Shadow of a Doubt: Testing for Divergences Between Software Versions

Hristina Palikareva Tomasz Kuchta Cristian Cadar

Imperial College London

SOFTWARE RELIABILITY GROUP

ICSE'16, 20th May 2016

This work is supported by EPSRC and Microsoft Research

Motivation

- Software patches
  - Frequent, at the core of software evolution
  - New features, bug fixes, better performance, usability
  - Poorly tested in practice
  - May introduce bugs

Motivation: an example

Old

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

Motivation: an example

Old

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

New

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

- Test cases: x = 0, x = 100, x = 101

Motivation: an example

Old

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

New

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

- Test cases: x = 0, x = 100, x = 101, 100% code coverage

Motivation: an example

Old

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

New

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

- Test cases: x = 0, x = 100, x = 101, 100% code coverage
- Only 50% new behaviour coverage
- Code coverage not sufficient!

Contributions

- Shadow symbolic execution technique
  - Focuses on the new behaviours of the patch
- Technique for unifying two versions of a program
  - Execute in a single symbolic execution instance
- A patch testing approach
  - Shadow symbolic execution
  - Enhanced cross-version checks

Symbolic execution

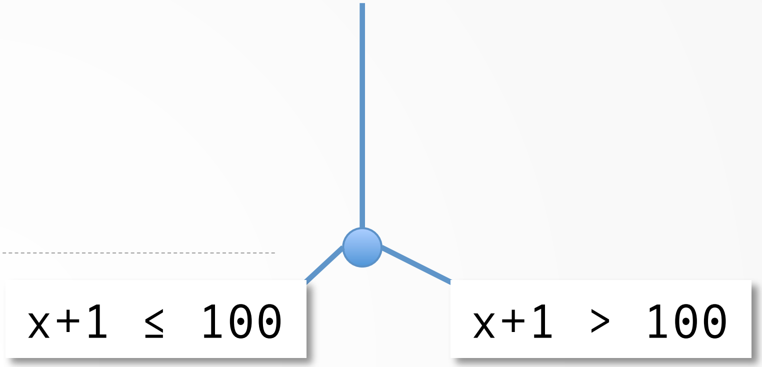
x is a symbolic variable

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

Symbolic execution

x is a symbolic variable

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```



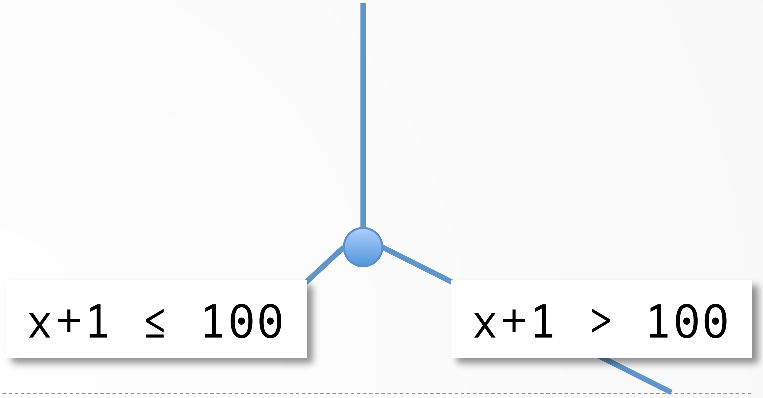
x+1 ≤ 100

x+1 > 100

Symbolic execution

x is a symbolic variable

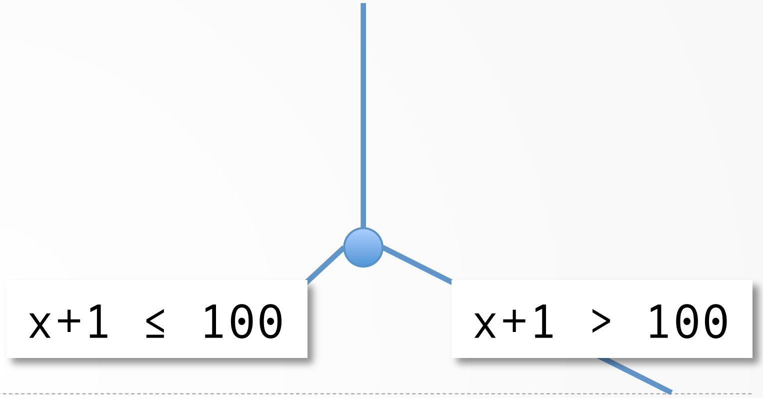
```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```



Symbolic execution

x is a symbolic variable

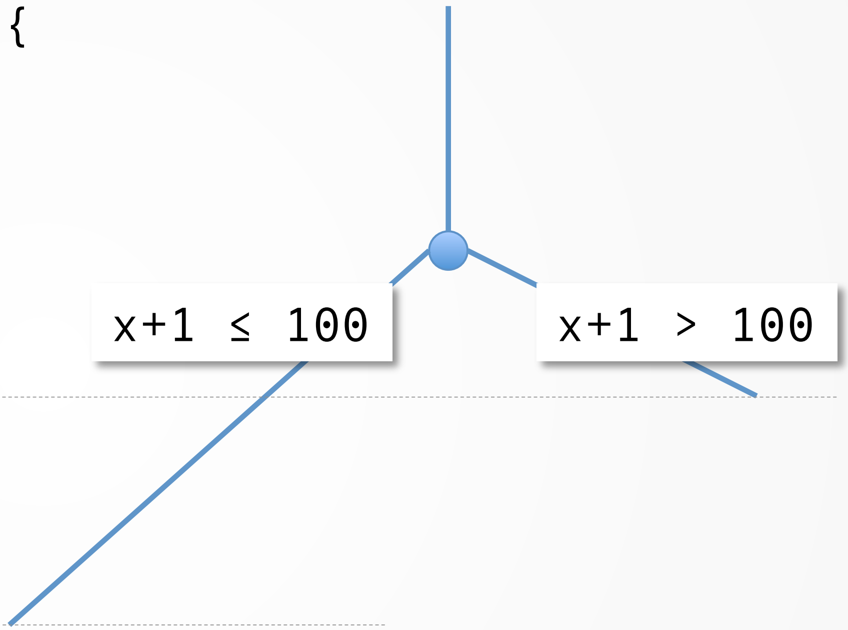
```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```



Symbolic execution

x is a symbolic variable

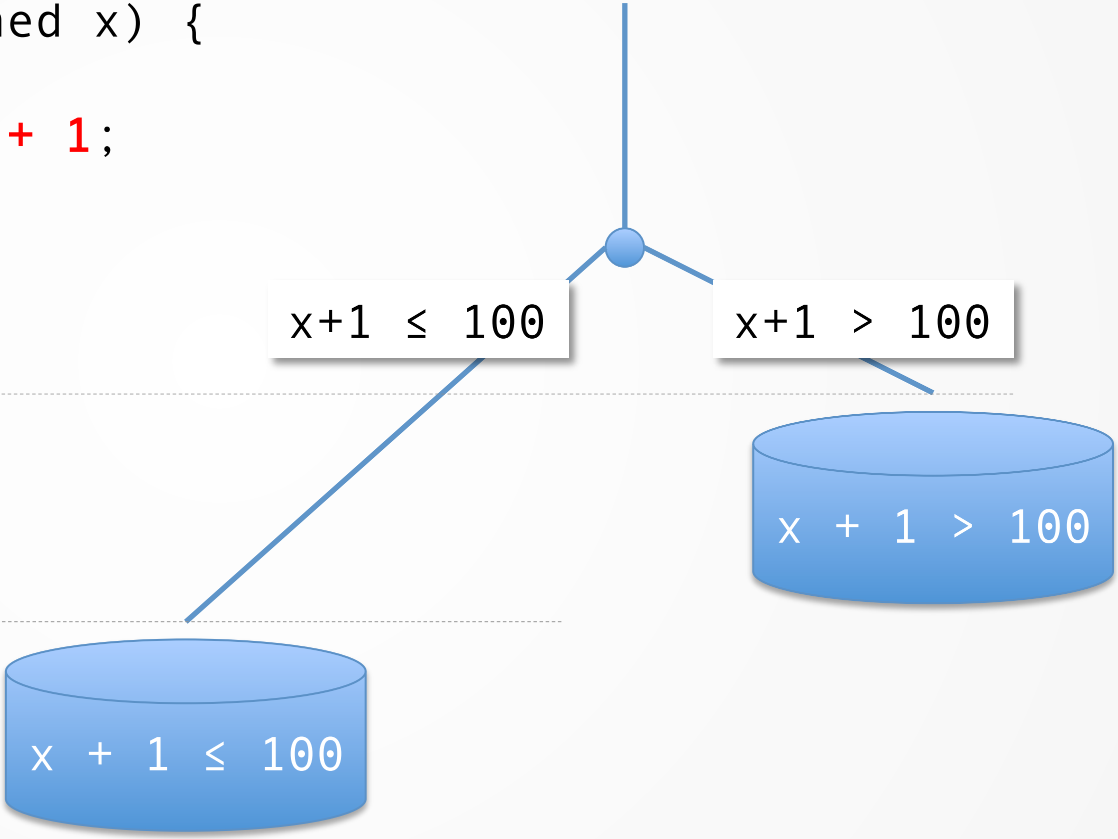
```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```



Symbolic execution

x is a symbolic variable

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```



x+1 ≤ 100

x+1 > 100

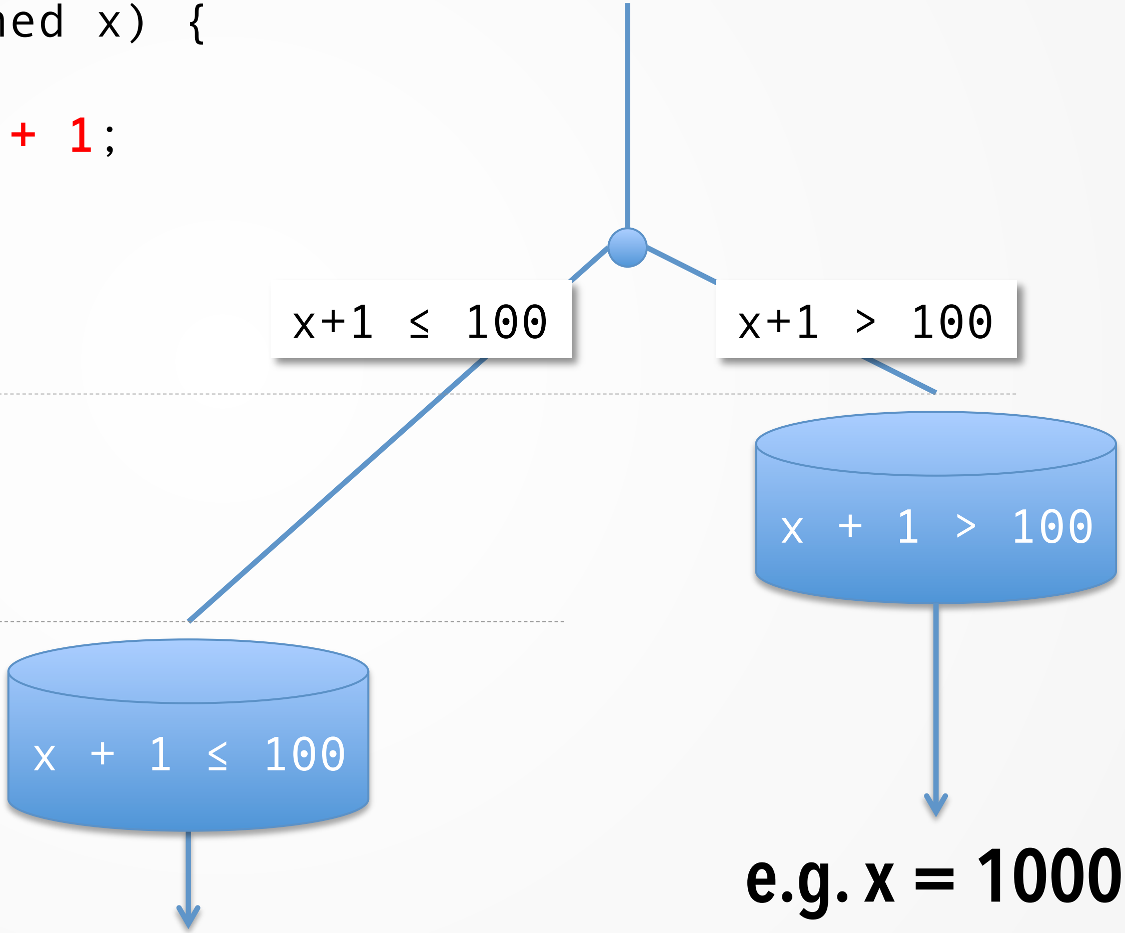
x + 1 > 100

x + 1 ≤ 100

Symbolic execution

x is a symbolic variable

```
int gt_100(unsigned x) {
 unsigned y = x + 1;
 if (y > 100)
  return 1;
 else
  return 0;
}
```



e.g. x = 1000

e.g. x = 0

Shadow symbolic execution

Shadow symbolic execution

- Old and new version in the same instance
  - The two versions are combined
  - Executed in lock-step fashion
  - The old version shadows the new one

- Focus on the new behaviour
  - Versions take different sides of a branch

Shadow symbolic execution

Old

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

New

```
int gt_100(unsigned x) {
  unsigned y = x + 1;
  if (y > 100)
    return 1;
  else
    return 0;
}
```

Combined

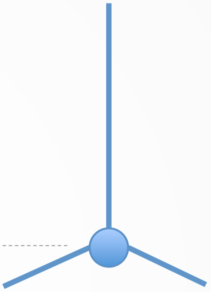
```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```

Shadow symbolic execution

```c
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```

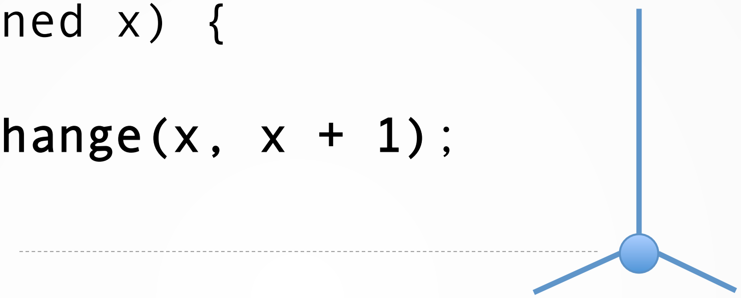
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



Shadow symbolic execution

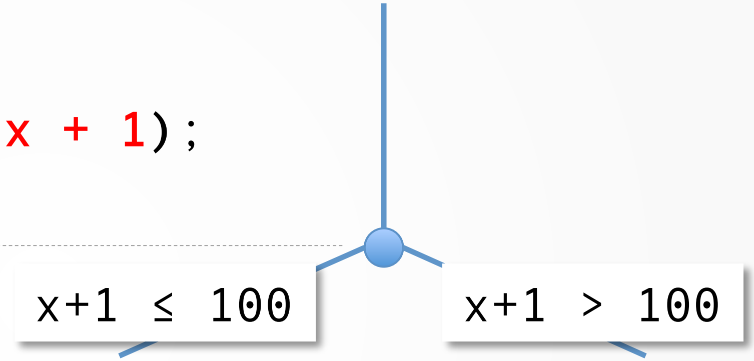
```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



4-way fork

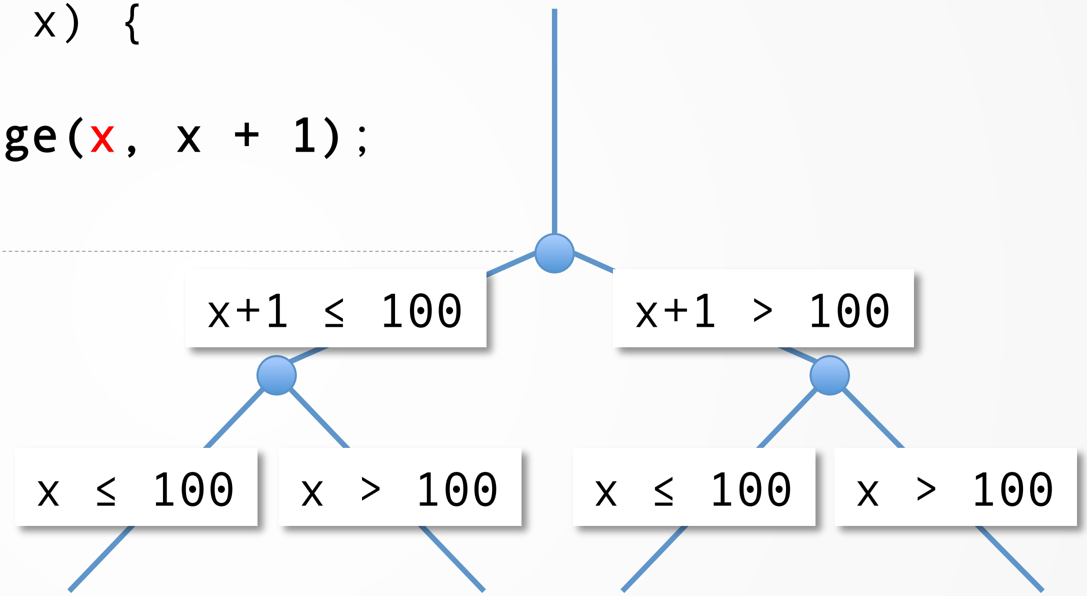
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



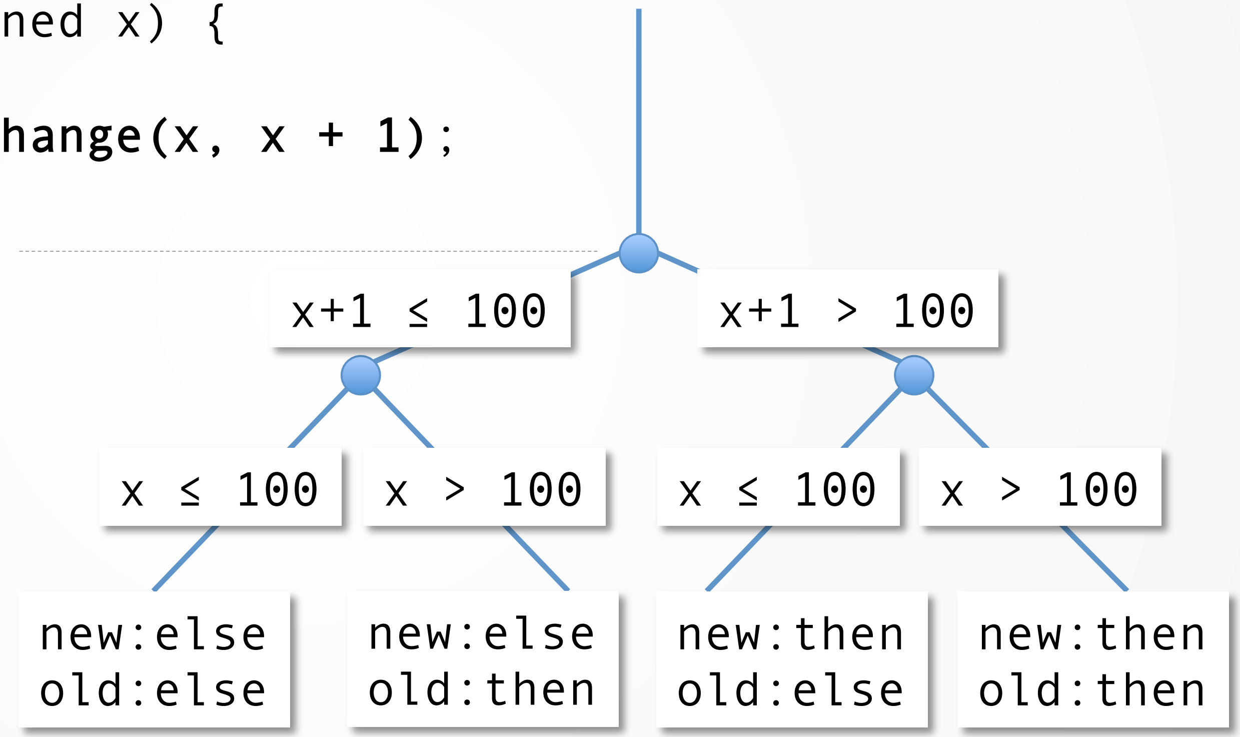
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



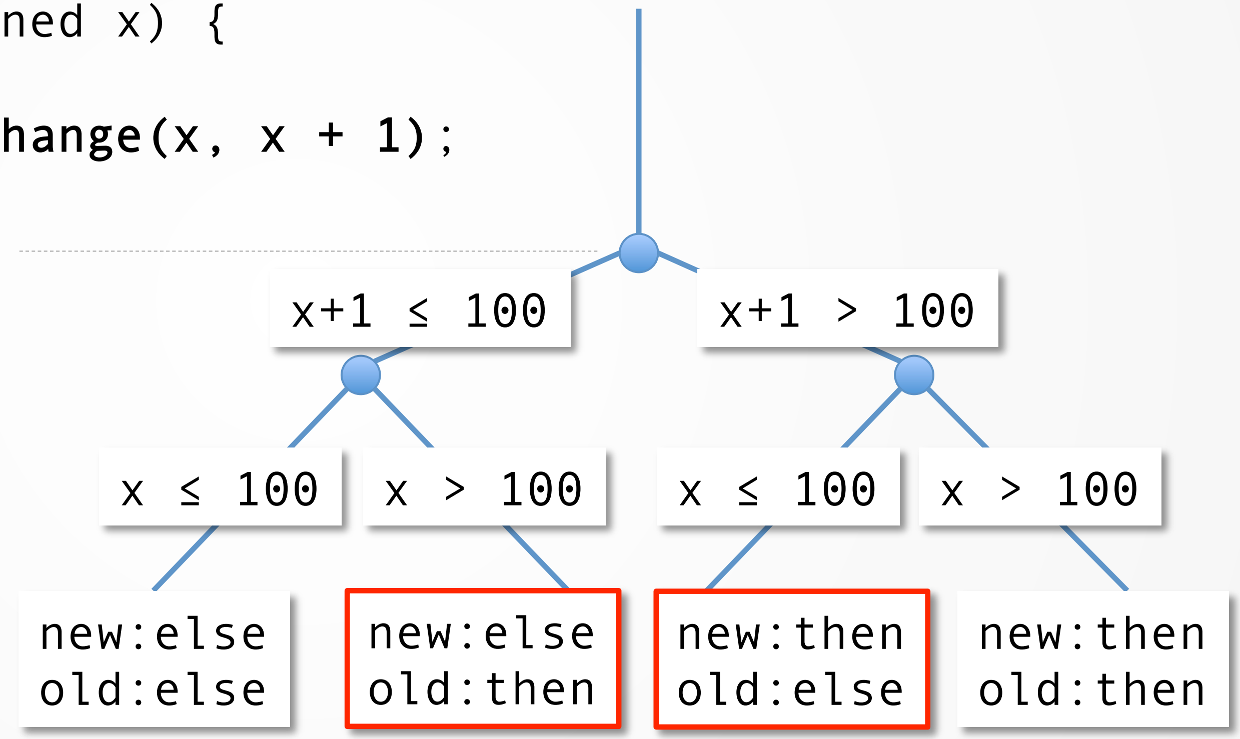
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



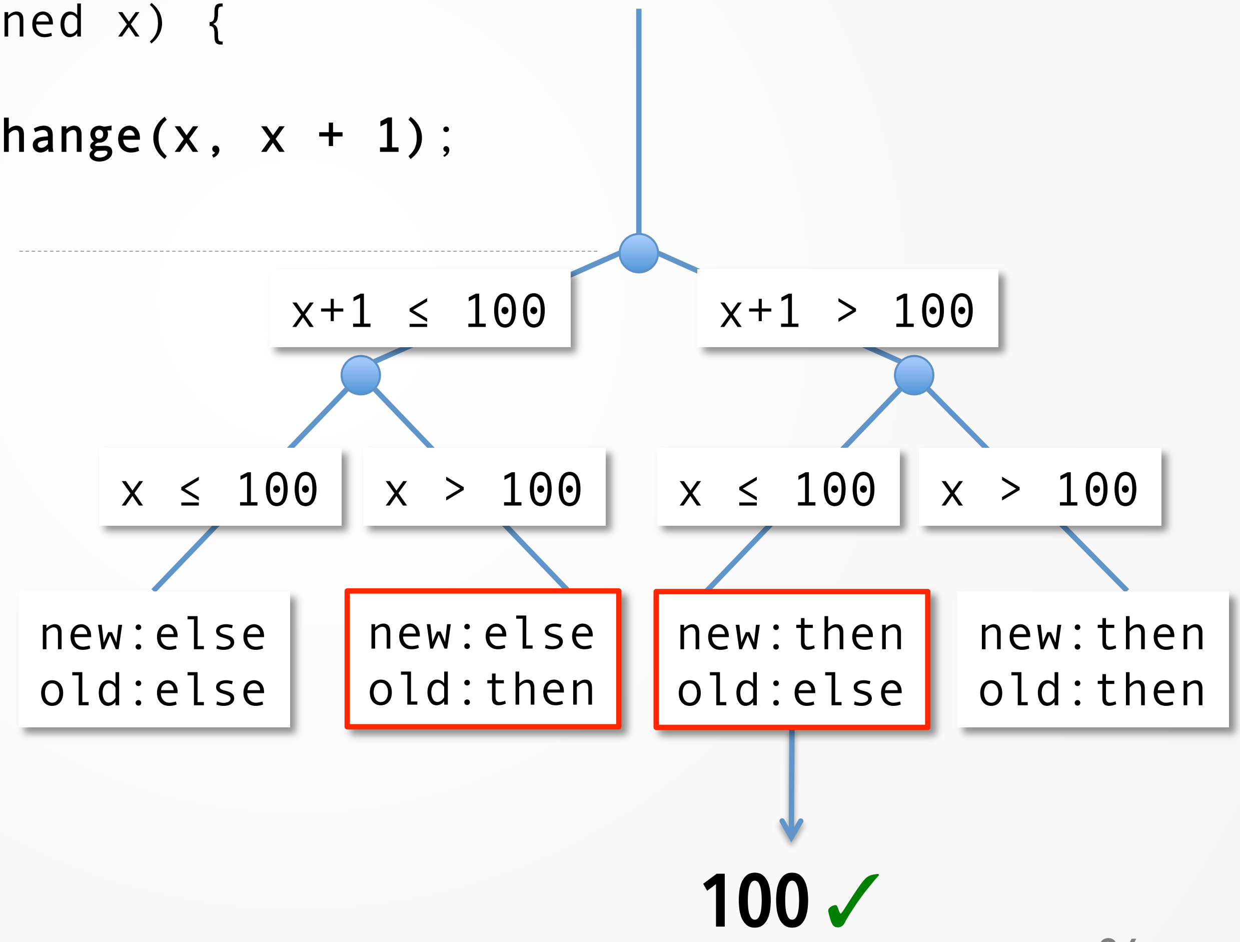
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



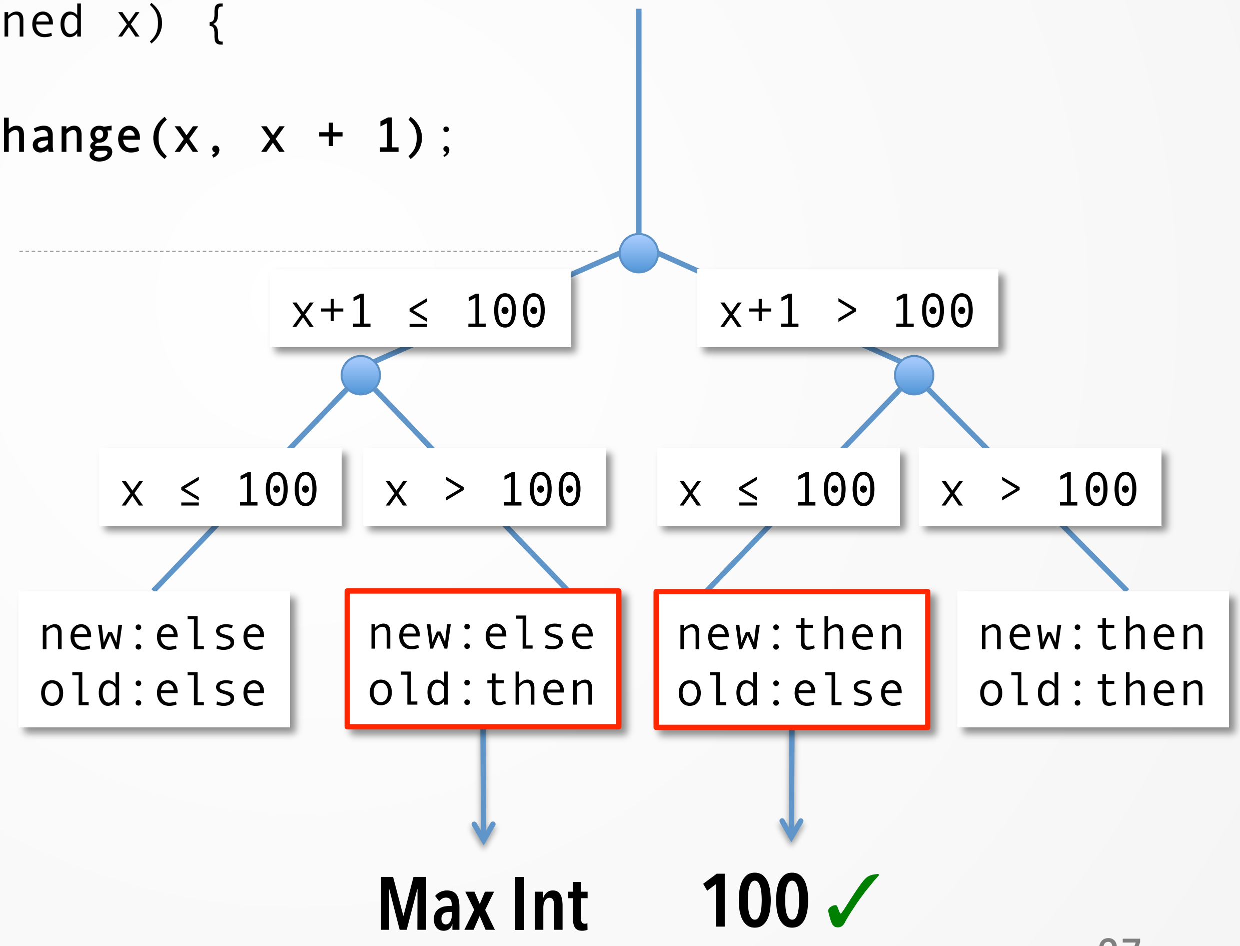
Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



Shadow symbolic execution

```
int gt_100(unsigned x) {
  unsigned y = change(x, x + 1);
  if (y > 100)
    return 1;
  else
    return 0;
}
```



x+1 ≤ 100

x+1 > 100

x ≤ 100

x > 100

x ≤ 100

x > 100

new:else
old:else

new:else
old:then

new:then
old:else

new:then
old:then

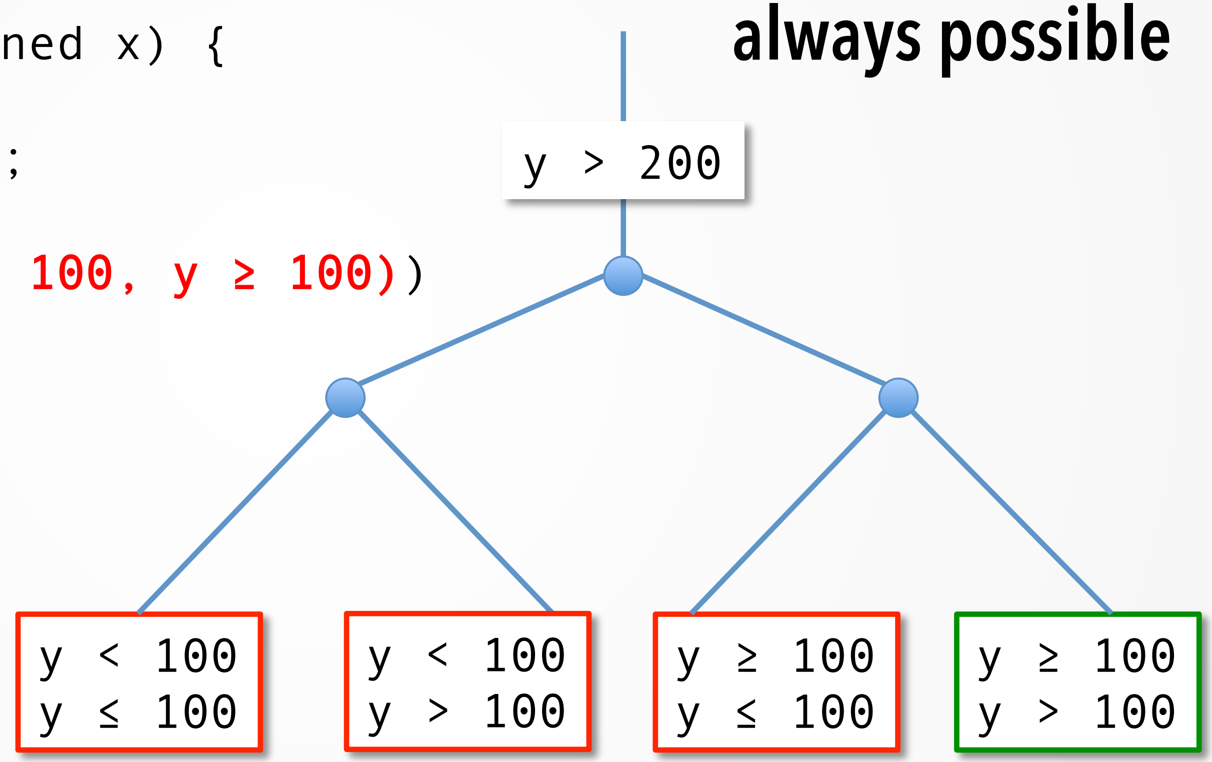
Max Int

100 ✓

Shadow symbolic execution

Divergence not always possible

```
int gt_100(unsigned x) {
  unsigned y = x;
  if (change(y > 100, y ≥ 100))
    return 1;
  else
    return 0;
}
```



y > 200

y < 100
y ≤ 100

y < 100
y > 100

y ≥ 100
y ≤ 100

y ≥ 100
y > 100

Shadow symbolic execution

- Advantages of **shadow** symbolic execution
  - Pruning execution paths – smaller search space
  - Space efficiency
    - Two versions combined into one
    - Expression sharing via shadow expressions
  - Does not execute unchanged path prefix twice

Patch annotations

Expressing patches

- Annotations
  - `change(old, new)` macro
  - Currently manual, automation possible
  - A set of 15 rules
  - See project web-site for annotated patches
    http://srg.doc.ic.ac.uk/projects/shadow/

Annotation rules

- Modifying an rvalue expression

Old

```
if (argc - optind < 1)
  {
    error (...);
    usage (EXIT_FAILURE);
  }
```

New

```
if (n_args < 1)
  {
    error (...);
    usage (EXIT_FAILURE);
  }
```

Combined

```
if (change(argc - optind, n_args) < 1)
  {
    error (...);
    usage (EXIT_FAILURE);
  }
```

Annotation rules

- Adding an assignment

Old

```
byte_idx = 0;
print_delimiter = false;

```

New

```
byte_idx = 0;
print_delimiter = false;
current_rp = rp;
```

Combined

```
byte_idx = 0;
print_delimiter = false;
current_rp = change(current_rp, rp);
```

Patch testing approach

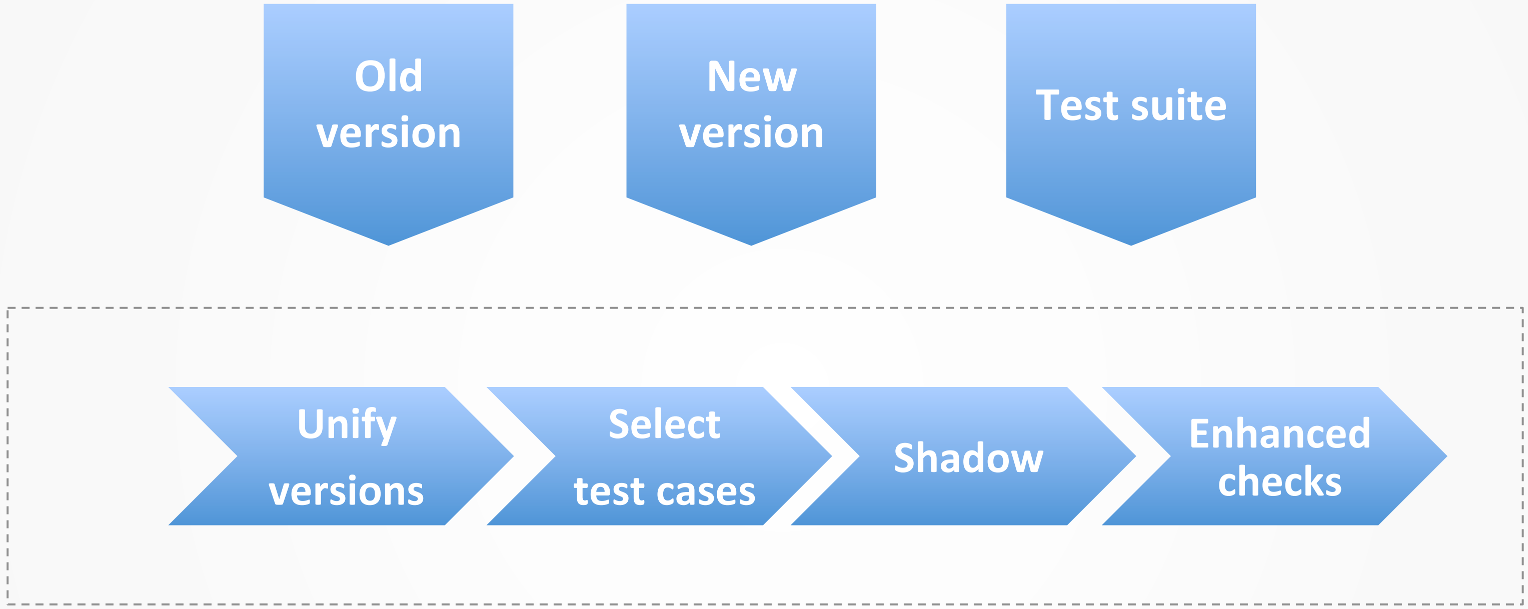
Shadow approach overview

Old version

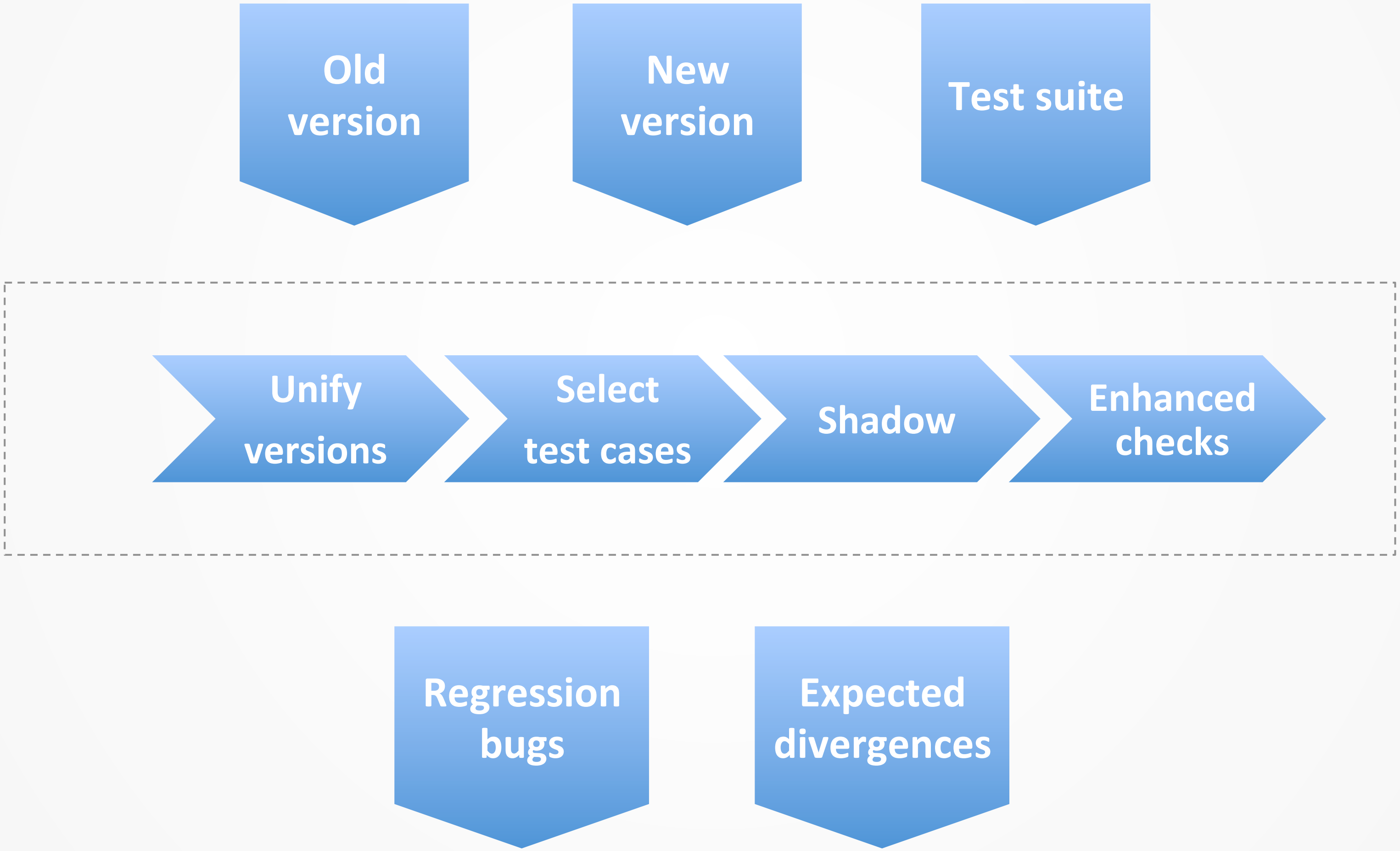
New version

Test suite

Shadow approach overview



Shadow approach overview



Old version

New version

Test suite

Unify versions

Select test cases

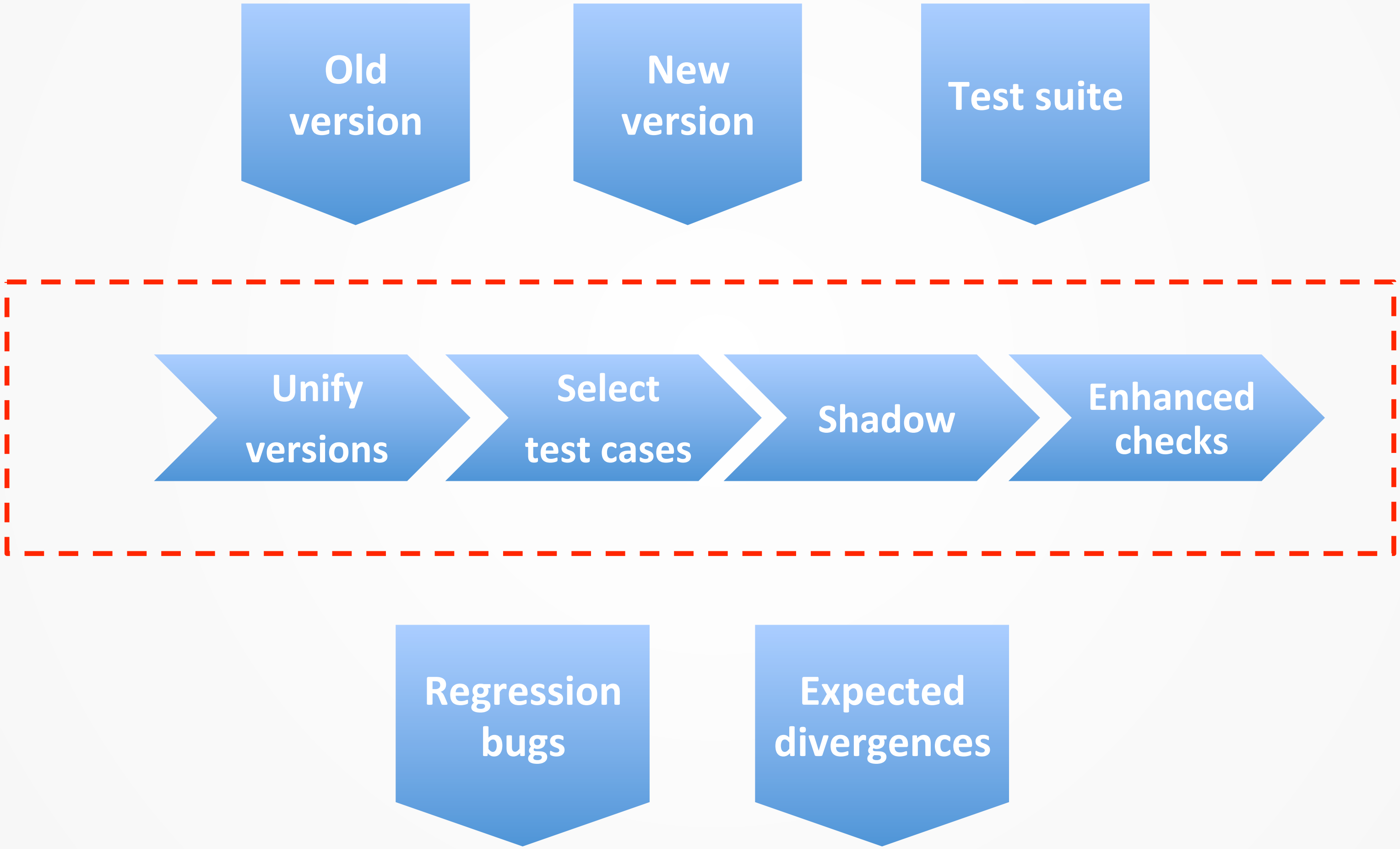
Shadow

Enhanced checks

Regression bugs

Expected divergences

Shadow approach overview



Old version

New version

Test suite

Unify versions

Select test cases

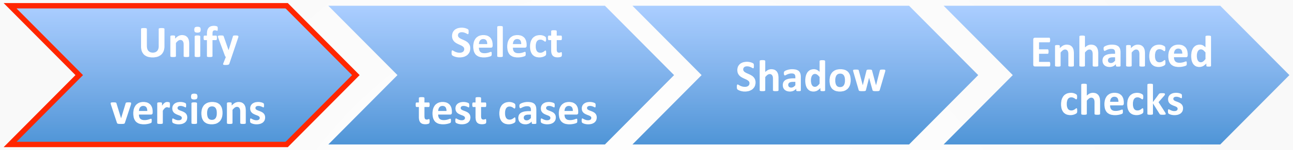
Shadow

Enhanced checks

Regression bugs

Expected divergences

Shadow approach overview



Unify versions

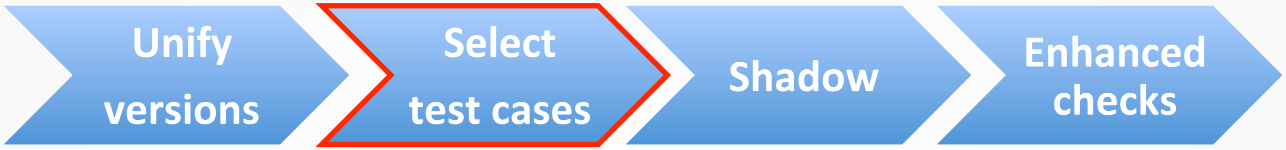
Select test cases

Shadow

Enhanced checks

- Combine old and new version
  - `change()` macro
  - Set of rules

Shadow approach overview



Unify versions

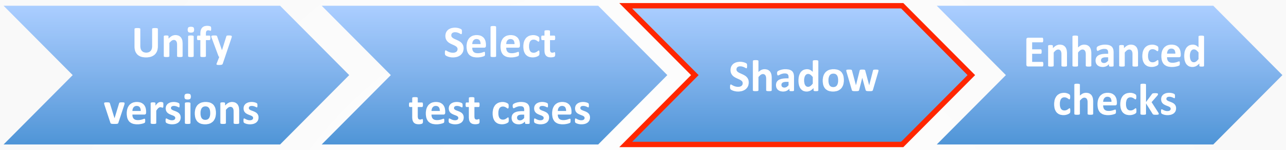
Select test cases

Shadow

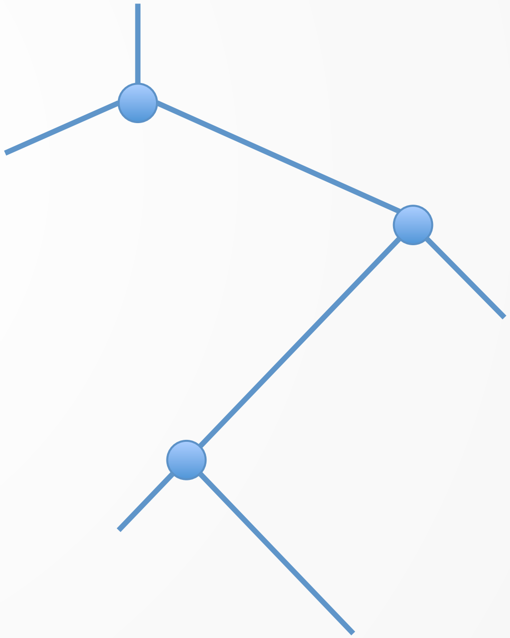
Enhanced checks

- Select test cases that touch the patch
  - Run test suite on the new version
  - Use coverage data
    - Cover at least one line of the patch

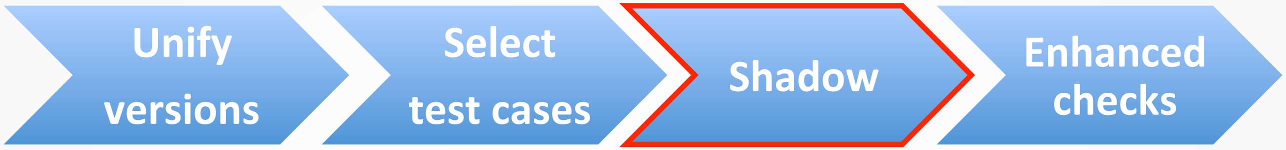
Shadow approach overview



- Use test suite inputs



Shadow approach overview



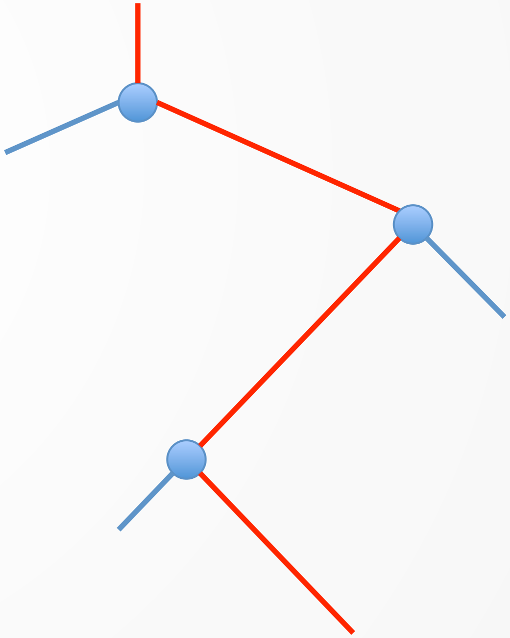
Unify versions

Select test cases

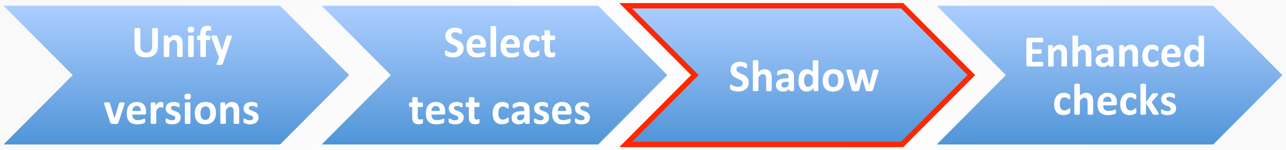
Shadow

Enhanced checks

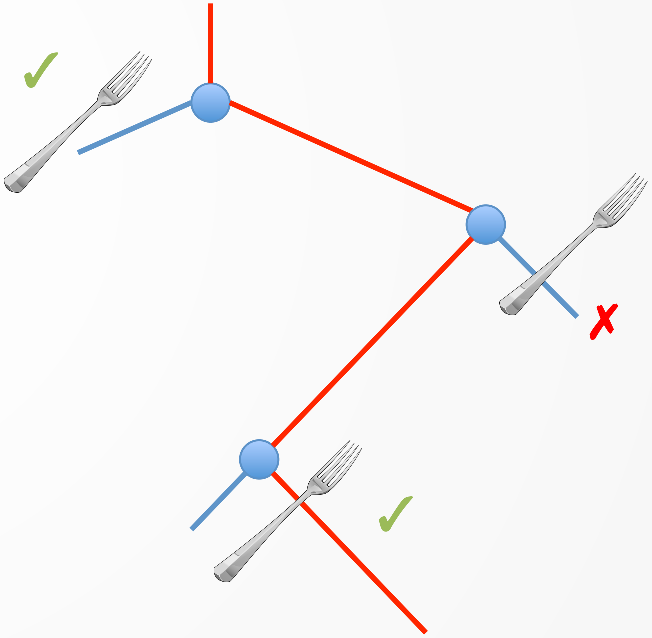
- Use test suite inputs



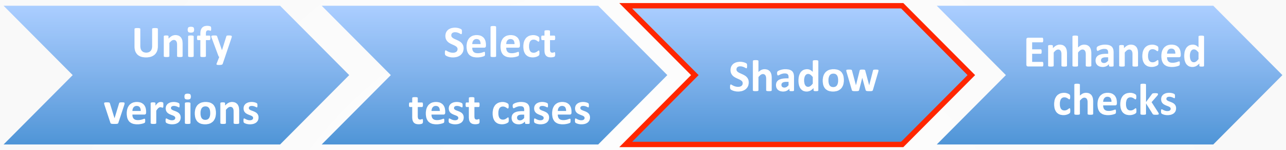
Shadow approach overview



- Use test suite inputs
- Try to find divergent paths



Shadow approach overview



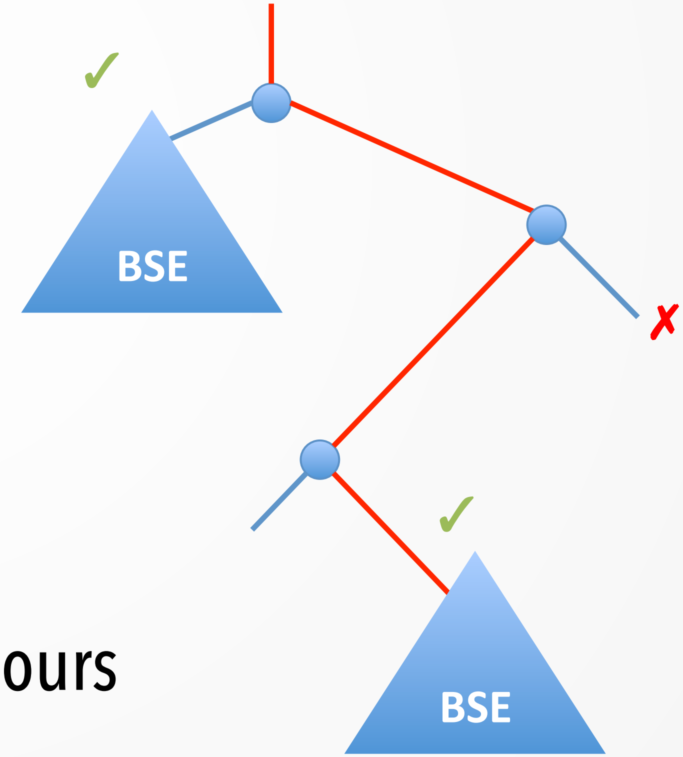
Unify versions

Select test cases

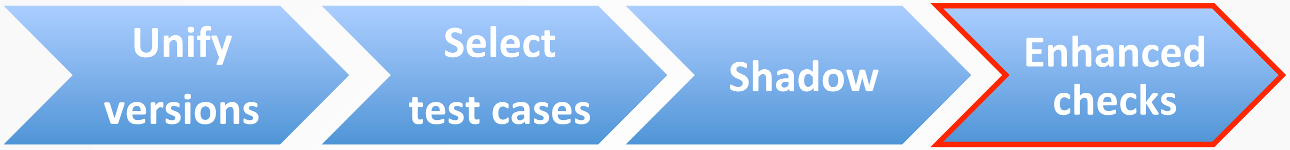
Shadow

Enhanced checks

- Use test suite inputs
- Try to find divergent paths
- Perform bounded symbolic execution
  - New test cases
  - Explore more divergent behaviours



Shadow approach overview



Unify versions

Select test cases

Shadow

Enhanced checks

- Run old and new versions on the generated inputs
- Compare:
  - program outputs
  - program exit codes
  - memory safety violations (ASAN)

Implementation and evaluation

Implementation

- Implemented on top of KLEE
- Uses concolic execution functionality from ZESTI and Docovery

http://klee.github.io

http://srg.doc.ic.ac.uk/projects/zesti

http://srg.doc.ic.ac.uk/projects/docovery

- Evaluated on patches from `CoREBench` study
  - http://www.comp.nus.edu.sg/~release/corebench/
  - 18 unique `Coreutils` patches which introduced bugs
  - Significantly more complex than typical patches used in the evaluation of previous work (e.g. SIR, Siemens)
  - The bug-fixing patches also known
  - Evaluated 16 out of 18 due to technical issues

Evaluation

Patch	Tool	Patch size		Annotations
		LOC	Hunks	
1	mv, rm	45	17	12
3	cut	294	35	14
4	tail	21	4	4
5=16	tail	275	13	1
6	cut	8	3	3
7	seq	148	5	5
8	seq	37	4	12
10	cp	16	8	2
11	cut	2	1	1
12=17	cut	110	17	4
13	ls	13	2	2
14	ls	15	5	4
15	du	3	1	1
19	seq	40	9	6
21	cut	31	10	6
22	expr	54	6	4

Evaluation

Patch	Tool	Patch size		Annotations
		LOC	Hunks	
1	mv, rm	45	17	12
3	cut	294	35	14
4	tail	21	4	4
5=16	tail	275	13	1
6	cut	8	3	3
7	seq	148	5	5
8	seq	37	4	12
10	cp	16	8	2
11	cut	2	1	1
12=17	cut	110	17	4
13	ls	13	2	2
14	ls	15	5	4
15	du	3	1	1
19	seq	40	9	6
21	cut	31	10	6
22	expr	54	6	4

Evaluation

Patch	Tool	Patch size		Annotations
		LOC	Hunks	
1	mv, rm	45	17	12
3	cut	294	35	14
4	tail	21	4	4
5=16	tail	275	13	1
6	cut	8	3	3
7	seq	148	5	5
8	seq	37	4	12
10	cp	16	8	2
11	cut	2	1	1
12=17	cut	110	17	4
13	ls	13	2	2
14	ls	15	5	4
15	du	3	1	1
19	seq	40	9	6
21	cut	31	10	6
22	expr	54	6	4

Expected

Generated input	Behaviour	
	Old	New
cut -s -d: -f0- <file> file contains “:::\n:1”	:::\n1	\n\n
cut -d: -f1,0- <file> file contains “a:b:c”	a:b:c	a
tail --retry ///s\x01\x00g\x00	tail: warning: --retry is useful mainly when following by name…	tail: warning: --retry ignored; --retry is useful only when following…

Bugs

Generated input	Behaviour	
	Old	New
cut -c1-3,8- --output-d=: <file> file contains “abcdefg”	abc	abc + buffer overflow
cut -c1-7,8- --output-d=: <file> file contains “abcdefg”	abcdefg	abcdefg + buffer overflow
cut -b0-2,2- --output-d=: <file> file contains “abc”	abc	signal abort

Bugs

Generated input	Behaviour	
	Old	New

`cut -c1-3,8- --output-d=: <file>`
`file contains “abcdefg”`
	`abc`	`abc + buffer overflow`

`cut -c1-7,8- --output-d=: <file>`
`file contains “abcdefg”`
	`abcdefg`	`abcdefg + buffer overflow`

`cut -b0-2,2- --output-d=: <file>`
`file contains “abc”`
	`abc`	`signal abort`

New bug, not part of CoREBench

Evaluation

Patch	Divergences	Output differences	
		Expected	Bug
1	39K	3	-
3	15K	-	-
4	39	36	-
5=16	14	-	2
6	1.4K	-	86
7	124	5	-
8	54K	-	-
10	6	-	2
11	874	9	-
12=17	4.2K	-	78
13	11	1	1
14	2	-	-
15	1	1	-
19	33K	7	-
21	21K	151	684
22	-	-	-

Evaluation

Patch	Divergences	Output differences	
		Expected	Bug
1	39K	3	-
3	15K	-	-
4	39	36	-
5=16	14	-	2
6	1.4K	-	86
7	124	5	-
8	54K	-	-
10	6	-	2
11	874	9	-
12=17	4.2K	-	78
13	11	1	1
14	2	-	-
15	1	1	-
19	33K	7	-
21	21K	151	684
22	-	-	-

Evaluation

Patch	Divergences	Output differences	
		Expected	Bug
1	39K	3	-
3	15K	-	-
4	39	36	-
5=16	14	-	2
6	1.4K	-	86
7	124	5	-
8	54K	-	-
10	6	-	2
11	874	9	-
12=17	4.2K	-	78
13	11	1	1
14	2	-	-
15	1	1	-
19	33K	7	-
21	21K	151	684
22	-	-	-

Evaluation

Patch	Divergences	Output differences	
		Expected	Bug
1	39K	3	-
3	15K	-	-
4	39	36	-
5=16	14	-	2
6	1.4K	-	86
7	124	5	-
8	54K	-	-
10	6	-	2
11	874	9	-
12=17	4.2K	-	78
13	11	1	1
14	2	-	-
15	1	1	-
19	33K	7	-
21	21K	151	684
22	-	-	-

- Unsuccessful cases
  - Refactorings
  - Non-functional changes
    - Memory consumption
    - Performance
  - Technical challenges:
    - Reasoning about file access rights
    - Symbolic directories support
    - Floating point support
    - Not reproducible

Summary

Shadow symbolic execution

- A symbolic execution technique for patch testing
  - Generates inputs that trigger new behaviours
  - Prunes large parts of the search space
  - Useful for: regression testing, test-suite augmentation, patch understanding

http://srg.doc.ic.ac.uk/projects/shadow/

SOFTWARE RELIABILITY
GROUP