Collaborators/funders:
APT / FM / S3 Research Groups
ARM Centre of Excellence
Centre for Digital Trust and Society
PPGEE, PPGI – UFAM
Innovate UK, UKRI, EPSRC, and EU Horizon
Industrial partners (ARM, Ethereum, Intel, Motorola, TII, Zscaler)

The University of Manchester

Towards Building Trustworthy Software and AI Systems

Lucas C. Cordeiro
Department of Computer Science
lucas.cordeiro@manchester.ac.uk
https://ssvlab.github.io/lucasccordeiro/


AVIS
ET'S
GO

Career Summary

1 7

BSc/MSc in Engineering and Lecturer

2

MSc in Embedded Systems

3

SIEMENS Mobile

Configuration and Build Manager

4

BenQ SIEMENS

Feature Leader

5

NXP

Set-top Box Software Engineer

6 Microsoft

PhD in Computer Science

8

Postdoctoral Researcher

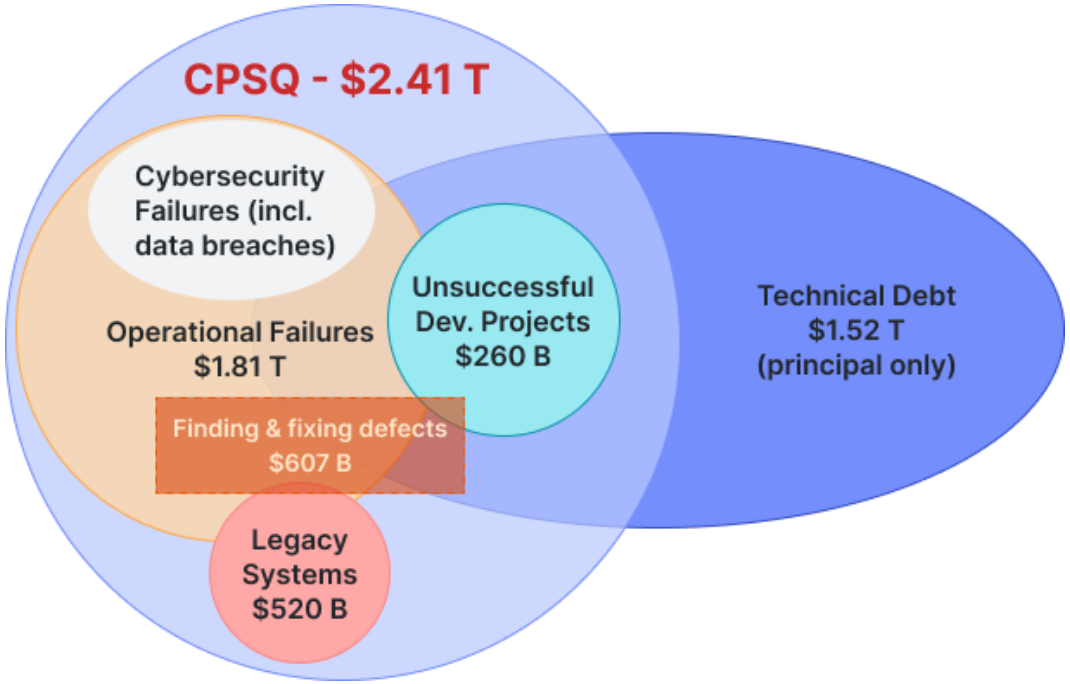
9

MANCHESTER 1824
The University of Manchester

Professor of Computer Science

How much could software errors cost your business?

Poor software quality cost US companies **$2.41 trillion in 2022**, while the **accumulated software Technical Debt** (TD) has grown to **~$1.52 trillion**



TD relies on temporary easy-to-implement solutions to achieve short-term results at the expense of efficiency in the long run

The cost of poor software quality in the US: A 2022 Report

CISQ
Consortium for Information & Software Quality™

How secure is AI-generated Code: A Large-Scale Comparison of Large Language Models

Category	Avg Prop. Viol. per Line	Rank	$\mathcal{VS}$	Rank	$\mathcal{VF}$	$\mathcal{VU}$ (Timeout)	Avg Prop. Viol. per File
GPT-4o-mini	**0.0165**	3	4.23%	2	57.14%	36.77%	3.40
Llama2-13B	0.0234	2	12.36%	**1**	**51.30%**	31.78%	3.62
Mistral-7B	0.0254	7	8.36%	4	62.08%	25.88%	**3.07**
CodeLlama-13B	0.0260	**1**	**15.48%**	3	52.71%	29.52%	4.13
Falcon-180B	0.0291	8	6.48%	5	62.07%	28.67%	3.38
GPT-3.5-turbo	0.0295	6	7.29%	7	65.07%	26.09%	4.42
Gemini Pro 1.0	0.0305	5	9.49%	6	63.91%	24.13%	4.70
Gemma-7B	0.0437	4	11.62%	8	67.01%	16.30%	4.20

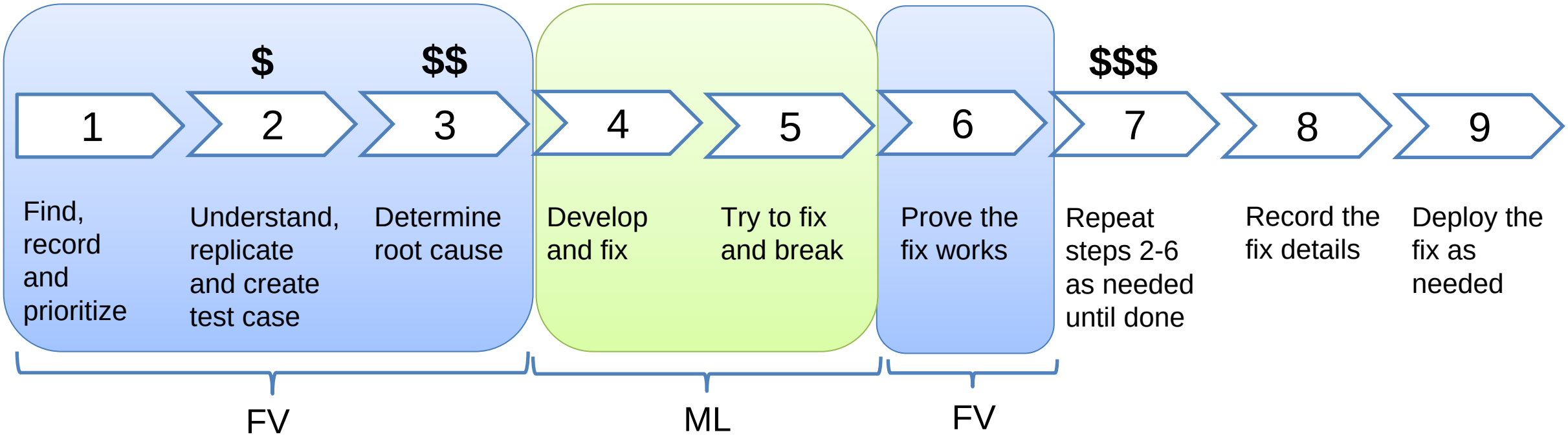
Legend:

$\mathcal{VS}$:0.0234 Verification Success; $\mathcal{VF}$: Verification Failed; $\mathcal{VU}$: Verification Unknown (Timeout).

Best performance in a category is highlighted with bold and/or Rank.

Tihanyi, N., Bisztray, T., Ferrag, M., Jain, R., Cordeiro, L.: How secure is AI-generated code: a large-scale comparison of large language models. Empir. Softw. Eng. 30(2): 47 (2025)

Find, Understand and Fix Bugs



*"A significant percentage (50%+) of a **software project's cost today is not spent on the creativity activity of software construction** but rather on the **corrective activity of debugging and fixing errors**"*

The cost of poor software quality in the US: A 2022 Report

CISQ
Consortium for Information & Software Quality™

What do you think, Julia?

That’s not good, that’s not nice, that’s not funny. It is scary!

Objective of this talk

Discuss **automated testing, verification, and synthesis** to establish a foundation **for building trustworthy software and AI systems**

- Introduce a **logic-based automated reasoning platform** to find and fix **software defects**
- Explain **testing, verification, and synthesis** techniques to build **trustworthy software and AI systems**
- Develop an **automated reasoning system** for **safeguarding software and AI systems** against vulnerabilities in an increasingly digital and interconnected world

Research Questions

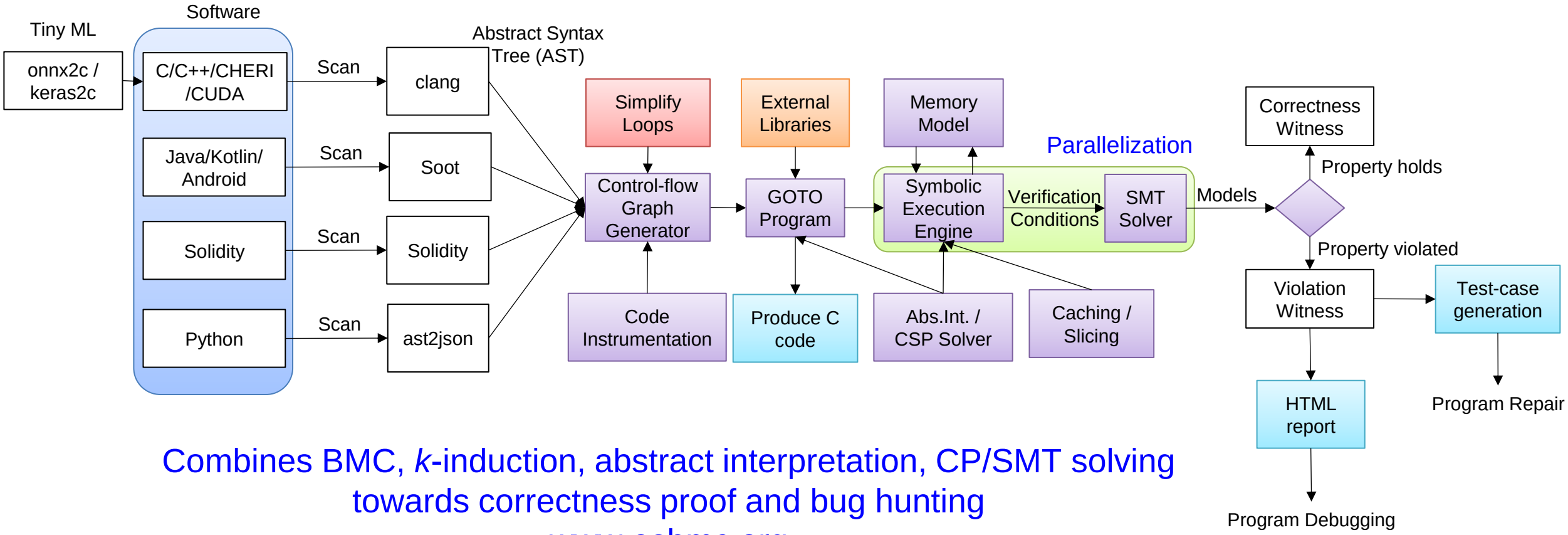
Given a **program** and a **specification**, can we automatically **verify** that the **program performs as specified**?

Can we leverage **program analysis/repair** to **discover and fix** more **software vulnerabilities** than existing state-of-the-art approaches?

Can we **improve engineers' productivity** to **find**, **understand**, and **fix software vulnerabilities**?

ESBMC: A Logic-based Verification Platform

Logic-based automated verification
for checking **safety** and **liveness**
properties in **AI** and **software systems**

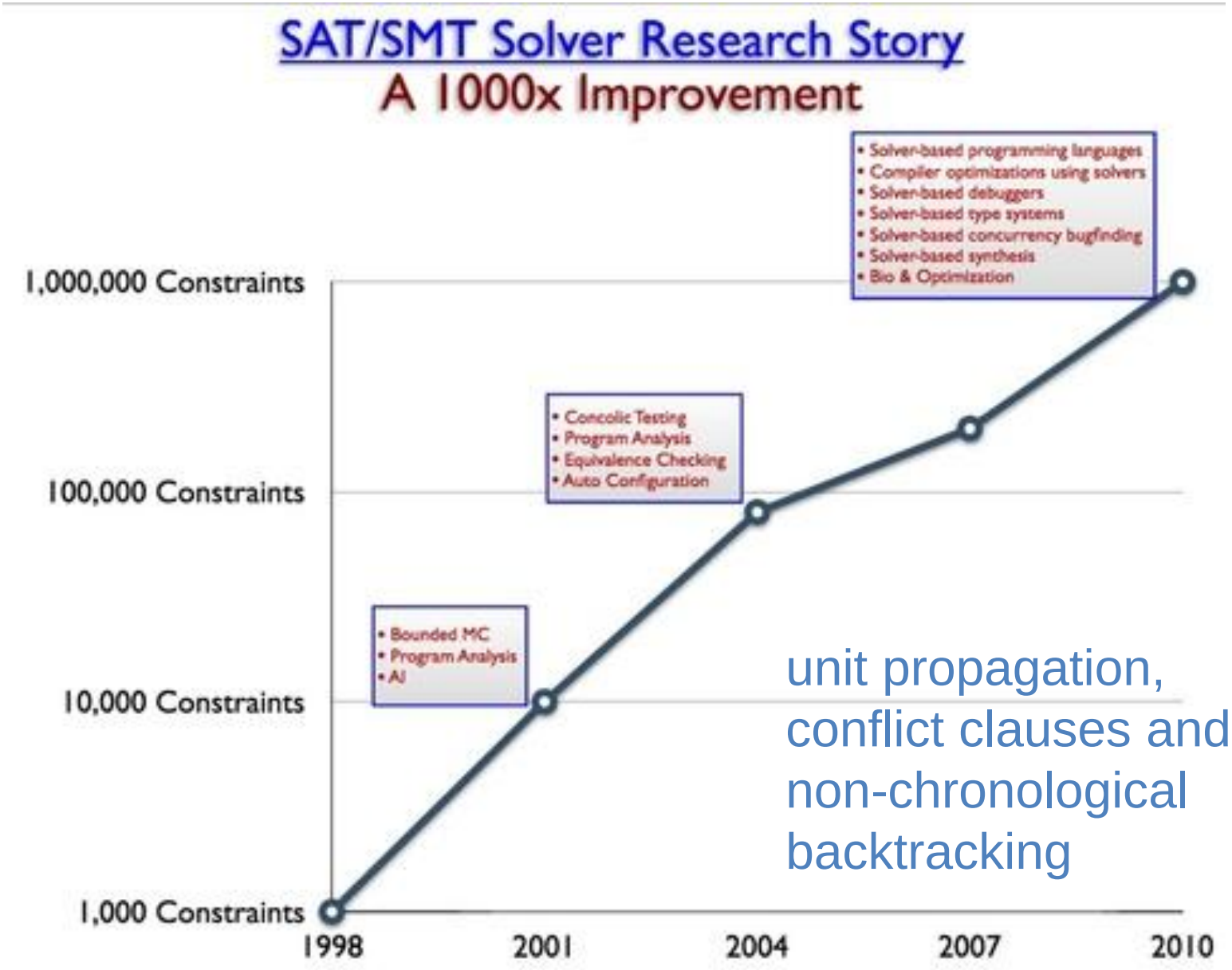


Combines BMC, *k*-induction, abstract interpretation, CP/SMT solving
towards correctness proof and bug hunting
www.esbmc.org

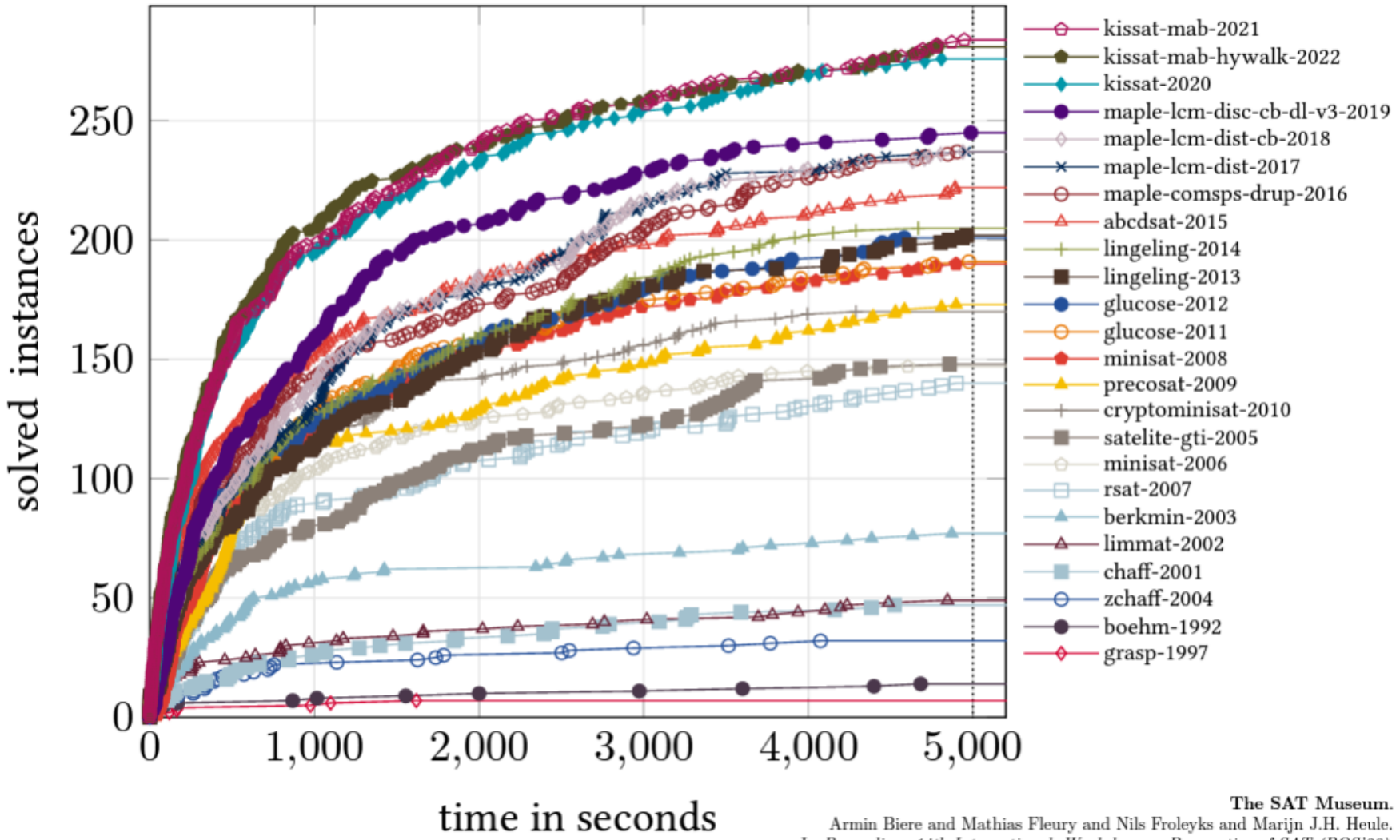
Agenda

- Automated program verification and testing to ensure conformance to specifications
- Advancing vulnerability detection to enhance software security beyond the state-of-the-art
- Automated program repair to improve the detection, comprehension, and fixing of software defects
- Summary and vision for future exploration

SAT solving as enabling technology



SAT Competition All Time Winners on SAT Competition 2022 Benchmarks

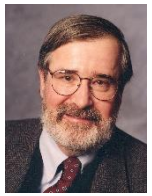


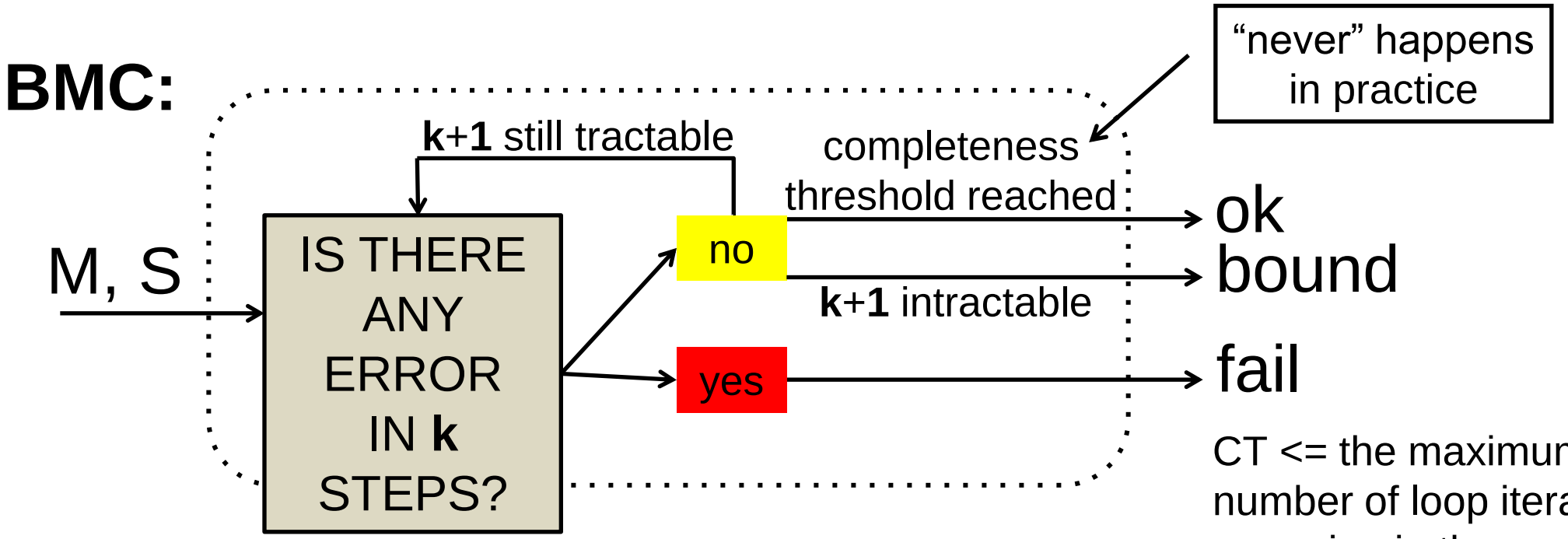
https://cca.informatik.uni-freiburg.de/satmuseum

The SAT Museum.
Armin Biere and Mathias Fleury and Nils Froleyks and Marijn J.H. Heule.
In *Proceedings 14th International. Workshop on Pragmatics of SAT (POS'23)*,
vol. 3545, CEUR Workshop Proceedings, pages 72-87, CEUR-WS.org 2023.
[paper - bibtex - data - zenodo - ceur - workshop - proceedings]

https://cca.informatik.uni-freiburg.de/satmuseum/

Bounded Model Checking (BMC)





Can the given property fail in *k*-steps?

$$I(S_0) \wedge T(S_0,S_1) \wedge \ldots \wedge T(S_{k-1},S_k) \wedge (\neg P(S_0) \vee \ldots \vee \neg P(S_k))$$

Initial state

k-steps

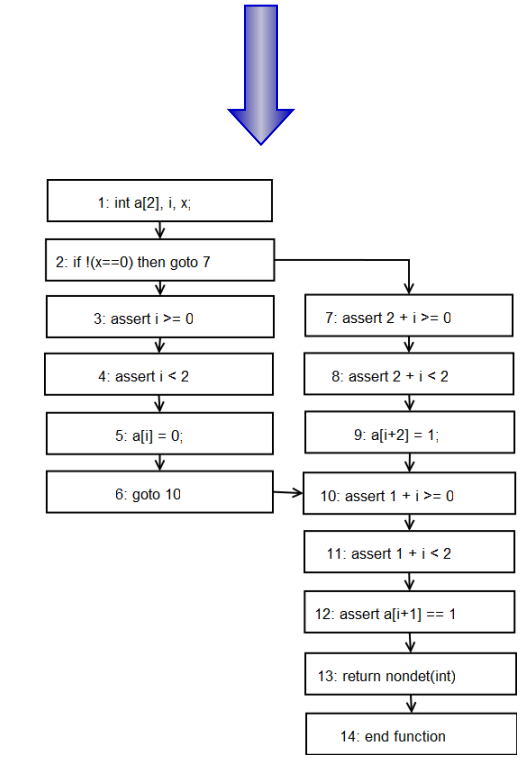
Property fails in some step

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: Symbolic Model Checking without BDDs. TACAS 1999: 193-207

Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph

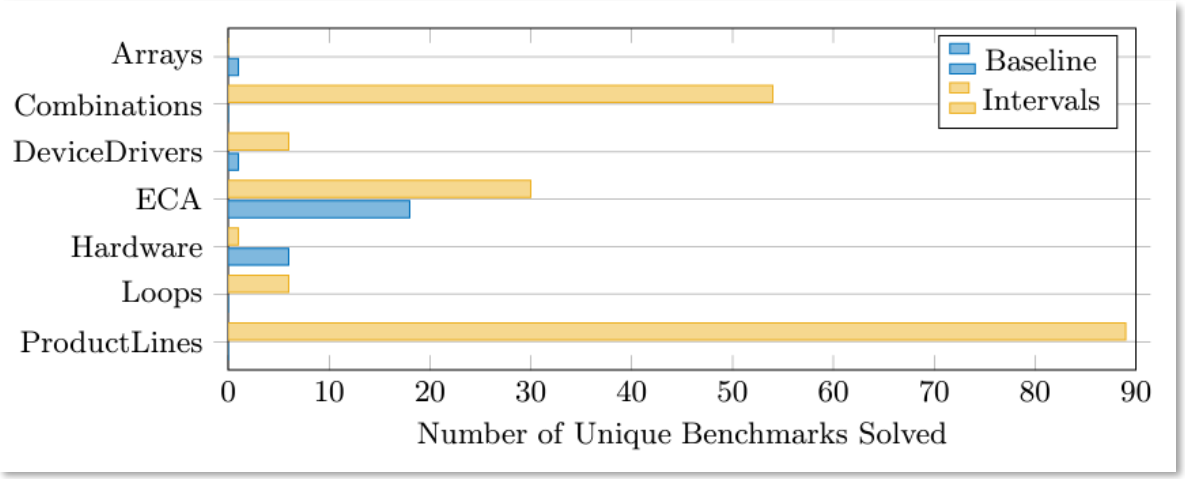
```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```



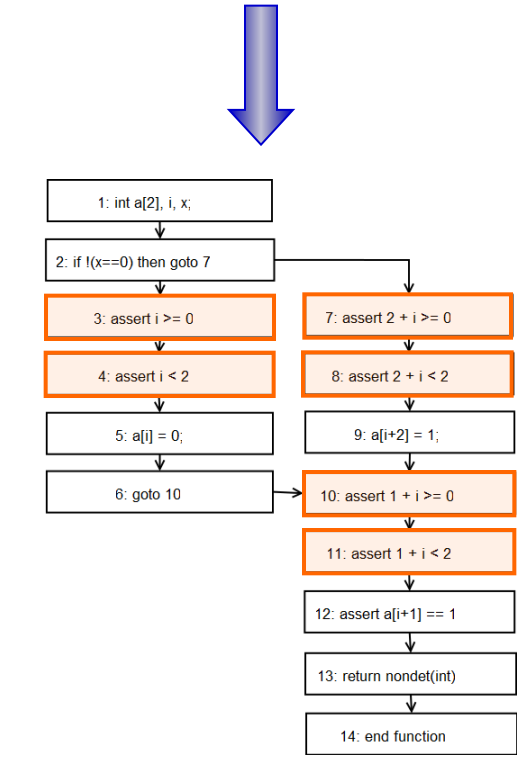
Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added assumptions/safety properties as extra nodes

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

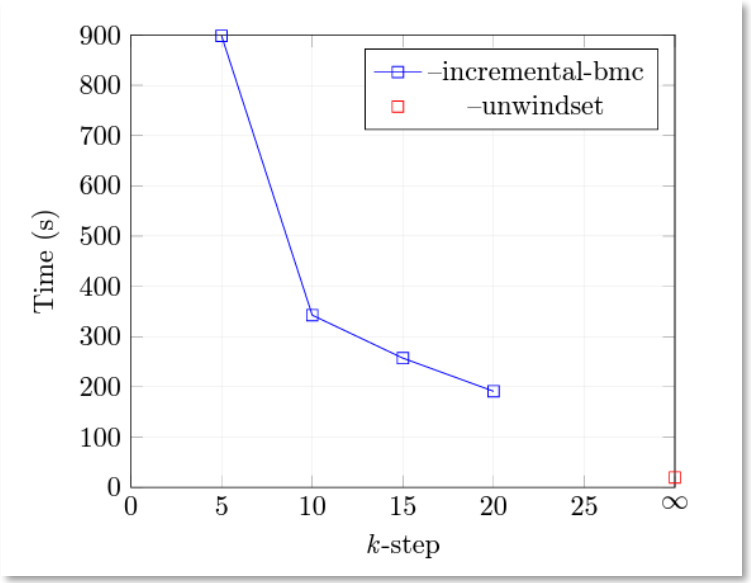


Menezes, R., Manino, E., Shmarov, F., Aldughaim, M., de Freitas, R., Lucas C. Cordeiro: Interval Analysis in Industrial-Scale BMC Software Verifiers: A Case Study.



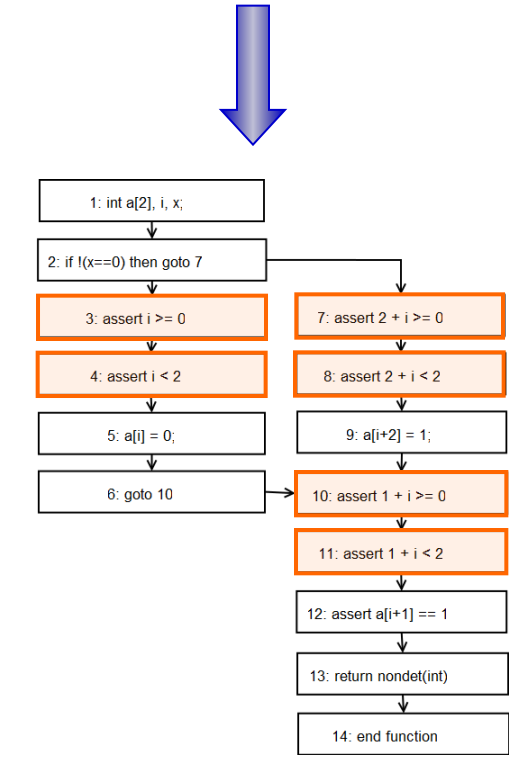
Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added assumptions/safety properties as extra nodes
- program unfolded up to given bounds



Wu T., Xiong, S., Manino, E., Stockwell, G., Cordeiro, L.: Verifying components of Arm(R) Confidential Computing Architecture with ESBMC. SAS 2024 (to appear)

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

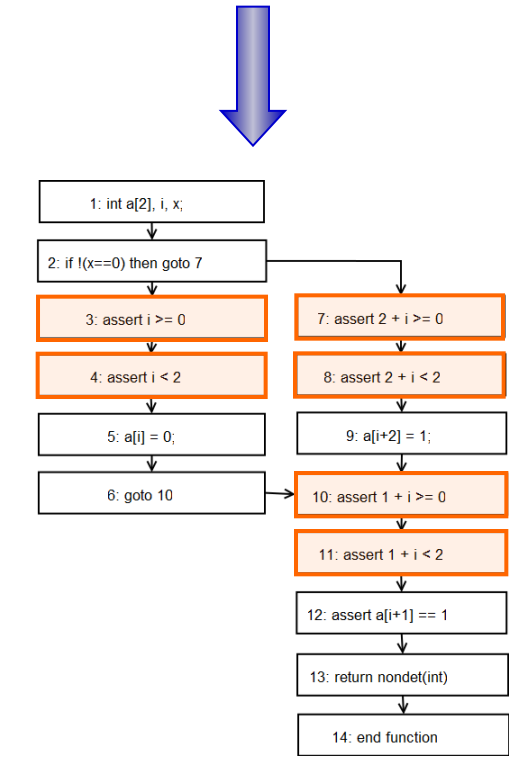


Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added assumptions/safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation/slicing
  - forward substitutions/caching
  - unreachable code/pointer analysis

} crucial

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

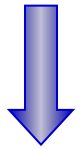


Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added assumptions/safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation/slicing
  - forward substitutions/caching
  - unreachable code/pointer analysis

  } crucial
- front-end converts unrolled and **optimized program into SSA**

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```



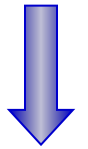
$g_1 = x_1 == 0$
$a_1 = a_0$ WITH $[i_0:=0]$
$a_2 = a_0$
$a_3 = a_2$ WITH $[2+i_0:=1]$
$a_4 = g_1 \ ? \ a_1 : a_3$
$t_1 = a_4 [1+i_0] == 1$

Software BMC

- program modeled as a state transition system
  - *state*: *pc* and program variables
  - derived from control-flow graph
  - added assumptions/safety properties as extra nodes
- program unfolded up to given bounds
- unfolded program optimized to reduce blow-up
  - constant propagation/slicing
  - forward substitutions/caching
  - unreachable code/pointer analysis

  } crucial
- front-end converts unrolled and **optimized program into SSA**
- extraction of *constraints C* and *properties P*
  - specific to selected SMT solver, uses theories
- satisfiability check of $C \wedge \neg P$

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```



$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{bmatrix}$$

Cordeiro, L., Fischer, B., Marques-Silva, J.: SMT-Based Bounded Model Checking for Embedded ANSI-C Software. IEEE Trans. Software Eng. 38(4): 957-974 (2012)

Most Influential Paper Award at ASE 2023

Context-Bounded Model Checking in ESBMC

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving

... combines

- **symbolic** model checking: on each individual interleaving
- **explicit state** model checking: explore all interleavings
  - bound the number of context switches allowed among threads

… implements

- **symbolic state hashing** (SHA1 hashes)
  - **monotonic partial order** reduction that combines dynamic POR with symbolic state space exploration

DISTINGUISHED PAPER AWARD

ICSE 2011

The 33rd International Conference on Software Engineering

May 21-28, 2011 Waikiki, Honolulu, Hawaii

Presented to

Lucas Cordeiro and Bernd Fischer

For

"Verifying Multi-threaded Software using SMT-based Context-Bounded

ETAPS
EUROPEAN JOINT CONFERENCES ON THEORY & PRACTICE OF SOFTWARE

TACAS 2012

Competition on Software Verification (SV-COMP)

ControlFlowInteger

1. CPAchecker-ABE 1.0.10
2. CPAchecker-Memo 1.0.10
3. QARMC-HSF
4. ESBMC 1.17
5. LLBMC 0.9

DeviceDrivers

1. LLBMC 0.9
2. Predator
3. BLAST 2.7
4. SATabs 3.0
5. Wolverine 0.5c

DeviceDrivers64

1. BLAST 2.7
2. CPAchecker-Memo 1.0.10
3. SATabs 3.0
4. CPAchecker-ABE 1.0.10
5. Wolverine 0.5c

HeapManipulation

1. Predator
2. LLBMC 0.9
3. CPAchecker-ABE 1.0.10
3. CPAchecker-Memo 1.0.10
5. ESBMC 1.17

SystemC

1. ESBMC 1.17
2. SATabs 3.0
3. CPAchecker-ABE 1.0.10
4. CPAchecker-Memo 1.0.10
5. Wolverine 0.5c

Concurrency

1. ESBMC 1.17
2. SATabs 3.0
3. --
4. --
5. --

Overall

1. CPAchecker-Memo 1.0.10
2. CPAchecker-ABE 1.0.10
3. ESBMC 1.17
4. SATabs 3.0
5. BLAST 2.7

Induction-Based Verification for Software

***k*-induction** checks loop-free programs...

- **base case ($base_k$):** find a counter-example with up to *k* loop unwindings (plain BMC)
- **forward condition (fwd_k):** check that *P* holds in all states reachable within *k* unwindings
- **inductive step ($step_k$):** check that whenever *P* holds for *k* unwindings, it also holds after next unwinding
  - havoc variables
  - assume loop condition
  - run loop body (*k* times)
  - assume loop termination

⇒ iterative deepening if inconclusive

Gadelha, M., Ismail, H., Cordeiro, L.: Handling loops in bounded model checking of C programs via k-induction. Int. J. Softw. Tools Technol. Transf. 19(1): 97-114 (2017)

Automatic Invariant Generation

- Infer invariants based on **intervals** as abstract domain via a dependence graph
  - *E.g., $a \leq x \leq b$ (integer and floating-point)*
  - Inject intervals as assumptions and contract them via CSP
  - Remove unreachable states

Line	*Interval for "a"*	*Restriction*
4	$(-\infty, +\infty)$	*None*
6	$(-\infty, 100]$	$a \leq 100$
7	$(100, +\infty)$	$a > 100$

```
1 int main()
2 {
3   int a = *;
4
5   while(a <= 100)
6     a++;
7   assert(a>10);
8   return 0;
9 }
```

k-Induction proof rule "hijacks" loop conditions to nondeterministic values, thus computing intervals become essential

_k_-Induction can prove the correctness of more programs when the invariant generation is enabled

Menezes, et al.: ESBMC v7.4: Harnessing the Power of Intervals - (Competition Contribution). TACAS (3) 2024: 376-380

BMC of Software Using Interval Methods via Contractors

1) Analyze intervals and properties
   - Static Analysis / Abstract Interpretation
2) Convert the problem into a CSP
   - Variables, Domains and Constraints
3) Apply contractor to CSP
   - Forward-Backward Contractor
4) Apply reduced intervals back to the program

```
1 unsigned int x=nondet_uint();
2 unsigned int y=nondet_uint();
3 __ESBMC_assume(x >= 20 && x <= 30);
4 __ESBMC_assume(y <= 30);
5 assert(x >= y);
```

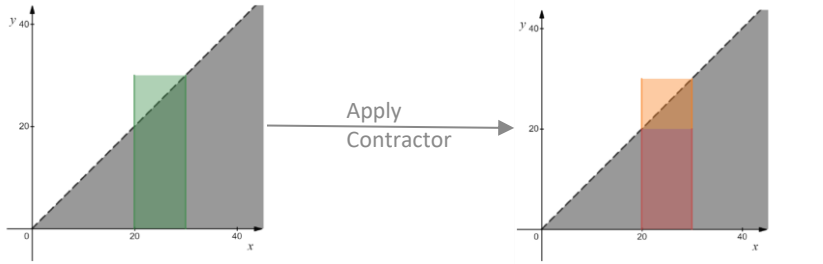
```
__ESBMC_assume(y <= 30 && y >= 20);
```

This **assumption** prunes our search space to the **orange area**

```
1 unsigned int x=nondet_uint();
2 unsigned int y=nondet_uint();
3 __ESBMC_assume(x >= 20 && x <= 30);
4 __ESBMC_assume(y <= 30);
5 assert(x >= y);
```

Domain: $[x] = [20, 30]$ and $[y] = [0, 30]$

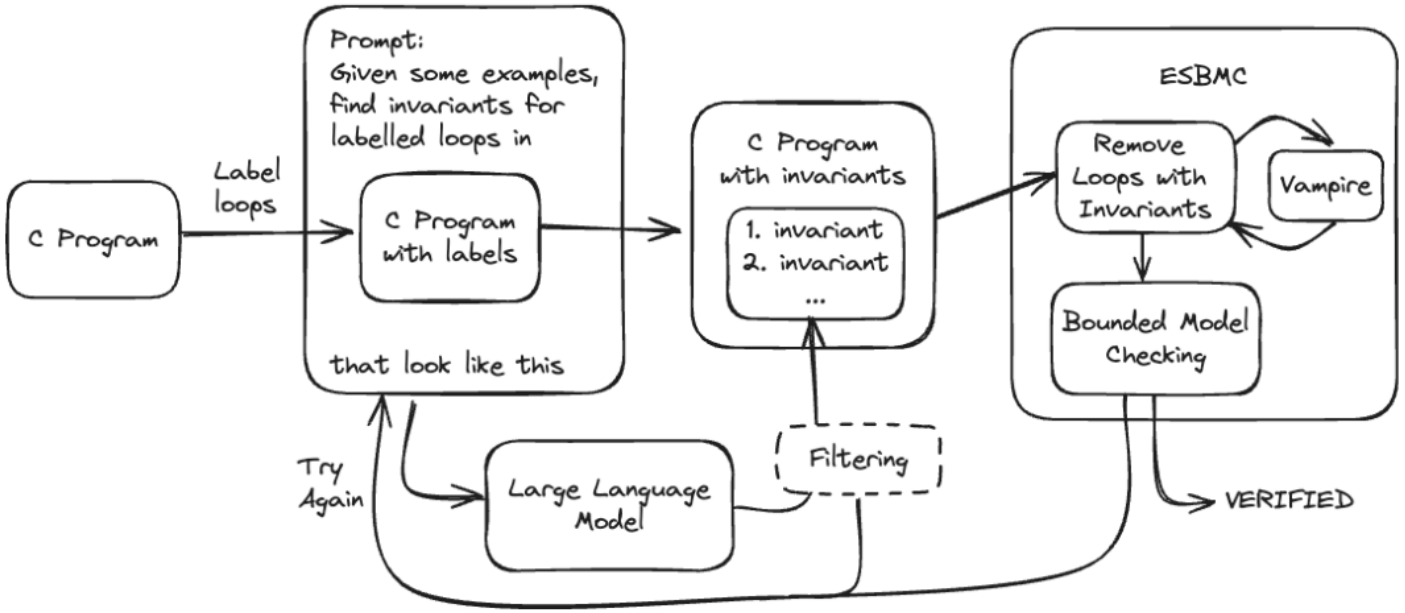
Constraint: $y - x \leq 0$



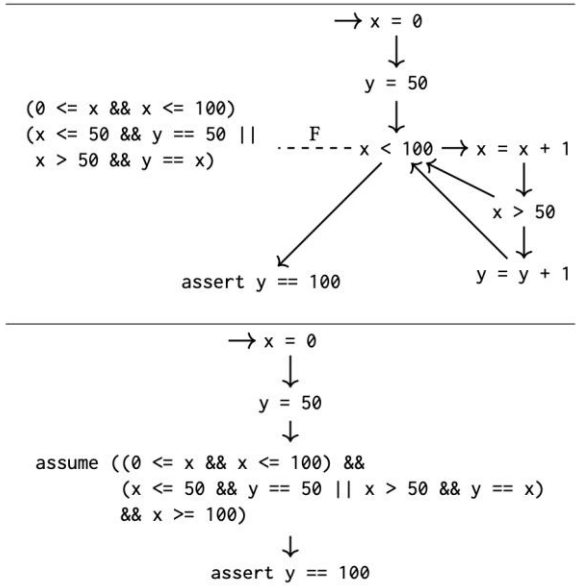
$[x] = [20, 30]$ and $[y] = [0, 30]$ $[x] = [20, 30]$ and $[y] = [20, 30]$

$f(x) > 0$	$I = [0, \infty)$	
$f(x) = y - x$	$[f(x)_1] = I \cap [y_0] - [x_0]$	Forward-step
$x = y - f(x)$	$[x_1] = [x_0] \cap [y_0] - [f(x)_1]$	Backward-step
$y = f(x) + x$	$[y_1] = [y_0] \cap [f(x)_1] + [x_1]$	Backward-step

LLM-Generated Invariants for Bounded Model Checking Without Loop Unrolling




```c
int main()
{
  int x = 0;
  int y = 50;
  __invariant(0 <= x && x <= 100);
  __invariant(x <= 50 && y == 50
          || x > 50 && y == x);
  while (x < 100) {
    x = x + 1;
    if(x > 50){
      y = y + 1;
    }
  }
  __VERIFIER_assert(y == 100 );
}
```



Pirzada, M., Bhayat, A., Reger, G., Cordeiro, L.: LLM-Generated Invariants for Bounded Model Checking Without Loop Unrolling. In ASE 2024

Distinguished Paper Award

ASE 2024

IEEE/ACM International Conference on Automated Software Engineering

October 27 - November 1 Sacramento, California

Presented to

Muhammad A. A. Pirzada, Giles Reger, Ahmed Bhayat, Lucas C. Cordeiro

for

LLM-Generated Invariants for Bounded Model Checking Without Loop Unrolling

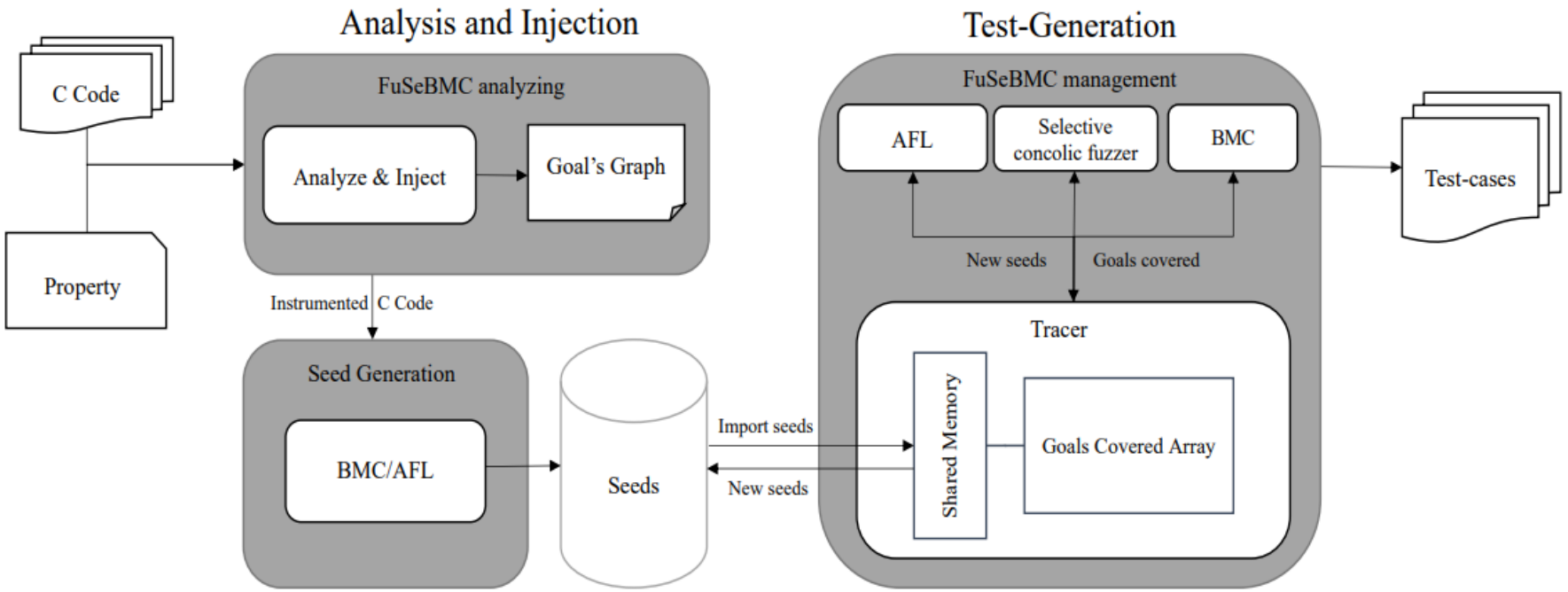
Vladimir Filkov
General Chair

Baishakhi Ray
Research PC Co-Chair

Minghui Zhou
Research PC Co-Chair

FuSeBMC v4 Framework

- Use **Clang** tooling infrastructure
- Employ three engines in its **reachability analysis**: **one BMC and two fuzzing engines**
- Use a **tracer** to coordinate the various engines

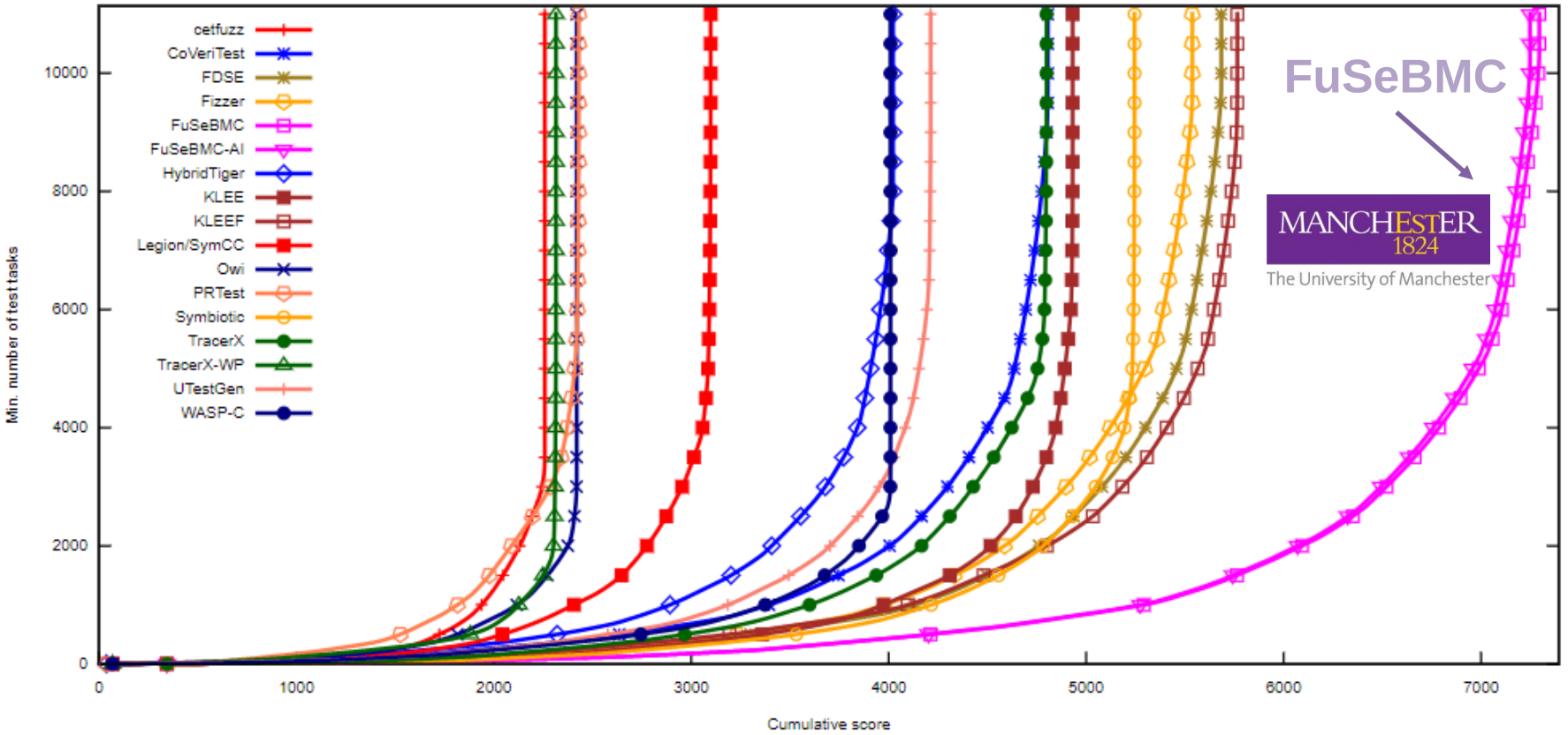


Alshmrany, K., Aldughaim, M., Bhayat, A., Cordeiro, L.: FuSeBMC v4: Smart Seed Generation for Hybrid Fuzzing - (Competition Contribution). FASE 2022: 336-340

Agenda

- Automated program verification and testing to ensure conformance to specifications
- Advancing vulnerability detection to enhance software security beyond the state-of-the-art
- Automated program repair to improve the detection, comprehension, and fixing of software defects
- Summary and vision for future exploration

Competition on Software Testing 2024: Results of the `Overall` Category



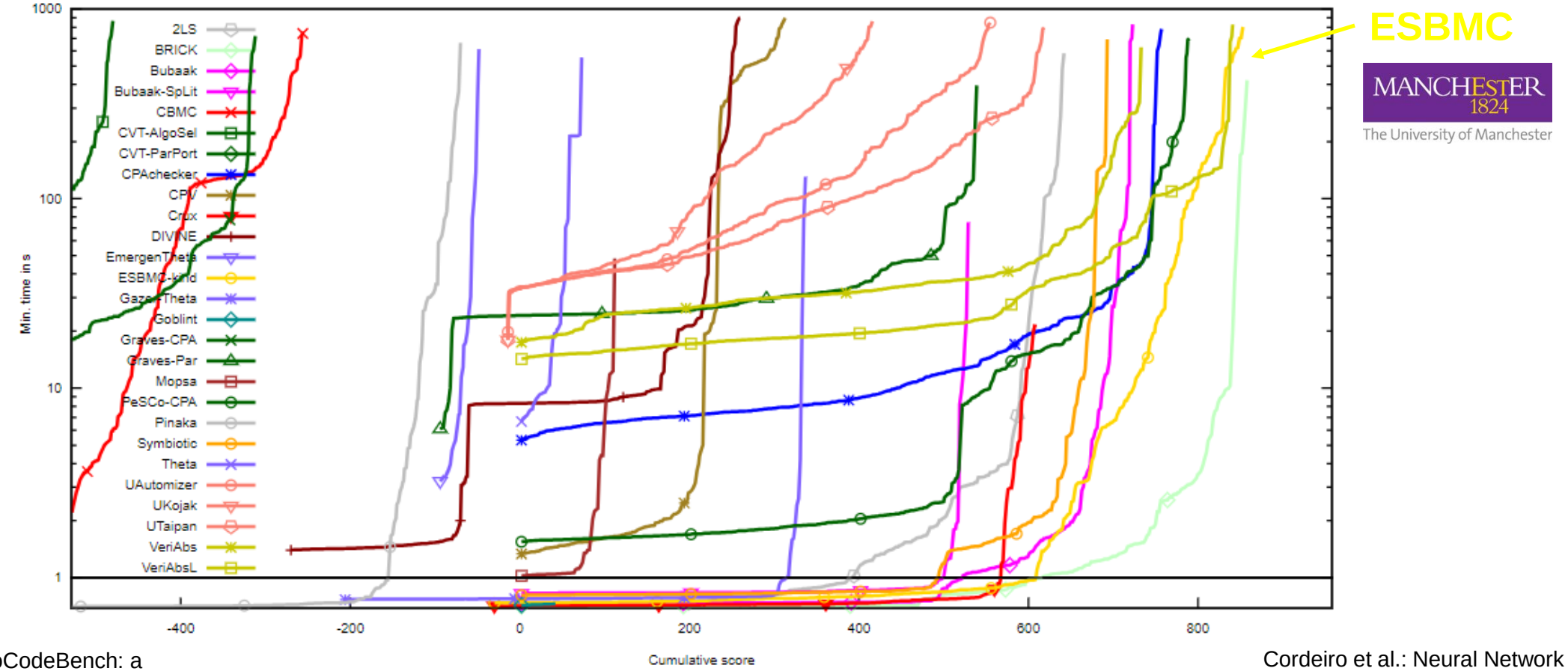
FuSeBMC achieved 3 awards: 1st place in `Cover-Error`, 1st place in `Cover-Branches`, and 1st place in `Overall`

Alshmrany, K., Aldughaim, M., Bhayat, A., Cordeiro, L.: FuSeBMC v4: Smart Seed Generation for Hybrid Fuzzing - (Competition Contribution). FASE 2022: 336-340

https://test-comp.sosy-lab.org/2024/

From Floating-Point Programs to Neural Network Implementations

- **Known ground truth**, width (1-1024 neurons), depth (1-4 layers), feedforward & recurrent, 8 activation functions



Verification of the `ReachSafety-Floats` Category

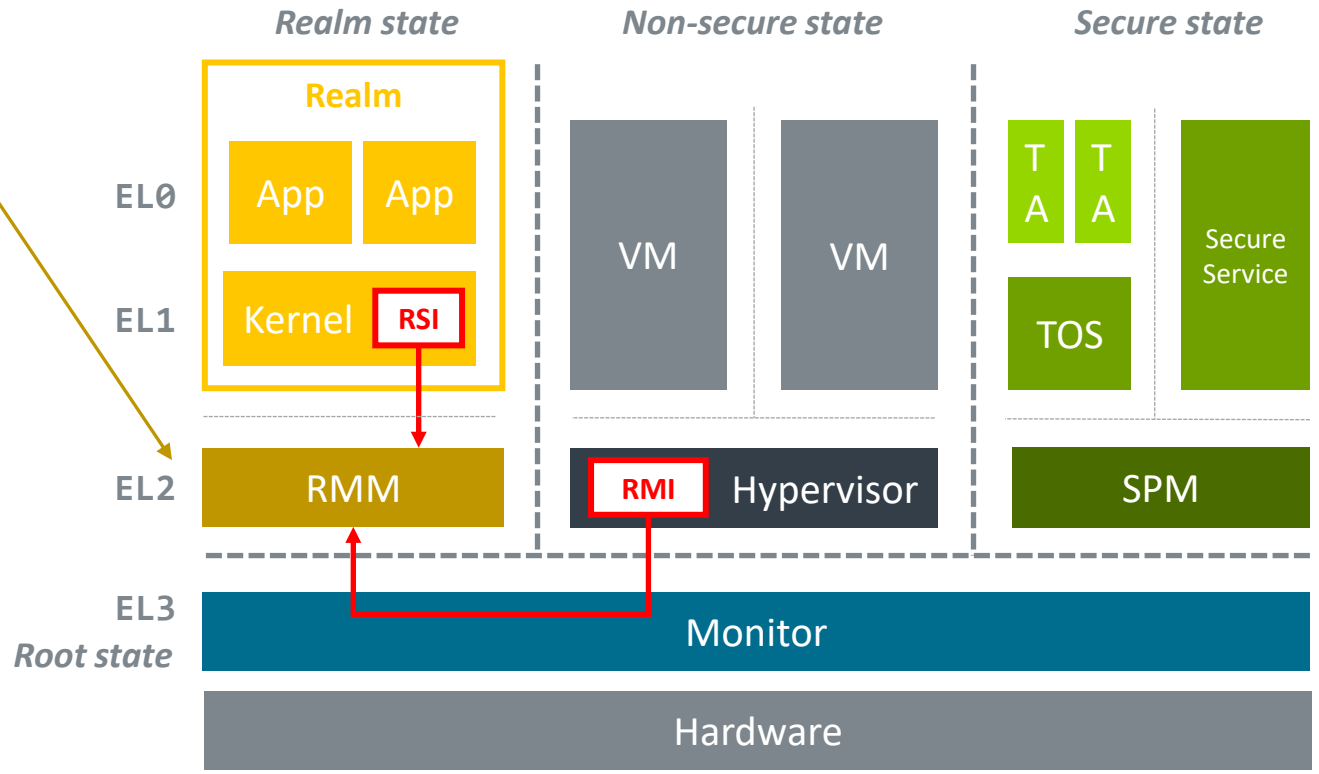
Manino, E. et al.: NeuroCodeBench: a plain C neural network benchmark for software verification. In AFRiTS 2023

Cordeiro et al.: Neural Network Verification is a Programming Language Challenge. In ESOP 2024

Verifying Components of Arm® Confidential Computing Architecture with ESBMC

Realm Management Monitor (RMM)

- Provides services to Host and Realm
  - Contains no policy
  - Performs no dynamic memory allocation
- Realm Management Interface (RMI)
  - Secure Monitor Call Calling Convention (SMCCC) interface called by Host
  - Create/destroy Realms
  - Manage Realm memory, manipulating stage 2 translation tables
  - Context switch between Realm VCPUs
- Realm Services Interface (RSI)
  - SMCCC interface called by Realm
  - Measurement and attestation
  - Handshakes involved in some memory management flows



Arm CCA is an architecture that provides Protected Execution Environments called Realms

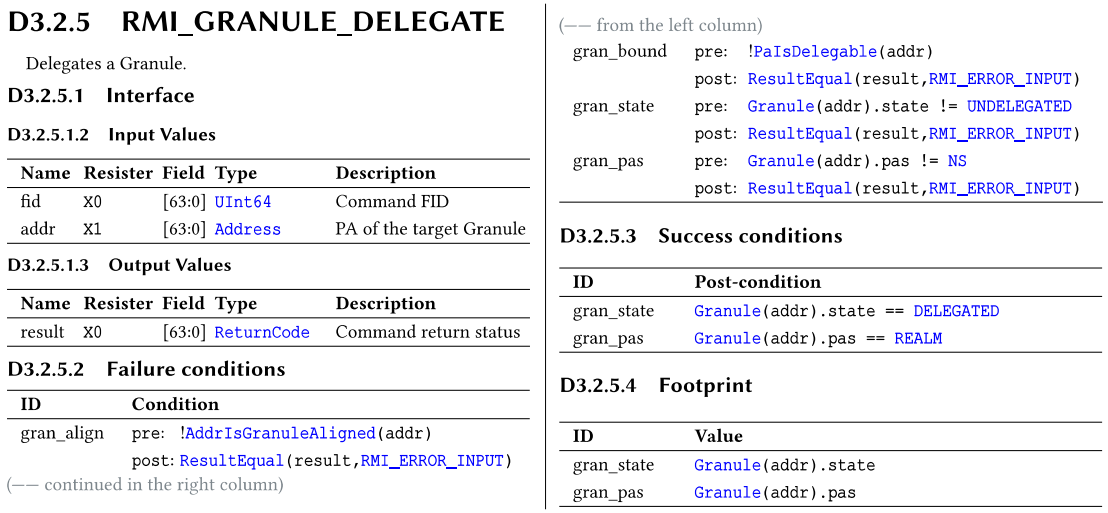
Wu, T., Xiong, S., Manino, E., Stockwell, G., Cordeiro, L. Verifying components of Arm(R) Confidential Computing Architecture with ESBMC. In SAS 2024.

Verifying Components of Arm® Confidential Computing Architecture with ESBMC

- The specification document[1] is in the style of:
  - rules-based writing

    R_{TMGSL} When the state of a Granule has transitioned from *P* to DELEGATED and then to any other state, any content associated with *P* has been *wiped*.

  - pre/post-condition pairs.



D3.2.5 RMI_GRANULE_DELEGATE

Delegates a Granule.

D3.2.5.1 Interface

D3.2.5.1.2 Input Values

Name	Resister	Field	Type	Description
fid	X0	[63:0]	UInt64	Command FID
addr	X1	[63:0]	Address	PA of the target Granule

D3.2.5.1.3 Output Values

Name	Resister	Field	Type	Description
result	X0	[63:0]	ReturnCode	Command return status

D3.2.5.2 Failure conditions

ID	Condition
gran_align	pre: !AddrIsGranuleAligned(addr)
	post: ResultEqual(result,RMI_ERROR_INPUT)

(–– continued in the right column)

(–– from the left column)

gran_bound	pre:	!PaIsDelegable(addr)
	post:	ResultEqual(result,RMI_ERROR_INPUT)
gran_state	pre:	Granule(addr).state != UNDELEGATED
	post:	ResultEqual(result,RMI_ERROR_INPUT)
gran_pas	pre:	Granule(addr).pas != NS
	post:	ResultEqual(result,RMI_ERROR_INPUT)

D3.2.5.3 Success conditions

ID	Post-condition
gran_state	Granule(addr).state == DELEGATED
gran_pas	Granule(addr).pas == REALM

D3.2.5.4 Footprint

ID	Value
gran_state	Granule(addr).state
gran_pas	Granule(addr).pas

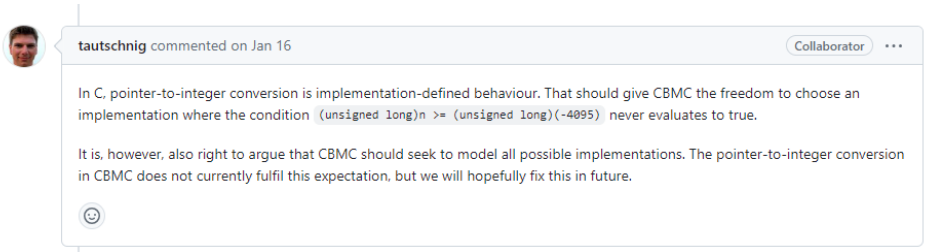
- The document is generated from a **machine-readable specification** (MRS)

[1] https://developer.arm.com/documentation/den0137/latest, the examples in this slide are taken when the paper was drafted.

Verifying Components of Arm® Confidential Computing Architecture with ESBMC

Test_benchmarks	esbmc multi	cbmc multi
RMI_REC_DESTROY	20	20
RMI_GRANULE_DELEGATE	safe	safe
RMI_GRANULE_UNDELEGATE	1	1
RMI_REALM_ACTIVATE	**3**	**safe**
RMI_REALM_DESTROY	**15**	**1**
RMI_REC_AUX_COUNT	1	1
RMI_FEATURES	safe	safe
RMI_DATA_DESTROY	**>=24**	**22**

```c
#include <assert.h>
extern int nondet_int();
int main() {
  int m = nondet_int();
  int *n = &m;
  if((unsigned long)n >= (unsigned long)(-4095))
    assert((unsigned int)(-1 * (long)n) < 6);
  int a = -2048;
  if((unsigned long)a >= (unsigned long)(-4095))
    assert((unsigned int)(-1 * (long)a) < 6);
}
```



https://github.com/diffblue/cbmc/issues/8161

Wu, T., Xiong, S., Manino, E., Stockwell, G., Cordeiro, L., Verifying components of Arm® Confidential Computing Architecture with ESBMC. In SAS 2024

Intel Core Power Management Firmware

Intel routinely employs ESBMC to **automate firmware analysis**

ESBMC has been applied to the **Authenticated Code Module**, where it **found over 30 vulnerabilities**

ESBMC is part of the CI pipeline for developing microcode for the Core family of processors

P6 Microcode Can Be Patched

Intel Discloses Details of Download Mechanism for Fixing CPU Bugs

"Taking an unusual approach to fixing bugs, Intel has implemented a microcode patch capability in its P6 processors, including Pentium Pro and Pentium II. This capability allows the microcode to be altered after the processor is fabricated, repairing bugs that are found after the processor is designed. Intel has already used this feature several times to correct minor bugs, and in the future, it may save the company from recalling CPUs if a major problem is discovered."

Menezes, R., Manino, E., Shmarov, F., Aldughaim, M., de Freitas, R. Cordeiro, L.: *Interval Analysis in Industrial-Scale BMC Software Verifiers: A Case Study*. CoRR abs/2406.15281 (2024)

WolfMQTT Verification

- **wolfMQTT** library is a client implementation of the MQTT protocol written in C for **IoT devices**

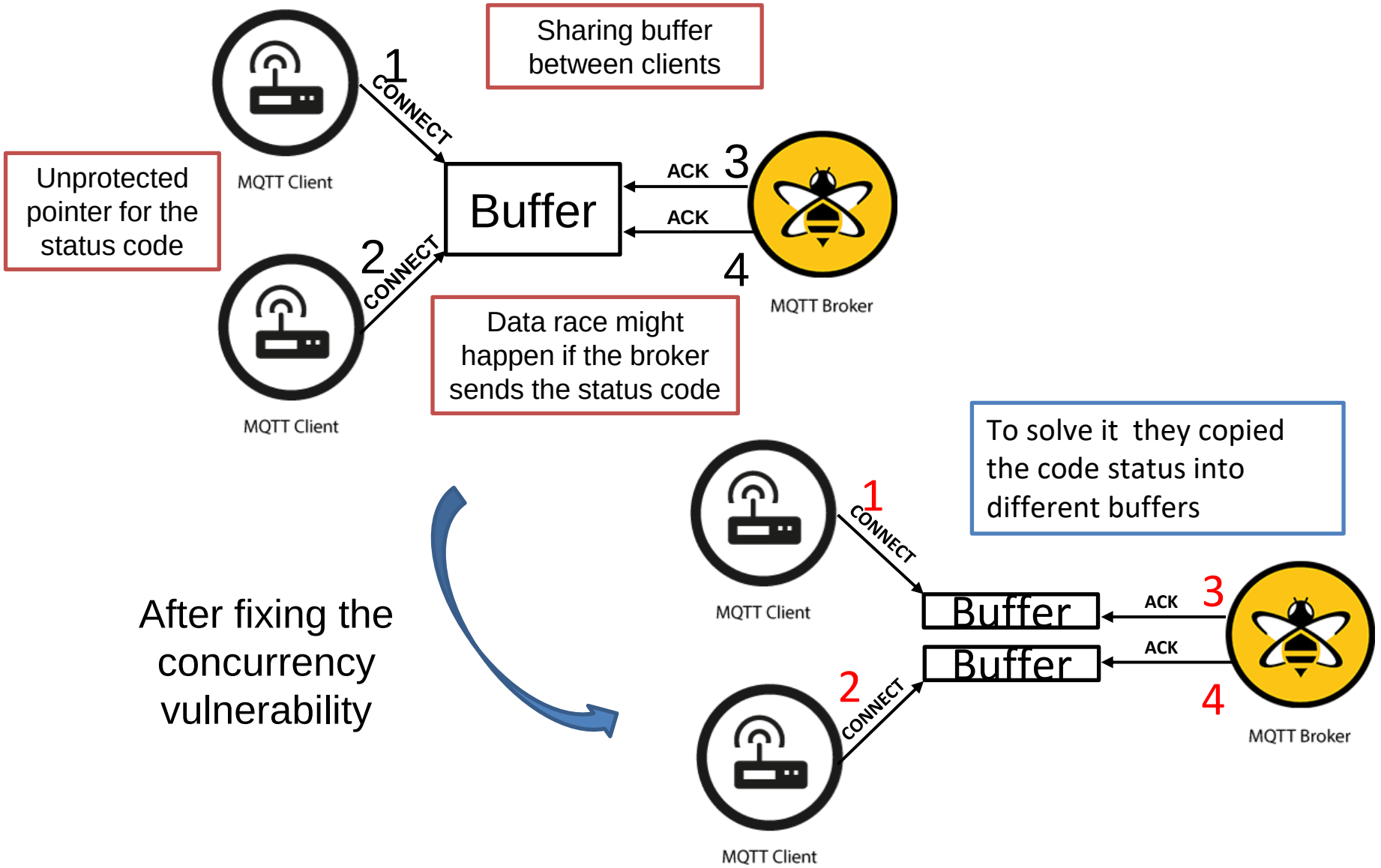
`subscribe_task` and `waitMessage_task` are called through different threads accessing `packet_ret`, causing a data race in `MqttClient_WaitType`

Here is where the data race might happen! Unprotected pointer

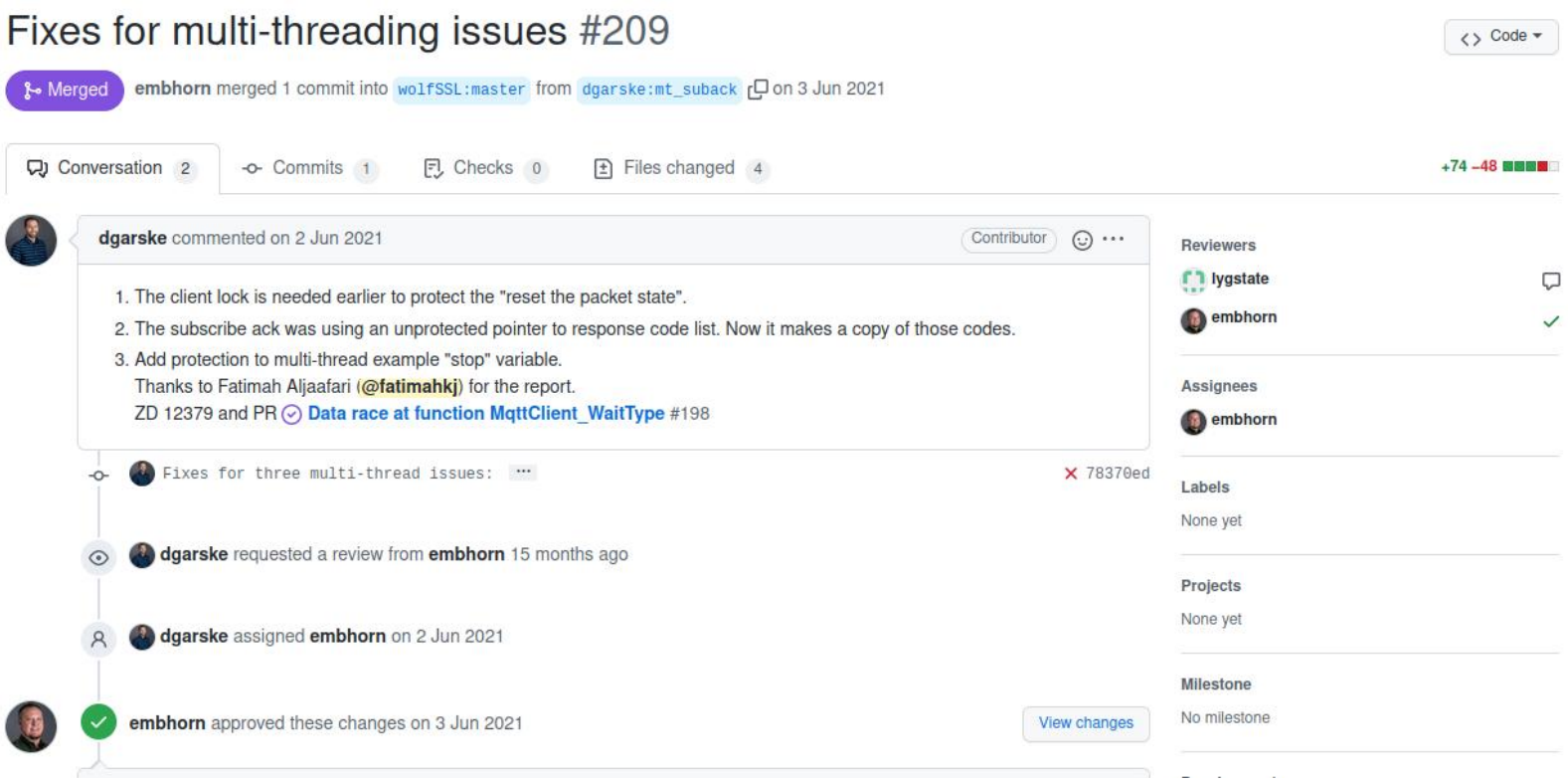
```
Int main(){
Pthread_t th1, th2;
static MQTTCtx mqttCtx;
pthread_create(&th1, subscribe_task, &mqttCtx))
pthread_create(&th2, waitMessage_task, &mqttCtx))}

static void *subscribe_task(void *client){
.....
MqttClient_WaitType(client,msg,MQTT_PACKET_TYPE_ANY,
0,timeout_ms);
.....}
static void *waitMessage_task(void *client){
…
MqttClient_WaitType(client, msg, MQTT_PACKET_TYPE_ANY,
0,timeout_ms);
.....}
static int MqttClient_WaitType(MqttClient *client,
void *packet_obj,
   byte wait_type, word16 wait_packet_id, int timeout_ms)
{
.....
            rc = wm_SemLock(&client->lockClient);
            if (rc == 0) {
                if (MqttClient_RespList_Find(client,
(MqttPacketType)wait_type,
                        wait_packet_id, &pendResp)) {
                    if (pendResp->packetDone) {
                        rc = pendResp->packet_ret;
.....}
```

WolfMQTT Verification



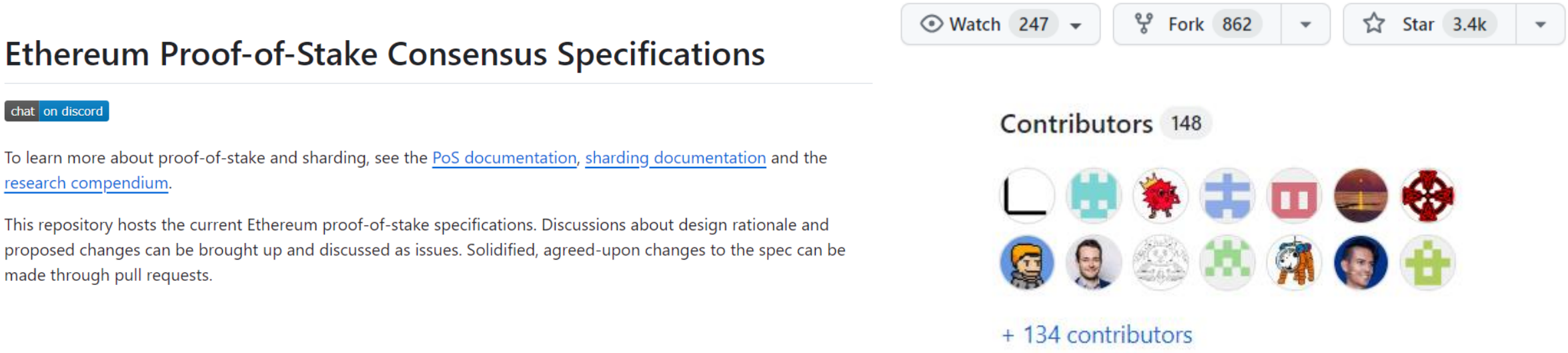
Bug Report



https://github.com/wolfSSL/wolfMQTT

Ethereum Consensus Specifications

- Consensus protocol dictates how the participants in Ethereum agree on the validity of transactions and the system's state
- Git repository with **Markdown** documents describing specifications
- Infrastructure to generate **Python** libraries from Markdown



ESBMC-Python Benchmark

Ethereum Consensus Specification

Markdown

consensus-specs / specs / phase0 / **beacon-chain.md**

Preview | Code | Blame | 1939 lines (1617 loc) · 71.4 KB

Math

`integer_squareroot`

```
def integer_squareroot(n: uint64) -> uint64:
    """
    Return the largest integer ``x`` such that ``x**2 <= n``.
    """
    x = n
    y = (x + 1) // 2
    while y < x:
        x = y
        y = (x + n // x) // 2
    return x
```

`xor`

```
def xor(bytes_1: Bytes32, bytes_2: Bytes32) -> Bytes32:
    """
    Return the exclusive-or of two 32-byte strings.
    """
    return Bytes32(a ^ b for a, b in zip(bytes_1, bytes_2))
```

eth2spec Python Library

mainnet.py

lib > python3.10 > site-packages > eth2spec-1.4.0b4-py3.10.egg > eth2spec > bellatrix > mainnet.py > integer_squareroot

```
1461
1462
1463 def integer_squareroot(n: uint64) -> uint64:
1464     """
1465     Return the largest integer ``x`` such that ``x**2 <= n``.
1466     """
1467     x = n
1468     y = (x + 1) // 2
1469     while y < x:
1470         x = y
1471         y = (x + n // x) // 2
1472     return x
1473
1474
1475 def xor(bytes_1: Bytes32, bytes_2: Bytes32) -> Bytes32:
1476     """
1477     Return the exclusive-or of two 32-byte strings.
1478     """
1479     return Bytes32(a ^ b for a, b in zip(bytes_1, bytes_2))
1480
1481
1482 def bytes_to_uint64(data: bytes) -> uint64:
1483     """
1484     Return the integer deserialization of ``data`` interpreted as ``ENDIANNESS``-endian.
1485     """
1486     return uint64(int.from_bytes(data, ENDIANNESS))
```

Python Application

integer_squareroot.py

eth2bmc > samples > helpers > math > integer_squareroot.py > ...

```
1 from eth2spec.bellatrix import mainnet as spec
2 from eth2spec.utils.ssz.ssz_typing import (uint64)
3
4 x = uint64(16)
5 assert spec.integer_squareroot(x) == 4
6
7 x = uint64(25)
8 assert spec.integer_squareroot(x) == 5
```

ESBMC

Verification Output

Handle `integer_squareroot` bound case #3600

Merged hwwhww merged 3 commits into `dev` from `integer_squareroot` 2 weeks ago

Conversation 4 | Commits 3 | Checks 15 | Files changed 5

hwwhww commented 2 weeks ago • edited — Contributor

Credits to the University of Manchester Bounded Model Checking (BMC) project team: Bruno Farias, Youcheng Sun, and Lucas C. Cordeiro for reporting this issue! 🙏 💯

This team is an Ethereum Foundation ESP "Bounded Model Checking for Verifying and Testing Ethereum Consensus Specifications (FY22-0751)" project grantee. They used ESBMC model checker to find this issue.

Description

`integer_squareroot` raises ValueError exception when `n` is maxint of `uint64`, i.e., `2**64 - 1`.

However, we only use `integer_squareroot` in

1. `integer_squareroot(total_balance)`
2. `integer_squareroot(SLOTS_PER_EPOCH)`

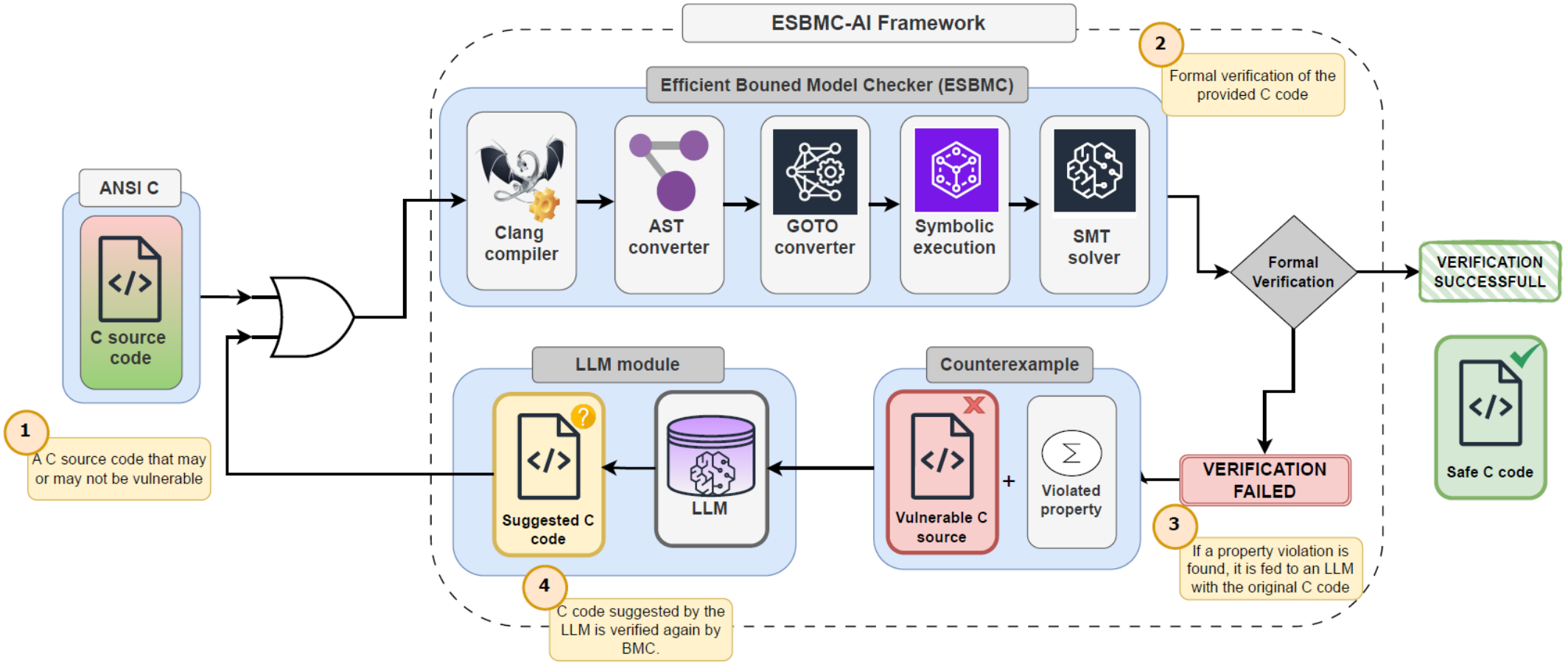
With the current Ether total supply + EIP-1559, it's unlikely to hit the overflow bound in a very long time. (🤞 🔊)

That said, it should be fixed to return the expected value.

Agenda

- Automated program verification and testing to ensure conformance to specifications
- Advancing vulnerability detection to enhance software security beyond the state-of-the-art
- Automated program repair to improve the detection, comprehension, and fixing of software defects
- Summary and vision for future exploration

ESBMC-AI: Automated Program Repair



Charalambous, Y., Tihanyi, N., Jain, R., Sun, Y., Ferrag, M. Cordeiro, L.: A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. In 6th ACM/IEEE International Conference on Automation of Software Test (AST 2025)

How secure is AI-generated Code: A Large-Scale Comparison of Large Language Models

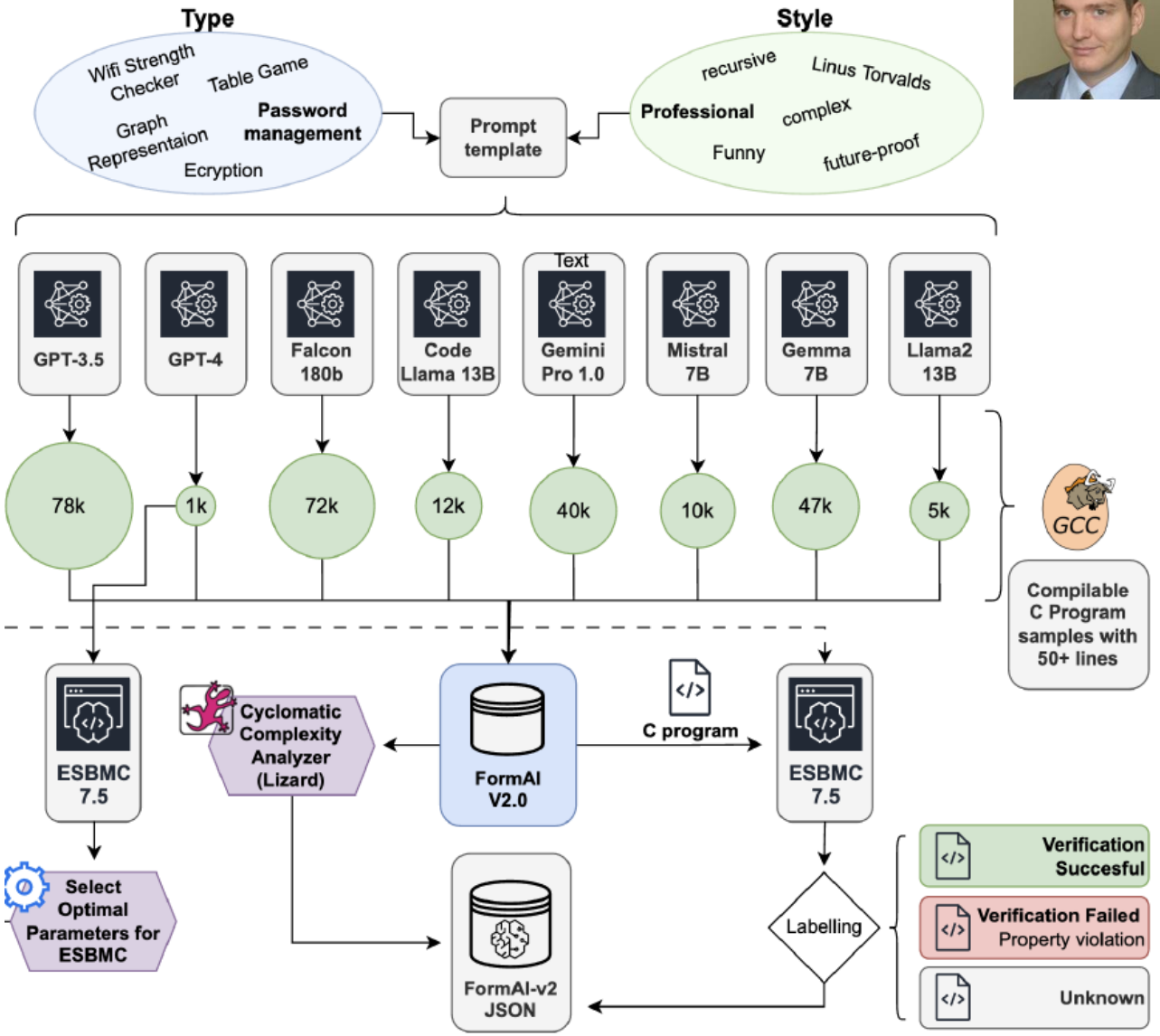
Norbert Tihanyi[1,2], Tamas Bisztray[3], Mohamed Amine Ferrag[4], Ridhi Jain[2], Lucas C. Cordeiro[5,6]

[1] Eötvös Loránd University (ELTE), Budapest, Hungary.
[2] Technology Innovation Institute (TII), Abu Dhabi, UAE.
[3] University of Oslo, Oslo, Norway.
[4]Guelma University, Guelma, Algeria.
[5]The University of Manchester, Manchester, UK.
[6]Federal University of Amazonas, Manaus, Brazil.

Specs \ Datasets	Big-Vul	Draper	SARD	Juliet	Devign	REVEAL	Diverse Vul	FormAI	FormAI v2
Language	C/C++	C/C++	Multi	Multi	C	C/C++	C/C++	C	C
Source	RW	Syn + RW	Syn + RW	Syn	RW	RW	RW	AI	AI
Dataset size	189k	1,274k	101k	106k	28k	23k	379k	112k	150k
Vul. Snippets	100%	5.62%	100%	100%	46.05%	9.85%	7.02%	51.24%	61%
Multi. Vulns.	✘	✔	✘	✘	✘	✘	✘	✔	✔
Compilable	✘	✘	✔	✔	✘	✘	✘	✔	✔
Granularity	Func	Func	Prog	Prog	Func	Func	Func	Prog	Prog
Class. Type	CVE CWE	CWE	CWE	CWE	CVE	CVE	CWE	CWE	CWE
Avg. LOC.	30	29	114	125	112	32	44	79	82
Labelling Method	P	S	B/S/M	B	M	P	P	F	F

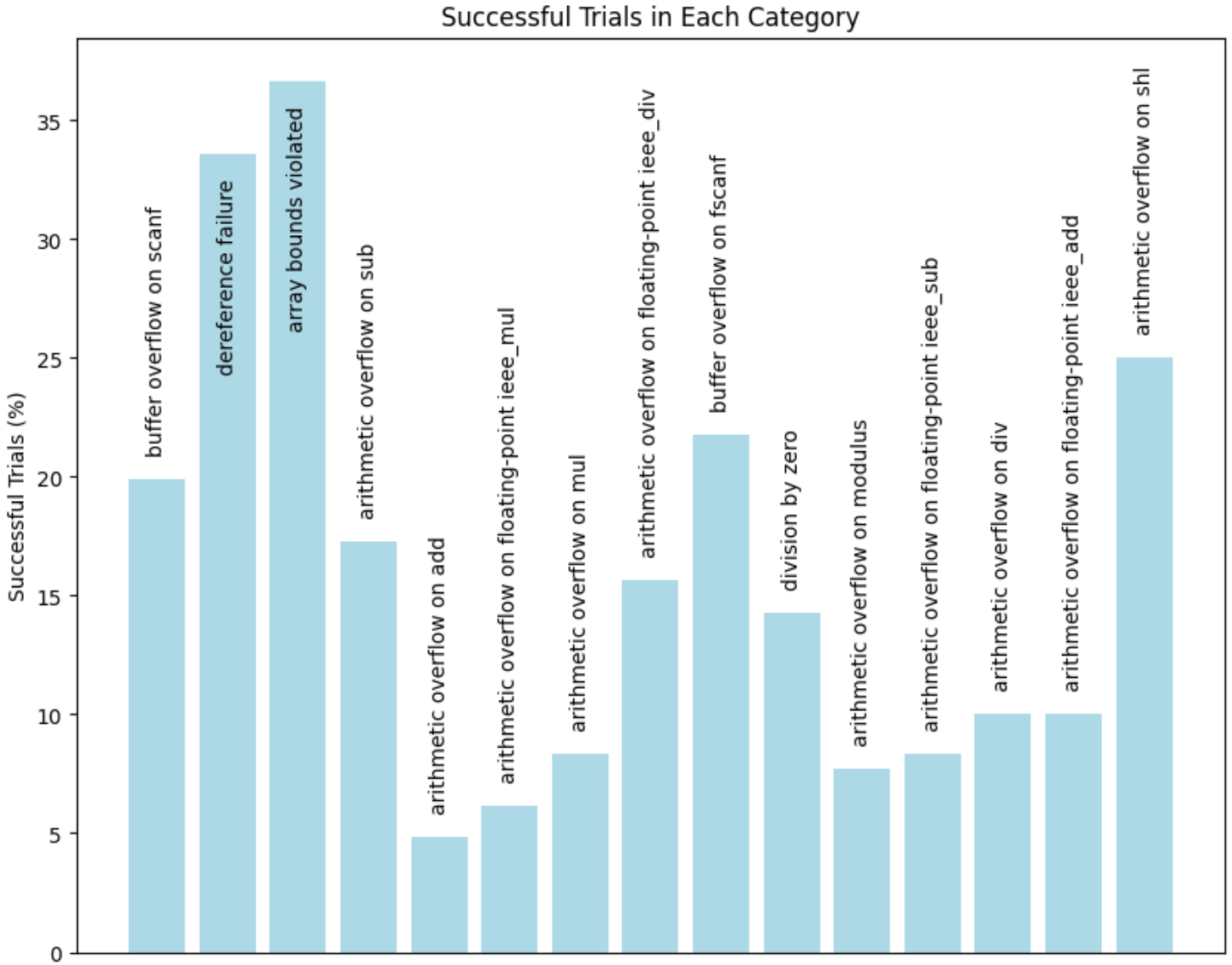
Legend:
Multi: Multi-Language Dataset, **RW**: Real World, **Syn**: Synthetic, **AI**: AI-generated,
Func: Function level granularity, **Prog**: Program level granularity,
CVE: Common Vulnerabilities and Exposures, **CWE**: Common Weakness Enumeration,
P: GitHub Commits Patching a Vulnerability, **S**: Static Analyzer,
B: By Design Vulnerable, **F**: Formal Verification with ESBMC, **M**: Manual Labeling

https://doi.org/10.1007/s10664-024-10590-1



Network Management, Table Games, Wi-Fi Signal Strength Analyzer, QR code reader, Image Steganography, Pixel Art Generator, Scientific Calculator Implementation, and Encryption, string manipulation, etc.

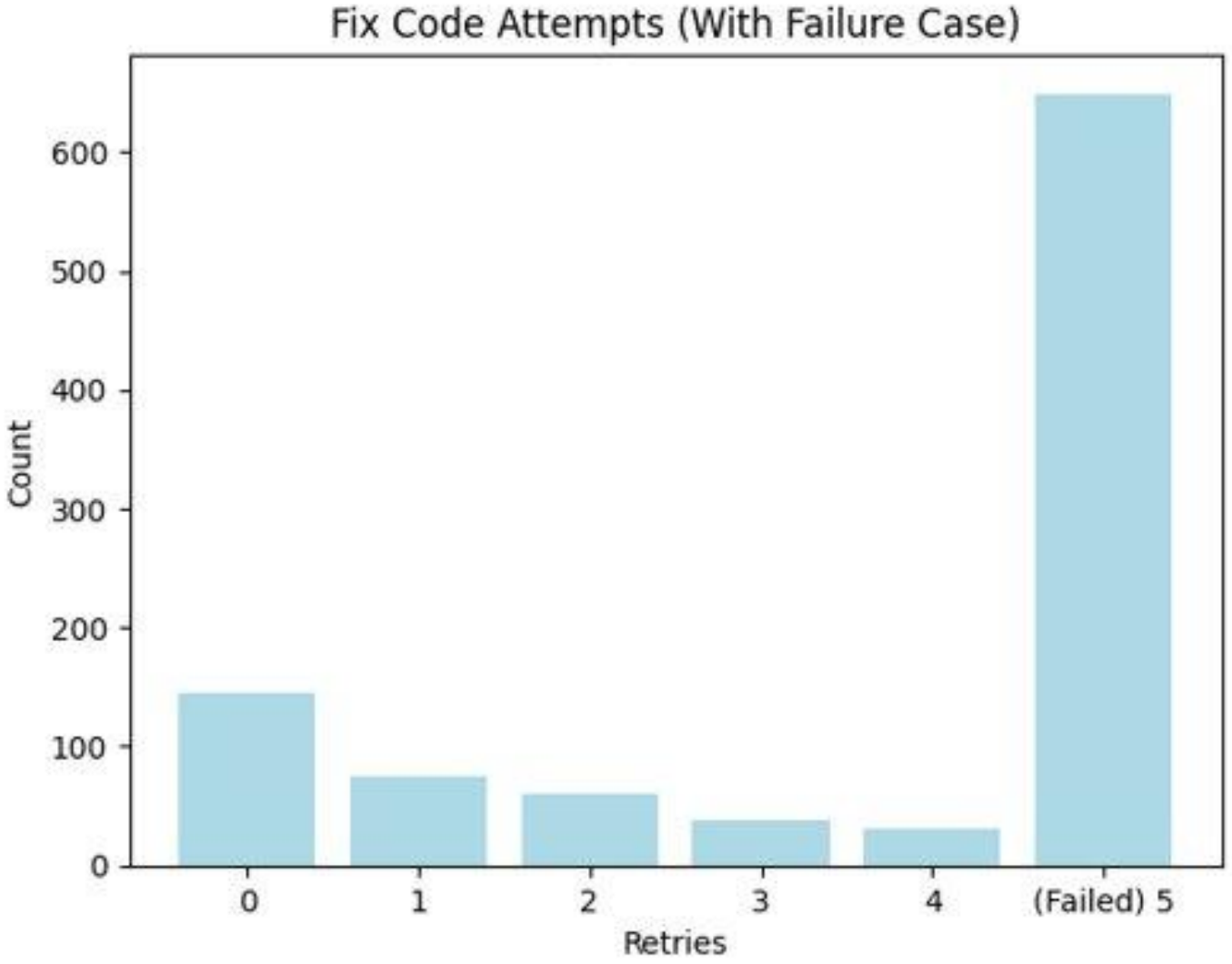
ESBMC-AI Fix Code Mode (FCM) Performance



- **Built the formAI dataset with 112k C programs**
- Randomly selected 1k vulnerable C programs
- Repaired 35.5% programs
- Lowest category was arithmetic overflow (~5%)
- Highest category was array out of bounds (~36%)
- **Generic prompts (room for improvement)**

Charalambous, Y. et al.: A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. Under review at AST 2025.

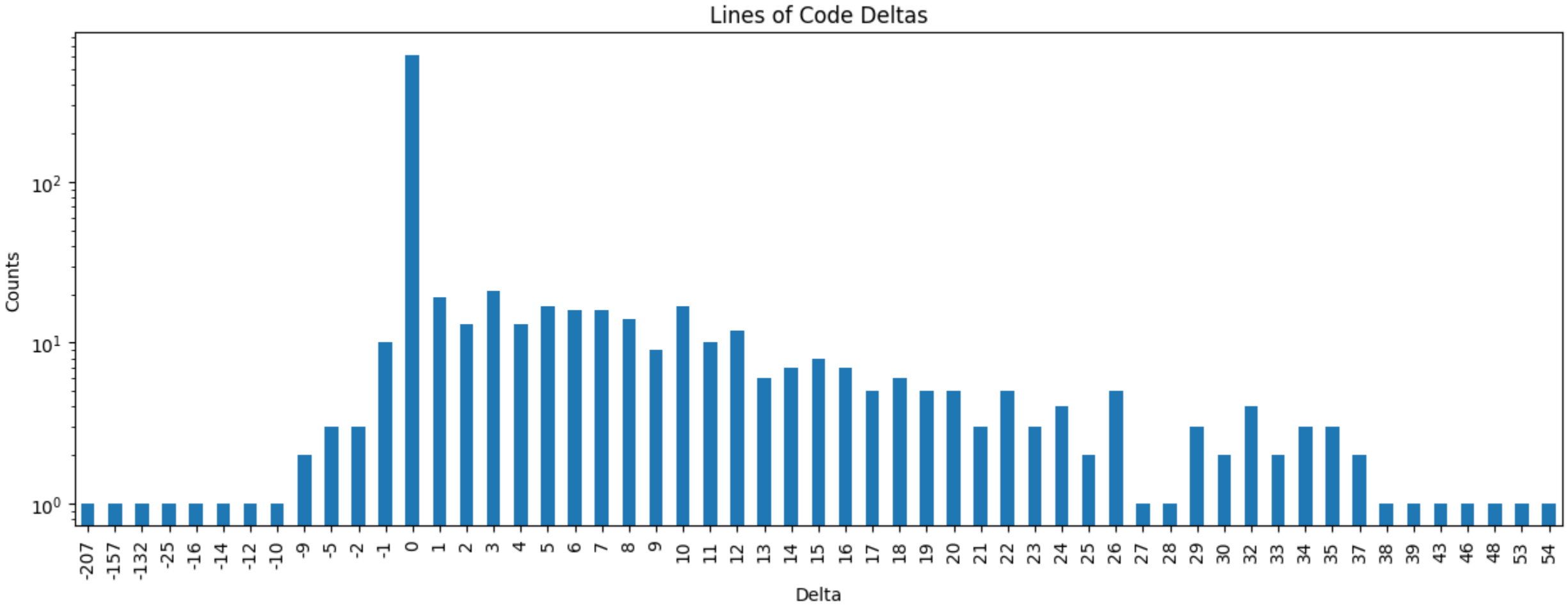
ESBMC-AI Fix Code Mode (FCM) Performance



- **Built the formAI dataset with 112k C programs**
- Randomly selected 1k vulnerable C programs
- Repaired 35.5% programs
- Lowest category was arithmetic overflow (~5%)
- Highest category was array out of bounds (~36%)
- **Generic prompts (room for improvement)**

Charalambous, Y. et al.: A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. Under review at AST 2025.

ESBMC-AI Fix Code Mode (FCM)

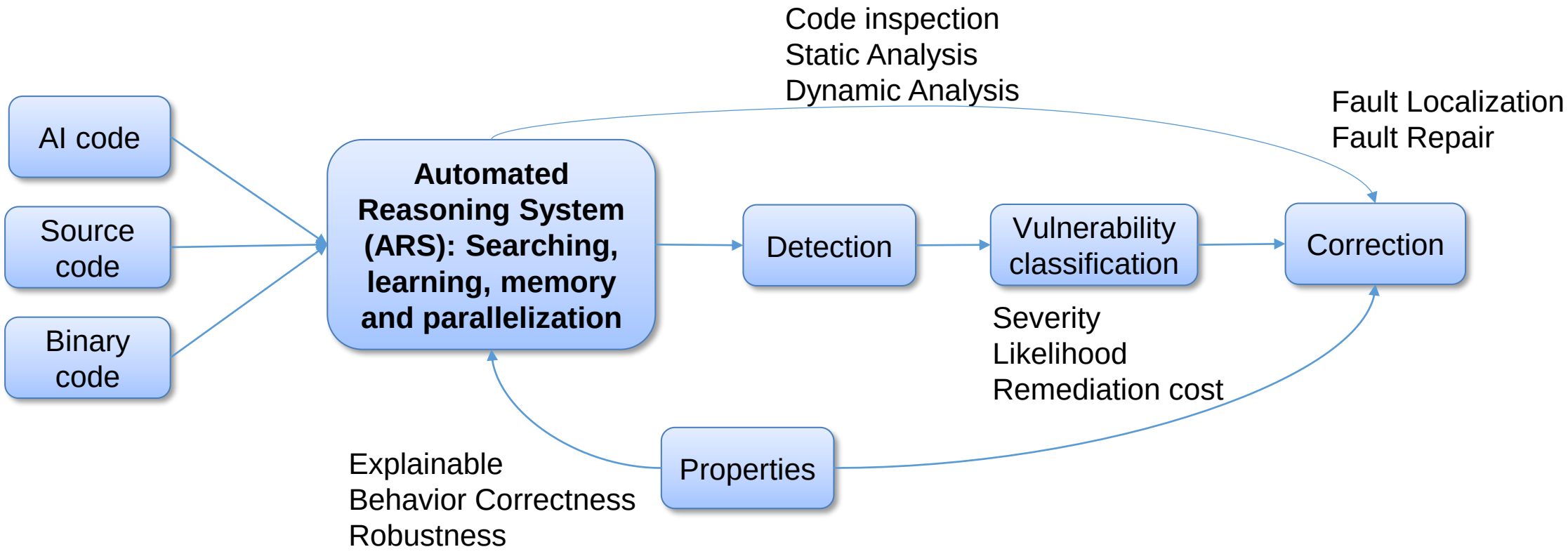


Agenda

- Automated program verification and testing to ensure conformance to specifications
- Advancing vulnerability detection to enhance software security beyond the state-of-the-art
- Automated program repair to improve the detection, comprehension, and fixing of software defects
- Summary and vision for future exploration

Vision: Building Trustworthy Software and AI Systems

Develop an automated reasoning system for **safeguarding software and AI systems** against vulnerabilities in an increasingly digital and interconnected world



(Real) Impact: Students and Contributors

- 5 PhD theses
- 30+ MSc dissertations
- 30+ final-year projects

- GitHub:
  - 43 contributors
  - 24,268 commits
  - 315 stars
  - 101 forks

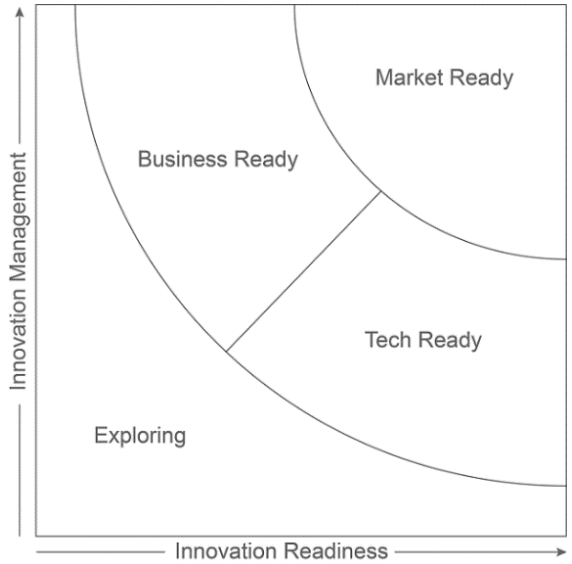
https://github.com/esbmc/esbmc

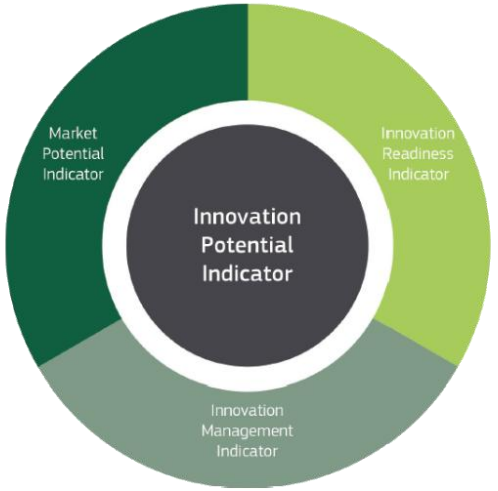
Impact: Awards and Industrial Deployment

- **Distinguished Paper Award** at ICSE'11 and ASE'24
- **Best Paper Award** at SBESC'15
- **Most Influential Paper Award** at ASE'23
- **Best Tool Paper Award** at SBSeg'23
- **46 awards** from intl. competitions on SW verification/testing at **TACAS/FASE**
  - Bug Finding and Code Coverage 🥇
- **Intel** deploys **ESBMC** in production as one of its verification engines for **verifying firmware in C**
- **Nokia and ARM** have found **security vulnerabilities** in **C/C++ software**
- **Funded by the government** (EPSRC, British Council, Royal Society, CAPES, CNPq, FAPEAM) and **industry** (Intel, Motorola, Samsung, Nokia, ARM)
- **Potential spin-out** about building trustworthy software and AI systems

The European Commission recognised our code verification framework as an outstanding innovation

- *“We believe that your organisation's inclusion in this initiative could open up new opportunities for you to partner with business or academic organisations and trigger interest from potential customers or investors in your innovations”*
    - **Innovation Title:** ELEGANT code verification mechanisms;
    - **Market Maturity of the Innovation:** Exploring
    - **Market Creation Potential of the innovation:** High





Commercial Impact

- UKRI IAA with SES Software Escrow
- Identifying security vulnerabilities in code
- FuSeBMC has secured more than £120,000 in funding from Innovate UK and the Innovation Factory, demonstrating external support for its innovation potential

Software spin-out wins funding from Innovation Factory

Wednesday, February 14, 2024 Research

Innovate UK | Business Connect

CyberASAP

The only pre-seed accelerator programme in the cybersecurity ecosystem, CyberASAP (Cyber Security Academic Startup Accelerator Programme) plays a unique and vital role in supporting cyber security innovation and commercialisation.

https://iuk.ktn-uk.org/programme/cyberasap/

VeriBee

https://www.veribee.co/

Advisory Committee by Deeptech Labs: £200,000

S3 Research Group

Security is one of the priority growth areas in the CS department's ~£10m expansion plan, and we already have **two new L/SL and one Professor posts advertised**

L. Cordeiro
(Software Security)

R. Banach
(Formal Methods)

M. Mustafa
(Applied Cryptography)

N. Zhang
(IoT Security)

E. Manino
(AI Security)

B. Magri
(Cryptography)

Y. Sun
(Trustworthy AI)

D. Dresner
(System Governance)

A. Creswell
(Cyber and AI)

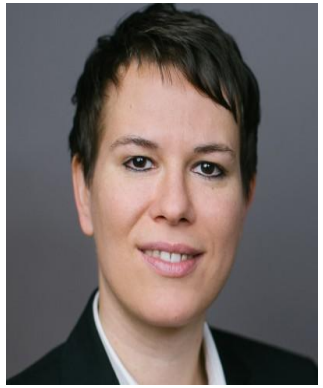
G. Smith
(Former Director General for Technology at GCHQ)

Collaborators across the University

K. Korovin
(Automated Reasoning)

G. Reger
(Automated Reasoning)

C. Kotselidis
(Computer Architecture)



Meropi Tzanetakis
(Digital Criminology)

Kaled Alshmrany
(Software Testing)

M. Luján
(Computer Architecture)

J. Goodacre
(Computer Architecture)

P. Olivier
(Computer Architecture)

Richard Allmendinger
(Applied AI)

S3 Research Objectives

We develop state-of-the-art algorithms, methods, and protocols to address **security** and **privacy** in **networked and distributed system** environments, and tools to build verifiable, **trustworthy software and AI/ML systems**

21 March 2024

University of Manchester recognised as Academic Centre of Excellence in Cyber Security Research

Acknowledgements