Collaborators/funders:
PPGEE, PPGI – UFAM
APT / FM / S3 Research Groups
ARM Centre of Excellence
Centre for Digital Trust and Society
UKRI, EPSRC, EU Horizon and industrial partners

UFAM

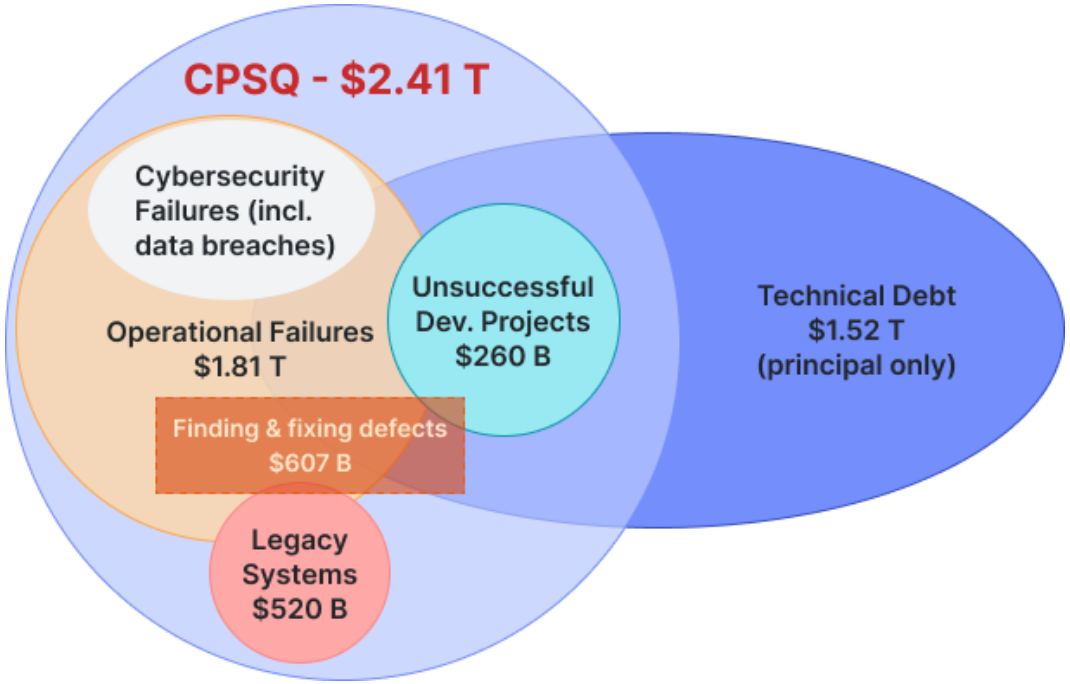
The University of Manchester

Building Trustworthy Software and AI Systems: Exploring Automated Testing, Formal Verification, and Repair Strategies

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

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™

Objective of this talk

Discuss **automated testing, formal verification, and repair** techniques to establish a **robust foundation for building trustworthy software and AI systems**

- Introduce a **logic-based automated verification platform** to find and repair **software vulnerabilities**
- Explain **testing, verification, and repair** 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

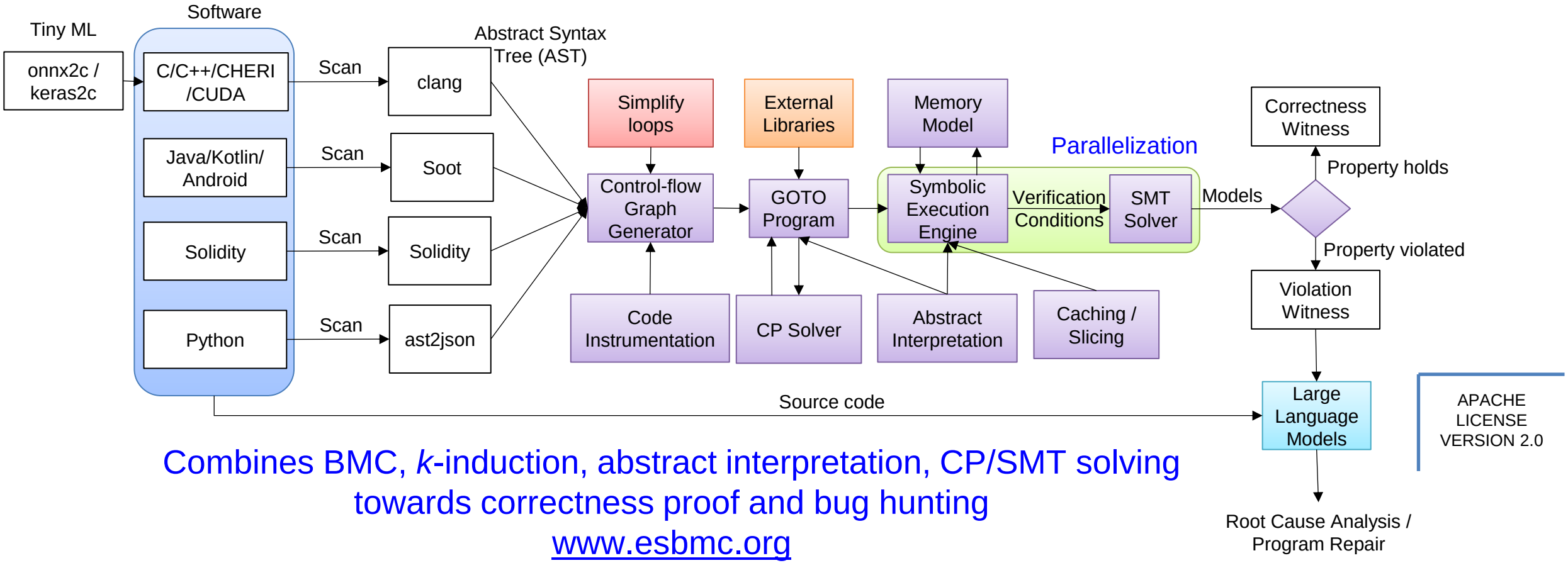
Given a **program** and a **specification**, can we automatically **verify** that the **program performs as specified**?

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**



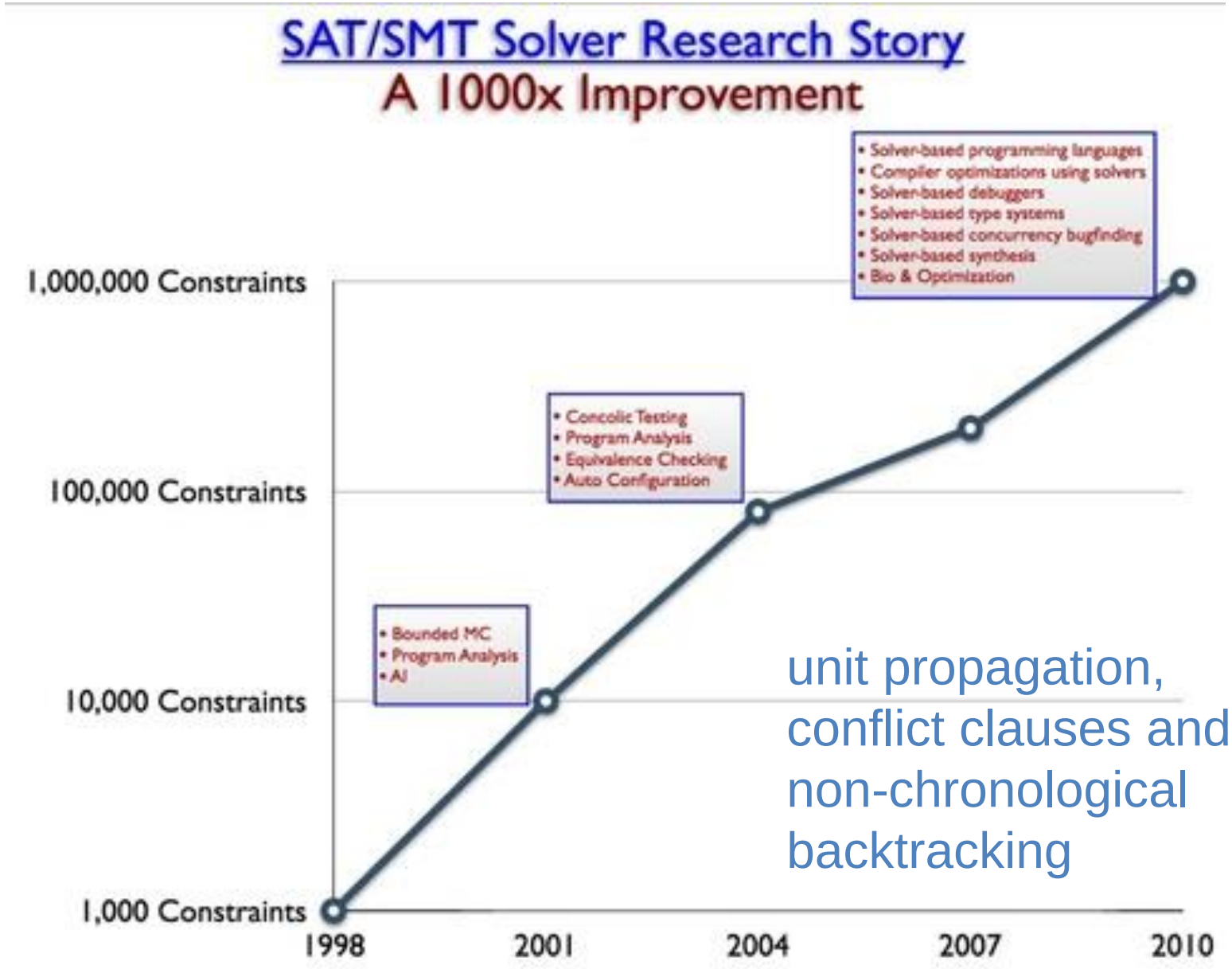
Combines BMC, *k*-induction, abstract interpretation, CP/SMT solving
towards correctness proof and bug hunting

www.esbmc.org

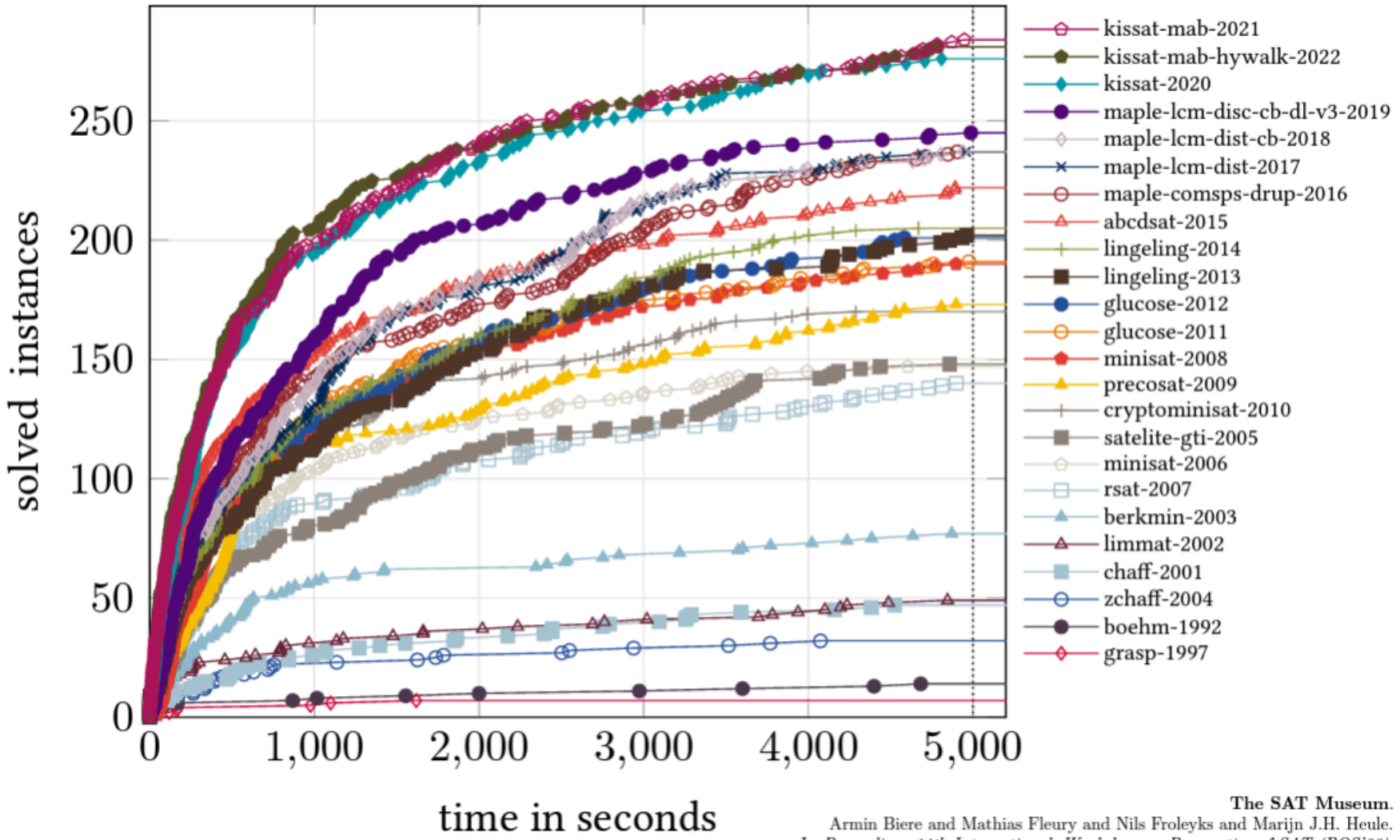
Agenda

- Automated Software Testing and Verification with the ESBMC Framework
- Towards Self-Healing Software via Large Language Models and Formal Verification
- Automated Reasoning System for Building Trustworthy SW and AI Systems

SAT solving as enabling technology



SAT Competition All Time Winners on SAT Competition 2022 Benchmarks

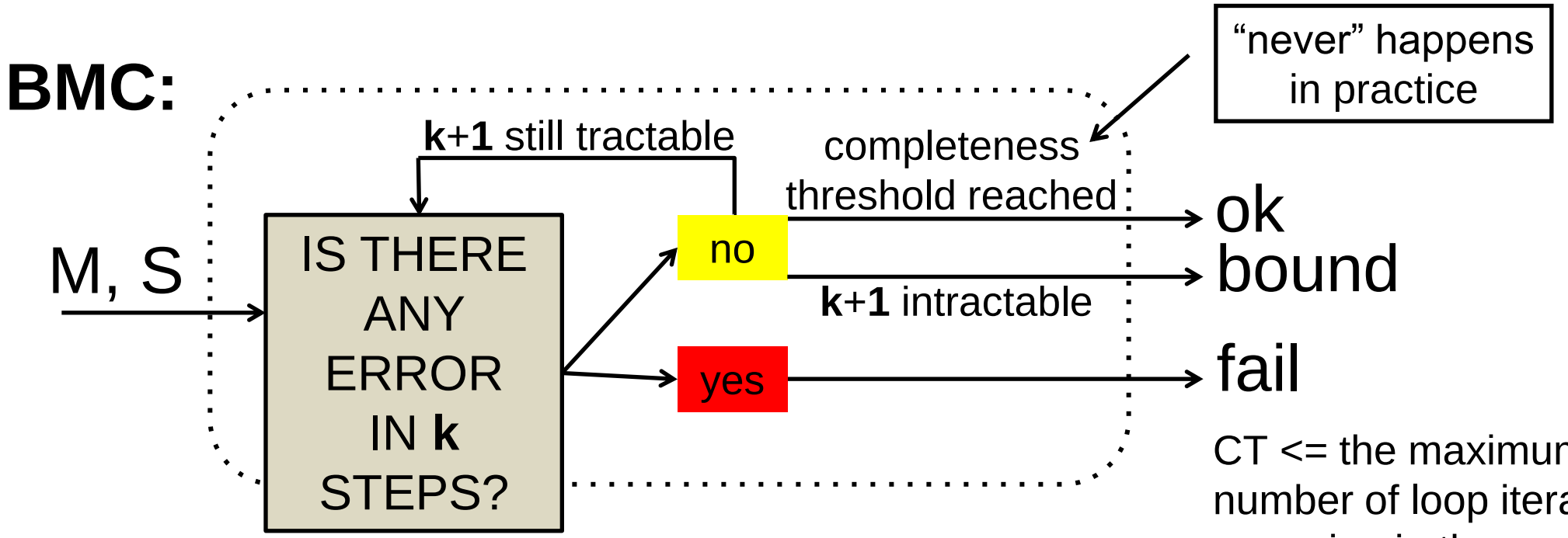


https://cca.informatik.uni-freiburg.de/satmuseum

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

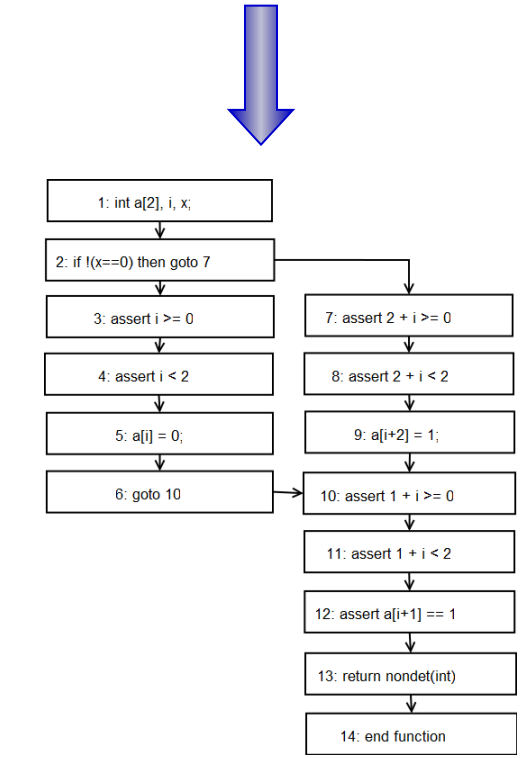
Property fails in some step

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

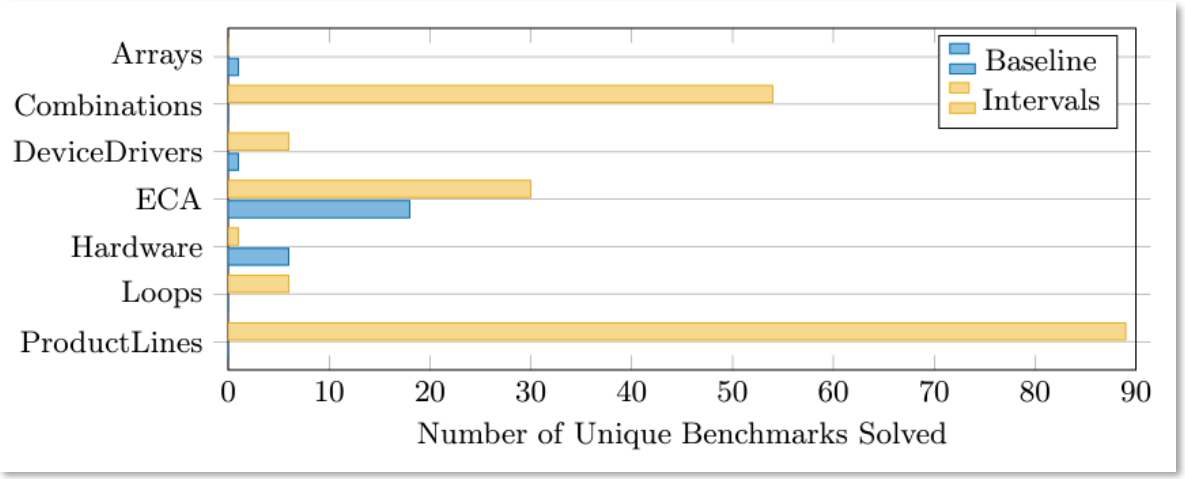
```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```



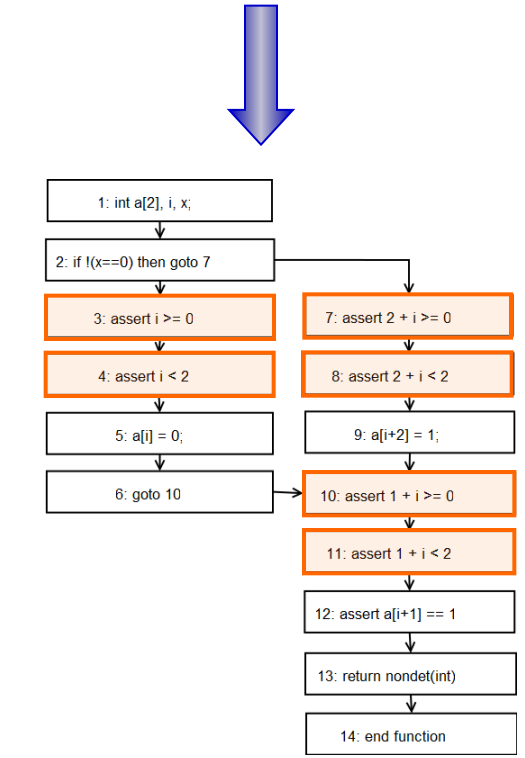
Software BMC

- 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);
}
```

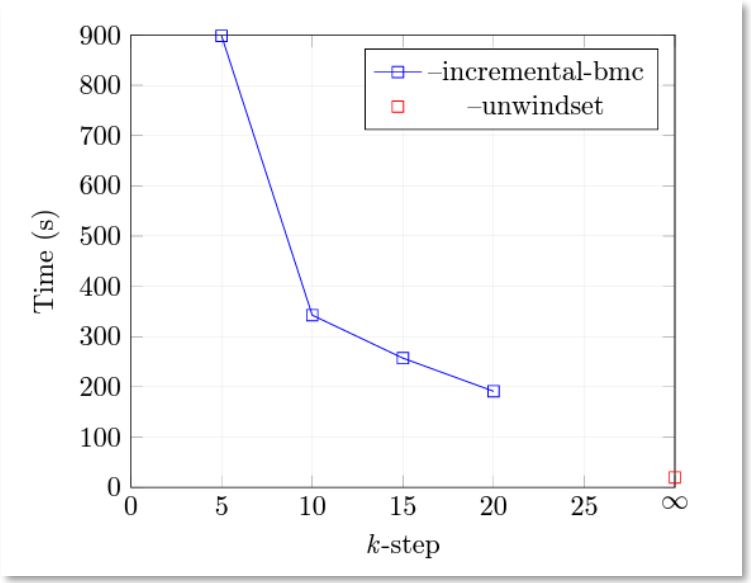


Menezes, R., Manino, E., Shmarov, F., Aldughaim, M., de Freitas, R., Lucas C. Cordeiro: Interval Analysis in Industrial-Scale BMC Software Verifiers: A Case Study.



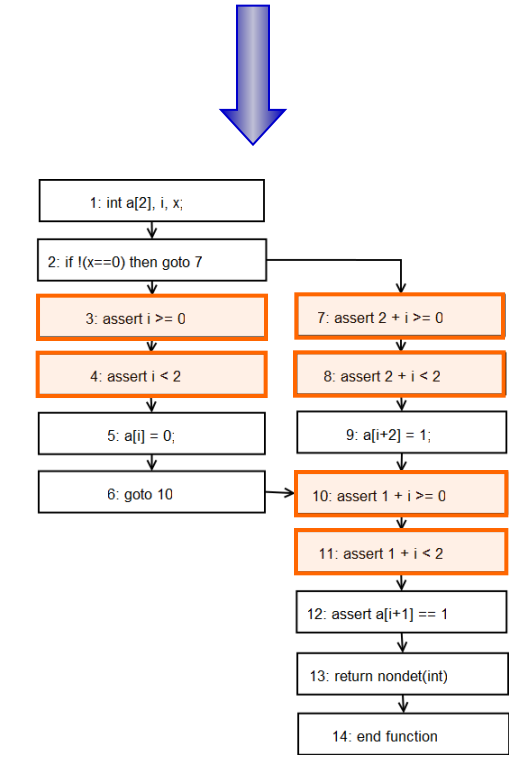
Software BMC

- 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);
}
```

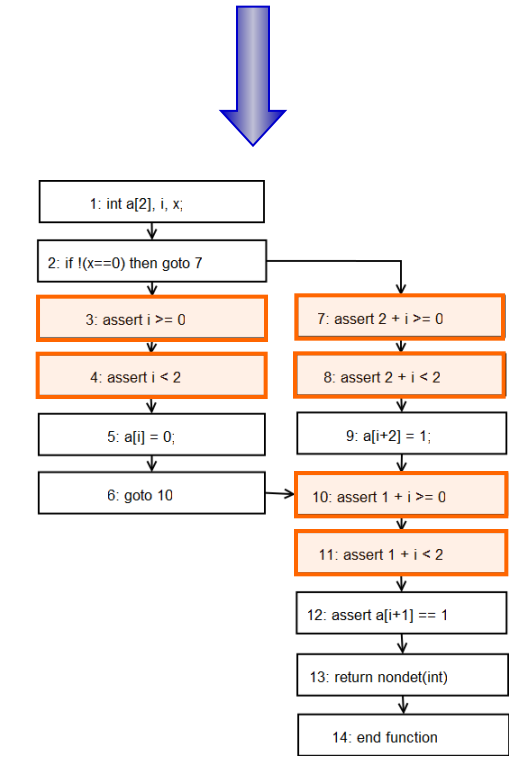


Software BMC

- 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);
}
```

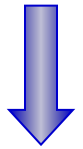


Software BMC

- 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*

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

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

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

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

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

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

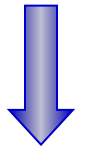
$$P := \begin{bmatrix} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{bmatrix}$$

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);
}
```



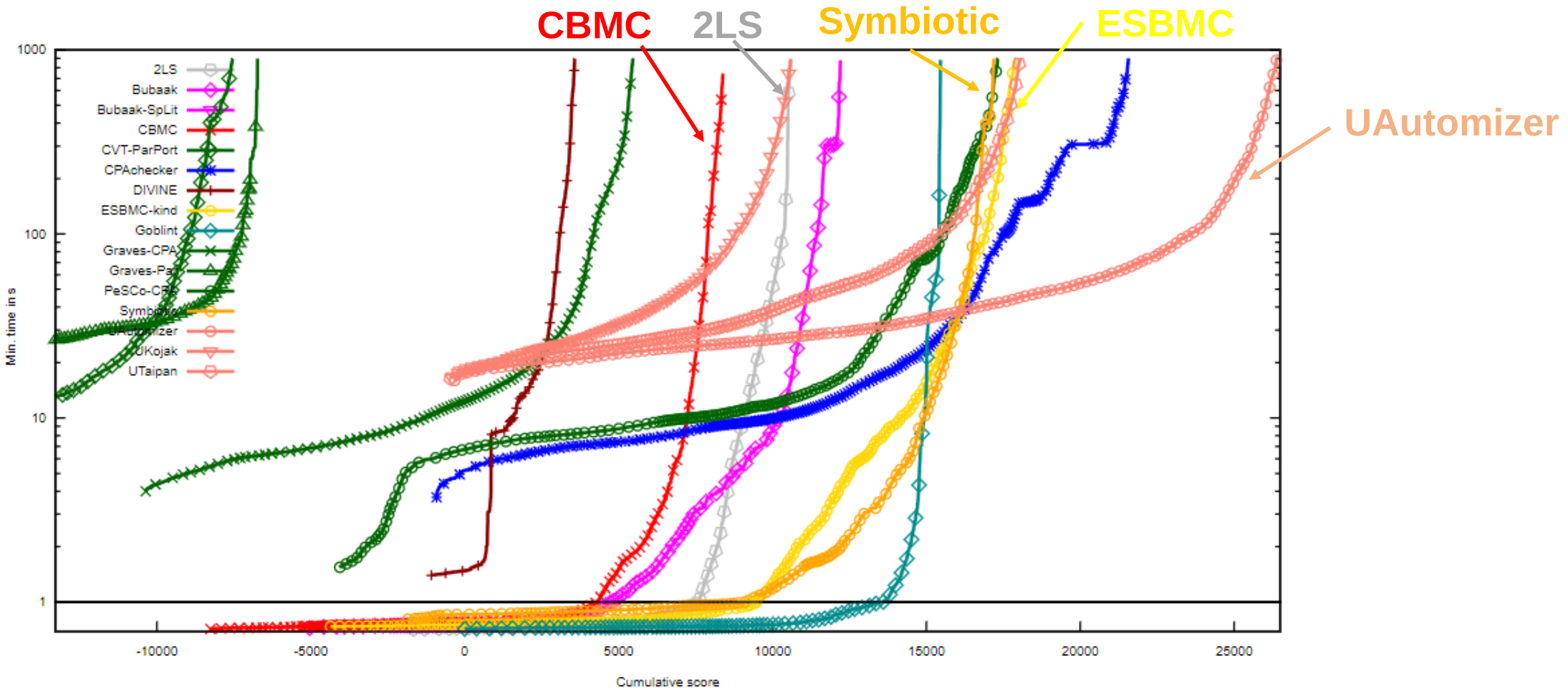
$$C := \begin{bmatrix} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{bmatrix}$$

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

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)

Intl. Software Verification Competition (SV-Comp 2024)

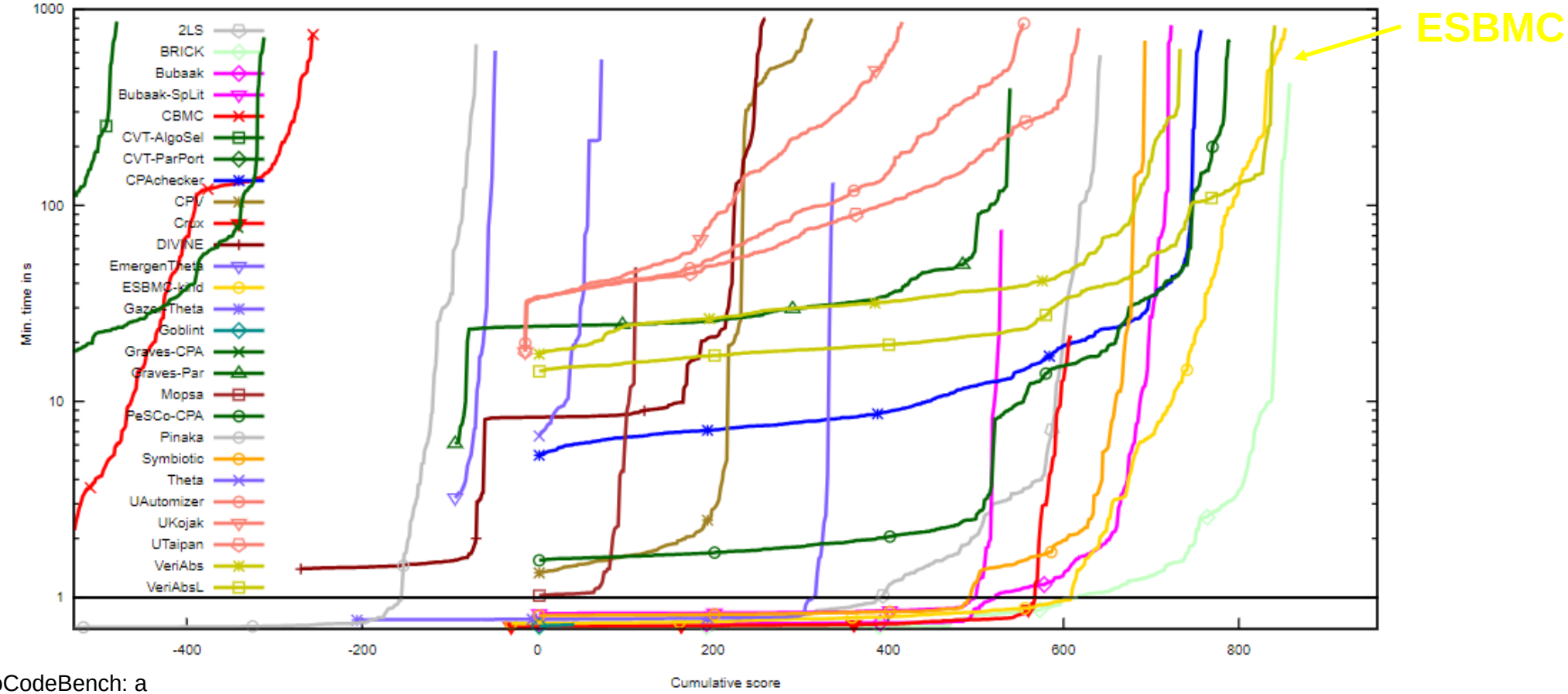
- SV-COMP 2024, 30300 verification tasks, max. score: 49097



Verification of the `Overall` Category

From Floating-Point Programs to Neural Network Implementations

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

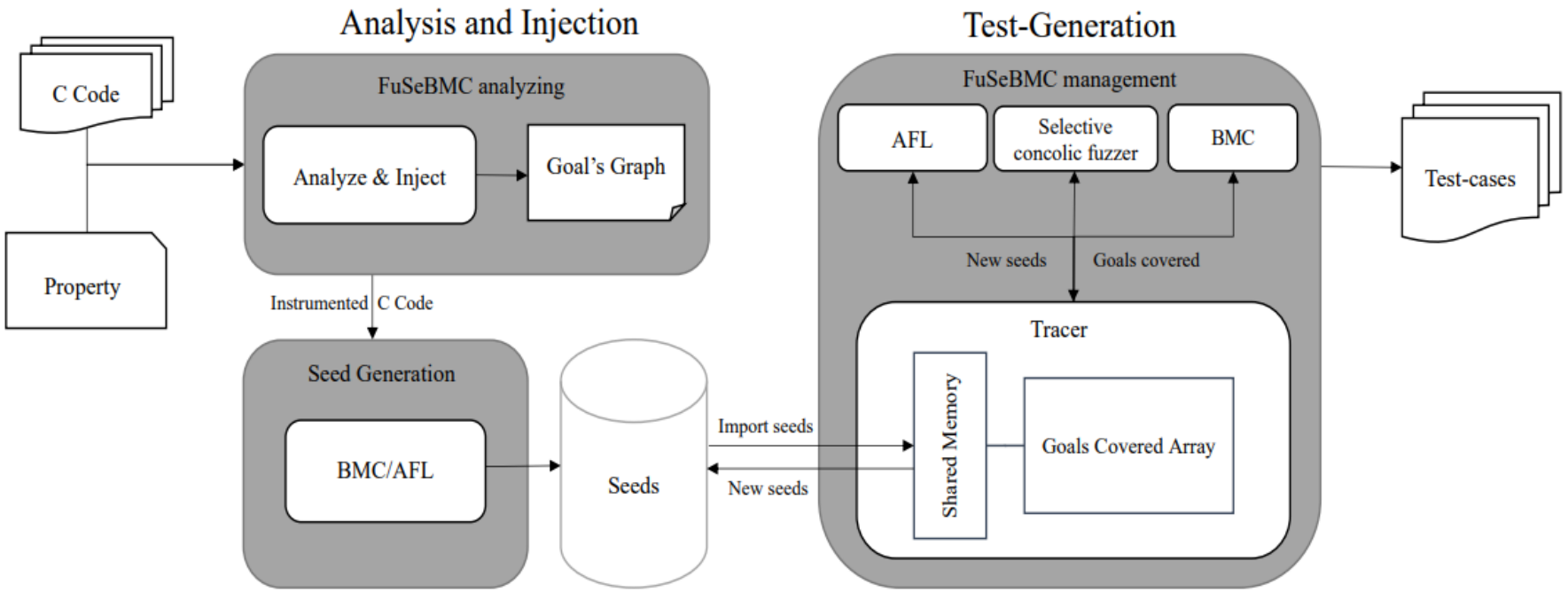


Verification of the `ReachSafety-Floats` Category

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

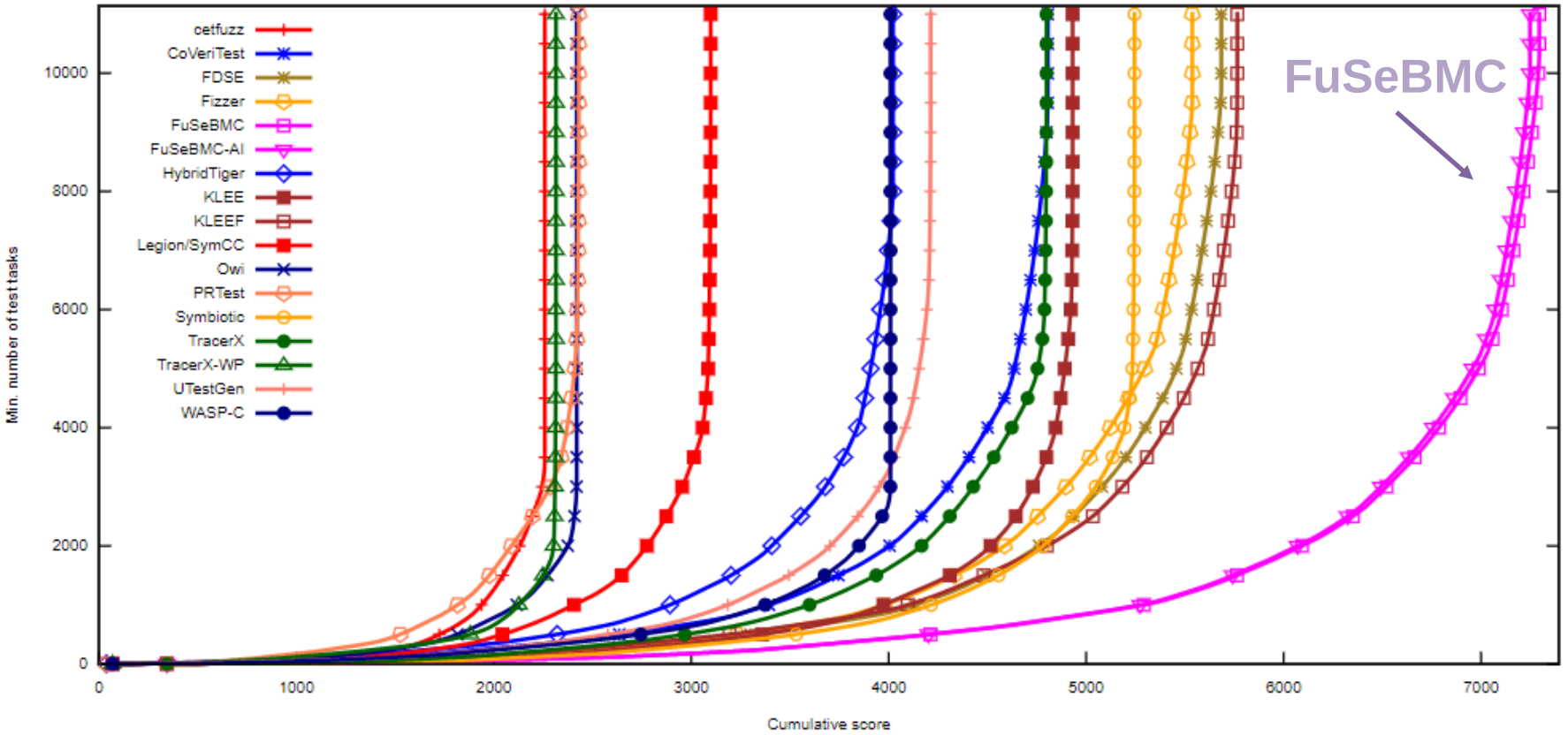
FuSeBMC v4 Framework

- 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

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



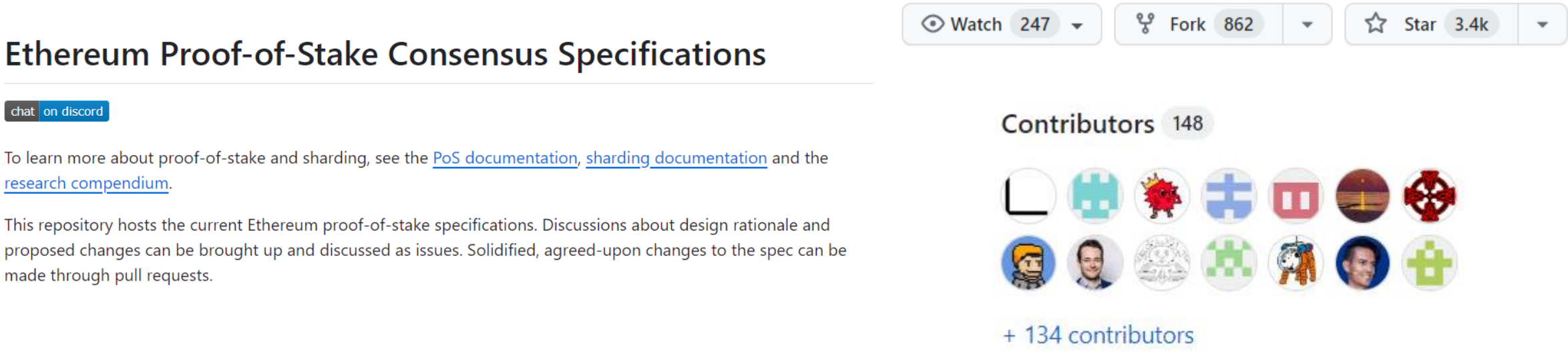
FuSeBMC achieved 3 awards: 1st place in `Cover-Error`, 1st place in `Cover-Branches`, and 1st place in `Overall`

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/

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.

WolfMQTT Verification

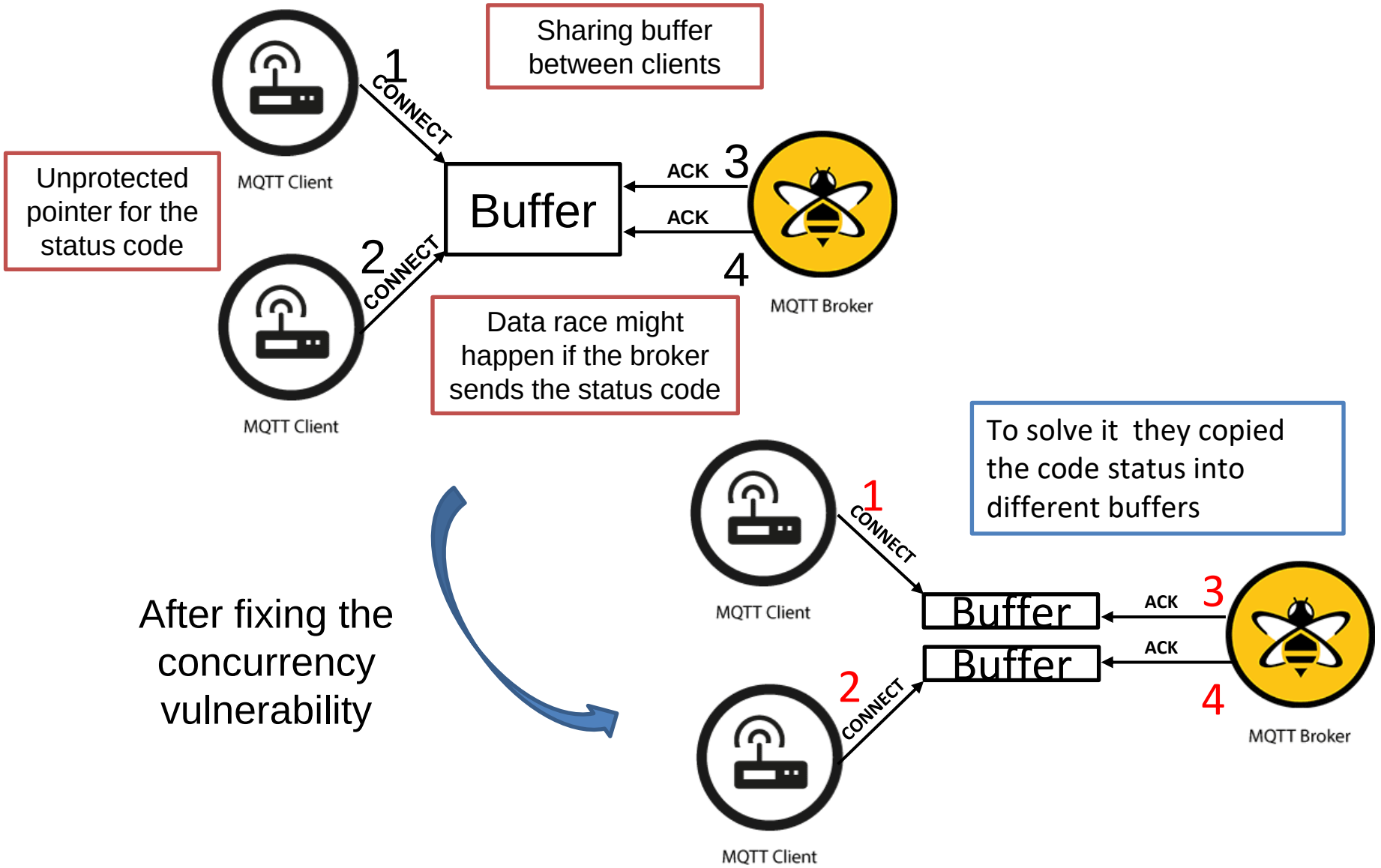
- **wolfMQTT** library is a client implementation of the MQTT protocol written in C for **IoT devices**

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

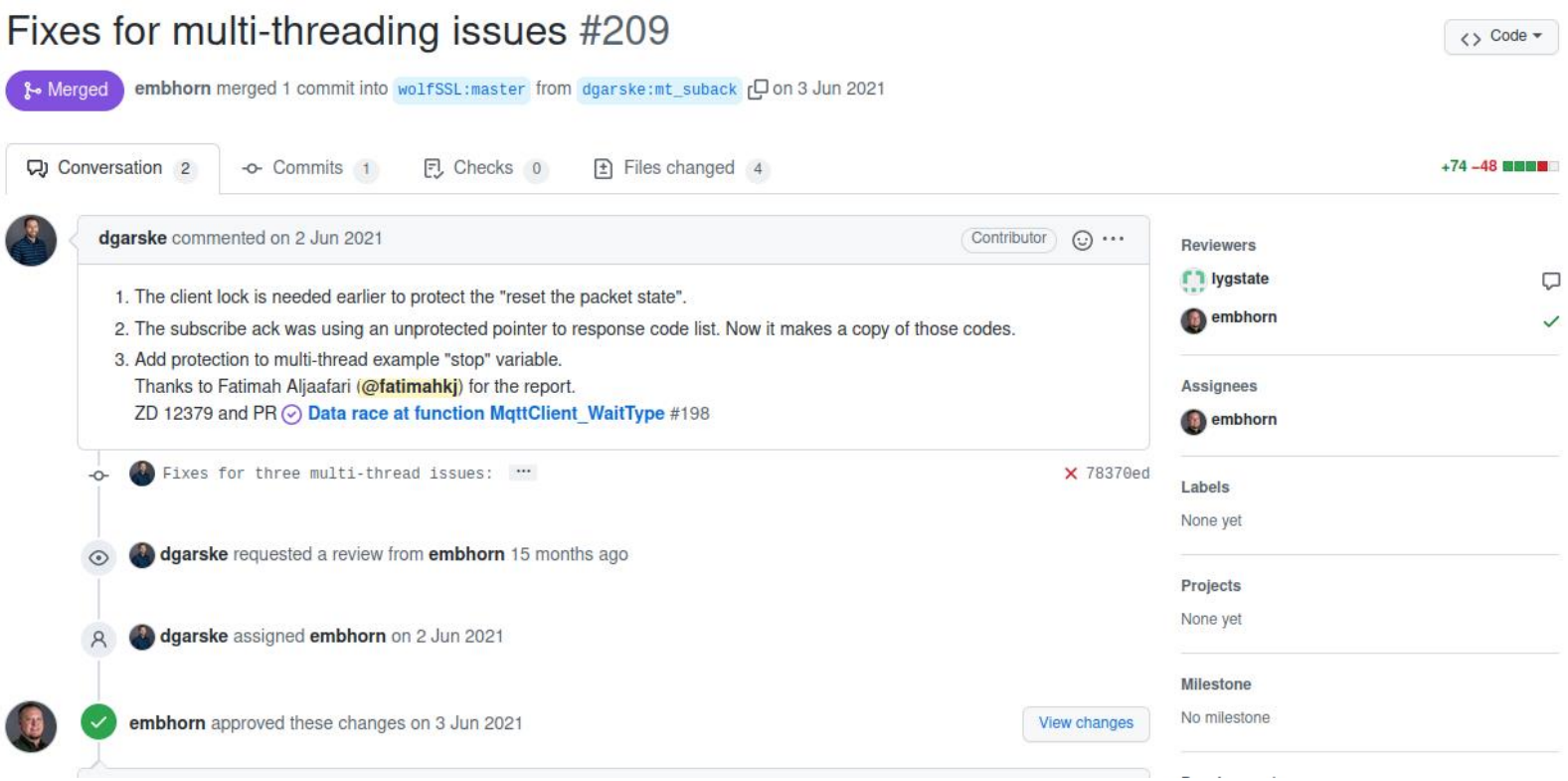
```
Int main(){
Pthread_t th1, th2;
static MQTTCtx mqttCtx;
pthread_create(&th1, subscribe_task, &mqttCtx))
pthread_create(&th2, waitMessage_task, &mqttCtx))}

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;
.....}
```

Here is where the data race might happen! Unprotected pointer

WolfMQTT Verification



Bug Report



Fixes for multi-threading issues #209

Code

Merged **embhorn** merged 1 commit into `wolfSSL:master` from `dgarske:mt_suback` on 3 Jun 2021

Conversation 2 · Commits 1 · Checks 0 · Files changed 4 · +74 −48

dgarske commented on 2 Jun 2021 — Contributor

1. The client lock is needed earlier to protect the "reset the packet state".
2. The subscribe ack was using an unprotected pointer to response code list. Now it makes a copy of those codes.
3. Add protection to multi-thread example "stop" variable.
   Thanks to Fatimah Aljaafari (@fatimahkj) for the report.
   ZD 12379 and PR Data race at function MqttClient_WaitType #198

`Fixes for three multi-thread issues: ···` ✕ 78370ed

dgarske requested a review from **embhorn** 15 months ago

dgarske assigned **embhorn** on 2 Jun 2021

embhorn approved these changes on 3 Jun 2021 — View changes

Reviewers: lygstate, embhorn

Assignees: embhorn

Labels: None yet

Projects: None yet

Milestone: No milestone

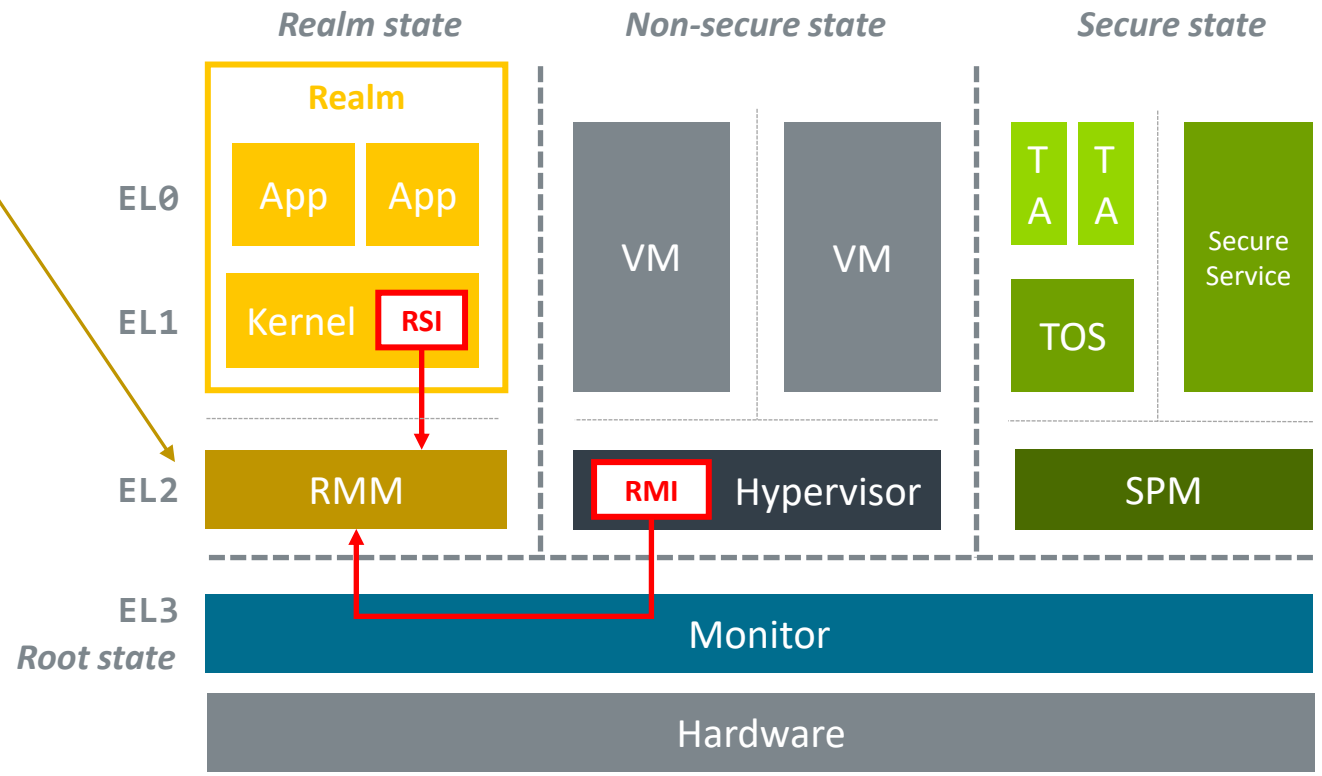
https://github.com/wolfSSL/wolfMQTT


wolfSSL
TLS 1.3
wolfSSL

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 (to appear)

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

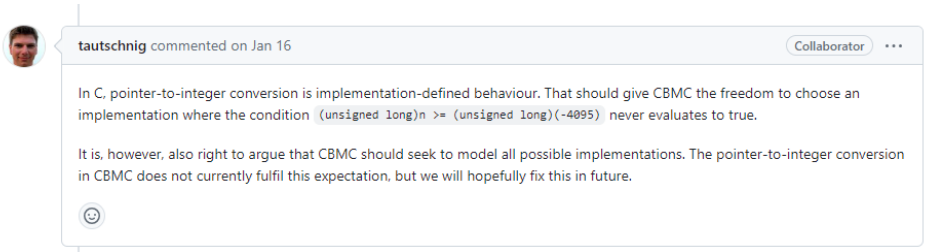
- The document is generated from a **machine-readable specification** (MRS).

[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);
}
```



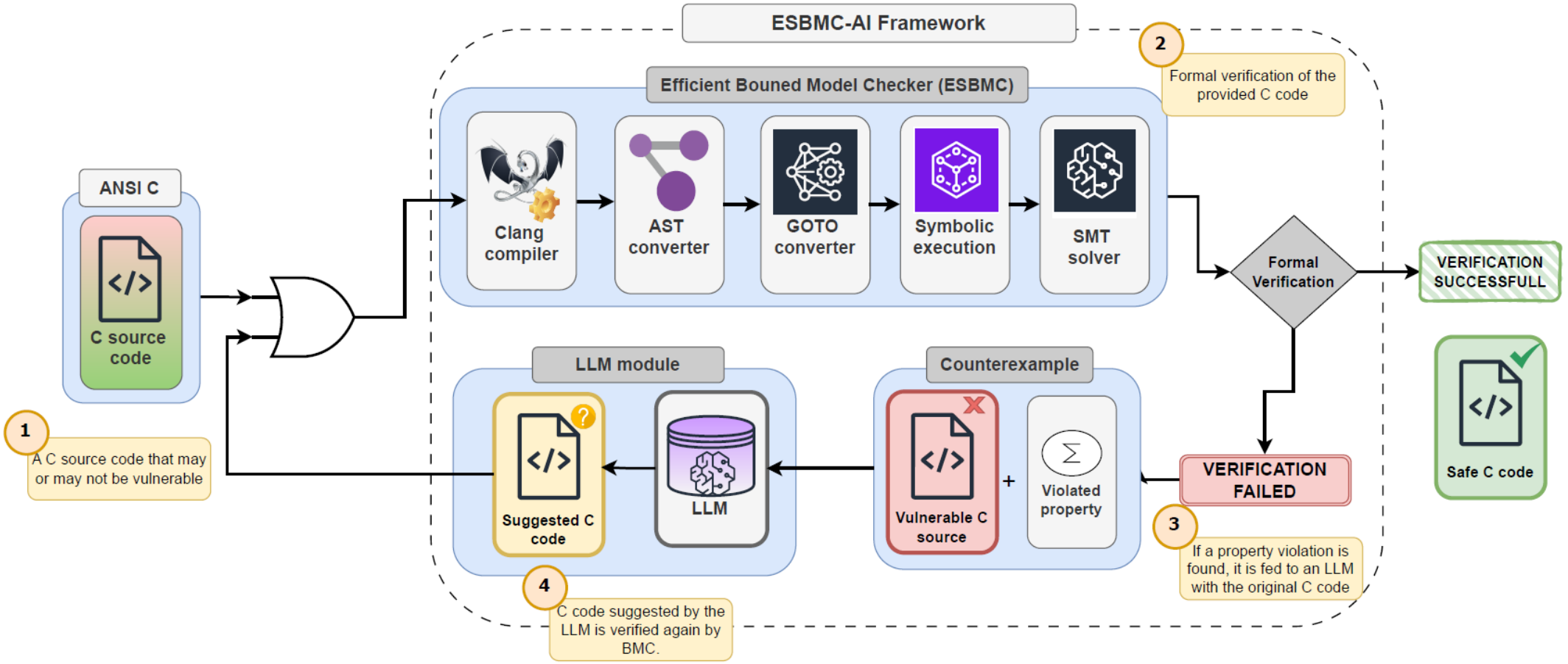
https://github.com/diffblue/cbmc/issues/8161

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

Agenda

- Automated Software Testing and Verification with the ESBMC Framework
- Towards Self-Healing Software via Large Language Models and Formal Verification
- Automated Reasoning System for Building Trustworthy SW and AI Systems

Towards Self-Healing Software via Large Language Models and Formal Verification



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. CoRR abs/2305.14752 (2023)

Do Neutral Prompts Produce Insecure Code? FormAI-v2 Dataset: Labelling Vulnerabilities in Code Generated by Large Language Models

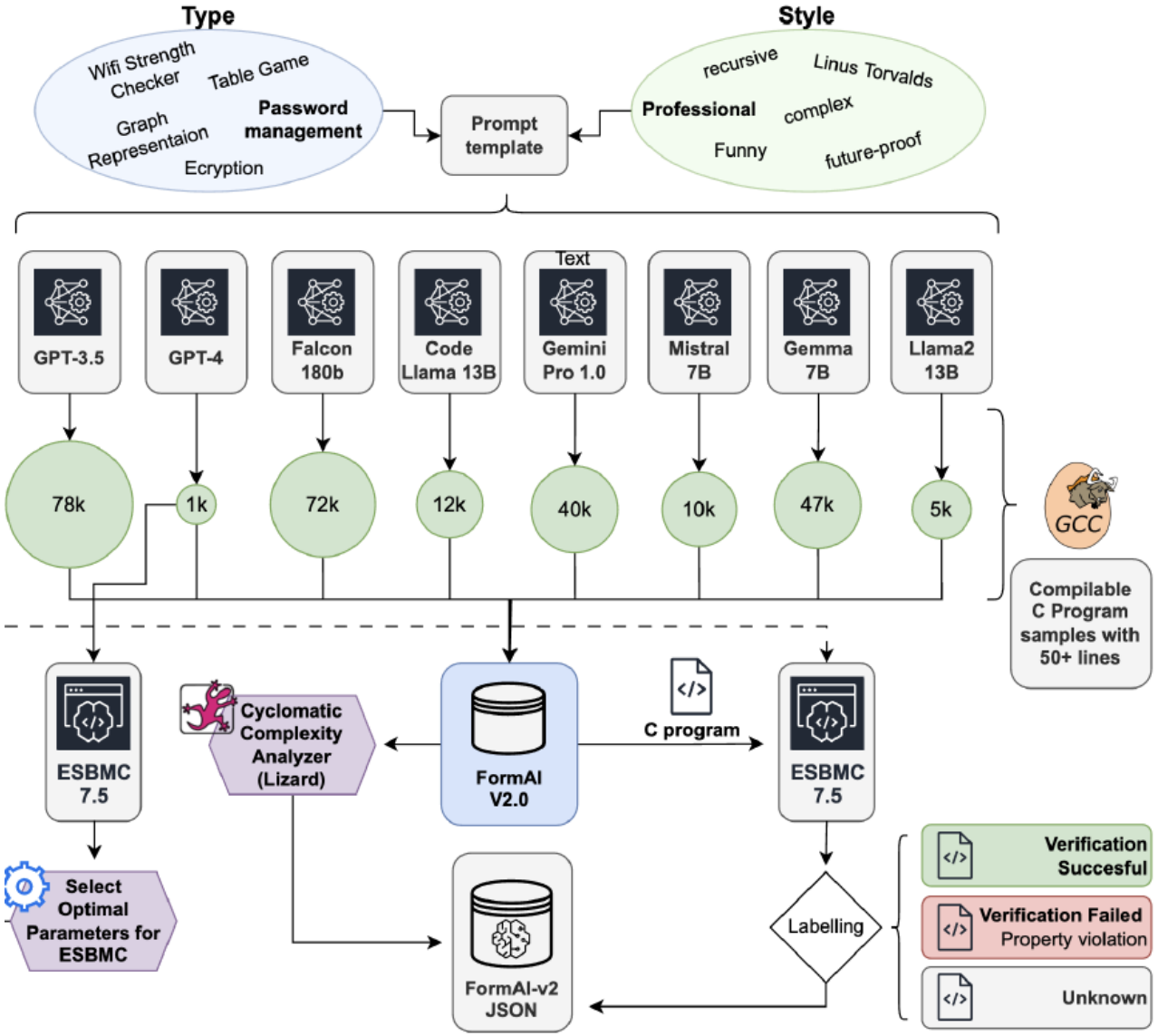
Norbert Tihanyi[1*], Tamas Bisztray[2], Mohamed Amine Ferrag[1], Ridhi Jain[1], Lucas C. Cordeiro[3]

[1] Technology Innovation Institute (TII), Abu Dhabi, UAE.
[2] University of Oslo, Oslo, Norway.
[3]University of Manchester, Manchester, UK.

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

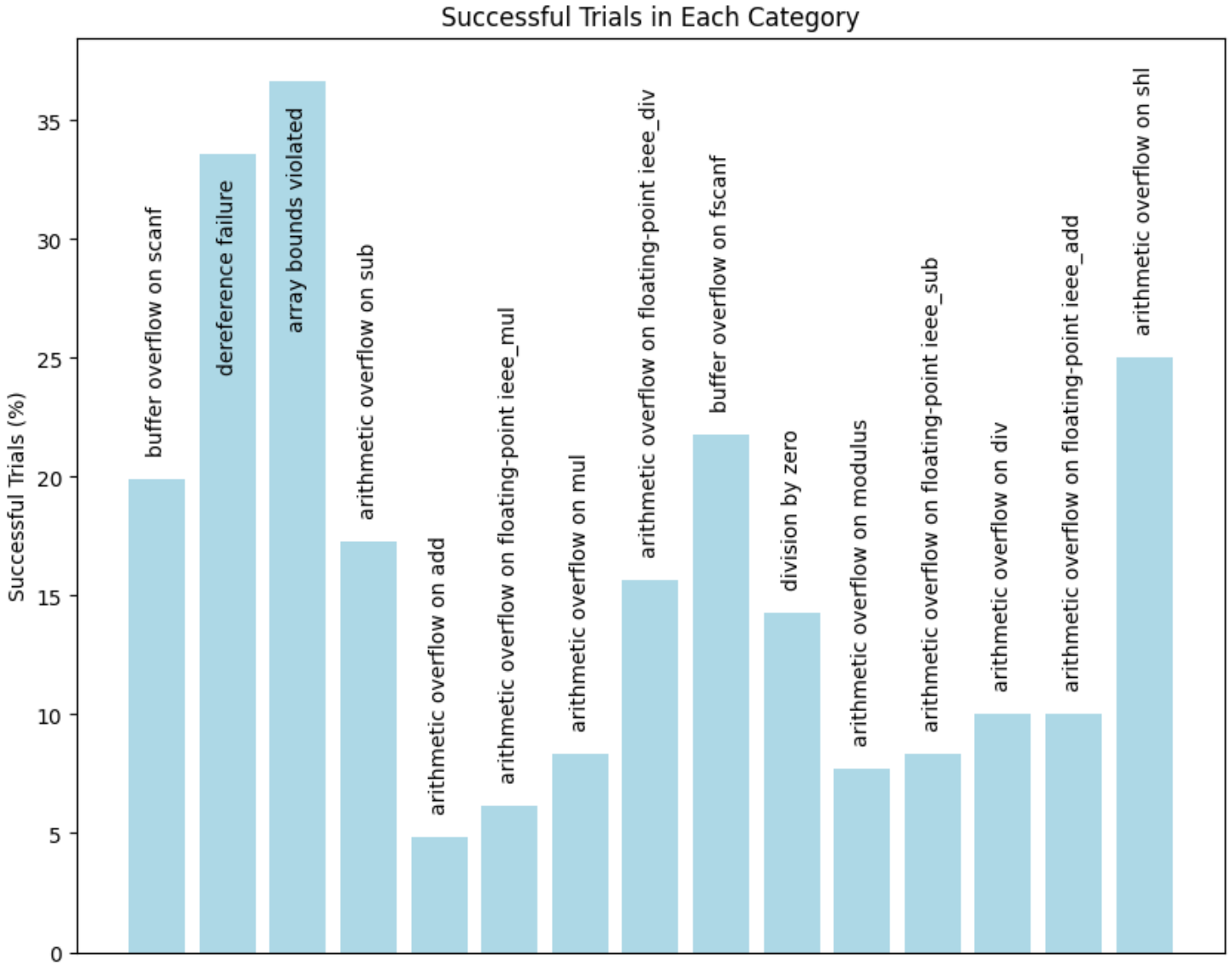
Legend:
Multi: Multi-Language Dataset, **RW**: Real World, **Syn**: Synthetic, **AI**: AI-generated,
Func: Function level granularity, **Prog**: Program level granularity,
CVE: Common Vulnerabilities and Exposures, **CWE**: Common Weakness Enumeration,
P: GitHub Commits Patching a Vulnerability, **S**: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

https://arxiv.org/abs/2404.18353



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.

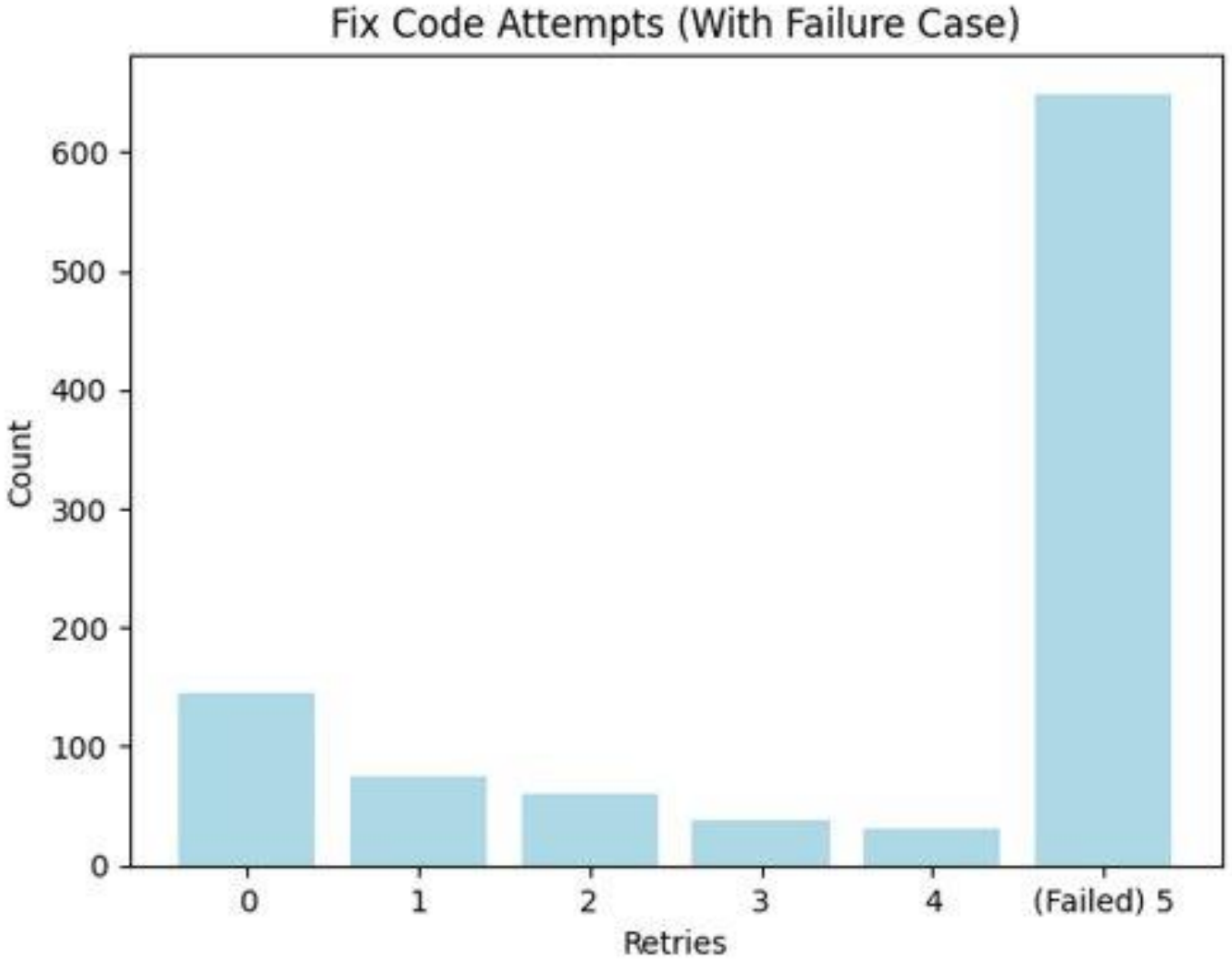
ESBMC-AI Fix Code Mode (FCM) Performance



- **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)**

Tihanyi et al.: The FormAI Dataset: Generative AI in Software Security through the Lens of Formal Verification. PROMISE 2023: 33-43

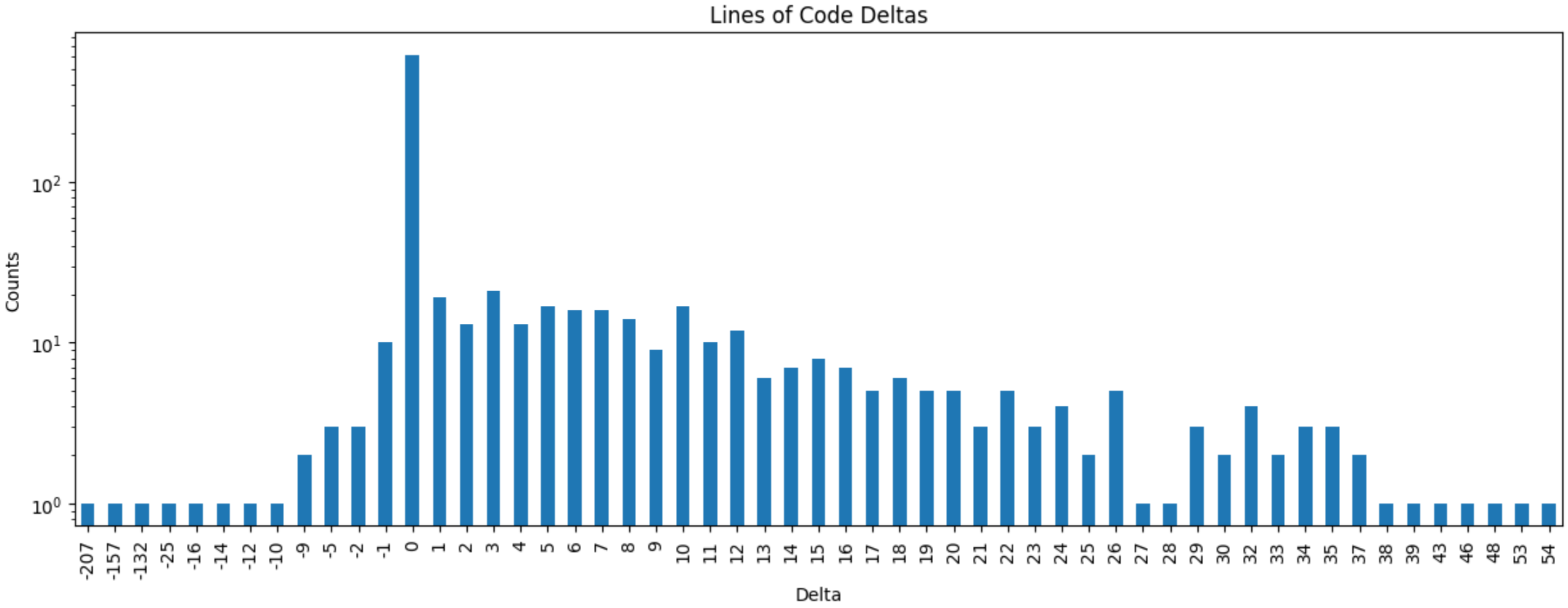
ESBMC-AI Fix Code Mode (FCM) Performance



- **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)**

Tihanyi et al.: The FormAI Dataset: Generative AI in Software Security through the Lens of Formal Verification. PROMISE 2023: 33-43

ESBMC-AI Fix Code Mode (FCM)

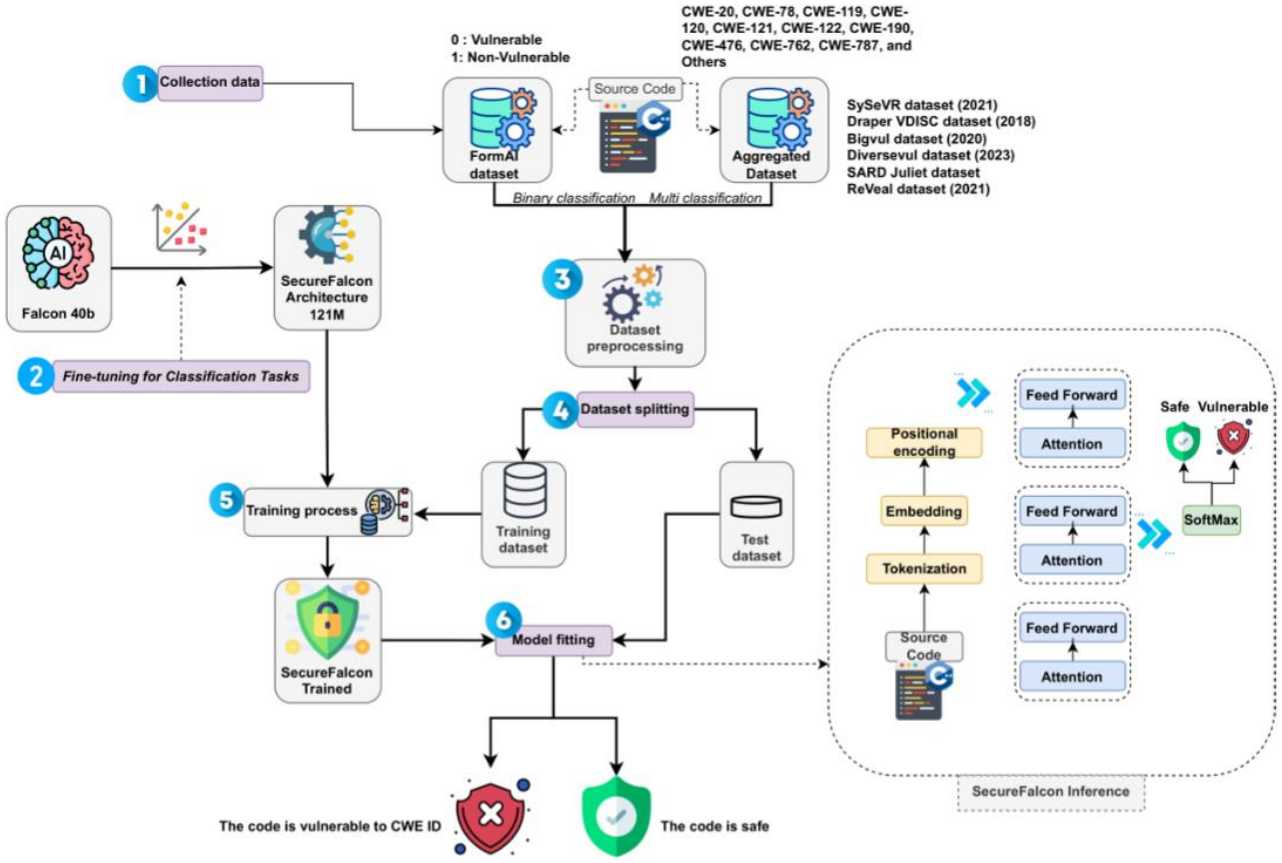


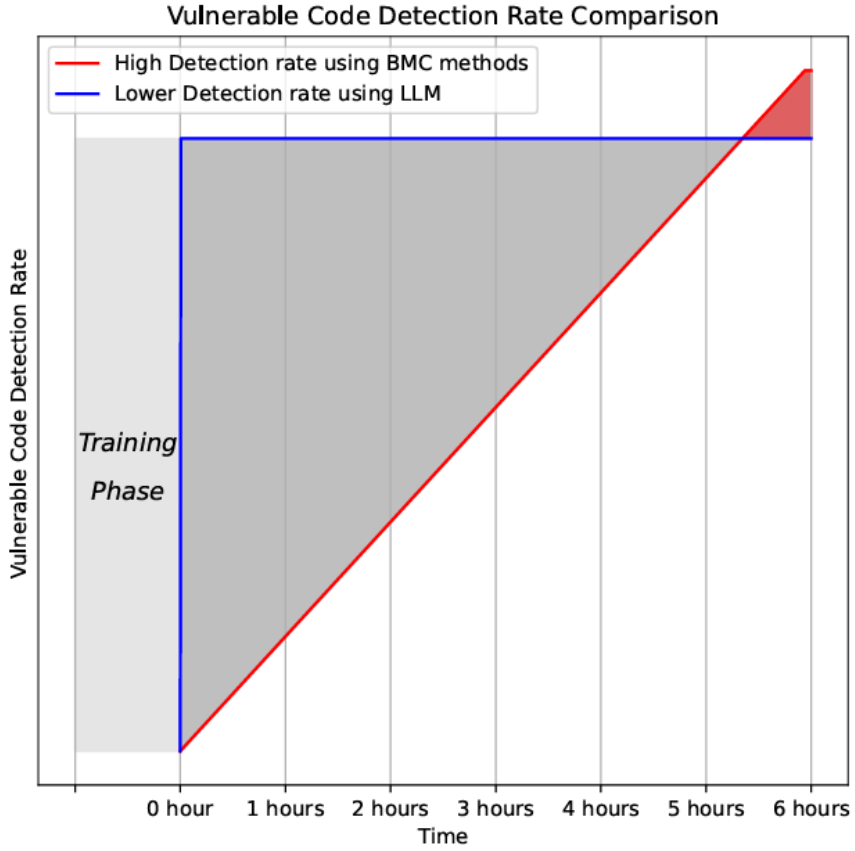
FCM LoC Delta 0 Examples

```
❯ delta FormAI_31585.c
1d0
< //FormAI DATASET v1.0 Category: File Encyptor ; Style: satisfied
27a27
>         fclose(file_ptr); // added line
```

```
❯ delta FormAI_80614.c
1d0
< //FormAI DATASET v1.0 Category: Palindrome Checker ; Style: accurate
3a3
> #include <stdlib.h>
10c10
<     scanf("%s", &str);
---
>     scanf("%99s", str); // Add a limit to the input size to prevent buffer overflow
```

SecureFalcon: Are We There Yet in Automated Software Vulnerability Detection with LLMs?





Train LLMs on known patterns (e.g., CWEs) to catch bugs in real-time as code is being written in an IDE

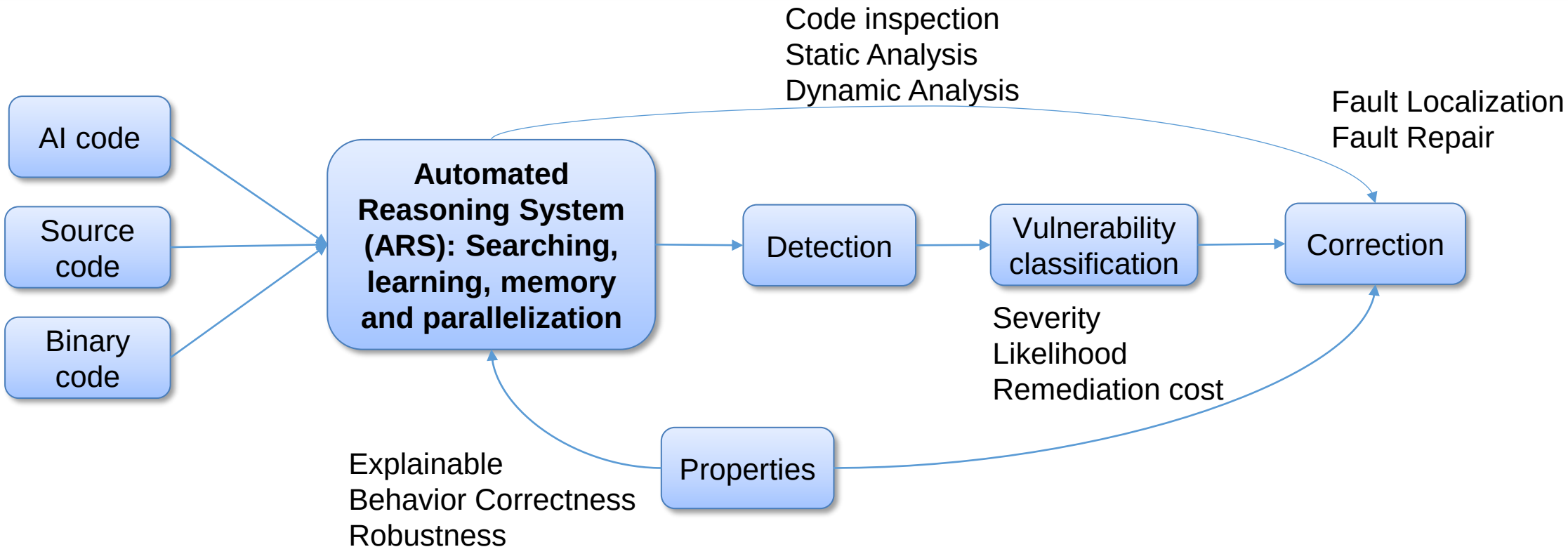
https://arxiv.org/abs/2307.06616

Agenda

- Automated Software Testing and Verification with the ESBMC Framework
- Towards Self-Healing Software via Large Language Models and Formal Verification
- Automated Reasoning System for Building Trustworthy SW and AI Systems

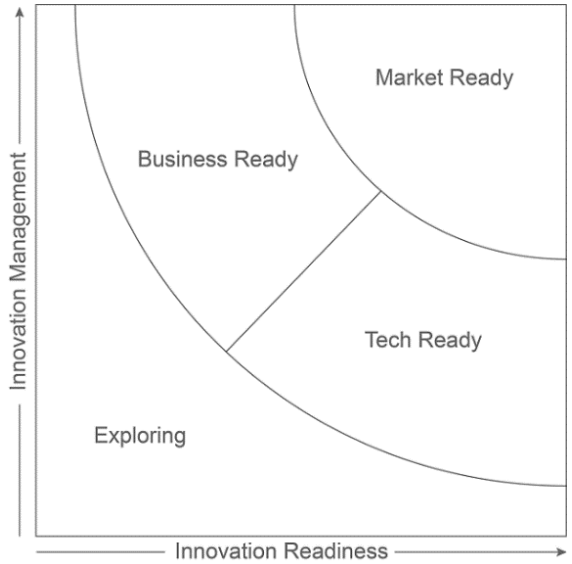
Vision: Automated Reasoning System for Building Trustworthy SW and AI Systems

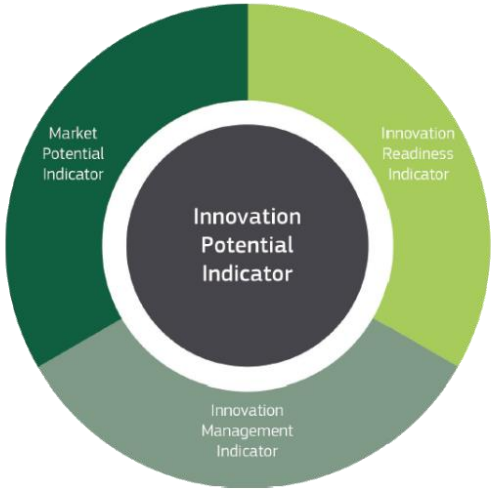
Develop an automated reasoning system for **safeguarding software and AI systems** against vulnerabilities in an increasingly digital and interconnected world



The European Commission recognized 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





(Real) Impact: Students and Contributors

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

- GitHub:
  - 33 contributors
  - 23,419 commits
  - 272 stars
  - 91 forks
  - 4.3k downloads

https://github.com/esbmc/esbmc

Impact: Awards and Industrial Deployment

- **Distinguished Paper Award** at ICSE'11
- **Best Paper Award** at SBESC'15
- **Most Influential Paper Award** at ASE'23
- **Best Tool Paper Award** at SBSeg'23
- **35 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

Acknowledgements