Automatic Self-Validation for Code Coverage Profilers

Yibiao Yang, *Yanyan Jiang* 蒋炎岩 (*presenter*), Zhiqiang Zuo,

Yang Wang, Hao Sun, Hongmin Lu, Yuming Zhou, Baowen Xu

南京大学
Nanjing University, China

Outline

- Code coverage profilers
- Testing code coverage profilers
- Automatic self-validation for code coverage profilers

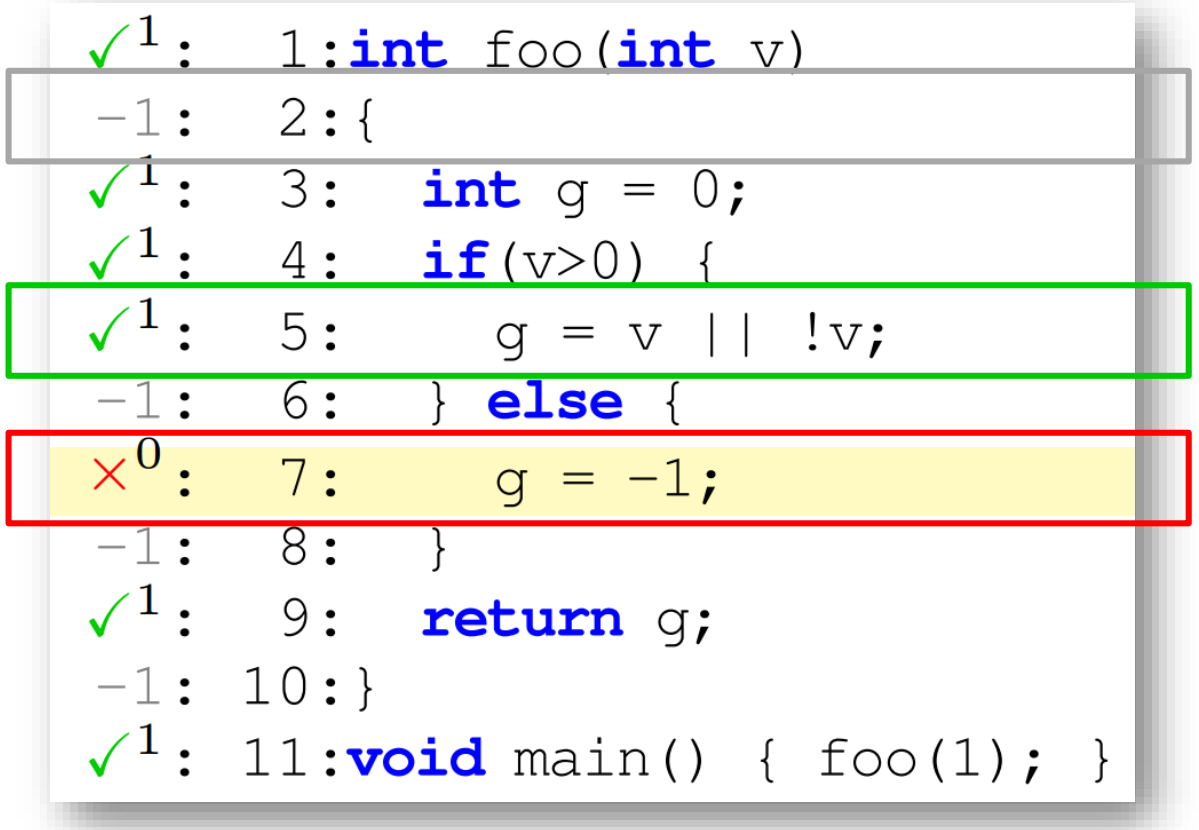
Code Coverage Profilers

Code Coverage

- Which code (normally a line) is executed or not?

- How many times each code is executed?

Code Coverage: Profilers



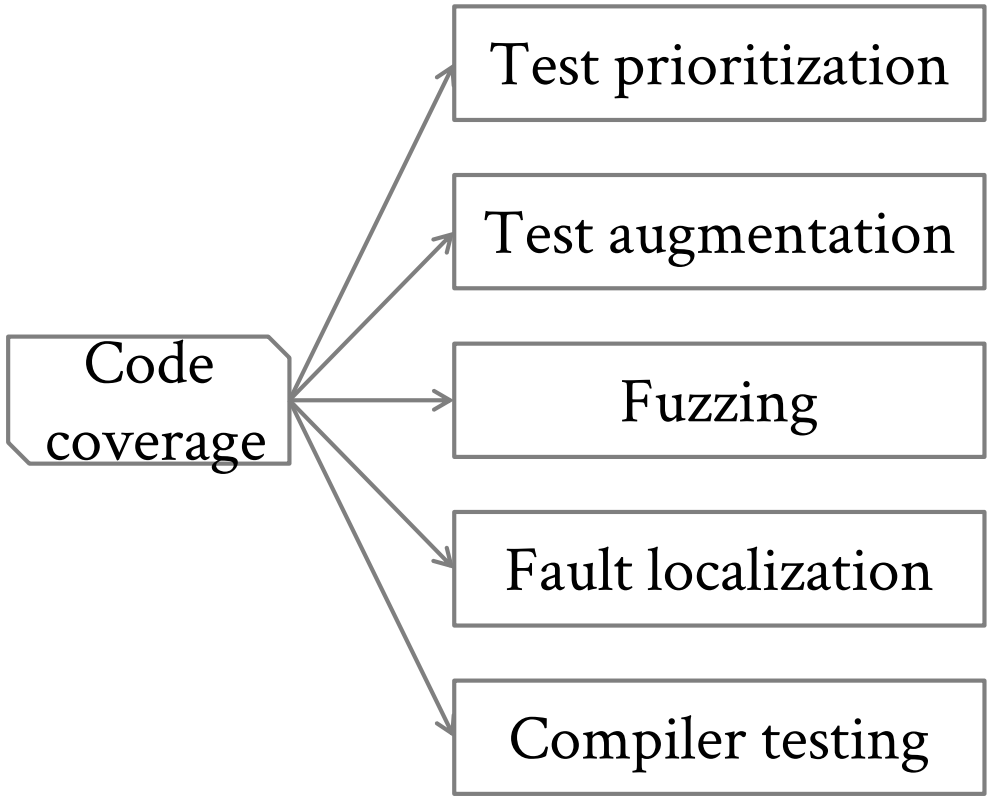
Line #2: no coverage information.

Line #5: executed once (1)

Line #7: not executed (0)

Example coverage report by Gcov (GCC)

Code Coverage: Usages



10:40 - 12:20: Papers - Testing and Coverage at Cortez 1

Chair(s): Jonathan Bell George Mason University

Talk	Automatic Self-Validation for Code Coverage Profilers
Talk	Efficient Test Generation Guided by Field Coverage Criteria
Talk	Exploring Output-Based Coverage for Testing PHP Web App
Talk	PHANTA: Diversified Test Code Quality Measurement for M
Demo	TestCov: Robust Test-Suite Execution and Coverage Measu
Demo	VisFuzz: Understanding and Intervening Fuzzing with Intera

Testing Code Coverage Profilers

Code Coverage Statistics Went Wrong?

- Oooops…
  - testers are misled
  - fuzzers lose directions

- That's why we should *test* code coverage profilers!

Differential Testing[1]

Compare outputs of two independent code coverage profilers on a same program

Gcov

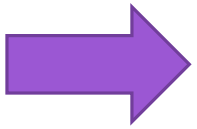
```
-1: 12:   }
✓4: 13:   for(;g;)
✓2: 14:     return 0;
×0: 15:   return 0;
-1: 16:}
```

LLVM-cov

```
12| ✓2|   }
13| ✓2|   for(;g;)
14| ✓2|     return 0;
15| ✓2|   return 0;
16| ✓2|}
```

Wrong execution count: ☑4 vs. ☑2

Mis-reporting: ☑2 vs. ☒0

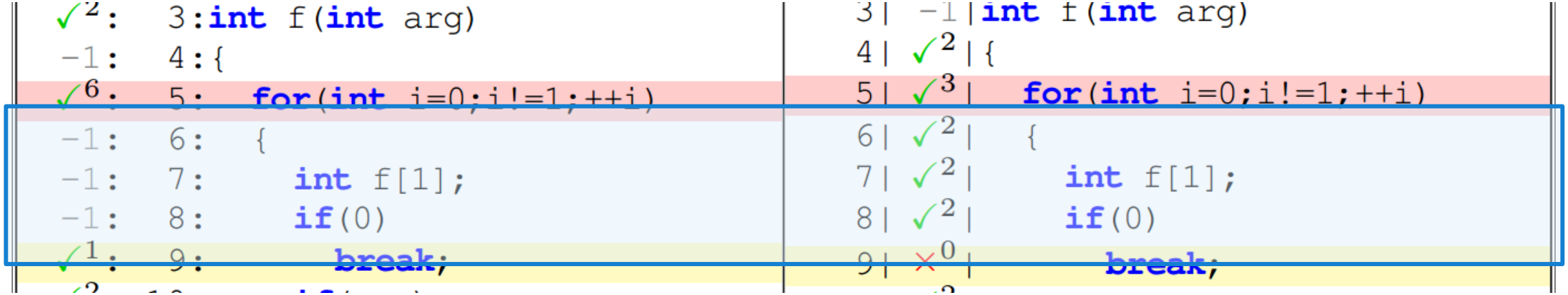


Something goes wrong!

[1] Y. Yang, et al. Hunting for bugs in code coverage tools via randomized differential testing. (ICSE'19)

But…

- Compilers do not have consensus on the definition of "covering a line"

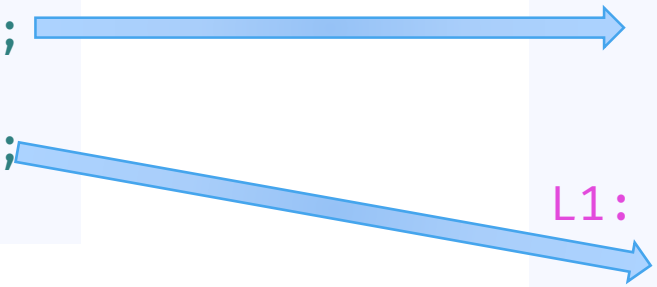


  - differential testing has lots of false positives
  - differential testing has to be conservative if a profiler reports -1

Why?

- Coverage profiler instruments basic blocks
  - add `__gcov0.foo[#bb]++;` to each basic block #bb

```
1 void foo(int x)
2 {
3   if (x >= 0)
4     asm volatile ("XXX");
5   else
6     asm volatile ("YYY");
7 }
```



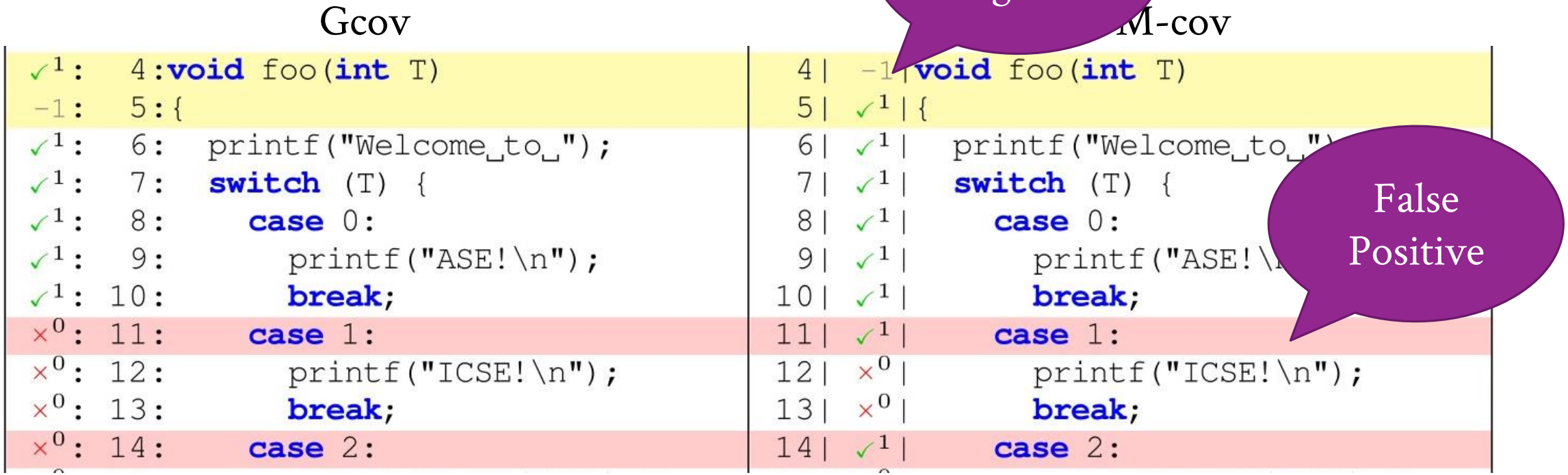
```
    test %edi,%edi
    js L1
    incq (__gcov0.foo[0])
    XXX
    incq (__gcov0.foo[2])
    retq
L1: incq (__gcov0.foo[1])
    YYY
    incq (__gcov0.foo[3])
    retq
```

`gcc -Os -c --coverage a.c`

(-Os to make assembly code easier to read)

Why?

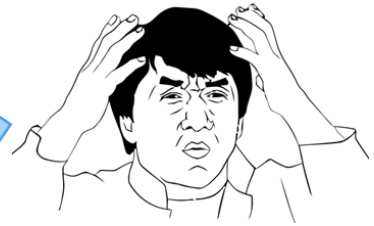
- Different compilers hold different opini[illegible] what are basic blocks, when they are executed, and [illegible]k to code!



Even Worse…

- Different compilers can do different optimizations even at zero optimization level
  - optimized code has no coverage info (-1)

```c
1 #define ONE 1
2 void foo(int x) {
3   if (ONE == 1) {
4     asm volatile ("XXX");
5   } else {
6     // optimized out
7   }
8 }
```



Optimization levels

O2 > O1 > Og > **O0**

```
1 push %rbp
2 mov %rsp,%rbp
3 mov %edi,-0x4(%rbp)
4 incl (__gcov0.foo[0])
5 XXX
6 incl (__gcov0.foo[1])
7 pop %rbp
8 retq
```

Image source: 暴走漫画

Automatic Self-Validation for Code Coverage Profilers

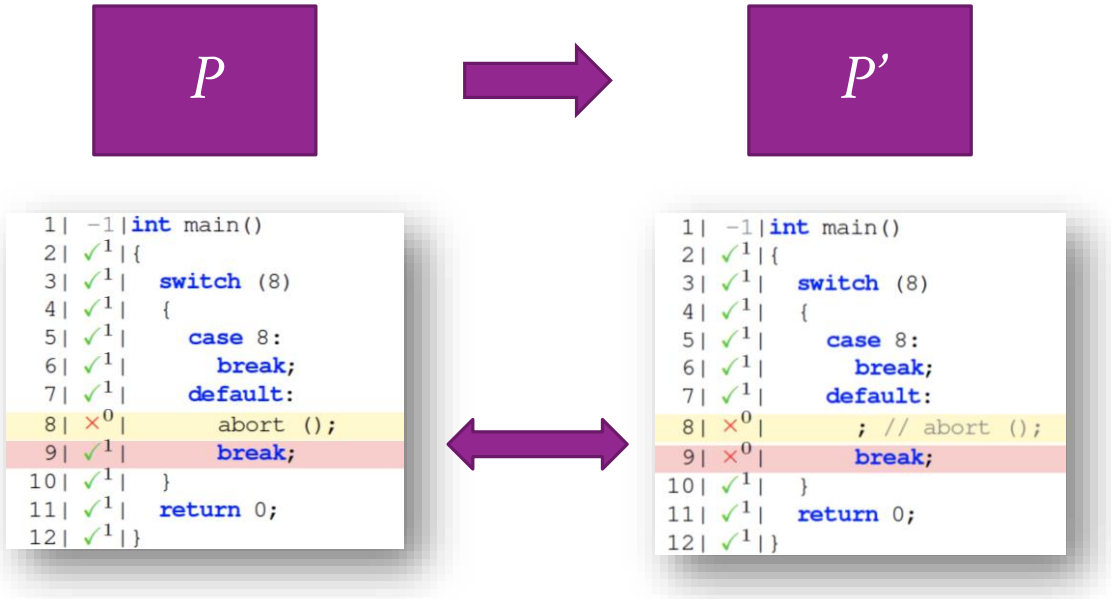
Our Goals

- ***Automatic self-validation*** of code coverage profilers
  - getting rid of the need of a reference implementation

- With ***zero false positive***
  - getting rid of heuristic clustering and human inspections

Basic Idea: Self-Validation

- Can't find two profilers? Just find two programs with ***correlated*** code coverage statistics!
  - ***mutate*** a single piece of program without disturbing existing code coverage statistics



…By Changing "Non-Covered" Code!

```
 1|  -1|int main()
 2|  ✓1|{
 3|  ✓1|   switch (8)
 4|  ✓1|   {
 5|  ✓1|      case 8:
 6|  ✓1|         break;
 7|  ✓1|      default:
 8|  ×0|         abort ();
 9|  ✓1|         break;
10|  ✓1|   }
11|  ✓1|   return 0;
12|  ✓1|}
```

- Replacing Line #8 (unexecuted) with ***anything*** won't change the coverage of other parts
  - (only if Line #8 is indeed not executed)
- Otherwise, we ***found a bug in the coverage profiler***!

Example: LLVM-#41821

```
-1:   3:  switch (8)
-1:   4:  {
-1:   5:    case 8:
✓1:   6:      break;
-1:   7:    default:
-1:   8:      abort ();
-1:   9:      break;
-1:  10:  }
```

gcov: unreachable code
optimized out (correct)

Example: LLVM-#41821

```
-1:   3:  switch (8)
-1:   4:  {
-1:   5:    case 8:
✓1:   6:      break;
-1:   7:    default:
-1:   8:      abort ();
-1:   9:      break;
-1:  10:  }
```

```
   3| ✓1|  switch (8)
   4| ✓1|  {
   5| ✓1|    case 8:
   6| ✓1|      break;
   7| ✓1|    default:
   8| ×0|      abort ();
   9| ✓1|      break;
  10| ✓1|  }
```

llvm-cov: inconsistent coverage statistics for sequential code (incorrect)

Example: LLVM-#41821

```
-1:  3:  switch (8)
-1:  4:  {
-1:  5:    case 8:
✓1:  6:      break;
-1:  7:    default:
-1:  8:      abort ();
-1:  9:      break;
-1: 10:  }
```

```
 3| ✓1|  switch (8)
 4| ✓1|  {
 5| ✓1|    case 8:
 6| ✓1|      break;
 7| ✓1|    default:
 8| ×0|      abort ();
 9| ✓1|      break;
10| ✓1|  }
```

```
 3| ✓1|  switch (8)
 4| ✓1|  {
 5| ✓1|    case 8:
 6| ✓1|      break;
 7| ✓1|    default:
 8| ×0|      ; // abort ();
 9| ×0|      break;
10| ✓1|  }
```

llvm-cov: correct if not aborting

Example: LLVM-#41821

```
✓1:   1:int main()
-1:   2:{
-1:   3:   switch (8)
-1:   4:   {
-1:   5:     case 8:
✓1:   6:       break;
-1:   7:     default:
-1:   8:       abort ();
-1:   9:       break;
-1:  10:   }
✓1:  11:   return 0;
-1:  12:}
```

(a) $\mathcal{C}_{\mathcal{P}}$ (gcov)

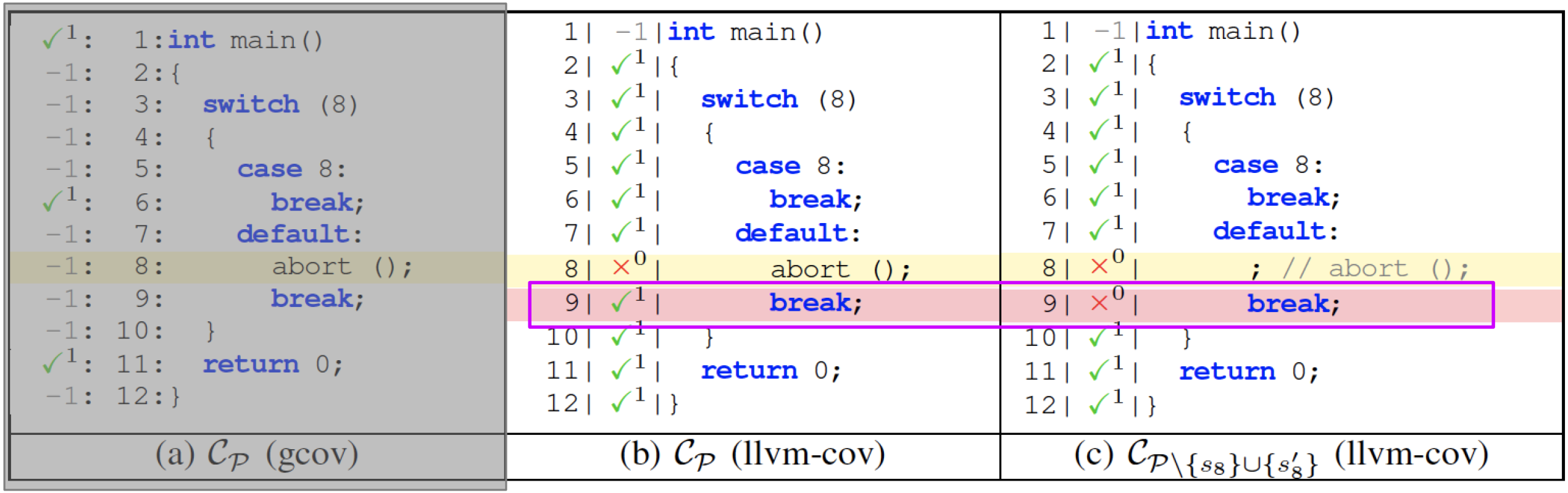
```
 1| -1|int main()
 2| ✓1|{
 3| ✓1|   switch (8)
 4| ✓1|   {
 5| ✓1|     case 8:
 6| ✓1|       break;
 7| ✓1|     default:
 8| ×0|       abort ();
 9| ✓1|       break;
10| ✓1|   }
11| ✓1|   return 0;
12| ✓1|}
```

(b) $\mathcal{C}_{\mathcal{P}}$ (llvm-cov)

```
 1| -1|int main()
 2| ✓1|{
 3| ✓1|   switch (8)
 4| ✓1|   {
 5| ✓1|     case 8:
 6| ✓1|       break;
 7| ✓1|     default:
 8| ×0|       ; // abort ();
 9| ×0|       break;
10| ✓1|   }
11| ✓1|   return 0;
12| ✓1|}
```

(c) $\mathcal{C}_{\mathcal{P}\setminus\{s_8\}\cup\{s'_8\}}$ (llvm-cov)

Example: LLVM-#41821



Example: GCC-#90439

- Removing unexecuted code should not rule out executed code
  - otherwise, buggy coverage profiler!

```
✓1:  1:void foo(int x, unsigned u) {
✓1:  2:  if ((1U << x) != 64
✓1:  3:       || (2 << x) != u
-1:  4:       || (1 << x) == 14
✓1:  5:       || (3 << 2) != 12)
×0:  6:    __builtin_abort ();
✓1:  7:}
✓1:  8:int main() {
✓1:  9:  foo(6, 128U);
✓1: 10:  return 0;
-1: 11:}
```

(a) $\mathcal{C}_{\mathcal{P}}$ (gcov)

```
✓1:  1:void foo(int x, unsigned u) {
✓1:  2:  if ((1U << x) != 64
✓1:  3:       || (2 << x) != u
-1:  4:       || (1 << x) == 14
-1:  5:       || (3 << 2) != 12)
-1:  6:    ; // __builtin_abort ();
✓1:  7:}
✓1:  8:int main() {
✓1:  9:  foo(6, 128U);
✓1: 10:  return 0;
-1: 11:}
```

(b) $\mathcal{C}_{\mathcal{P}\setminus\{s_5\}\cup\{s_5'\}}$ (gcov)

Discussion: Zero False Positive

- Argument: zero-level optimization should strictly follow the ***statement-level semantics*** of a program[1] (assuming no undefined behavior)
  - strong inconsistency ($x \neq y; -1 \notin \{x, y\}$)
    - $\vdash$ bug (LLVM-#41821)
  - weak inconsistency ($x \neq y;\ -1 \in \{x, y\}$)
    - $\vdash$ bug (GCC-#90439) or improperly aggressive optimization

[1] C. Ellison, G. Rosu. An executable formal semantics of C with applications. (POPL'12)

Experimental Results

- 23 previously unknown bugs in Gcov and LLVM-cov
  - 12 cannot be found by differential testing
  - other 11 were filtered out as false positives

ID	Profiler	Bugzilla ID	Priority	Status	Type	DiffTest
1	gcov	88913	P3	Fixed	Wrong Freq.	✓
2	gcov	88914	P3	Fixed	Wrong Freq.	✓
3	gcov	88924	P5	New	Wrong Freq.	✓
4	gcov	88930	P3	Fixed	Wrong Freq.	✓
5	gcov	89465	P3	Fixed	Missing	×
6	gcov	89467	P3	Fixed	Wrong Freq.	✓
7	gcov	89468	P5	New	Wrong Freq.	×
8	gcov	89469	P5	New	Wrong Freq.	✓
9	gcov	89470	P5	New	Wrong Freq.	✓
10	gcov	89673	P5	New	Spurious	×
11	gcov	89674	P5	New	Spurious	×
12	gcov	89675	P3	Fixed	Missing	×
13	gcov	90023	P5	New	Spurious	×
14	gcov	90054	P3	Fixed	Missing	✓
15	gcov	90057	P3	Fixed	Wrong Freq.	✓
16	gcov	90066	P5	New	Wrong Freq.	×
17	gcov	90091	P3	New	Wrong Freq.	✓
18	gcov	90104	P3	New	Wrong Freq.	×
19	gcov	90425	P5	New	Wrong Freq.	×
20	gcov	90439	P3	New	Missing	×
21	llvm-cov	41051	PN	New	Wrong Freq.	✓
22	llvm-cov	41821	PN	New	Spurious	×
23	llvm-cov	41849	PN	New	Missing	×

Summary & Thanks!

Our Goals

- ***Automatic self-validation*** of code coverage profilers
  - getting rid of the need of a reference implementation
- With ***zero false positive***
  - getting rid of heuristic clustering and human inspections

Basic Idea: Self-Validation

- Can't find two profilers? Just find two programs with ***correlated*** code coverage statistics!
  - ***mutate*** a single piece of program without disturbing existing code coverage statistics

...By Changing "Non-Covered" Code!

```
 1|  -1|int main()
 2|  ✓1|{
 3|  ✓1|  switch (8)
 4|  ✓1|  {
 5|  ✓1|    case 8:
 6|  ✓1|      break;
 7|  ✓1|    default:
 8|  ×0|      abort ();
 9|  ✓1|      break;
10|  ✓1|  }
11|  ✓1|  return 0;
12|  ✓1|}
```

- Replacing Line #8 (unexecuted) with ***anything*** won't change the coverage of other parts
  - (only if Line #8 is indeed not executed)
- Otherwise, we ***found a bug in the coverage profiler***!

Experimental Results

- 23 previously unknown bugs in Gcov and LLVM-cov
  - 12 cannot be found by differential testing
  - other 11 were filtered out as false positives

ID	Profiler	Bugzilla ID	Priority	Status	Type	DiffTest
1	gcov	88913	P3	Fixed	Wrong Freq.	✓
2	gcov	88914	P3	Fixed	Wrong Freq.	✓
3	gcov	88924	P5	New	Wrong Freq.	✓
4	gcov	88930	P3	Fixed	Wrong Freq.	✓
5	gcov	89465	P3	Fixed	Missing	×
6	gcov	89467	P3	Fixed	Wrong Freq.	✓
7	gcov	89468	P5	New	Wrong Freq.	×
8	gcov	89469	P5	New	Wrong Freq.	✓
9	gcov	89470	P5	New	Wrong Freq.	✓
10	gcov	89673	P5	New	Spurious	×
11	gcov	89674	P5	New	Spurious	×
12	gcov	89675	P3	Fixed	Missing	×
13	gcov	90023	P5	New	Spurious	×
14	gcov	90054	P3	Fixed	Missing	✓
15	gcov	90057	P3	Fixed	Wrong Freq.	✓
16	gcov	90066	P5	New	Wrong Freq.	×
17	gcov	90091	P3	New	Wrong Freq.	✓
18	gcov	90104	P3	New	Wrong Freq.	×
19	gcov	90425	P5	New	Wrong Freq.	×
20	gcov	90439	P3	New	Missing	×
21	llvm-cov	41051	PN	New	Wrong Freq.	✓
22	llvm-cov	41821	PN	New	Spurious	×
23	llvm-cov	41849	PN	New	Missing	×