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)

UFAM

MANCHESTER 1824
The University of Manchester

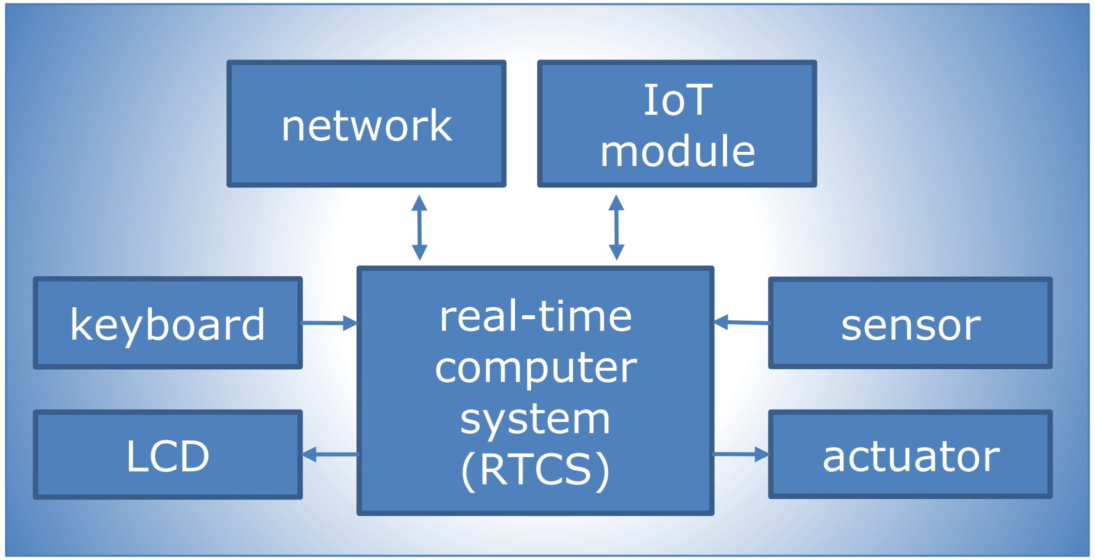
Specification and Verification of Embedded & CPS

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

Verifying Embedded & CPS is Hard

RTCS usually implemented in μC, DSP, and FPGA

AI code (neural nets, LLMs)



fixed- and dynamic, preemptive and non-preemptive scheduling

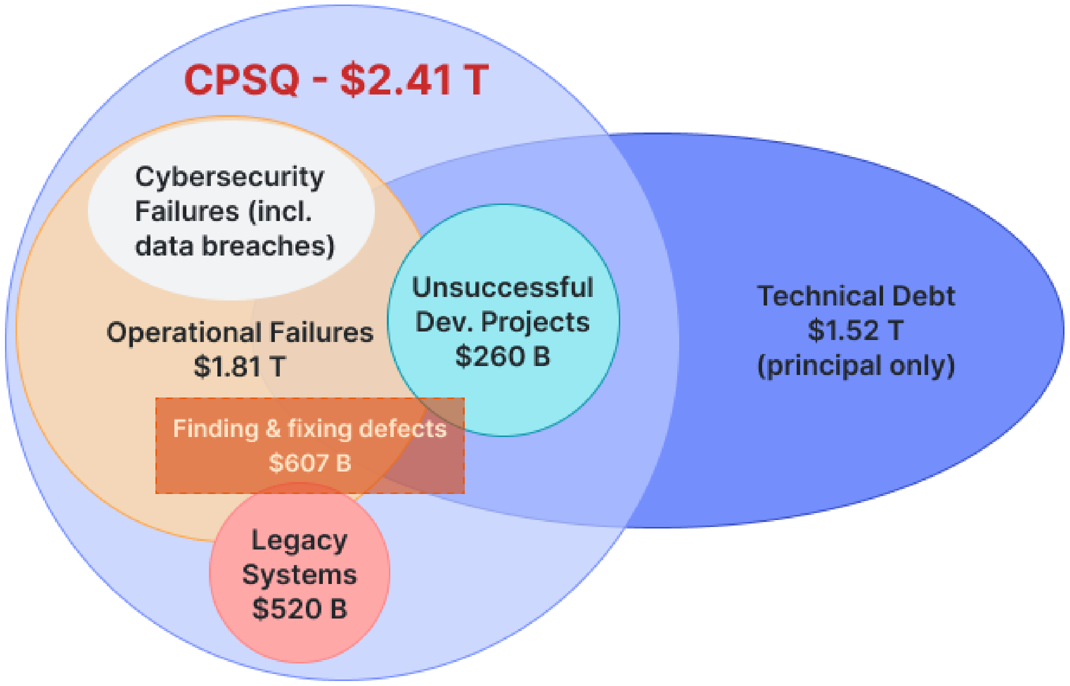
mass production

multi-core processors with limited amount of energy

safety-critical system

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

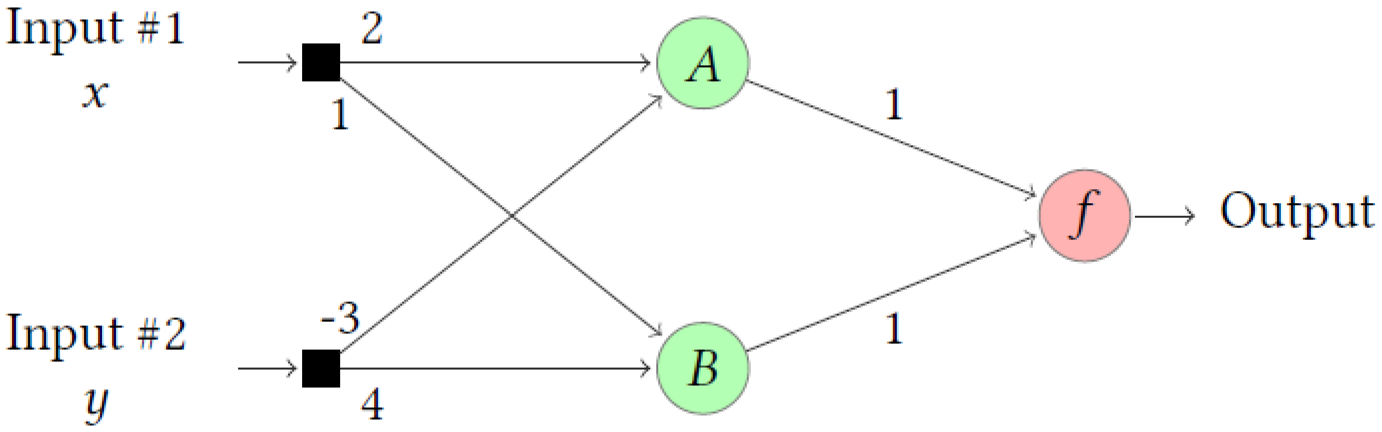
Legend:

$\mathcal{VS}$: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)

Verifying Neural Networks



numerical errors and disagreements
between DNN implementations
and their quantized versions [ICML2019]

$$x, y \in [0.749, 0.498]$$

$$\phi \leftrightarrow f \geq 2.7$$

$$f = A + B = ReLU(2x - 3y) + ReLU(x + 4y)$$

$$\boxed{\begin{aligned} A &= ReLU(2 \times 0.749 - 3 \times 0.498) = ReLU(0.004) = 0.004, \\ B &= ReLU(0.749 + 4 \times 0.498) = ReLU(2.741) = 2.741, \\ f &= A + B = 0.004 + 2.741 = 2.745, \end{aligned}}$$

$$\boxed{f = \mathcal{F}_{\langle 3,6 \rangle}\left(y_{1,2}\right) = 2.6867}$$

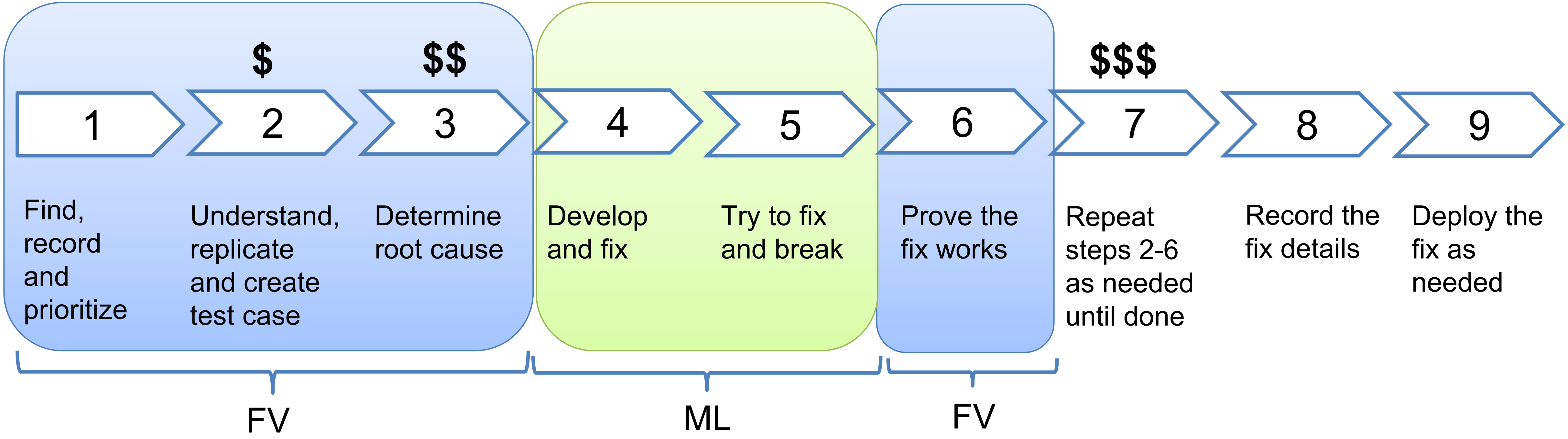
Verifying Neural Networks

```
import numpy as np
x = np.add(2147483647, 1, dtype=np.int32)
```

```
$ python3 main.py

$ esbmc main.py --overflow-check
```

```
[Counterexample]

State 1 file main.py line 3 column 0 thread 0
----------------------------------------------------
Violated property:
  file main.py line 3 column 0
  arithmetic overflow on add
  !overflow("+", 2147483647, 1)

VERIFICATION FAILED
```

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™

Objective of this tutorial

Present **automated testing and verification** to establish a foundation **for building trustworthy embedded & CPS**

- Introduce a **logic-based automated reasoning platform** to find and fix **software defects**
- Explain **testing and verification** techniques to build **trustworthy embedded & CPS**
- Apply an **automated reasoning system** for **safeguarding embedded & CPS** against vulnerabilities

Research Questions

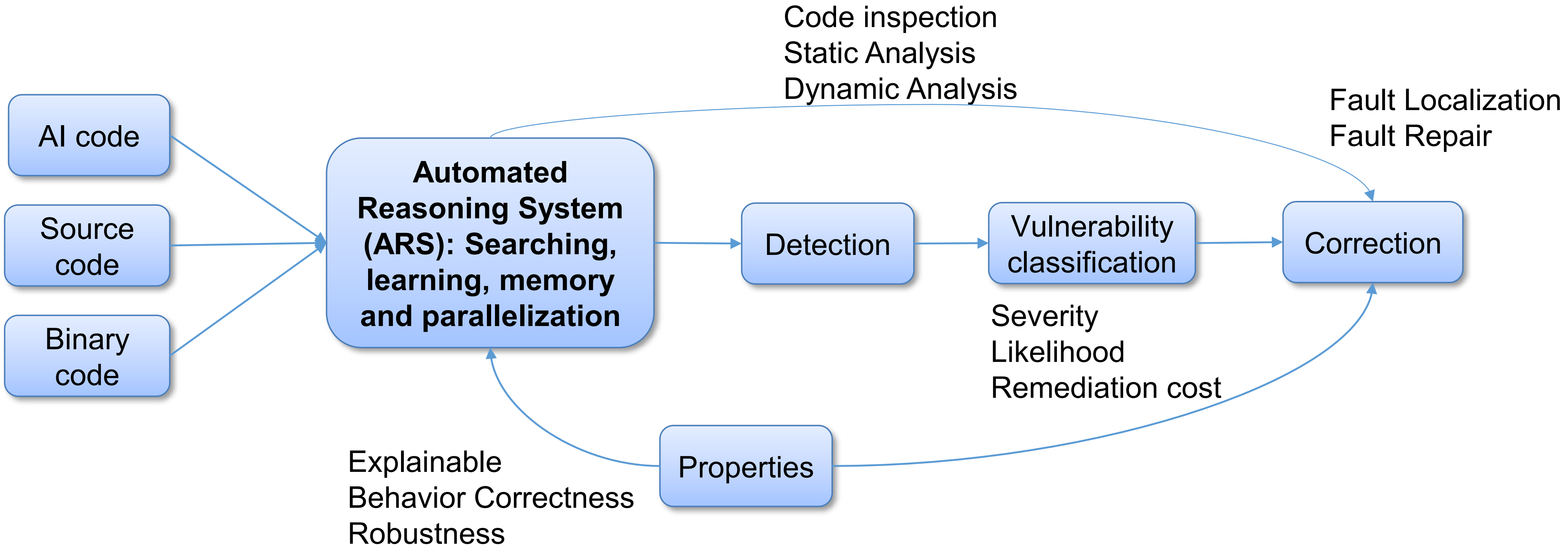
Given a **program** and a **specification**, can we automatically **verify** that the **embedded SW in CPS performs as specified**?

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

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

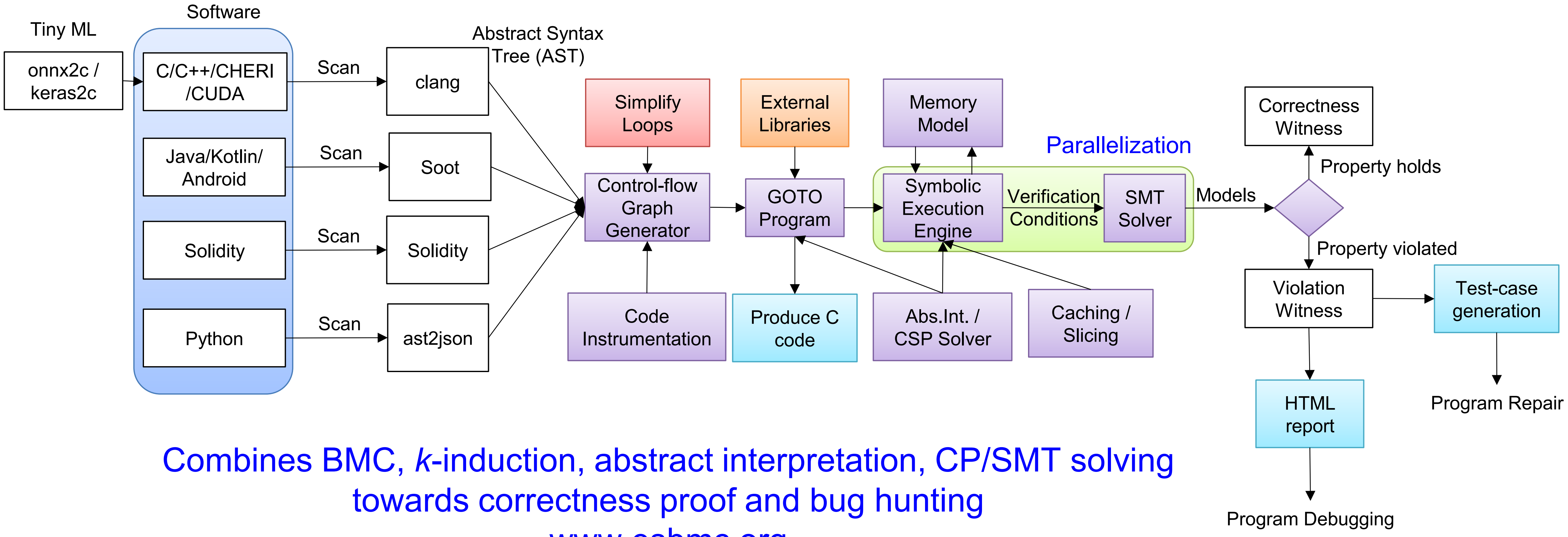
Vision: Building Trustworthy Software and AI Systems

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



ESBMC: A Logic-based Verification Platform

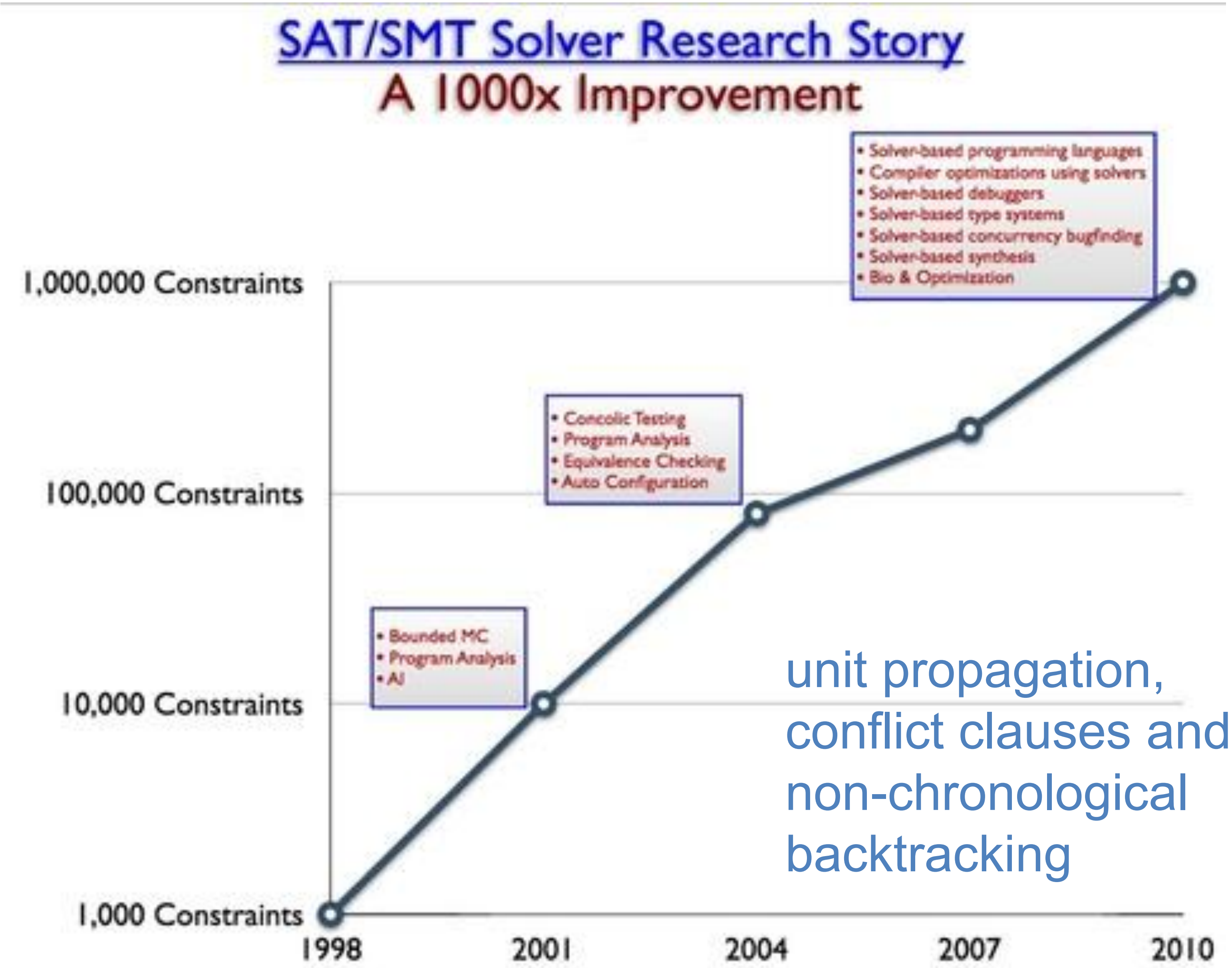
Logic-based automated verification for checking **safety** and **liveness** properties in **AI** and **software systems**



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

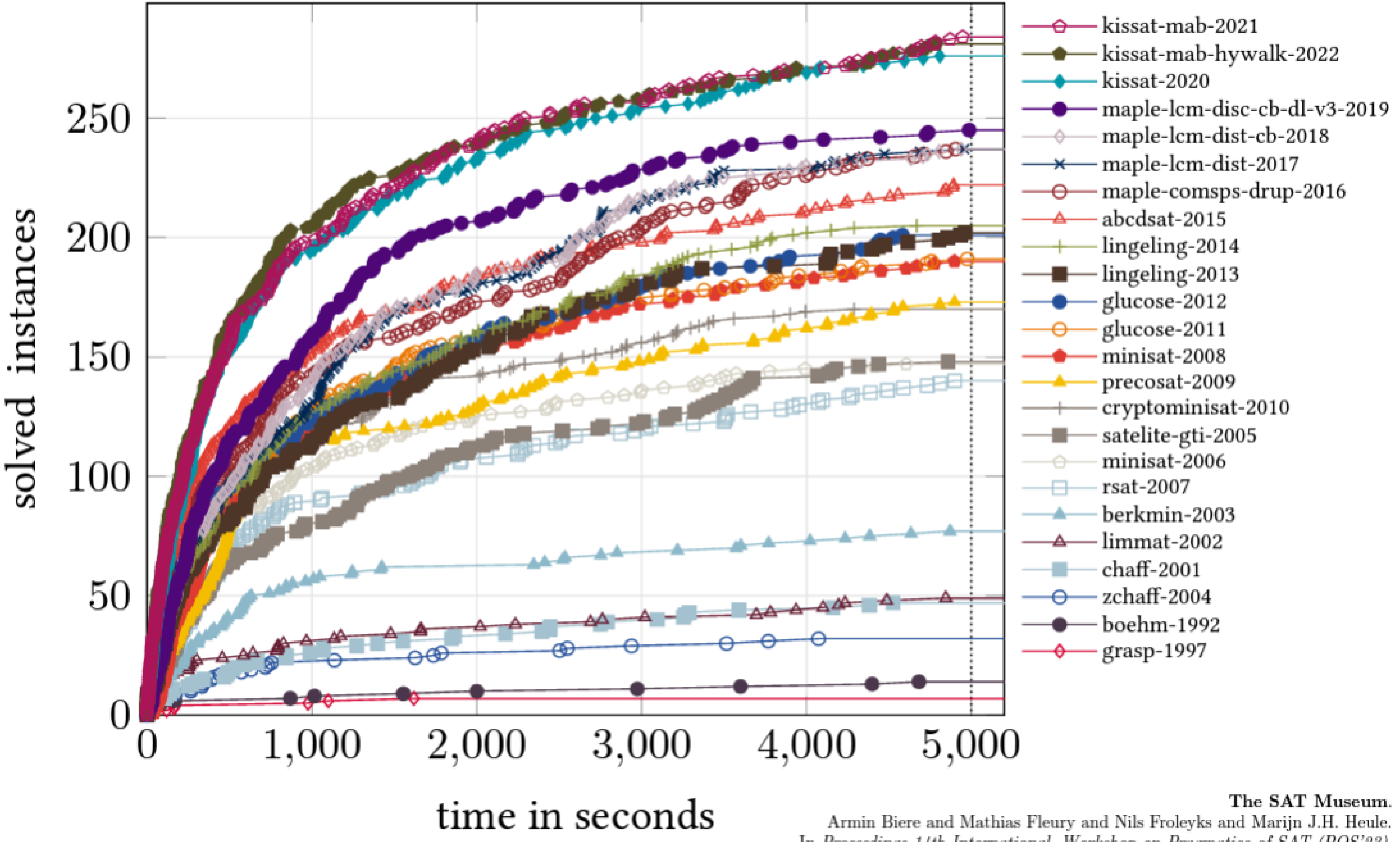
www.esbmc.org

SAT solving as enabling technology



unit propagation, conflict clauses and non-chronological backtracking

SAT Competition All Time Winners on SAT Competition 2022 Benchmarks



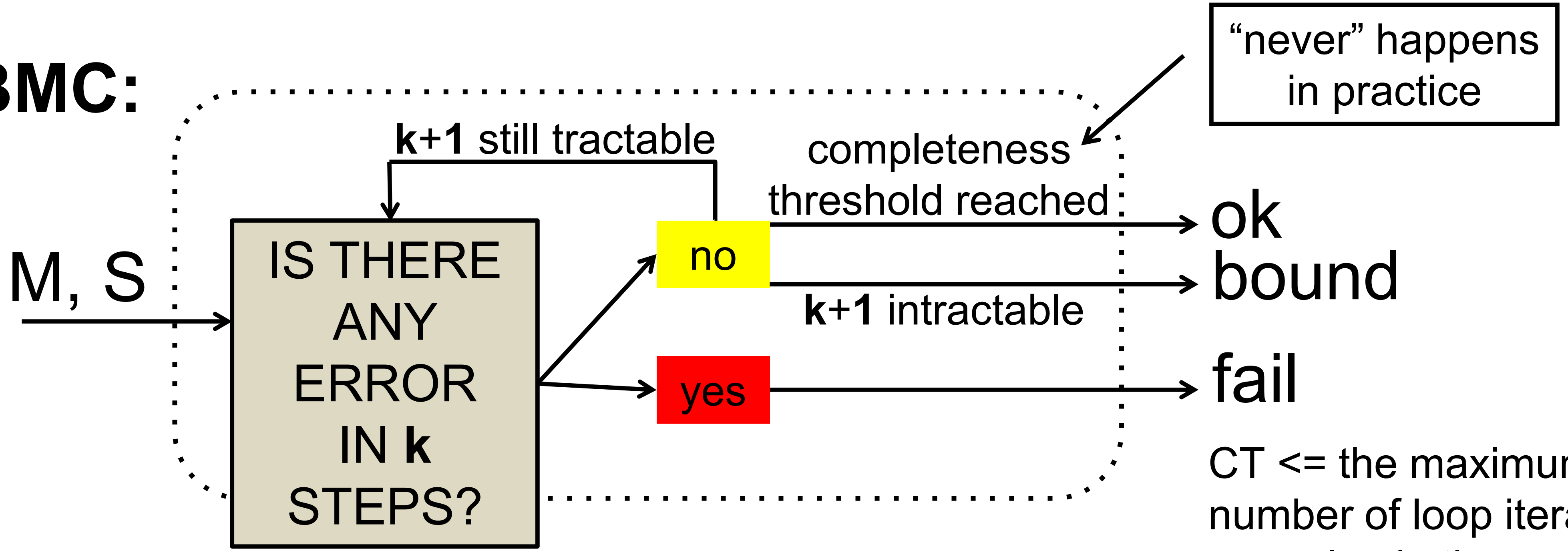
https://cca.informatik.uni-freiburg.de/satmuseum

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)

BMC:



CT <= the maximum number of loop iterations occurring in the program

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

$$\underbrace{I(S_0)}_{\text{Initial state}} \wedge \underbrace{T(S_0,S_1) \wedge \ldots \wedge T(S_{k-1},S_k)}_{\text{k-steps}} \wedge (\neg P(S_0) \vee \ldots \vee \neg P(S_k))$$

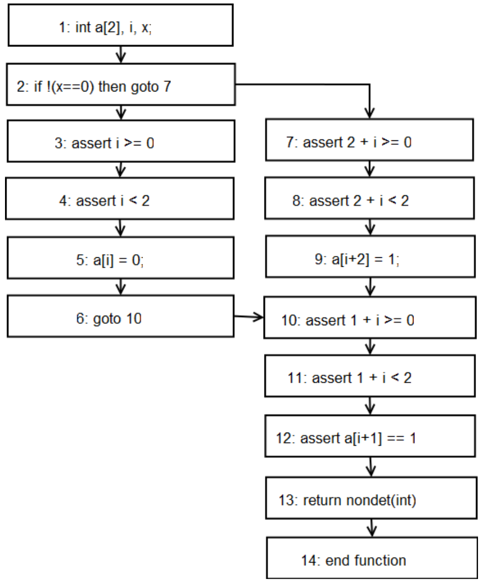
Property fails in some step

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

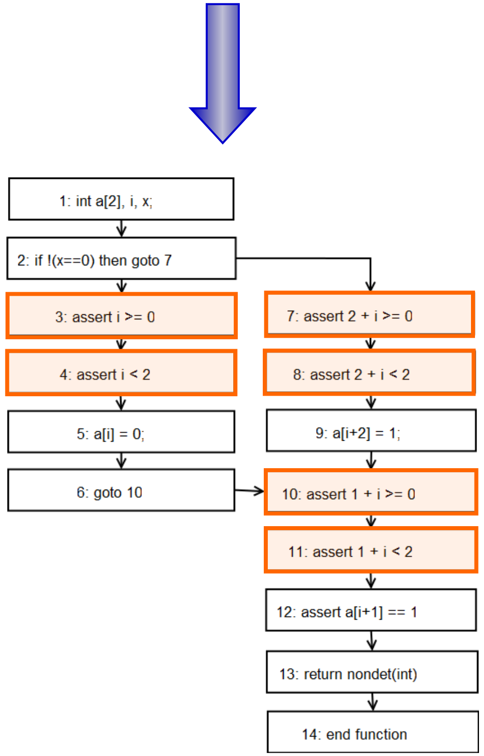
```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

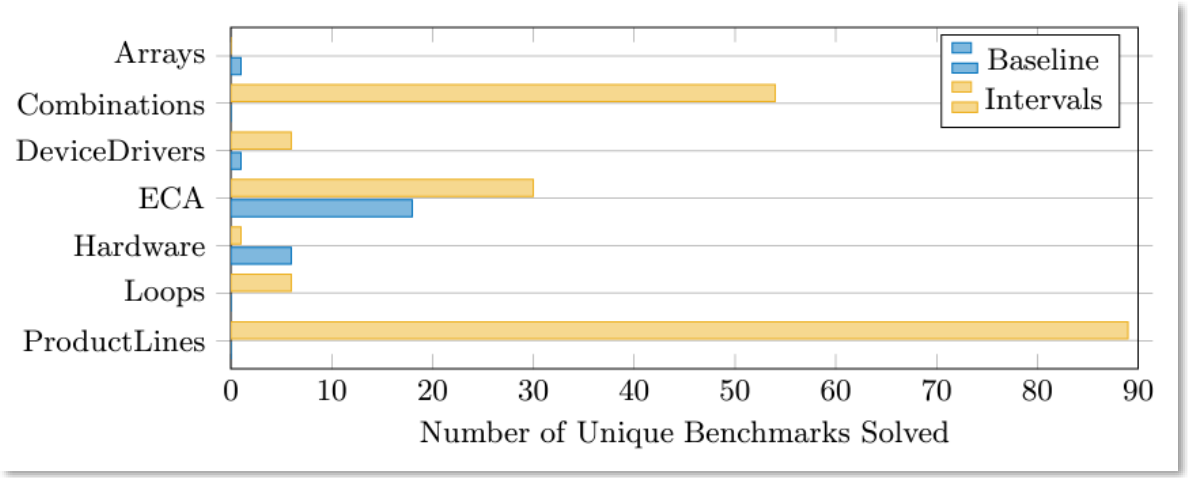


Software BMC

- 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);
}
```



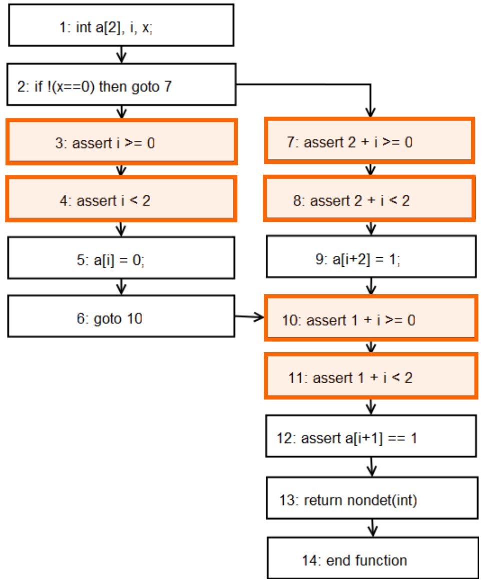


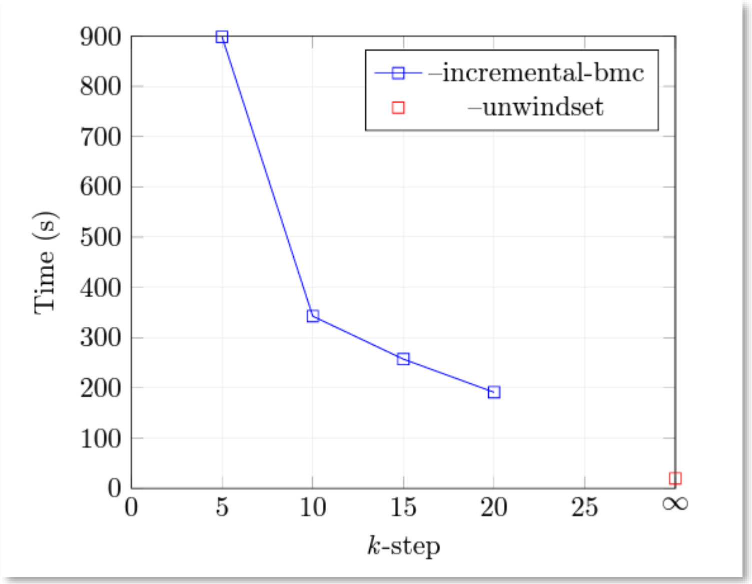
Menezes, R., Manino, E., Shmarov, F., Aldughaim, M., de Freitas, R., Lucas C. Cordeiro: Interval Analysis in Industrial-Scale BMC Software Verifiers: A Case Study.

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

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





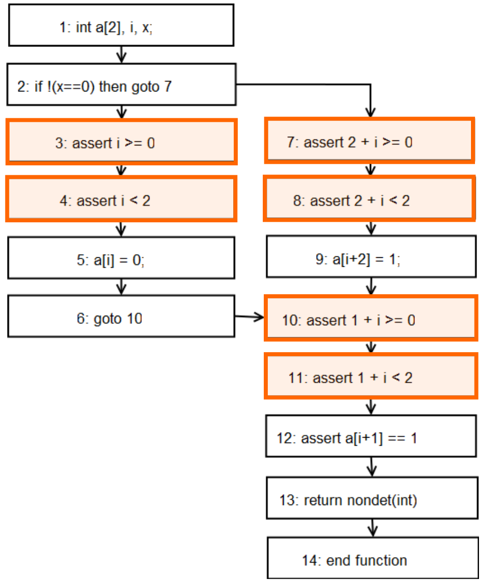
Wu T., Xiong, S., Manino, E., Stockwell, G., Cordeiro, L.: Verifying components of Arm(R) Confidential Computing Architecture with ESBMC. SAS 2024

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);
}
```

$$C := \left[\begin{array}{l} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2 + i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{array}\right]$$

$$P := \left[\begin{array}{l} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\, 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\, select(a_4, i_0 + 1) = 1 \end{array}\right]$$

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

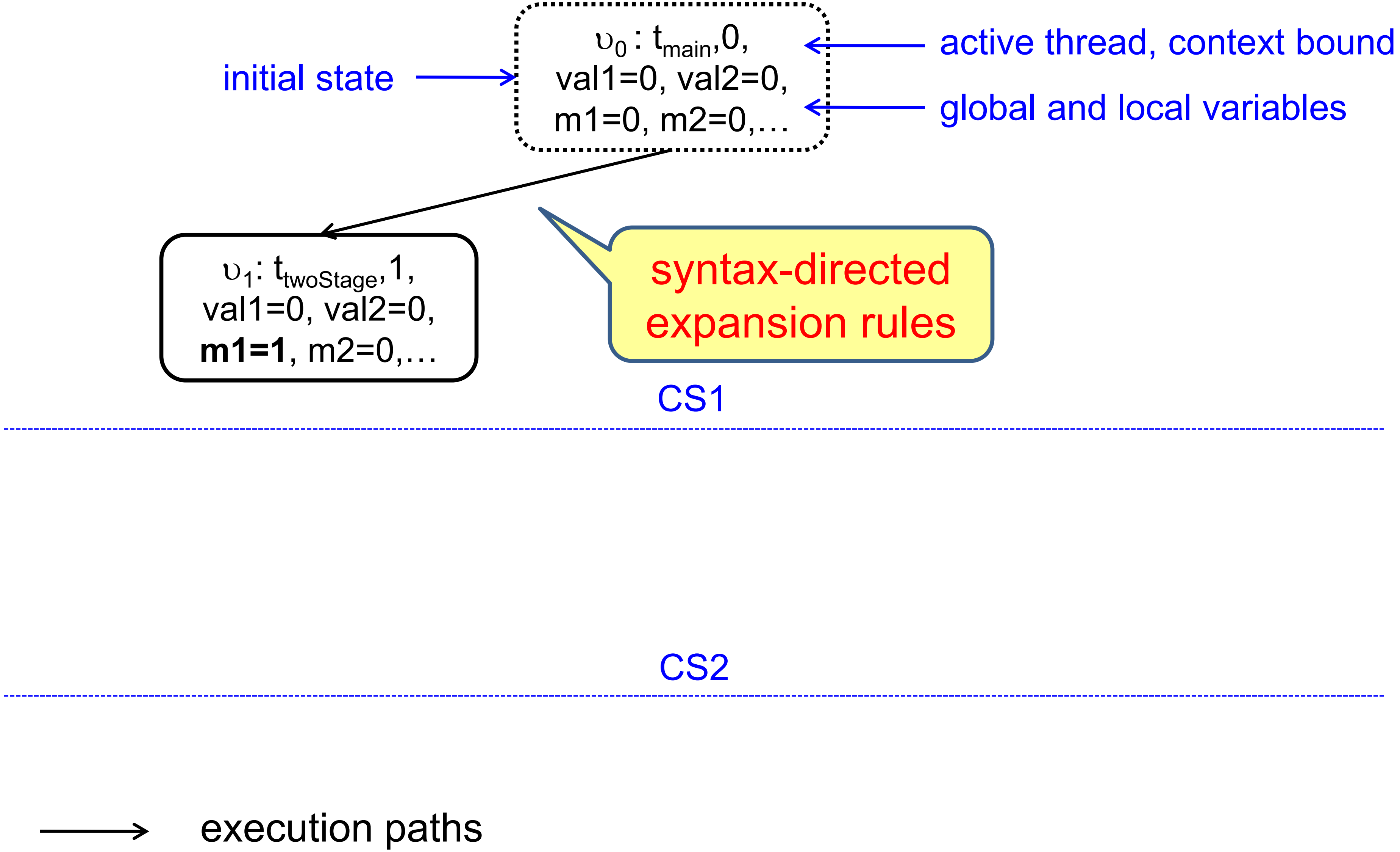
- **symbolic** model checking: on each individual interleaving
- **explicit state** model checking: explore all interleavings
  - bound the number of context switches allowed among threads

… implements

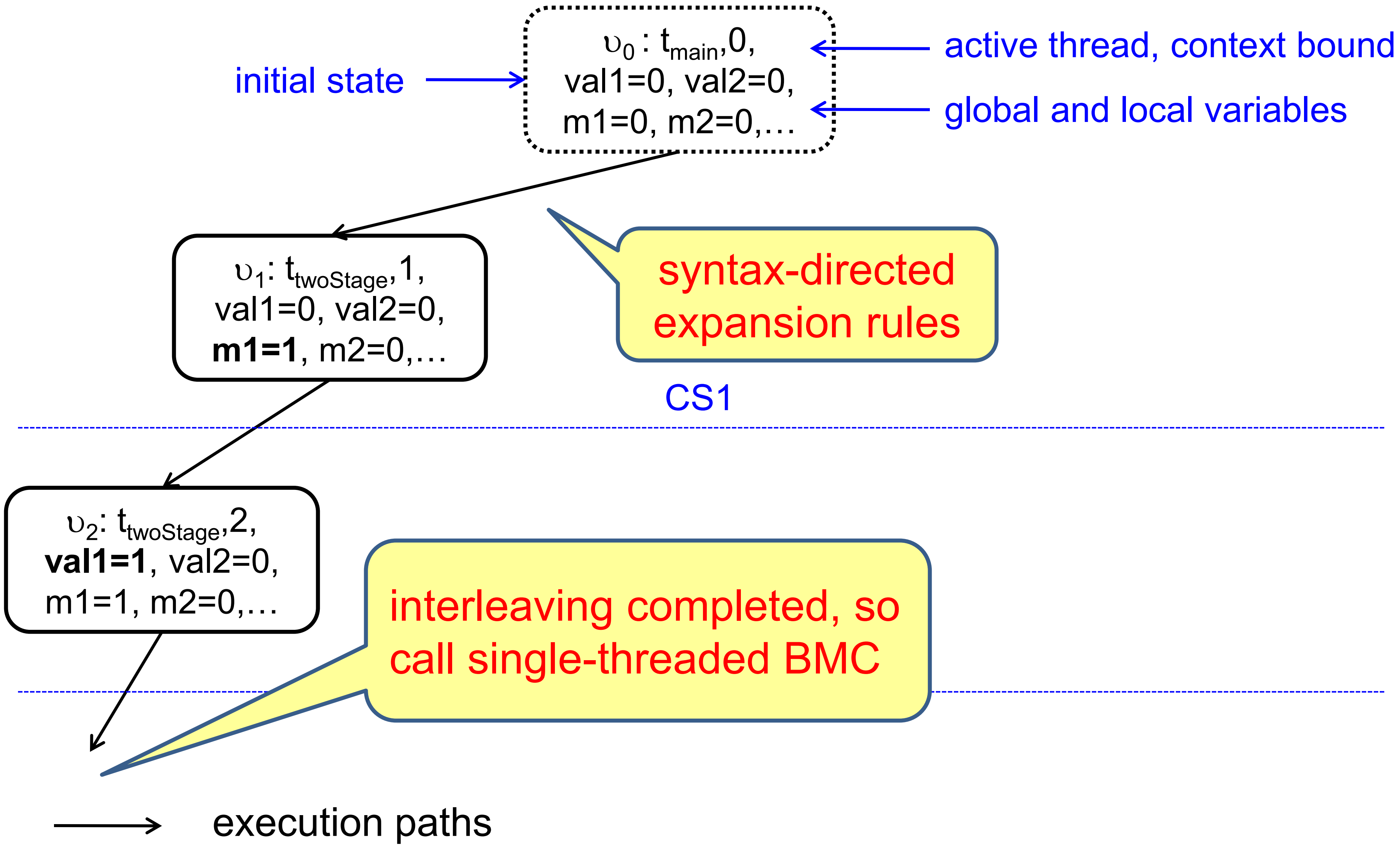
- **symbolic state hashing** (SHA1 hashes)
  - **monotonic partial order** reduction that combines dynamic POR with symbolic state space exploration

Lucas C. Cordeiro, Bernd Fischer: Verifying multi-threaded software using smt-based context-bounded model checking. ICSE 2011: 331-340

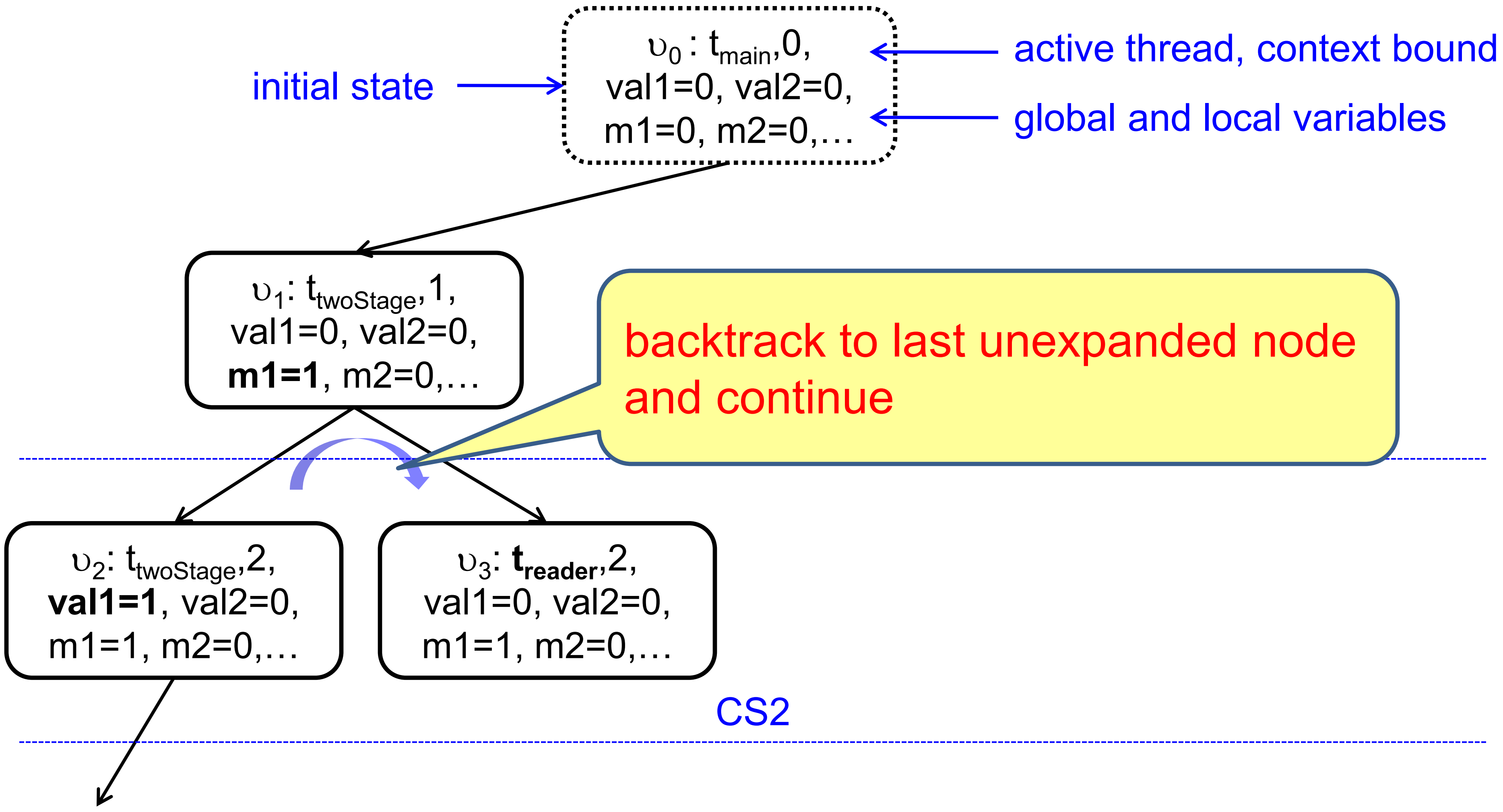
Lazy Exploration of the Reachability Tree



Lazy Exploration of the Reachability Tree



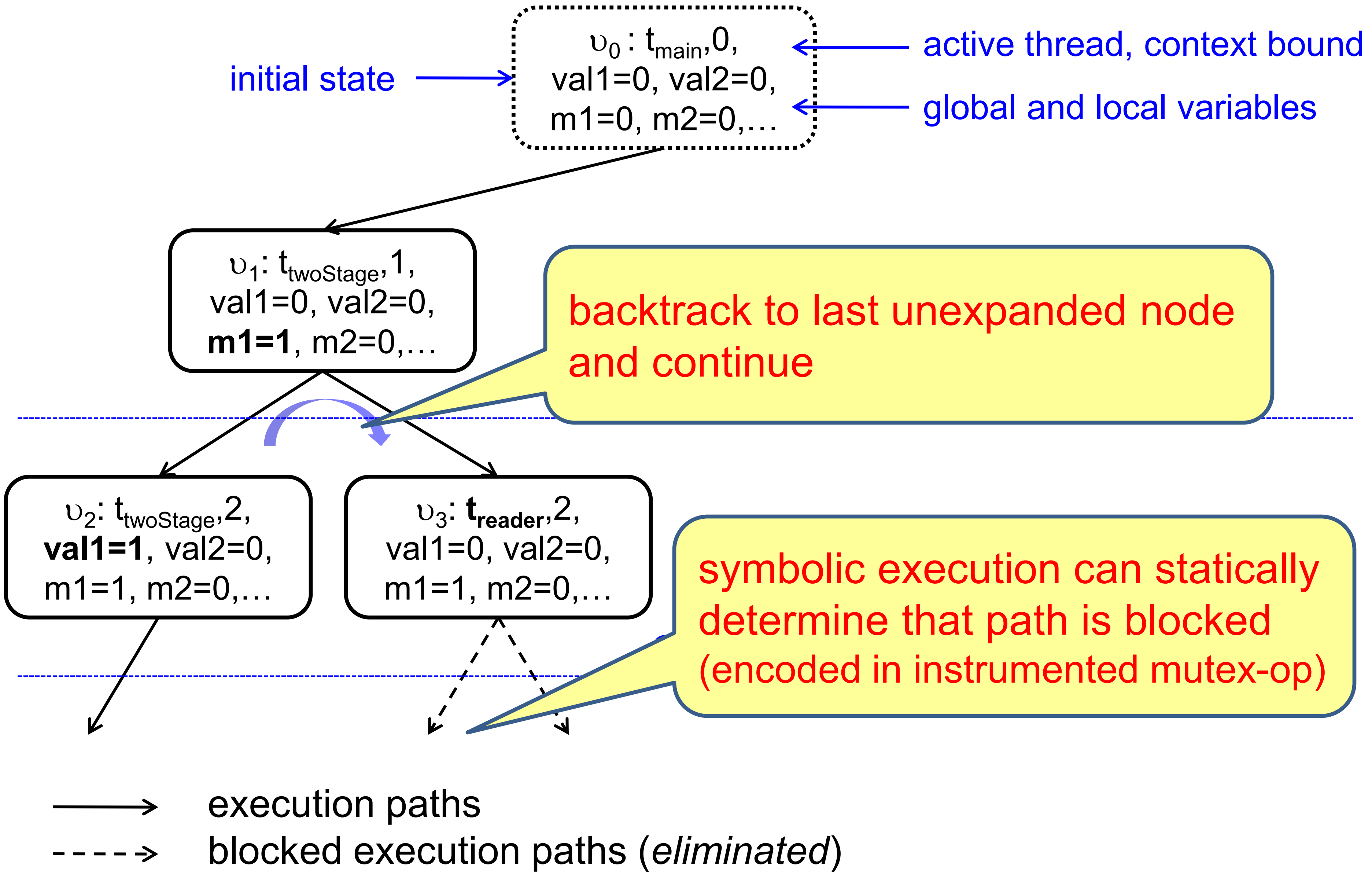
Lazy Exploration of the Reachability Tree



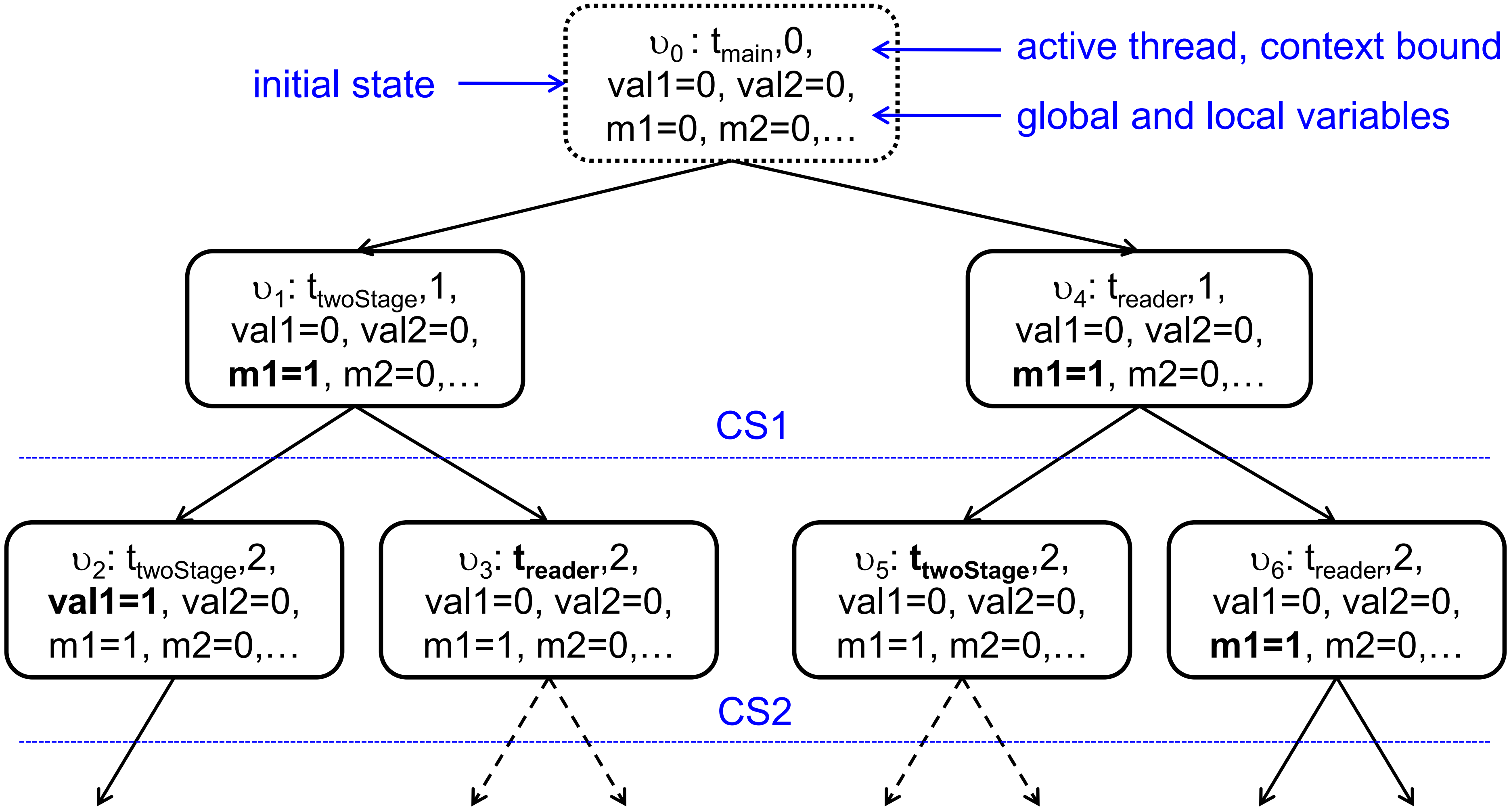
⟶ execution paths

- - - -> blocked execution paths (*eliminated*)

Lazy Exploration of the Reachability Tree



Lazy Exploration of the Reachability Tree



→ execution paths

- - -> blocked execution paths (*eliminated*)

Wu, T., Li, X., Manino, E., Sa Menezes, R., Gadelha, M. R., Xiong, S., Tihanyi, N., Petoumenos, P., Cordeiro, L., ESBMC v7.7: Efficient Concurrent Software Verification with Scheduling, Incremental SMT and Partial Order Reduction. TACAS, 2025. (to appear).

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 ≤ x ≤ 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

```
unsigned int x=nondet_uint();
unsigned int y=nondet_uint();
__ESBMC_assume(x >= 20 && x <= 30);
__ESBMC_assume(y <= 30);
assert(x >= y);
```

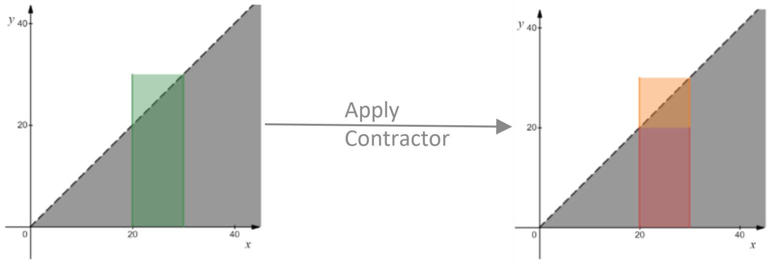
```
__ESBMC_assume(y <= 30 && y >= 20);
```

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

```
unsigned int x=nondet_uint();
unsigned int y=nondet_uint();
__ESBMC_assume(x >= 20 && x <= 30);
__ESBMC_assume(y <= 30);
assert(x >= y);
```

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

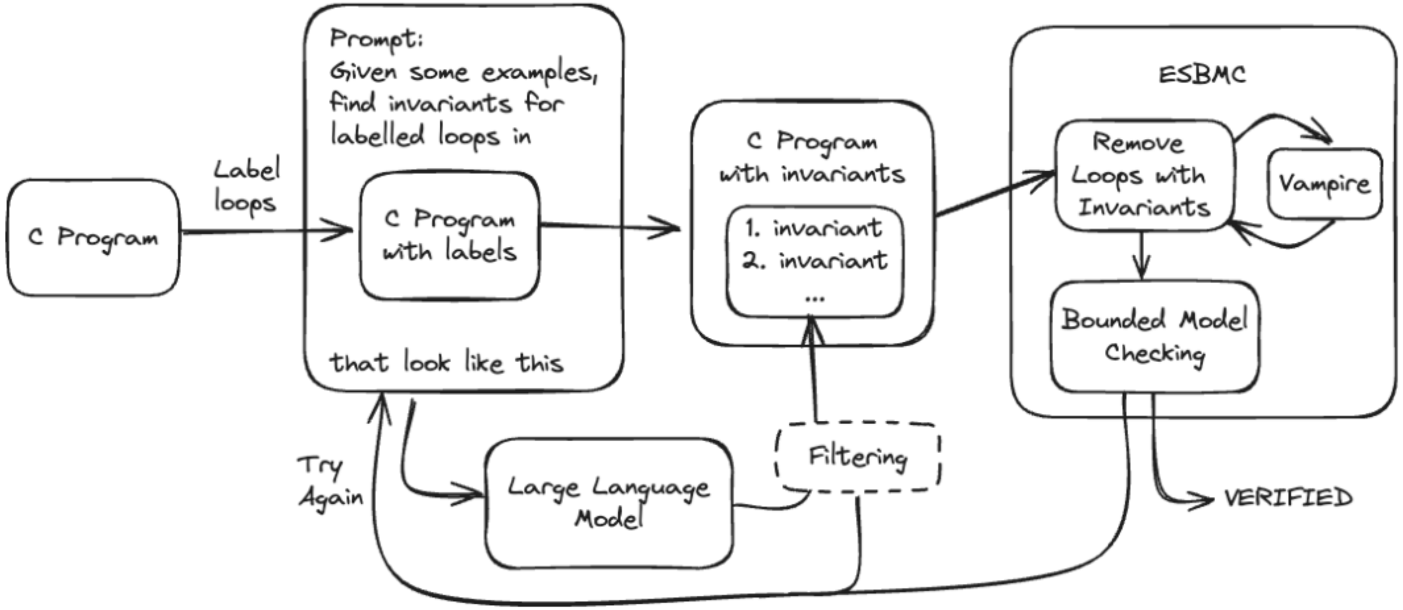
Constraint: $y - x \leq 0$



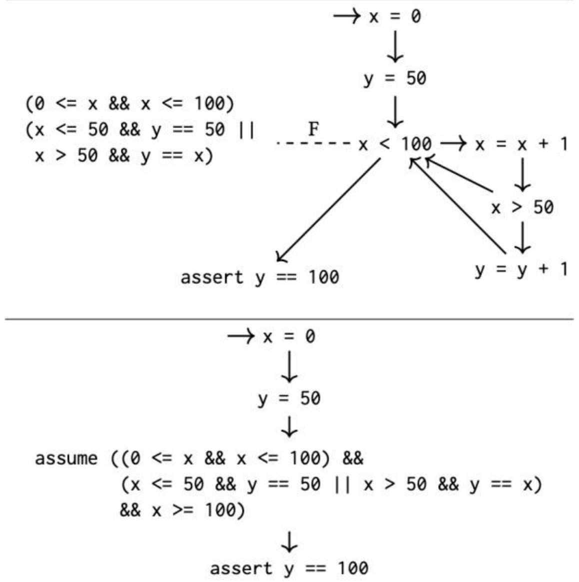
$[x] = [20, 30]$ and $[y] = [0, 30]$ $\quad$ $[x] = [20, 30]$ and $[y] = [20, 30]$

$f(x) > 0$	$I = [0, \infty)$	
$f(x) = y - x$	$[f(x)_1] = I \cap [y_0] - [x_0]$	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

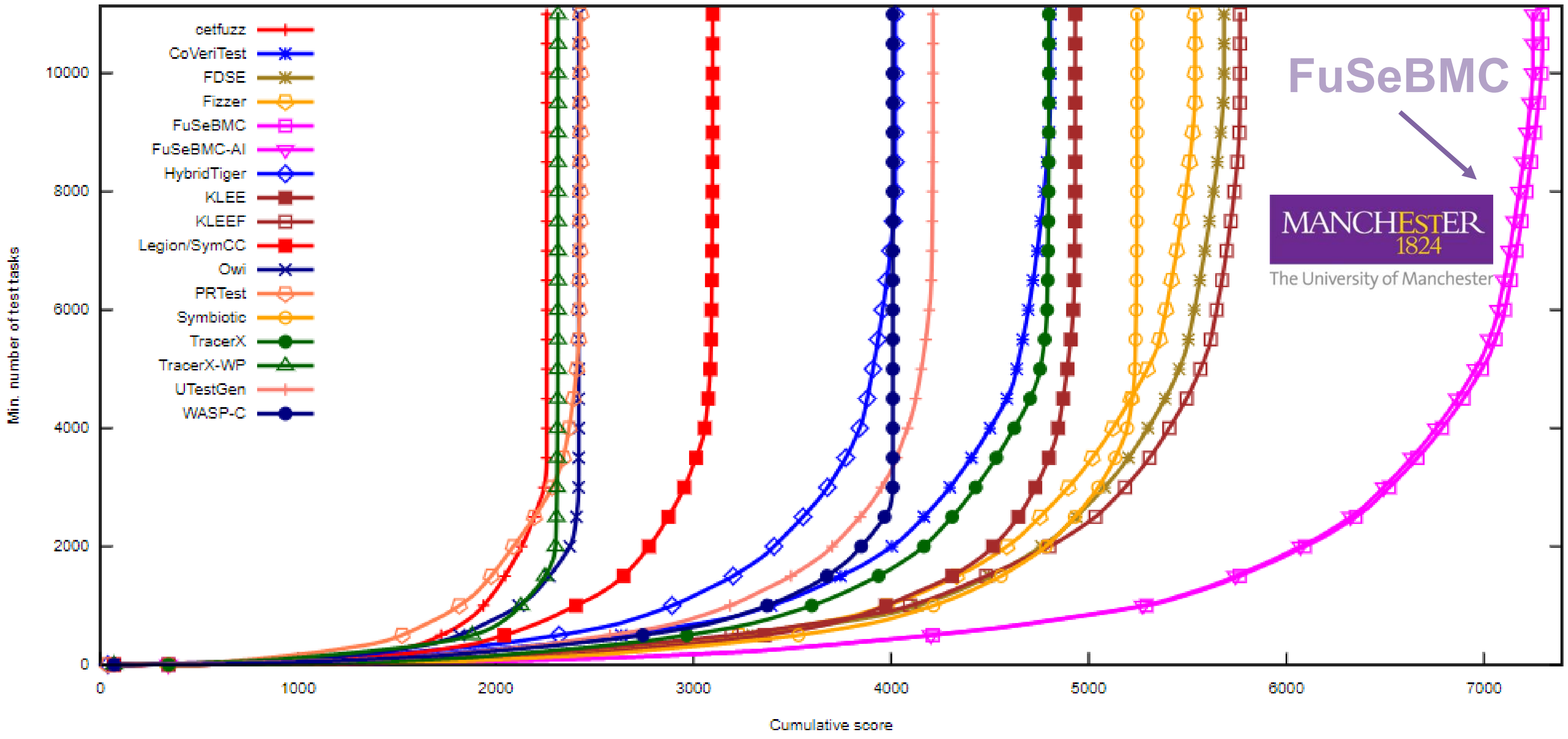
LLM-Generated Invariants for Bounded Model Checking Without Loop Unrolling

Vladimir Filkov
General Chair

Baishakhi Ray
Research PC Co-Chair

Minghui Zhou
Research PC Co-Chair

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



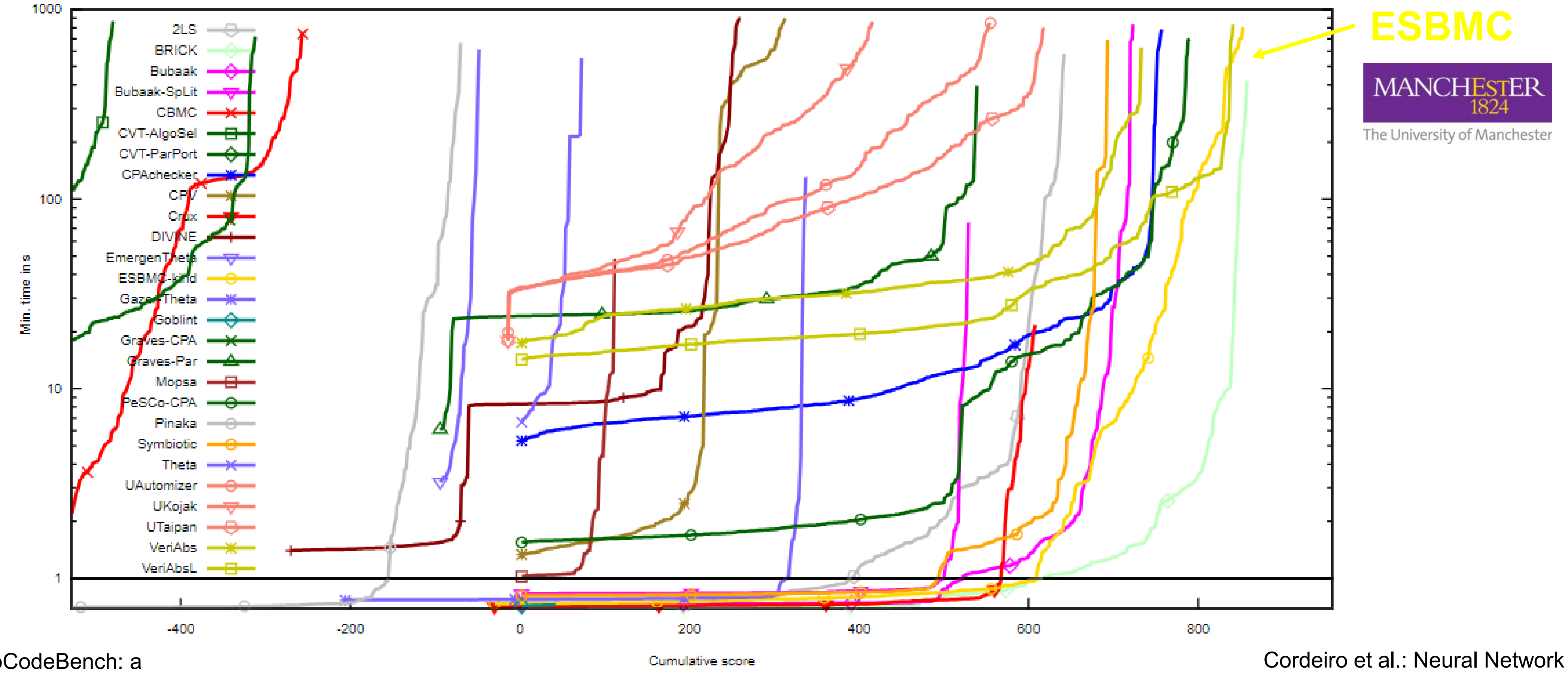
FuSeBMC achieved 3 awards: 1st place in `Cover-Error`, 1st place in `Cover-Branches`, and 1st place in `Overall`

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



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

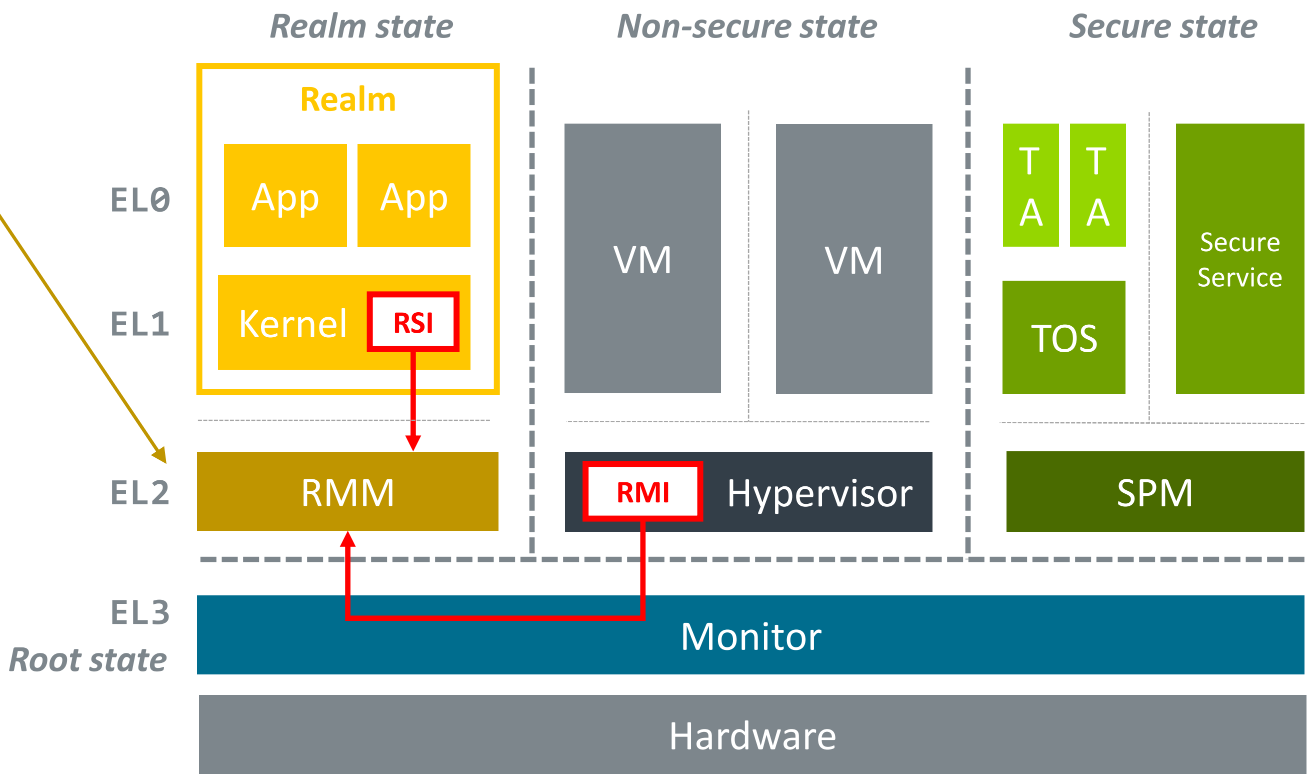
Verification of the `ReachSafety-Floats` Category

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)`
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)`

(—— continued in the right column)

(—— from the left column)

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`

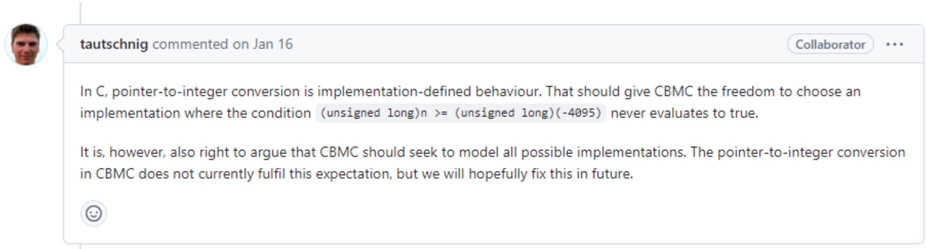
- The document is generated from a **machine-readable specification** (MRS)

[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);
}
```


tautschnig commented on Jan 16 — Collaborator

In C, pointer-to-integer conversion is implementation-defined behaviour. That should give CBMC the freedom to choose an implementation where the condition `(unsigned long)n >= (unsigned long)(-4095)` never evaluates to true.

It is, however, also right to argue that CBMC should seek to model all possible implementations. The pointer-to-integer conversion in CBMC does not currently fulfil this expectation, but we will hopefully fix this in future.

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

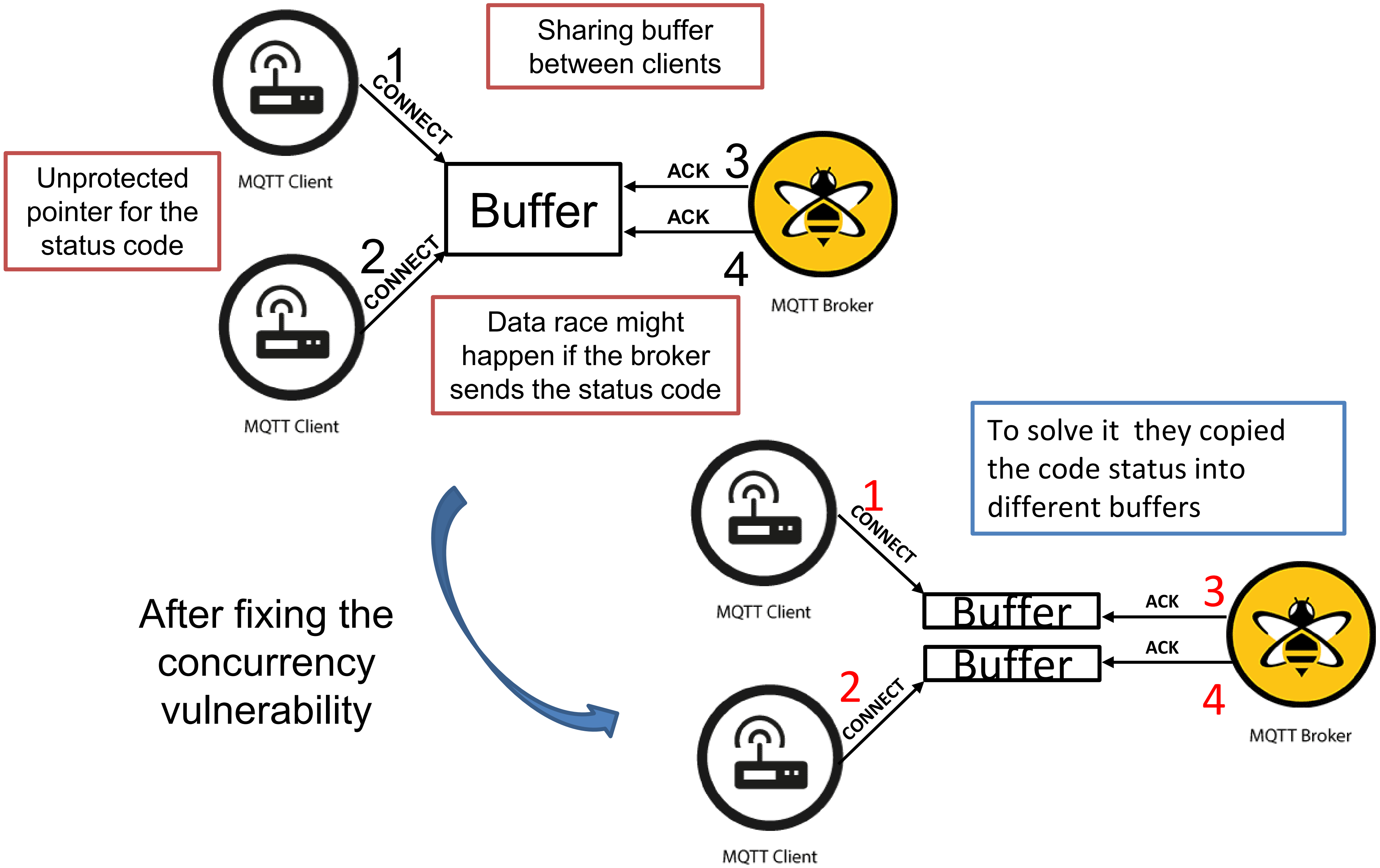
- **wolfMQTT** library is a client implementation of the MQTT protocol written in C for **IoT devices**

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

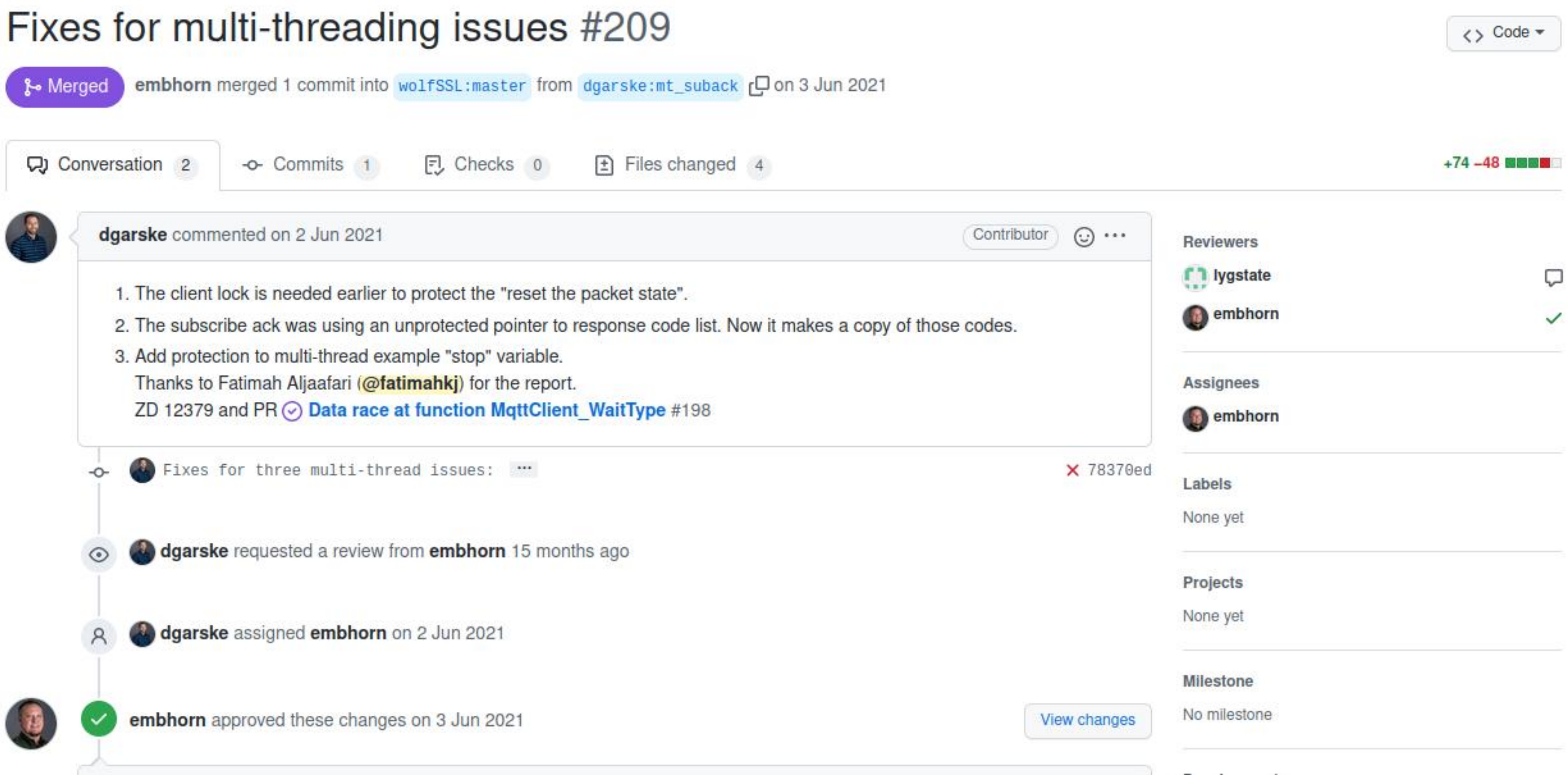
Here is where the data race might happen! Unprotected pointer

```
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

Ethereum Proof-of-Stake Consensus Specifications

chat on discord

To learn more about proof-of-stake and sharding, see the PoS documentation, sharding documentation and the research compendium.

This repository hosts the current Ethereum proof-of-stake specifications. Discussions about design rationale and proposed changes can be brought up and discussed as issues. Solidified, agreed-upon changes to the spec can be made through pull requests.

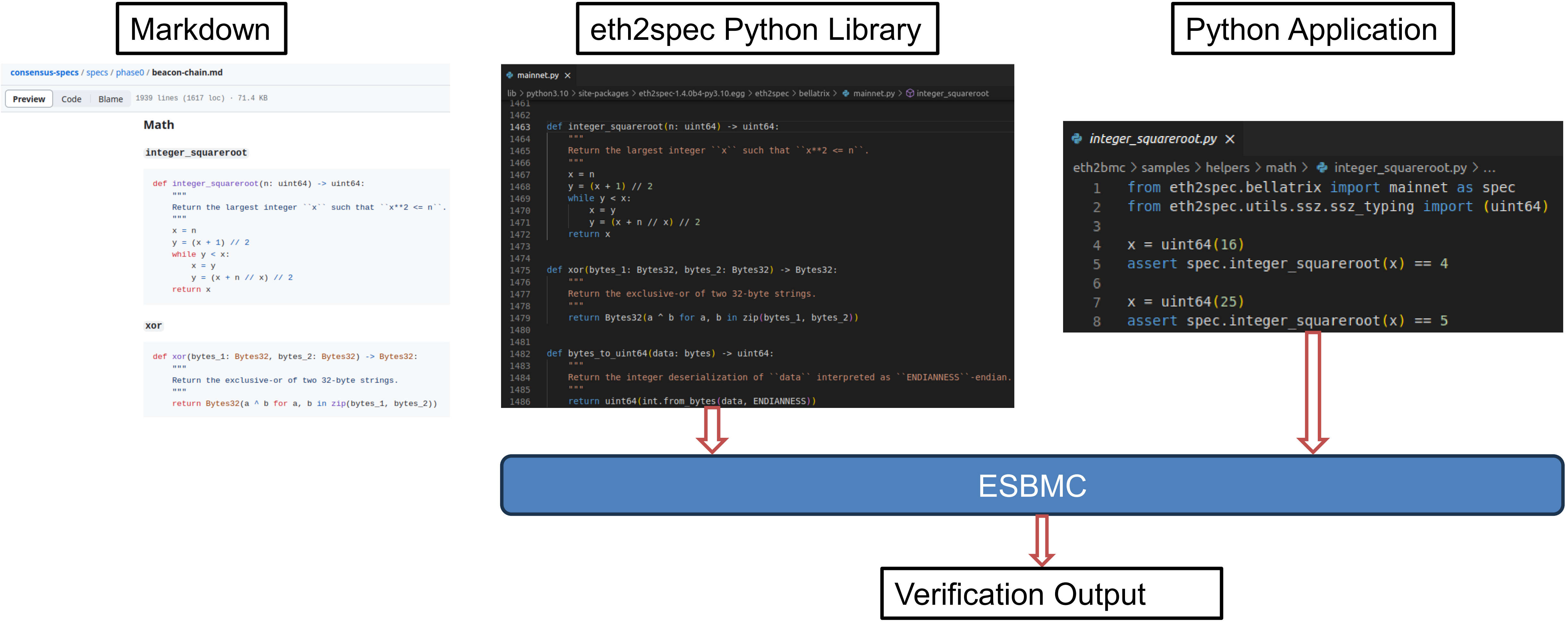
Watch 247 | Fork 862 | Star 3.4k

Contributors 148

+ 134 contributors

ESBMC-Python Benchmark

Ethereum Consensus Specification



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.

https://github.com/ethereum/consensus-specs/pull/3600

Acknowledgements

EPSRC
Engineering and Physical Sciences Research Council

UKRI
UK Research and Innovation

CNPq

CAPES

BRITISH COUNCIL

intel®

ethereum

motorola

FLEXTRONICS®

NOKIA

arm