MANCHESTER 1824

The University of Manchester

Coverage Checking and Optimizations in ESBMC

Supervisor: Lucas C. Cordeiro

Co-Supervisor: Youcheng Sun

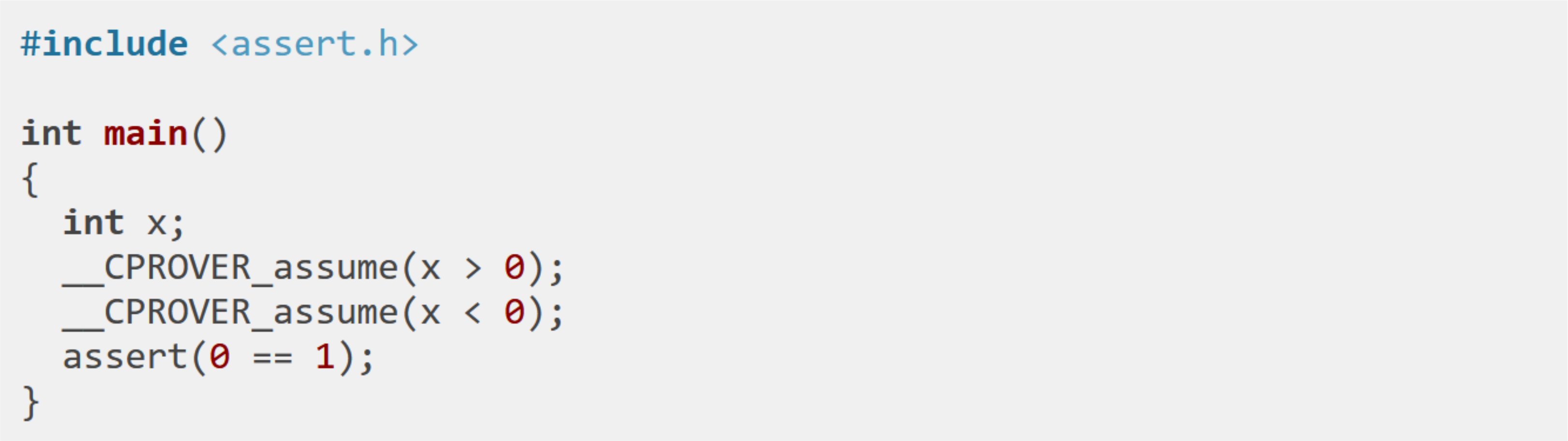
Presenter: Chenfeng Wei

Coverage

- *“Coverage is a metric that helps you understand how much of your source is tested*
- Common coverage criteria include branch coverage, condition coverage, Modified Condition/Decision Coverage (MC/DC) …

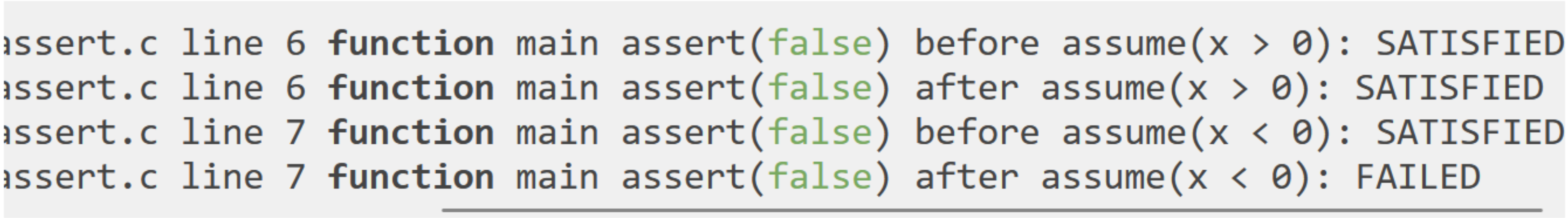
Coverage in Verifiers

- CBMC: give coverage information regarding **__CPROVER_assume** statements
  - to check which assume statements may have led to an emptying of the search state space, resulting in assert statements being vacuously passed.



```c
#include <assert.h>

int main()
{
  int x;
  __CPROVER_assume(x > 0);
  __CPROVER_assume(x < 0);
  assert(0 == 1);
}
```

CBMC invoked with `cbmc --cover assume test.c` will report:



```
assert.c line 6 function main assert(false) before assume(x > 0): SATISFIED
assert.c line 6 function main assert(false) after assume(x > 0): SATISFIED
assert.c line 7 function main assert(false) before assume(x < 0): SATISFIED
assert.c line 7 function main assert(false) after assume(x < 0): FAILED
```

- This indicates that this specific assume statement (on the line reported) is one that is emptying the search space for model checking

Coverage

- CBMC: give coverage information regarding **__CPROVER_assume** statements
  - to check which assume statements may have led to an emptying of the search state space, resulting in assert statements being vaccuously passed.
- ESBMC: give coverage information regarding **__ESBMC_assert(0)** statements
  - Automatically instrumented by ESBMC
  - Report “**Assertion Coverage**”

Coverage Criteria: Assertions

- **Assertion-based coverage** is a method of measuring the quality of functional verification of program designs using **formal verification** techniques.
- It involves writing **assertions**, which are formal specifications of the expected behavior of the design, and then analyzing the coverage of those assertions over the design.

Why Assertions Coverage in ESBMC?

- ESBMC is good at **adding** assertions
  - Property checking (e.g. assert(arr_bounds ≥ 0);)
- ESBMC is good at **mutating** assertions
  - API provided to mutate assertions in Goto level
- ESBMC is good at **verifying** assertions

Strategies

- **add-false-assert**:
  - this inserts a false assertion at the beginning of each function/branch and the end of each function/branch
- **make-assert-false**:
  - this converts every assertion to false

Before jumping into coverage …

- ESBMC utilizes some features from **multi-property**, which need to be explain first

What is ESBMC Multi-Property

- ESBMC can verify the satisfiability of all the claims of a given bound.
- During this multi-property verification, ESBMC does not terminate when a counterexample is found; instead, it continues to run until all bugs have been discovered

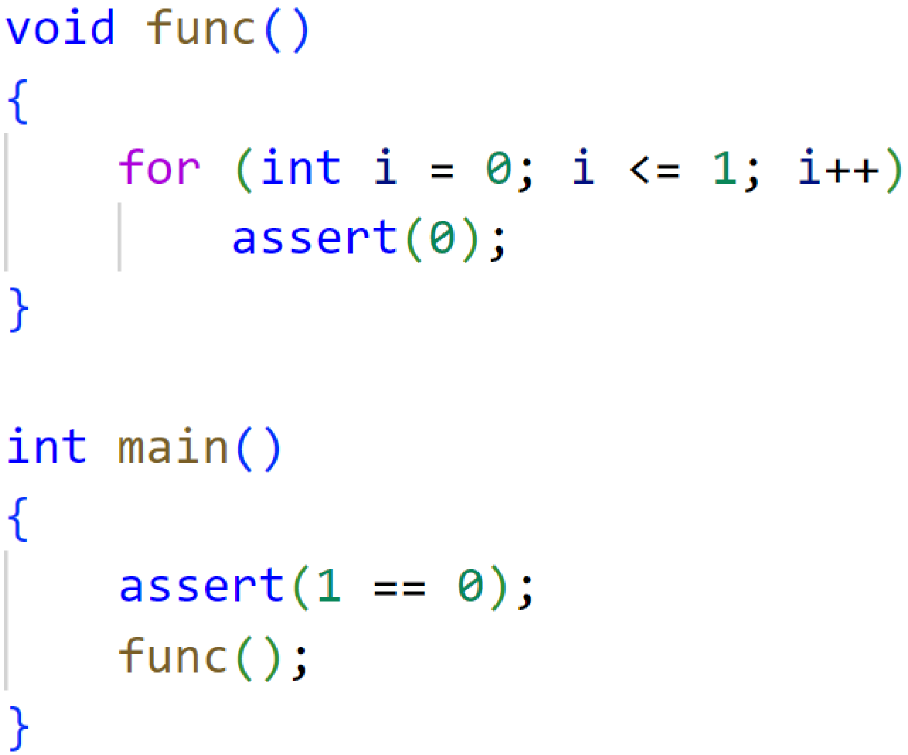
Options

- **multi-property:** verifies the satisfiability of all claims of the current bound.

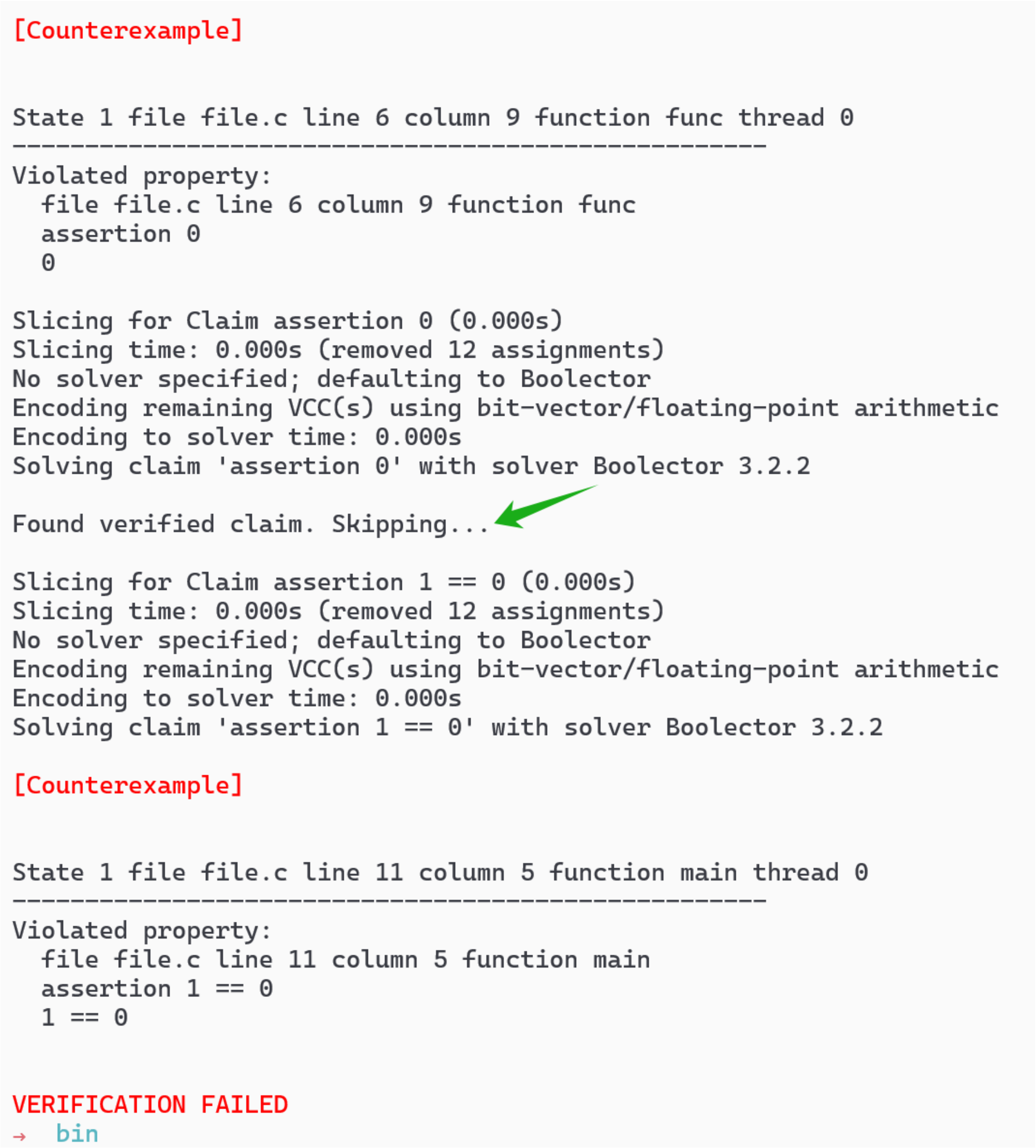
Suboptions

- **multi-fail-fast n:** stops after first **n** VCC violation found in multi-property mode
  - A debug option to check if a solver was stuck
- **keep-verified-claims:** do not skip verified claims in multi-property verification.
  - With this option enabled, the assertions inside the loop body will be verified repeatedly during the unwinding.
  - with this option disabled, each claim will only get verified once.

Example



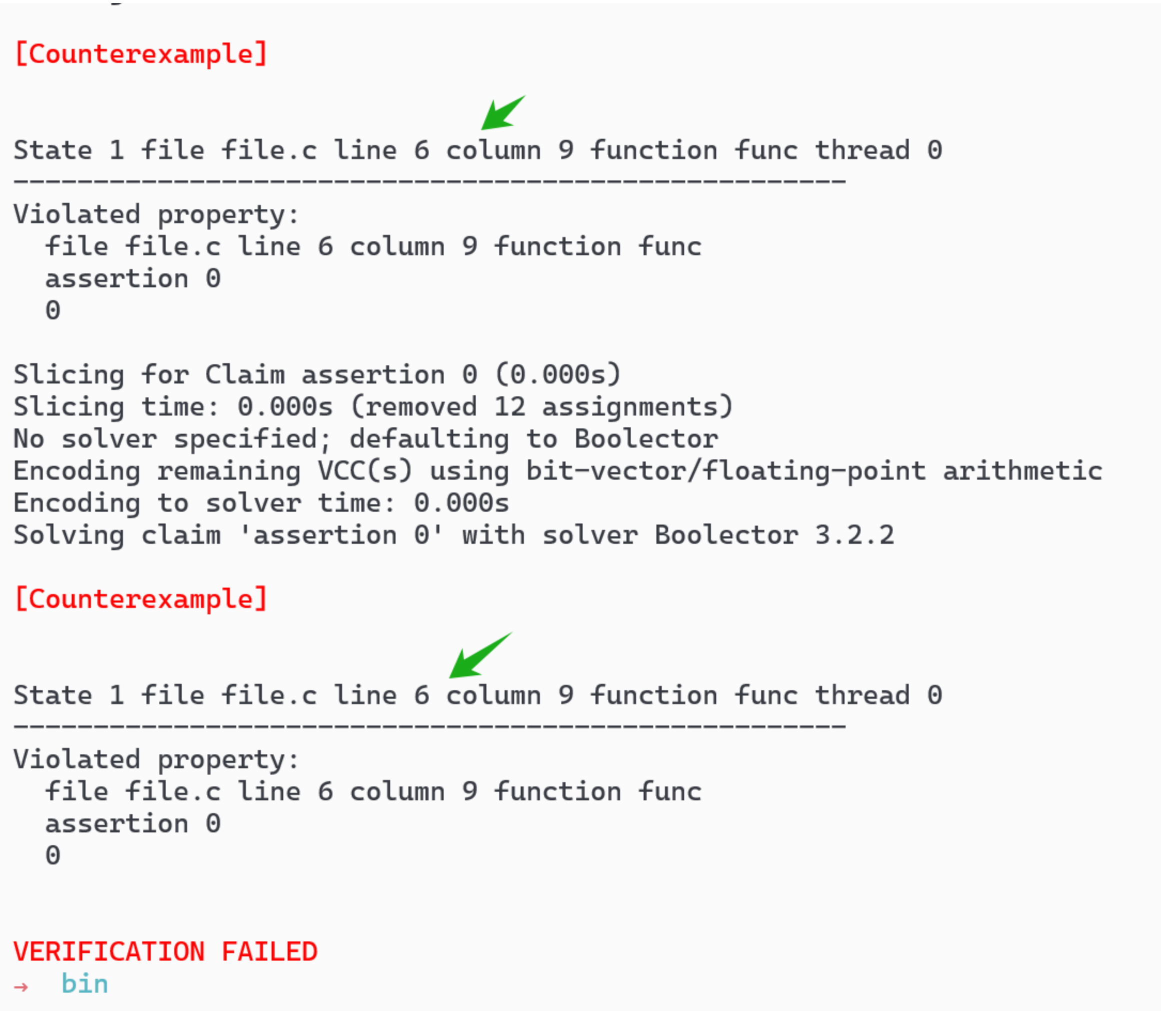
```
void func()
{
    for (int i = 0; i <= 1; i++)
        assert(0);
}

int main()
{
    assert(1 == 0);
    func();
}
```



```
[Counterexample]


State 1 file file.c line 6 column 9 function func thread 0
----------------------------------------------------
Violated property:
  file file.c line 6 column 9 function func
  assertion 0
  0

Slicing for Claim assertion 0 (0.000s)
Slicing time: 0.000s (removed 12 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'assertion 0' with solver Boolector 3.2.2

Found verified claim. Skipping...

Slicing for Claim assertion 1 == 0 (0.000s)
Slicing time: 0.000s (removed 12 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'assertion 1 == 0' with solver Boolector 3.2.2

[Counterexample]


State 1 file file.c line 11 column 5 function main thread 0
----------------------------------------------------
Violated property:
  file file.c line 11 column 5 function main
  assertion 1 == 0
  1 == 0


VERIFICATION FAILED
→  bin
```

./esbmc file.c --unwind 3
--multi-property --color

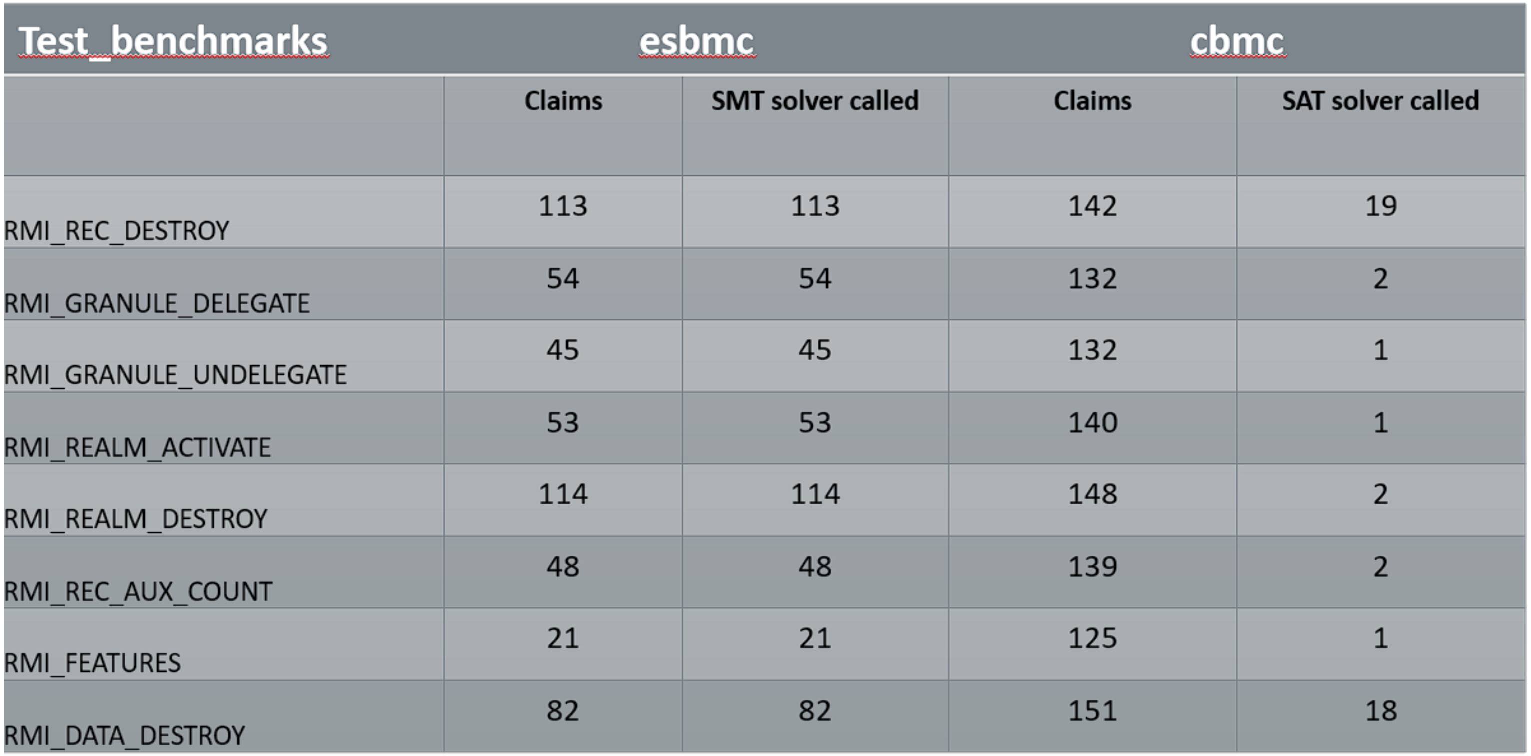
./esbmc file.c --unwind 3 --multi-property --keep-verified-claims --multi-fail-fast 2 --color



```
[Counterexample]


State 1 file file.c line 6 column 9 function func thread 0
----------------------------------------------------
Violated property:
  file file.c line 6 column 9 function func
  assertion 0
  0

Slicing for Claim assertion 0 (0.000s)
Slicing time: 0.000s (removed 12 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'assertion 0' with solver Boolector 3.2.2

[Counterexample]


State 1 file file.c line 6 column 9 function func thread 0
----------------------------------------------------
Violated property:
  file file.c line 6 column 9 function func
  assertion 0
  0


VERIFICATION FAILED
→  bin
```

Future Plan

- Reduce solver called to save time



Test_benchmarks	esbmc		cbmc	
	Claims	SMT solver called	Claims	SAT solver called
RMI_REC_DESTROY	113	113	142	19
RMI_GRANULE_DELEGATE	54	54	132	2
RMI_GRANULE_UNDELEGATE	45	45	132	1
RMI_REALM_ACTIVATE	53	53	140	1
RMI_REALM_DESTROY	114	114	148	2
RMI_REC_AUX_COUNT	48	48	139	2
RMI_FEATURES	21	21	125	1
RMI_DATA_DESTROY	82	82	151	18

Coverage Criteria: Assertions

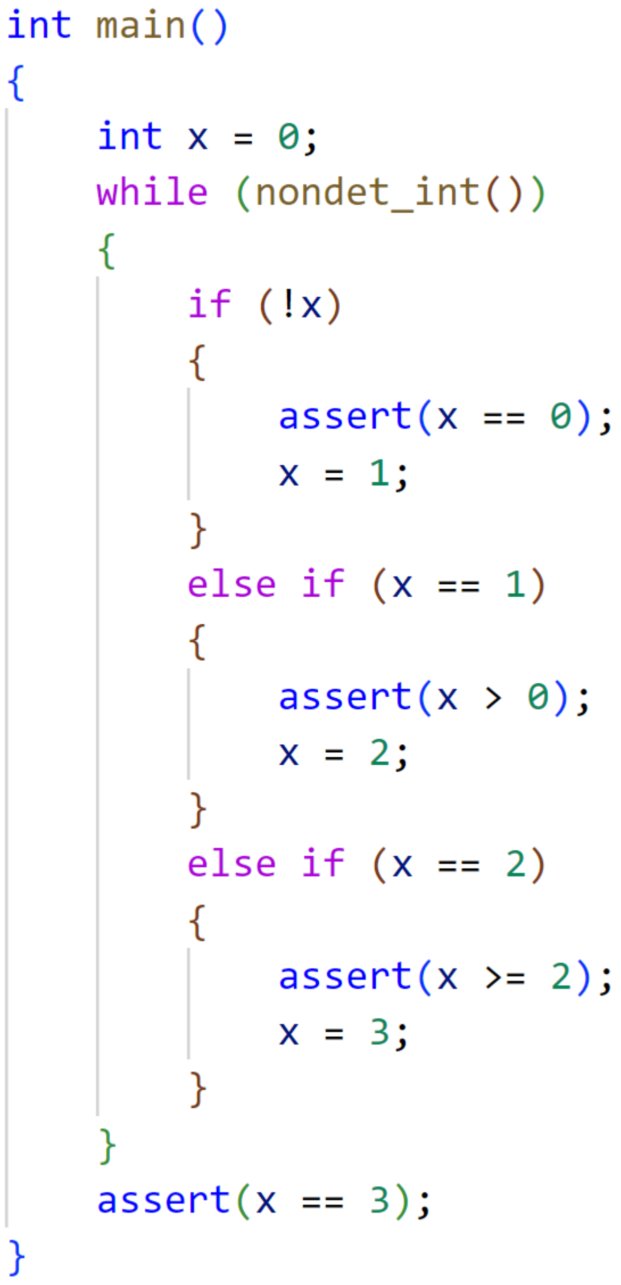
- **Assertion-based coverage** is a method of measuring the quality of functional verification of program designs using **formal verification** techniques.
- It involves writing **assertions**, which are formal specifications of the expected behavior of the design, and then analyzing the coverage of those assertions over the design.
- In ESBMC, we show **assertion instances coverage**!

Control Options

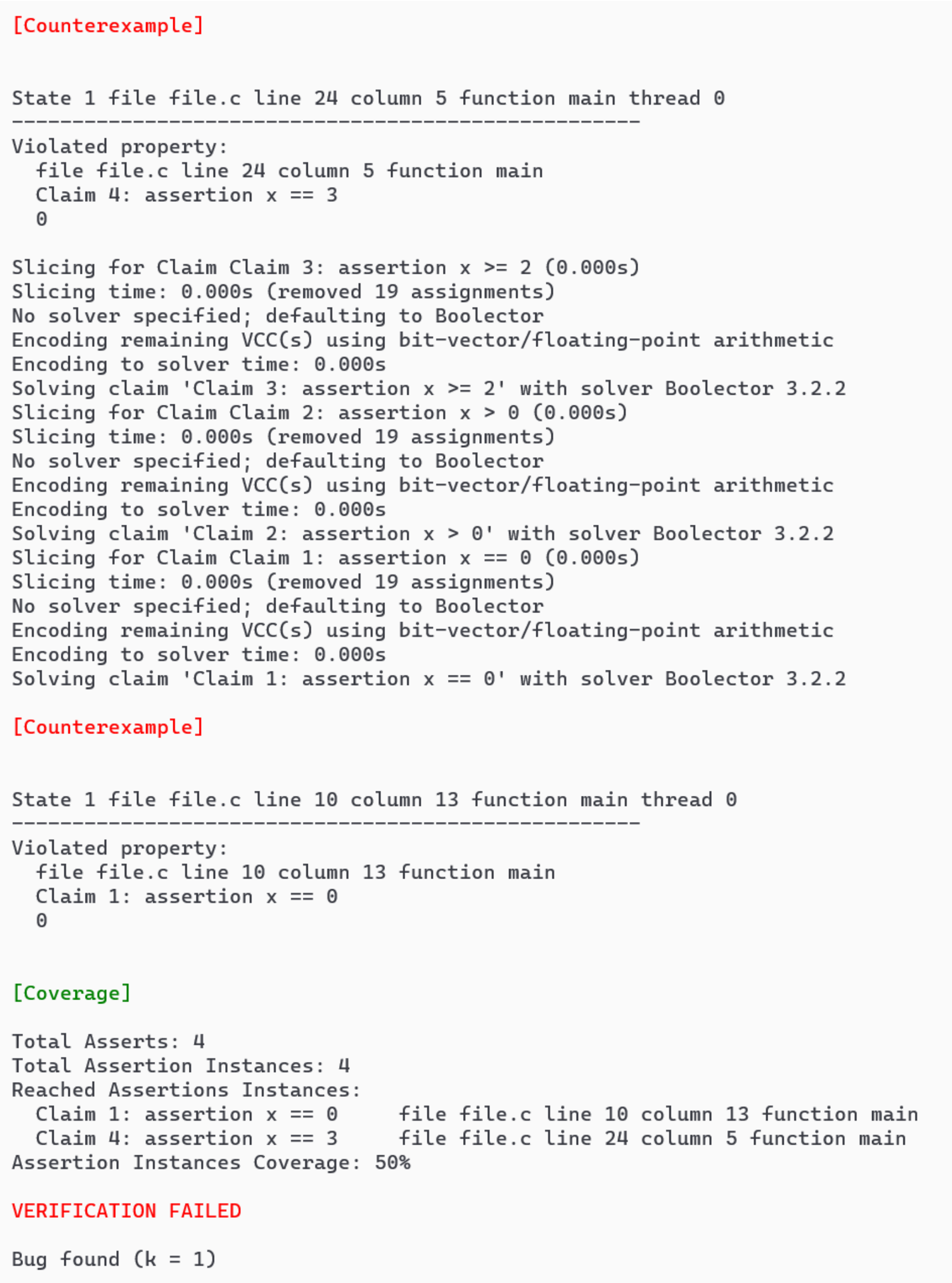
- **goto-coverage**:
  - this activates **--make-assert-false** and **--multi-property**, deactivates **--keep-verified-claims**, and shows the coverage of assertion instances
- **goto-coverage-claims**:
  - this activates **--goto-coverage** and shows all reached claim instances

Example 1

- esbmc file.c --k-induction --goto-coverage-claims
- Inside, the make-assert-false function will convert all asserts to assert

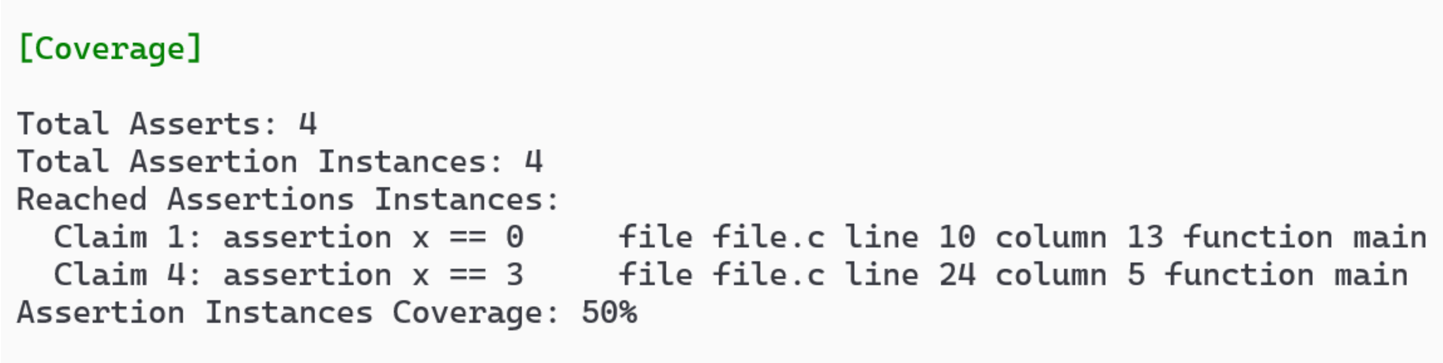


```c
int main()
{
    int x = 0;
    while (nondet_int())
    {
        if (!x)
        {
            assert(x == 0);
            x = 1;
        }
        else if (x == 1)
        {
            assert(x > 0);
            x = 2;
        }
        else if (x == 2)
        {
            assert(x >= 2);
            x = 3;
        }
    }
    assert(x == 3);
}
```



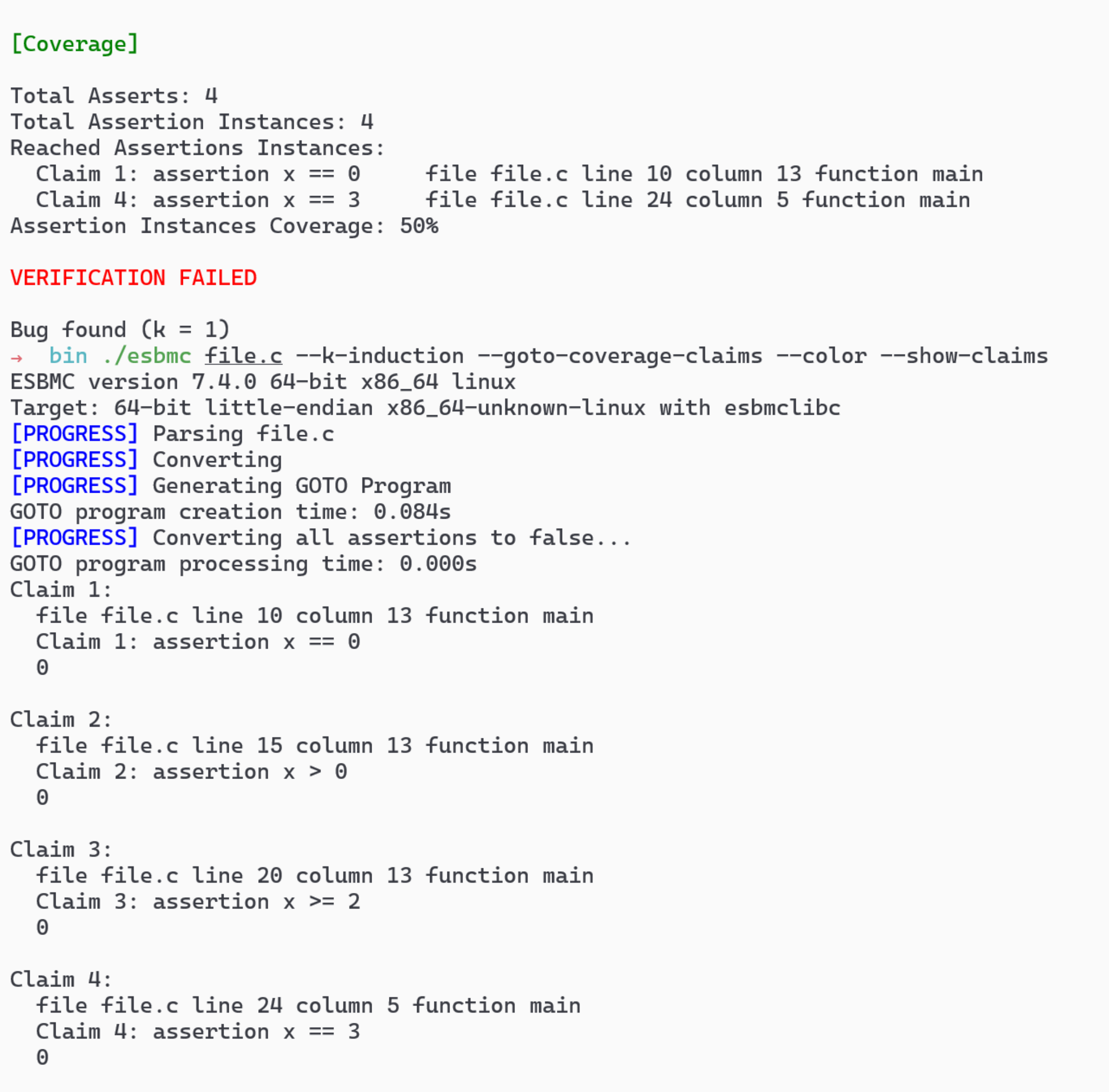
```
[Counterexample]


State 1 file file.c line 24 column 5 function main thread 0
----------------------------------------------------
Violated property:
  file file.c line 24 column 5 function main
  Claim 4: assertion x == 3
  0

Slicing for Claim Claim 3: assertion x >= 2 (0.000s)
Slicing time: 0.000s (removed 19 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'Claim 3: assertion x >= 2' with solver Boolector 3.2.2
Slicing for Claim Claim 2: assertion x > 0 (0.000s)
Slicing time: 0.000s (removed 19 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'Claim 2: assertion x > 0' with solver Boolector 3.2.2
Slicing for Claim Claim 1: assertion x == 0 (0.000s)
Slicing time: 0.000s (removed 19 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'Claim 1: assertion x == 0' with solver Boolector 3.2.2

[Counterexample]


State 1 file file.c line 10 column 13 function main thread 0
----------------------------------------------------
Violated property:
  file file.c line 10 column 13 function main
  Claim 1: assertion x == 0
  0


[Coverage]

Total Asserts: 4
Total Assertion Instances: 4
Reached Assertions Instances:
  Claim 1: assertion x == 0     file file.c line 10 column 13 function main
  Claim 4: assertion x == 3     file file.c line 24 column 5 function main
Assertion Instances Coverage: 50%

VERIFICATION FAILED

Bug found (k = 1)
```

Explain



```
[Coverage]

Total Asserts: 4
Total Assertion Instances: 4
Reached Assertions Instances:
  Claim 1: assertion x == 0     file file.c line 10 column 13 function main
  Claim 4: assertion x == 3     file file.c line 24 column 5 function main
Assertion Instances Coverage: 50%
```

- **Total Asserts:** the total number of assertions that are contained in the flow that ESBMC covers.
- **Total Assertion Instances:** the number of times that assertion can be triggered after ESBMC folds the code. For example, if a loop with 4 iterations contains an assertion, this assertion has 4 instances
- The **coverage** is obtained by dividing reached assertion instances by total assertion instances.
- The **unreached claims** can be checked by comparing them with the output of **--show-claims**.



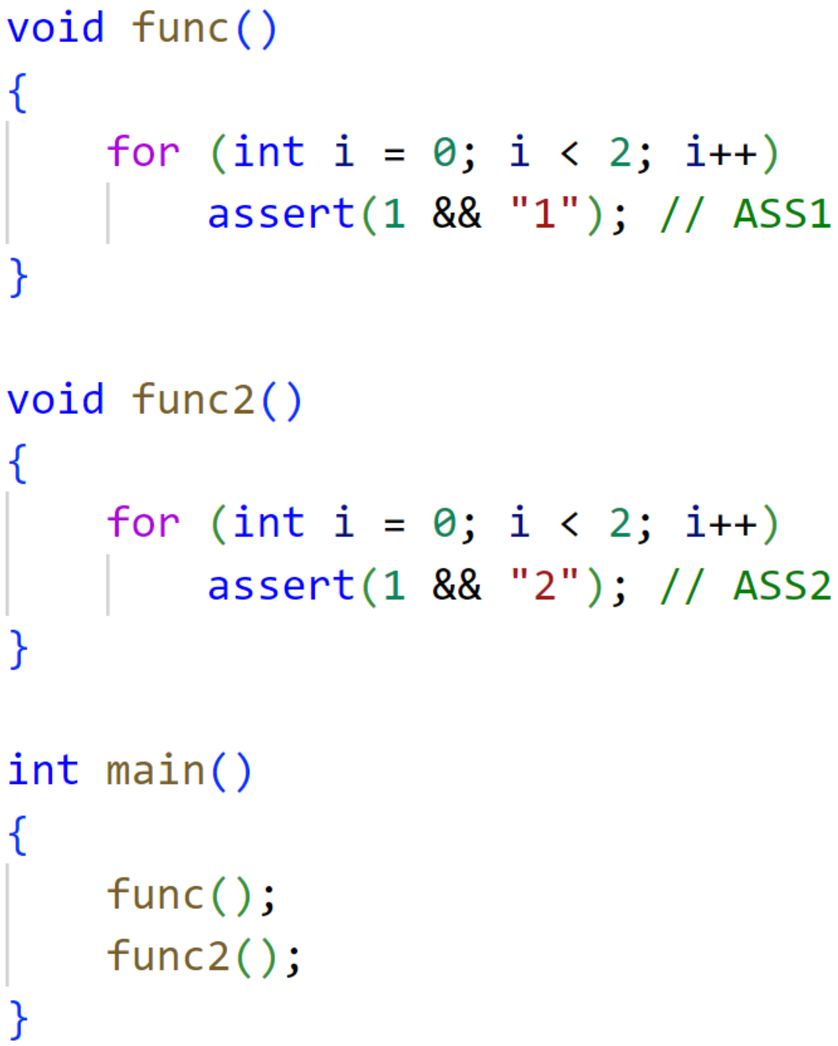
```
[Coverage]

Total Asserts: 4
Total Assertion Instances: 4
Reached Assertions Instances:
  Claim 1: assertion x == 0     file file.c line 10 column 13 function main
  Claim 4: assertion x == 3     file file.c line 24 column 5 function main
Assertion Instances Coverage: 50%

VERIFICATION FAILED

Bug found (k = 1)
→  bin ./esbmc file.c --k-induction --goto-coverage-claims --color --show-claims
ESBMC version 7.4.0 64-bit x86_64 linux
Target: 64-bit little-endian x86_64-unknown-linux with esbmclibc
[PROGRESS] Parsing file.c
[PROGRESS] Converting
[PROGRESS] Generating GOTO Program
GOTO program creation time: 0.084s
[PROGRESS] Converting all assertions to false...
GOTO program processing time: 0.000s
Claim 1:
  file file.c line 10 column 13 function main
  Claim 1: assertion x == 0
  0

Claim 2:
  file file.c line 15 column 13 function main
  Claim 2: assertion x > 0
  0

Claim 3:
  file file.c line 20 column 13 function main
  Claim 3: assertion x >= 2
  0

Claim 4:
  file file.c line 24 column 5 function main
  Claim 4: assertion x == 3
  0
```

Conclusion

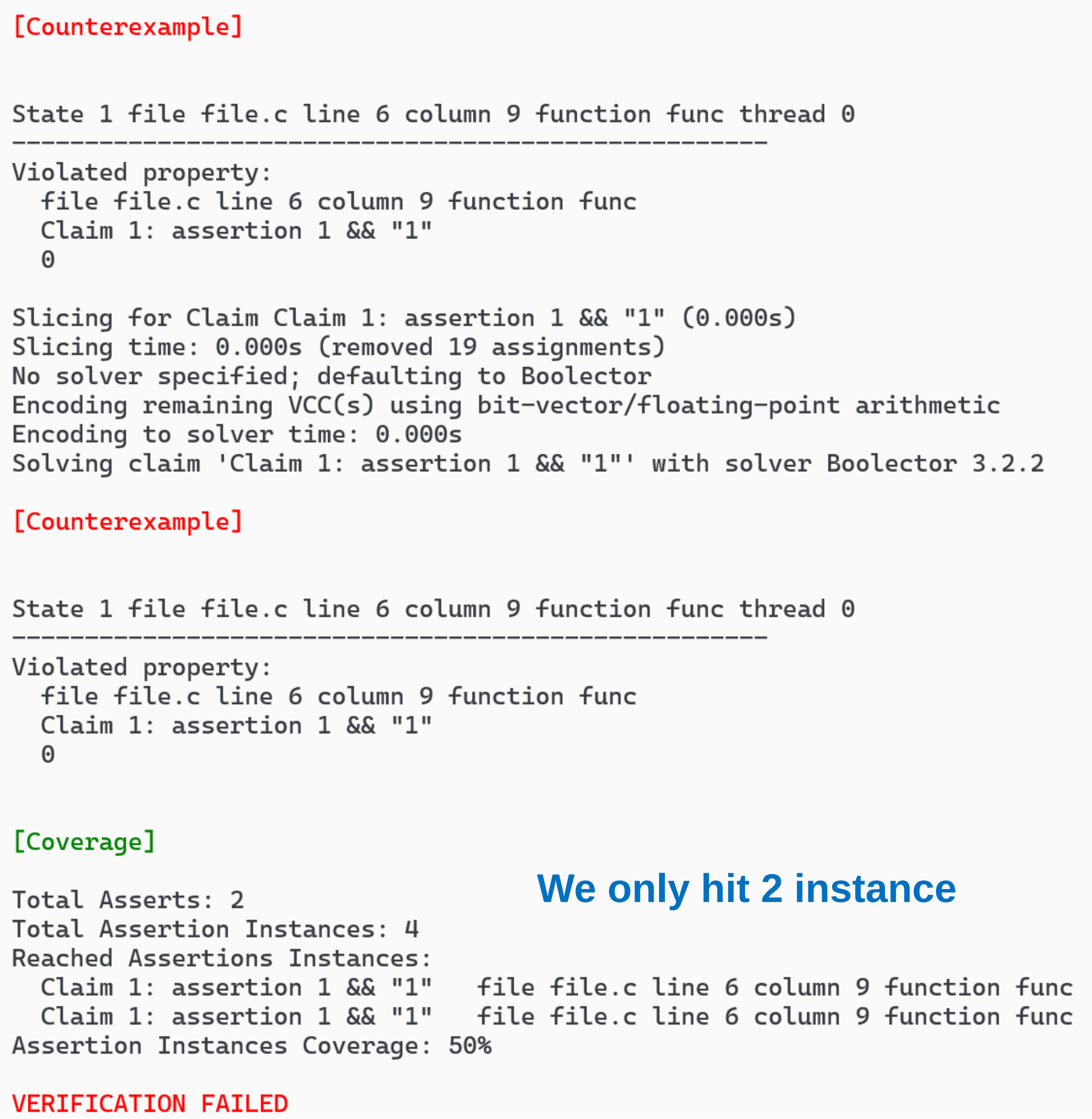
- Comparing our result with –show-claims, we find out in k-induction mode, we fail to reach two branches.

Example 2

- esbmc file.c --unwind 2 --no-unwinding-assertions --goto-coverage-claims
- 4 assertion instances in total



```
void func()
{
    for (int i = 0; i < 2; i++)
        assert(1 && "1"); // ASS1
}

void func2()
{
    for (int i = 0; i < 2; i++)
        assert(1 && "2"); // ASS2
}

int main()
{
    func();
    func2();
}
```

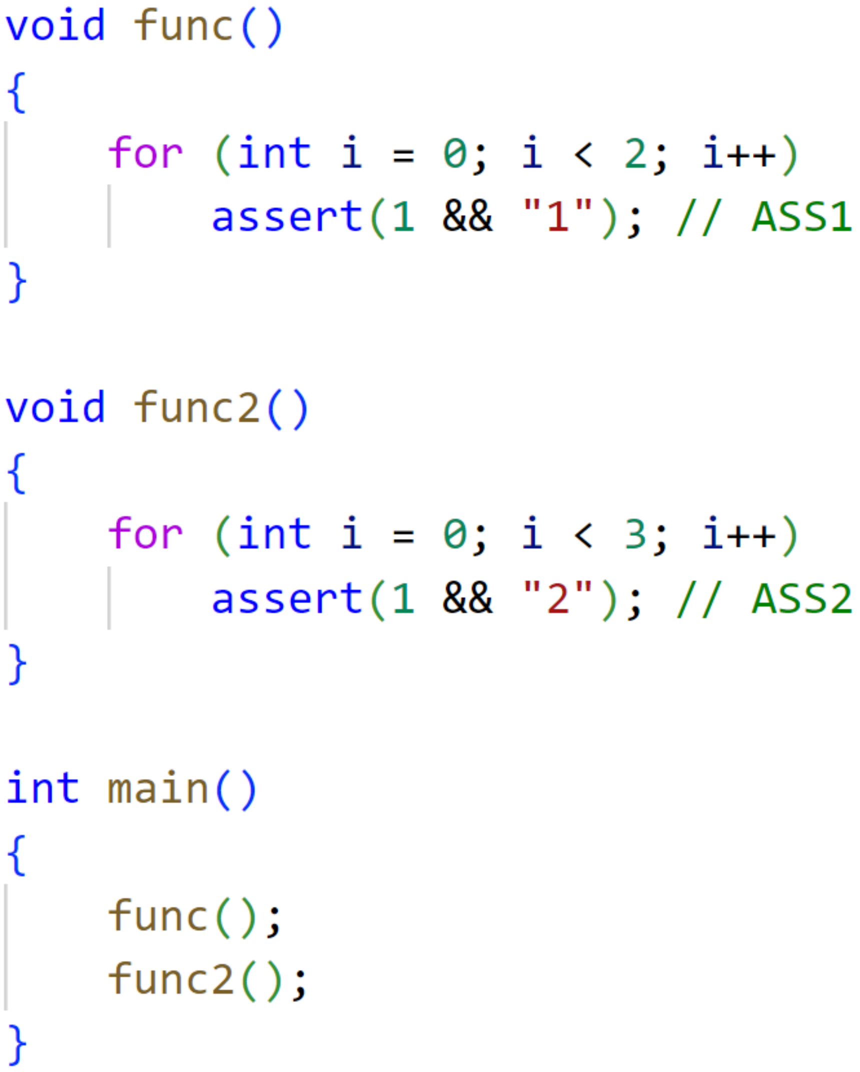


```
[Counterexample]


State 1 file file.c line 6 column 9 function func thread 0
----------------------------------------------------
Violated property:
  file file.c line 6 column 9 function func
  Claim 1: assertion 1 && "1"
  0

Slicing for Claim Claim 1: assertion 1 && "1" (0.000s)
Slicing time: 0.000s (removed 19 assignments)
No solver specified; defaulting to Boolector
Encoding remaining VCC(s) using bit-vector/floating-point arithmetic
Encoding to solver time: 0.000s
Solving claim 'Claim 1: assertion 1 && "1"' with solver Boolector 3.2.2

[Counterexample]


State 1 file file.c line 6 column 9 function func thread 0
----------------------------------------------------
Violated property:
  file file.c line 6 column 9 function func
  Claim 1: assertion 1 && "1"
  0


[Coverage]

Total Asserts: 2
Total Assertion Instances: 4
Reached Assertions Instances:
  Claim 1: assertion 1 && "1"   file file.c line 6 column 9 function func
  Claim 1: assertion 1 && "1"   file file.c line 6 column 9 function func
Assertion Instances Coverage: 50%

VERIFICATION FAILED
```

We only hit 2 instance

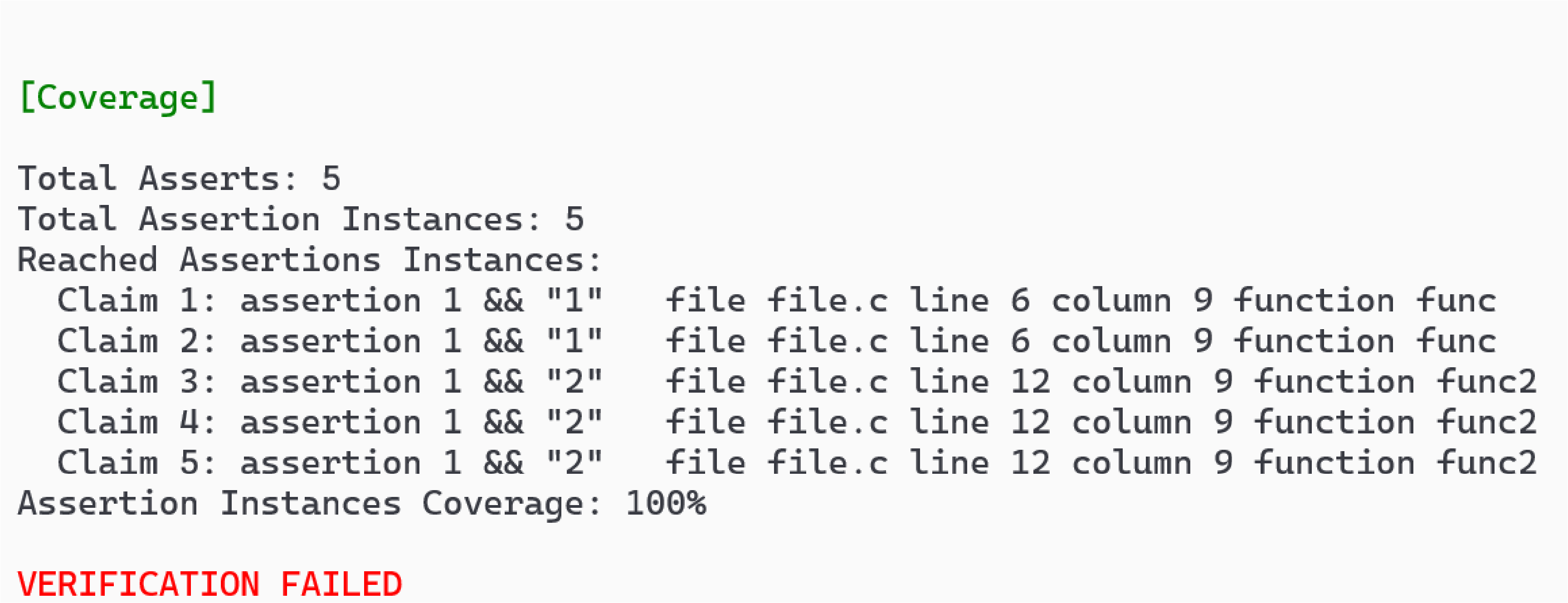
Example 3

- esbmc file.c –goto-unwind --goto-coverage-claims
- 5 assertion instances in total



```
void func()
{
    for (int i = 0; i < 2; i++)
        assert(1 && "1"); // ASS1
}

void func2()
{
    for (int i = 0; i < 3; i++)
        assert(1 && "2"); // ASS2
}

int main()
{
    func();
    func2();
}
```

Result



```
[Coverage]

Total Asserts: 5
Total Assertion Instances: 5
Reached Assertions Instances:
  Claim 1: assertion 1 && "1"   file file.c line 6 column 9 function func
  Claim 2: assertion 1 && "1"   file file.c line 6 column 9 function func
  Claim 3: assertion 1 && "2"   file file.c line 12 column 9 function func2
  Claim 4: assertion 1 && "2"   file file.c line 12 column 9 function func2
  Claim 5: assertion 1 && "2"   file file.c line 12 column 9 function func2
Assertion Instances Coverage: 100%

VERIFICATION FAILED
```

Conclusion

- We know which assert(s) remain unreached.
  - Which reflects the path that are unreached. (Width)
- We know which assert instances were reached (and unreached).
  - Which reflects the extent to which ESBMC has explored the loop/(…).(Depth)

MANCHESTER
1824
The University of Manchester

Thank you