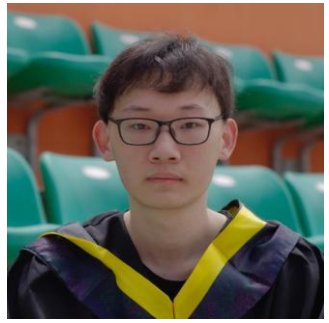
MANCHESTER 1824
The University of Manchester

ESBMC v7.7 ： Automating Branch Coverage Analysis Using CFG-Based Instrumentation and SMT Solving

Author: Chenfeng Wei
Chenfeng.wei@machester.ac.uk
Co-Authors: Tong Wu, Rafael Sa Menezes, Fedor Shmarov, Fatimah Aljaafari,
Sangharatna Godboley, Kaled Alshmrany,
Rosiane de Freitas, Lucas C. Cordeiro

Contributors



Mr. Chenfeng Wei

Mr. Tong Wu

Mr. Rafael Menezes

Dr. Fedor Shmarov

Dr. Fatimah Aljaafari

Dr.
Sangharatna Godboley

Dr.
Kaled Alshmrany

Dr.
Rosiane de Freitas

Dr.
Lucas C. Cordeiro

ESBMC

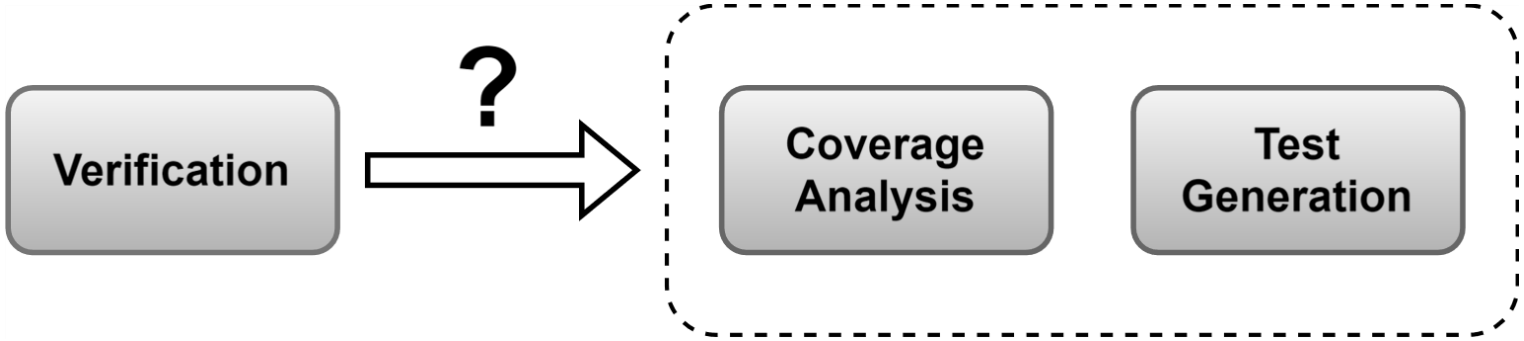
ESBMC Team

Motivation

ESBMC, a bounded model checking (BMC) verifier based on SMT solving, has proven its effectiveness in bug detection in recent competitions.

However, it has never participated in the evaluation of the Cover-Branches category due to the lack of:

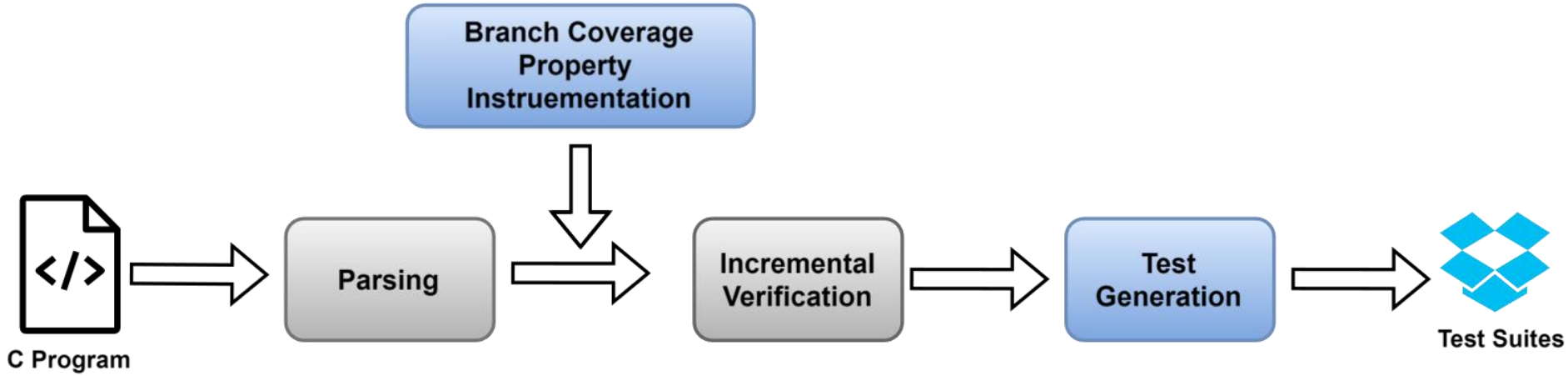
- Branch coverage analysis
- Automated test suite generation



Contribution

To bridge this gap, we present **ESBMC v7.7** with the following key contributions:

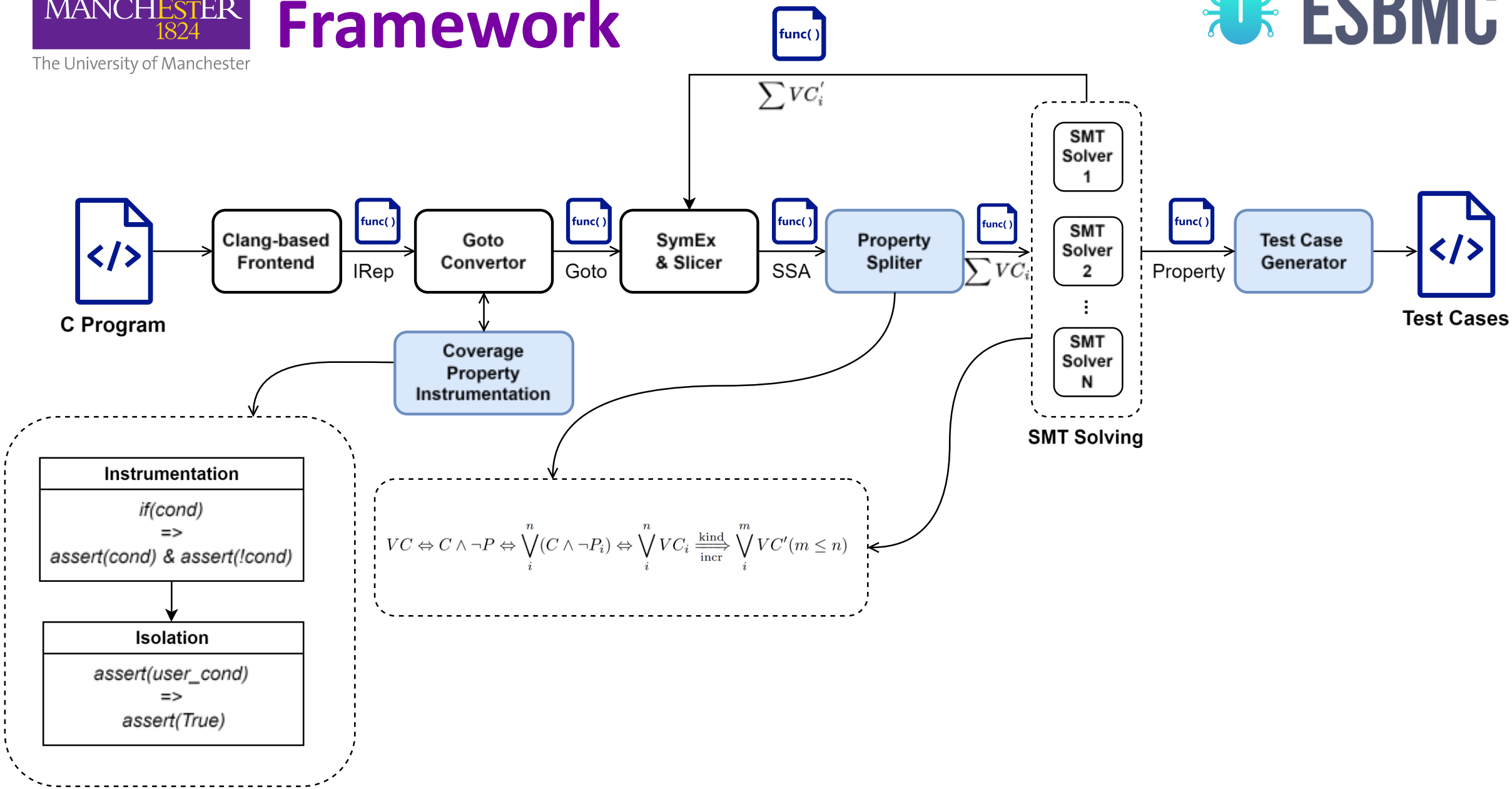
- Branch Coverage Instrumentation
  - Instrumentation & Isolation
- Incremental Multiple Property Verification
  - Property Splitting & Incremental Reasoning
  - Re-verification & Termination
  - Test Generation



MANCHESTER 1824
The University of Manchester

Framework

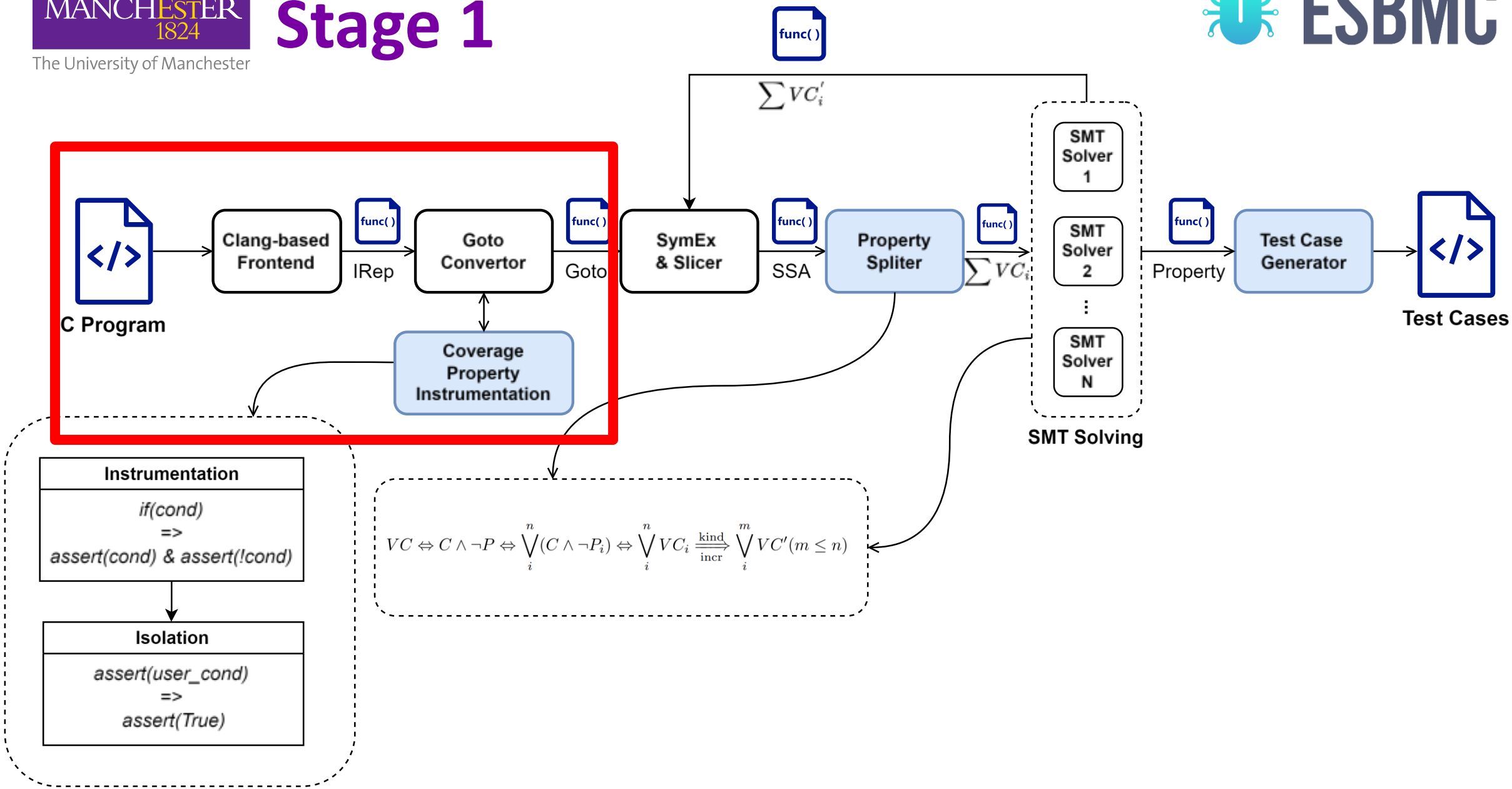
ESBMC



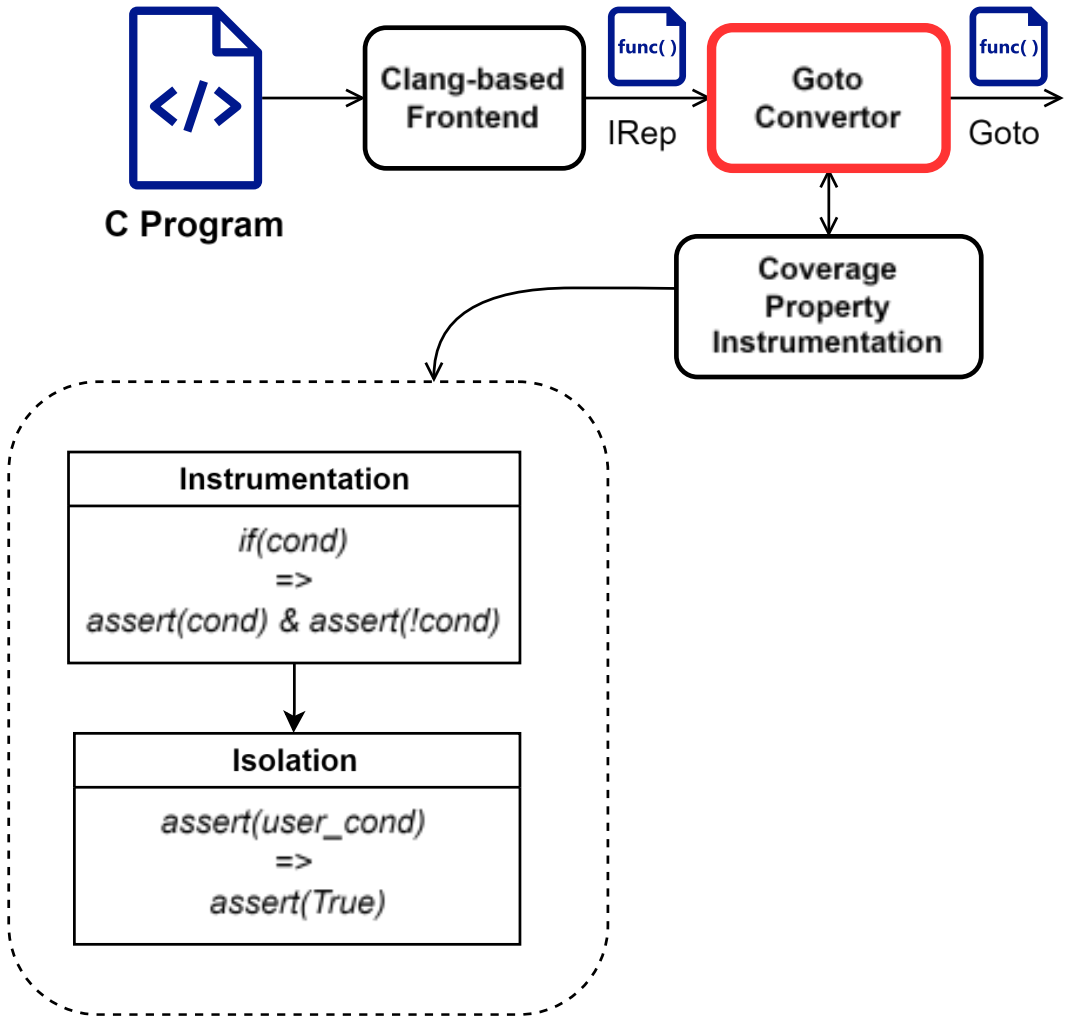
MANCHESTER 1824
The University of Manchester

Stage 1

ESBMC



Simplification



We apply our algorithm upon **GOTO program,** i.e., Control Flow Graph (CFG).

Simplification: In the Goto program, control-flow constructs such as **if-else** statements, **while** loops, **for** loops, and **switch-case** statements are normalized into **if-goto** structures.

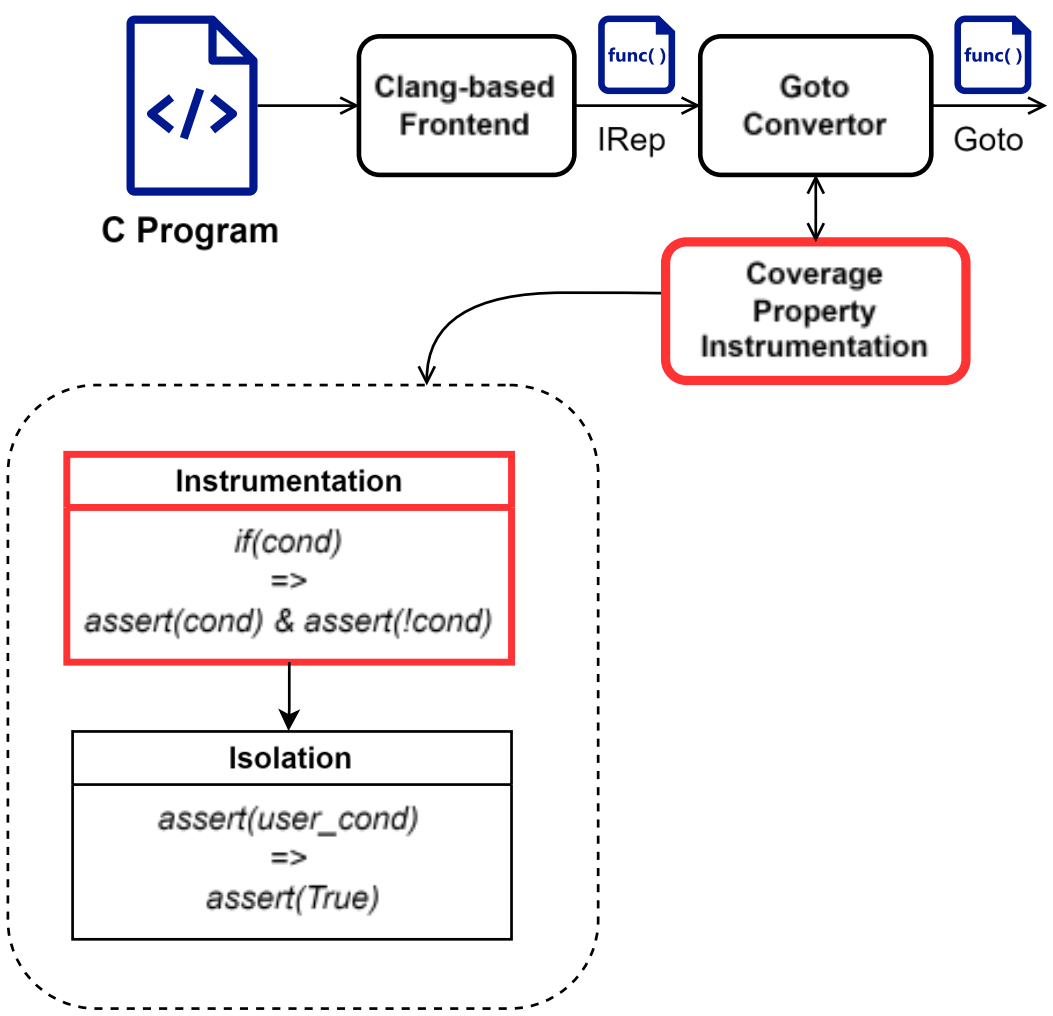
MANCHESTER 1824
The University of Manchester

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

```
esbmc exp.c --goto2c &> exp2.c
```

```c
int main()
{
    {
        int x = __VERIFIER_nondet_int();
    __ESBMC_goto_label_1:;
        _Bool return_value____VERIFIER_nondet_bool_1;
        return_value____VERIFIER_nondet_bool_1 = __VERIFIER_nondet_bool();
        if (!return_value____VERIFIER_nondet_bool_1) // while (nondet_bool())
            goto __ESBMC_goto_label_4;
        if (!(!(_Bool)x))                            // if (!x)
            goto __ESBMC_goto_label_2;
        assert(x == 0);
        assert(!overflow("+", x, 1));                // overflow-check
        x = x + 1;
        goto __ESBMC_goto_label_3;
    __ESBMC_goto_label_2:;
        if (!(x == 1))                               // else if (x == 1)
            goto __ESBMC_goto_label_3;
        assert(x > 0);
        x = 2;
    __ESBMC_goto_label_3:;
        goto __ESBMC_goto_label_1;
    __ESBMC_goto_label_4:;
        __VERIFIER_assert((int)(x == 2));
    }
}
```

Instrumentation



Instrumentation: to entry a branch guarded by condition like `if(cond)`, there must exist an assignment that satisfies cond. This is equivalent to checking if a counterexample satisfies `assert(!cond)`

- True Branch: the body executes
- False Branch: the body is skipped

```
if(cond) => True branch:  assert(!cond)
         => False branch: assert(!(!cond))
```

MANCHESTER 1824
The University of Manchester

Instrumentation

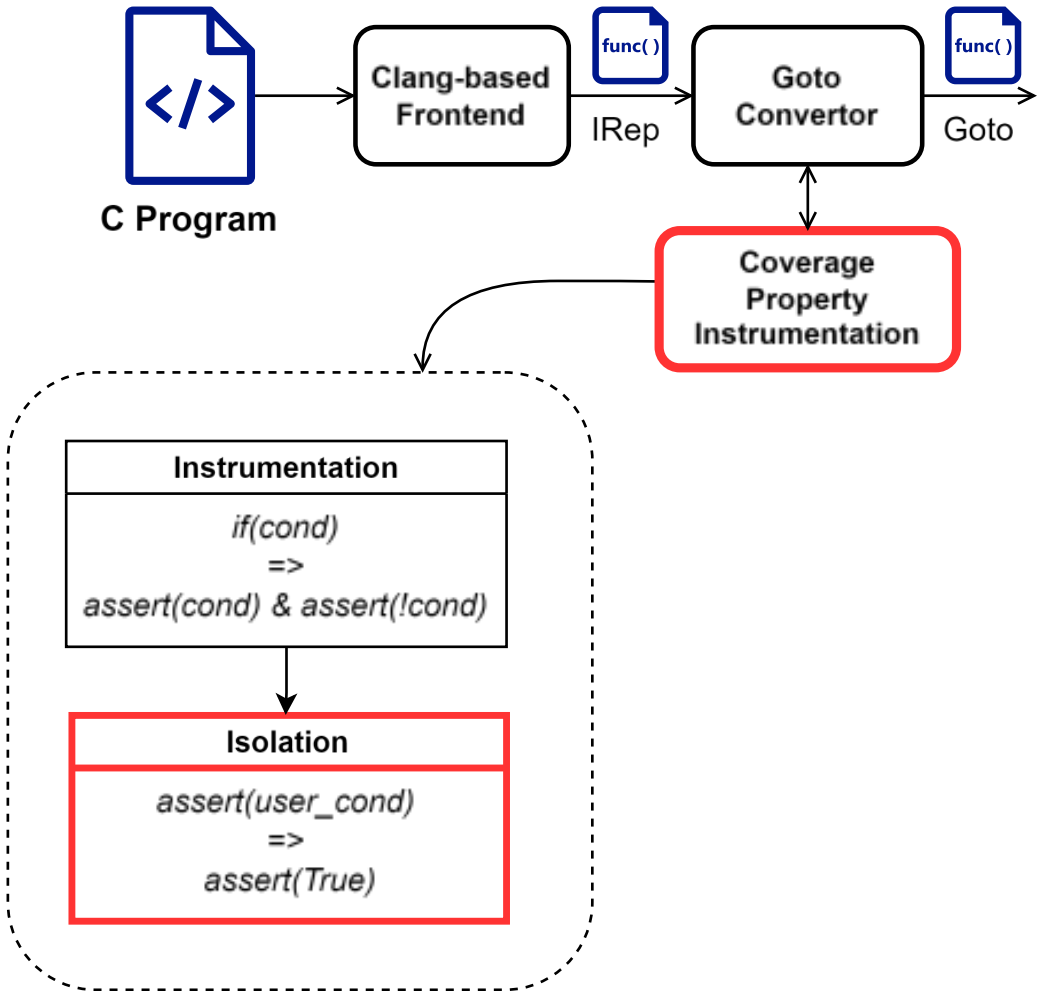
ESBMC

```
_Bool return_value___VERIFIER_nondet_bool_1;
return_value___VERIFIER_nondet_bool_1 = __VERIFIER_nondet_bool();
assert(!return_value___VERIFIER_nondet_bool_1);    // !return_value$___VERIFIER_nondet_bool$1
assert(!(!return_value___VERIFIER_nondet_bool_1)); // !(!return_value$___VERIFIER_nondet_bool$1)
if (!return_value___VERIFIER_nondet_bool_1)
    goto __ESBMC_goto_label_4;
```

```
assert(!(!(_Bool)x));    // !(!(_Bool)x)
assert(!(!(!(_Bool)x))); // !(!(!(_Bool)x))
if (!(!(_Bool)x))
    goto __ESBMC_goto_label_2;
```

```
assert(!(x == 1));    // !(x == 1)
assert(!(!(x == 1))); // !(!(x == 1))
if (!(x == 1))
    goto __ESBMC_goto_label_3;
```

Isolation



Isolation: Potential interferences are excluded to isolate the analysis of instrumented coverage properties from others.

- User-defined properties, i.e. assertions, are converted into tautologies

```
assert(a[i]>0) => assert(True)
```

- Internal safety checks within ESBMC are disabled

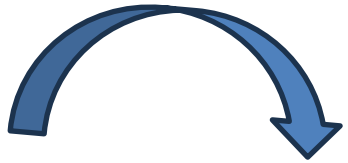
```
/* assert(y!=0) */
Double z = x / y
```

MANCHESTER 1824
The University of Manchester

Isolation

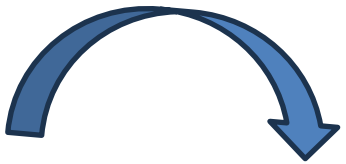
ESBMC

```
__ESBMC_goto_label_4:;
    __VERIFIER_assert((int)(x == 2));
}
```



```
__ESBMC_goto_label_4:;
    __VERIFIER_assert(1);  // x == 2
}
```

```
assert(x == 0);
assert(!overflow("+", x, 1));
x = x + 1;
```

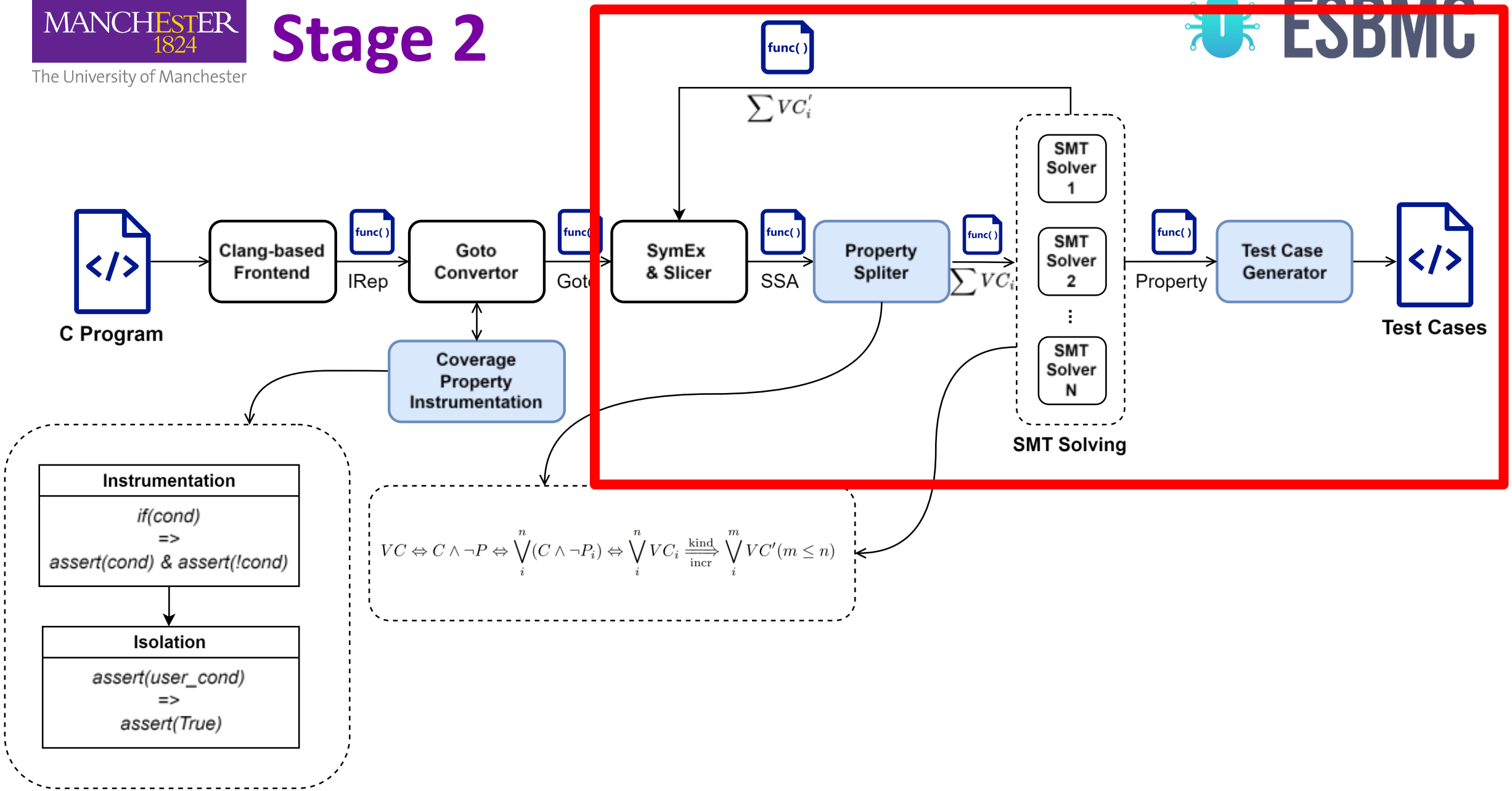


```
assert(1);
/* assert(!overflow("+", x, 1)); */
x = x + 1;
```

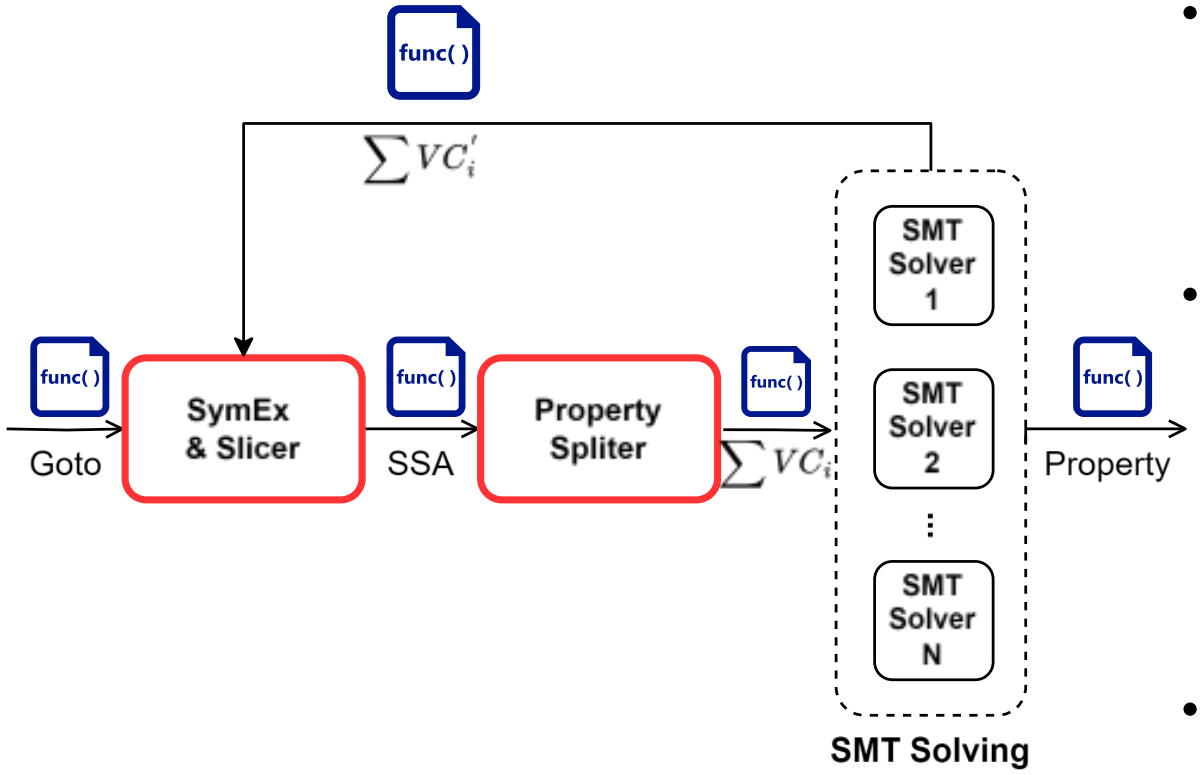
MANCHESTER 1824
The University of Manchester

Stage 2

ESBMC



SymEx & Splitting



- The CFG get symbolic executed within defined bounds (e.g., through loop unrolling) and is eventually encoded as a **verification condition (VC)**.

- In ESBMC, the VC is an SMT formula incorporating:
  - **Constraints** (execution conditions, **C**)
  - **Properties** (expected behaviours, **P**)

$$VC \Leftrightarrow C \wedge \neg P$$

- To let ESBMC verify multiple properties, we split the property ***P*** into a set of unit properties $\boldsymbol{P_i}$

$$C \wedge \neg P \Leftrightarrow \bigvee_{i}^{n} (C \wedge \neg P_i) \Leftrightarrow \bigvee_{i}^{n} VC_i$$

Incremental Reasoning

Issue:

- Normally, BMC verifies system behavior only up to a fixed bound ***k***, once this threshold is reached, it terminates. The verification result becomes unknown.
- As a consequence, some branch paths may be missed.

How about we set a relatively large ***k*** (e.g. ***100***)?

- **No guarantee of soundness**: Larger ***k*** increases depth but doesn't ensure that all paths are covered or that unreachability is proved.
- **Inefficiency**: Large bounds might lead to state-space explosion

https://ssvlab.github.io/esbmc/documentation.html#k-induction

Incremental Reasoning

Aid: Use incremental reasoning (e.g. k-induction) to automatically extend verification beyond bound ***k***

- It checks that a property holds up to a bound (k)
- It proves that if it holds at (k), it also holds at $(k+1)$

In ESBMC, this can be summaries as three steps:

- **Base Case**: Check if the program is correct up to (k) steps (normal BMC).
- **Forward Reasoning**: prove that all loops in the program were fully unrolled.
- **Inductive Step**: If it is good up to (k), prove it's still good for $(k+1)$.

$$\begin{aligned} \neg\ & B(k) && \rightarrow && \text{program contains bug} \\ B(k) \ \wedge\ & F(k) && \rightarrow && \text{program is correct} \\ B(k) \ \wedge\ & I(k) && \rightarrow && \text{program is correct} \end{aligned}$$

https://ssvlab.github.io/esbmc/documentation.html#k-induction

MANCHESTER 1824
The University of Manchester

Recall: Instrumentation

ESBMC

```
_Bool return_value___VERIFIER_nondet_bool_1;
return_value___VERIFIER_nondet_bool_1 = __VERIFIER_nondet_bool();
assert(!return_value___VERIFIER_nondet_bool_1);    // !return_value$___VERIFIER_nondet_bool$1
assert(!(!return_value___VERIFIER_nondet_bool_1)); // !(!return_value$___VERIFIER_nondet_bool$1)
if (!return_value___VERIFIER_nondet_bool_1)
    goto __ESBMC_goto_label_4;
```

```
assert(!(!(_Bool)x));    // !(!(_Bool)x)
assert(!(!(!(_Bool)x))); // !(!(!(_Bool)x))
if (!(!(_Bool)x))
    goto __ESBMC_goto_label_2;
```

```
assert(!(x == 1));    // !(x == 1)
assert(!(!(x == 1))); // !(!(x == 1))
if (!(x == 1))
    goto __ESBMC_goto_label_3;
```

Incremental Reasoning

```
!(!return_value$___VERIFIER_nondet_bool$1)

!return_value$___VERIFIER_nondet_bool$1

!(!(!(_Bool)x))

!(!(_Bool)x)

!(!(x == 1))

!(x == 1)
```

K=1

```
!(!(!(_Bool)x))

!(!(x == 1))

!(x == 1)
```

K=2

```
!(!(x == 1))
```

K=3

K=4 **Terminate**

Termination:

a) all remaining coverage properties are proven during forward reasoning

b) all properties are reduced to tautologies and removed through slicing, leaving no properties for further verification

Re-verification: if any property P_i remains unknown, a verification re-run is initiated

$$\bigvee_{i}^{n} VC_i \xrightarrow[\text{incr}]{\text{kind}} \bigvee_{i}^{m} VC'(m \leq n)$$

MANCHESTER 1824
The University of Manchester

Result

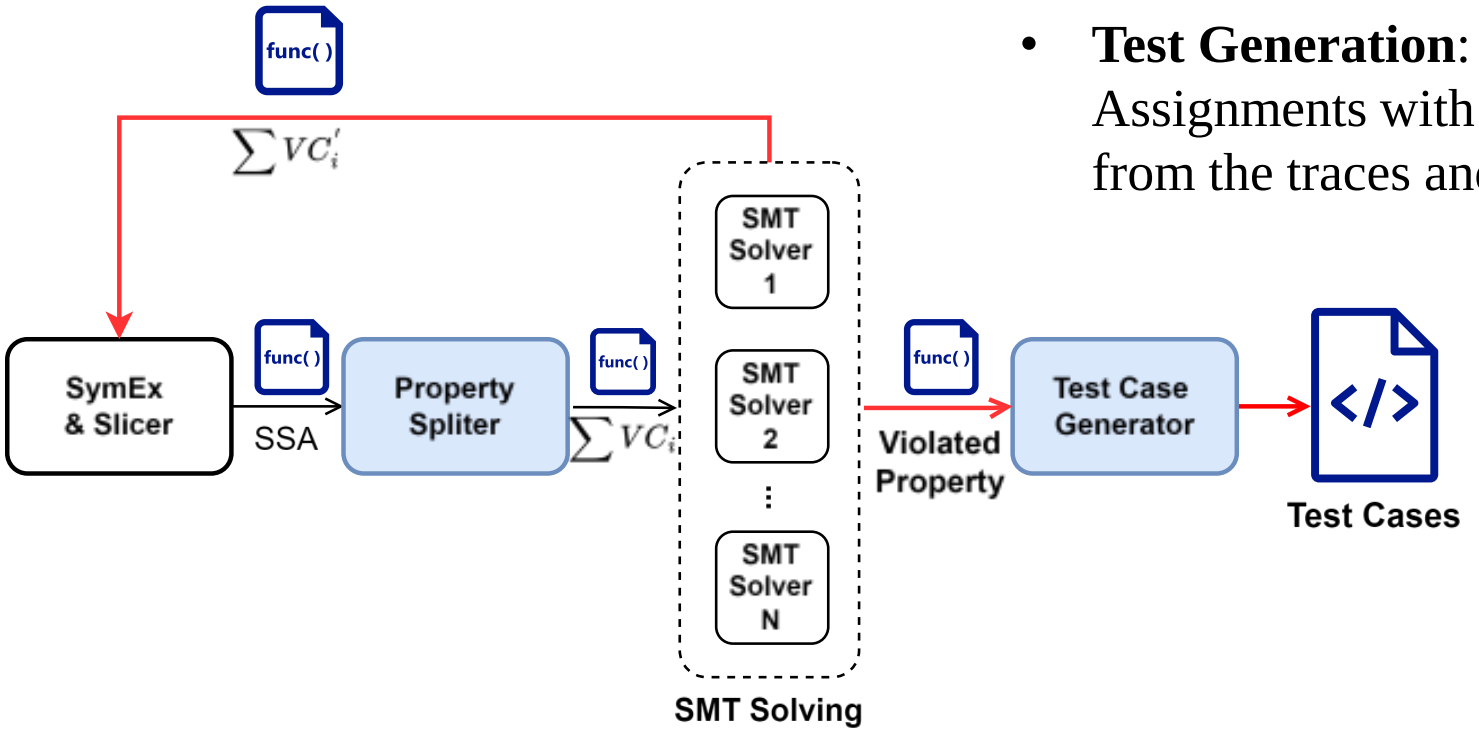
ESBMC

```
Solution found by the inductive step (k = 4)

[Coverage]

Branches : 6
Reached : 6
  !(!(!(_Bool)x))       file exp.c line 11 column 5 function main
  !(x == 1)     file exp.c line 16 column 10 function main
  !return_value$___VERIFIER_nondet_bool$1       file exp.c line 9 column 3 function main
  !(!return_value$___VERIFIER_nondet_bool$1)    file exp.c line 9 column 3 function main
  !(!(x == 1))  file exp.c line 16 column 10 function main
  !(!(_Bool)x)  file exp.c line 11 column 5 function main
Branch Coverage: 100%
```

Test Generation

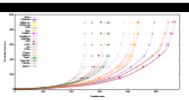


- **Test Generation**: whenever a property P_i violation is reported. Assignments with nondeterministic initial values are extracted from the traces and transformed into corresponding test suites.

```
testcase-0.xml
testcase-1.xml
testcase-2.xml
testcase-3.xml
testcase-4.xml
testcase-5.xml
```

Result

ESBMC ranked around 7th–8th among all participants in Cover-Branches category at Test-Comp 2025.

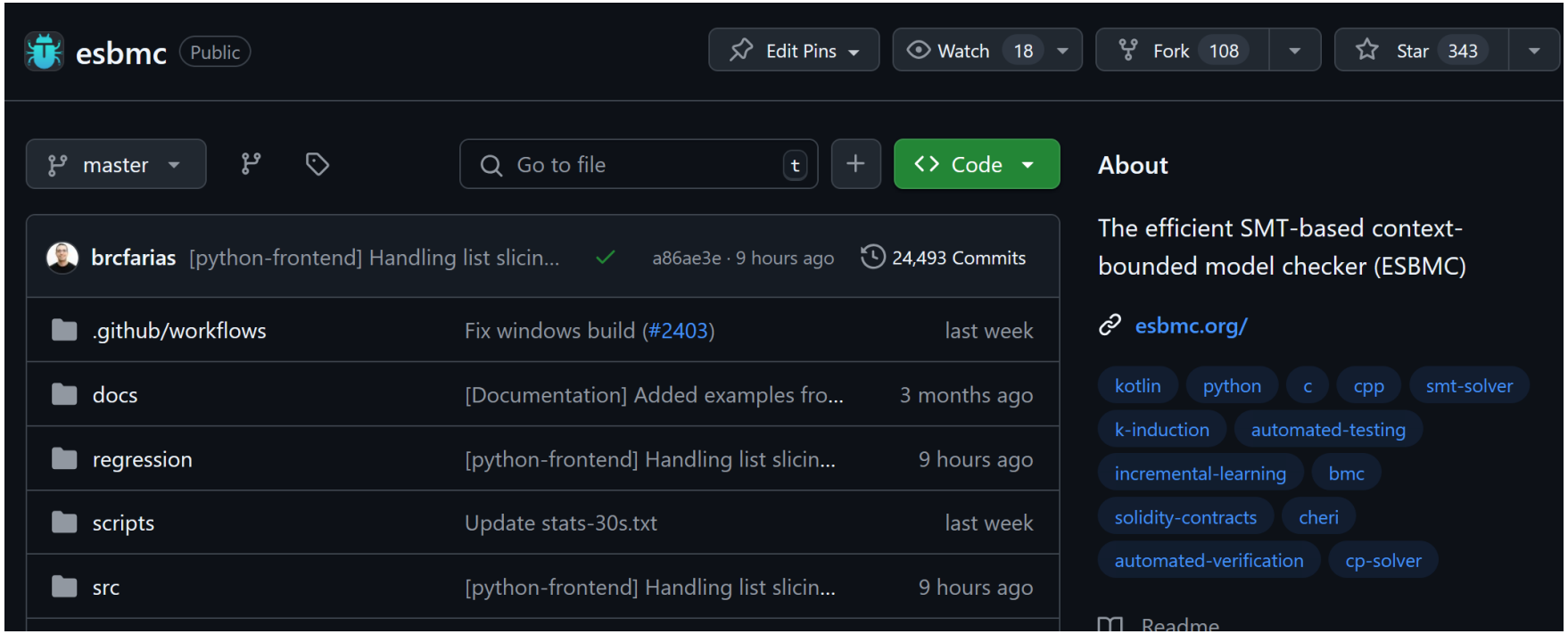
Participants	Plots	cetfuzz	CoVeriTest	ESBMC-incr	ESBMC-kind	FDSE	Fizzer	FuSeBMC	FuSeBMC-AI	HybridTiger	KLEE	KLEEF	Owi	PRTest	Rizzer	Sikraken	Symbiotic	TracerX	TracerX-WP	UTestGen	WASP-C
Cover-Branches 10011 valid tasks		2524	4959	4380	4323	5468	5429	5656	4077	3866	3065	5734	2462	3191	–	2469	4207	3327	3275	4393	2740
CPU time		1800000 s	6000000 s	1500000 s	980000 s	7900000 s	7300000 s	8900000 s	3600000 s	4800000 s	2700000 s	5400000 s	6200000 s	8200000 s		7300000 s	2800000 s	2000000 s	2300000 s	5900000 s	3600000 s

Here, ESBMC-incr performs slightly better, mostly due to different unwinding configuration.

Software Project

ESBMC is open-source under the Apache License 2.0, and its C++ source code is publicly available on GitHub: https://github.com/esbmc/esbmc/

The official website is available at: https://esbmc.org



MANCHESTER 1824
The University of Manchester

Software Project

ESBMC

Thank you!