller2018] Yannic Noller, Hoang Lam Nguyen, Minxing Tang, and Timo Kehrer. 2018. Shadow Symbolic Execution with Java PathFinder. SIGSOFT Softw. Eng. Notes 42, 4 (January 2018), 1-5. DOI: https://doi.org/10.1145/3149485.3149492

[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. DOI: https://doi.org/10.1145/2884781.2884845