UNIVERSITY OF Southampton
School of Electronics
and Computer Science

SMT-Based Bounded Model Checking for Embedded ANSI-C Software

Lucas Cordeiro, Bernd Fischer, Joao Marques-Silva
b.fischer@ecs.soton.ac.uk

Software Model Checking – State of The Art in 2009

Software model checking had made **significant progress** but faced challenges of scalability, concurrency, and integration into the software development process

1. **Tools**: Verifiers were available (***BLAST***, ***CBMC***, ***JPF***, ***NuSMV***, and ***Spin***), but had **limitations** regarding **scalability, modeling languages**, **types of properties**.
2. **SAT/SMT solvers**: Few verifiers could support SAT (***CBMC***, ***SLAM***) or even SMT solvers (***SMT-CBMC***). **The viability of using SMT solvers was unclear**.
3. **Concurrency**: Researchers were developing techniques to model and verify multi-threaded programs more effectively (**ongoing challenge**).
4. **Integration with Development Process**: Growing emphasis on integrating model checking into the software development process.
5. **Hybrid Approaches**: Combine model-checking with other techniques, such as testing and static analysis, to improve verification accuracy and efficiency.

Objective of this work

UNIVERSITY OF Southampton
School of Electronics and Computer Science

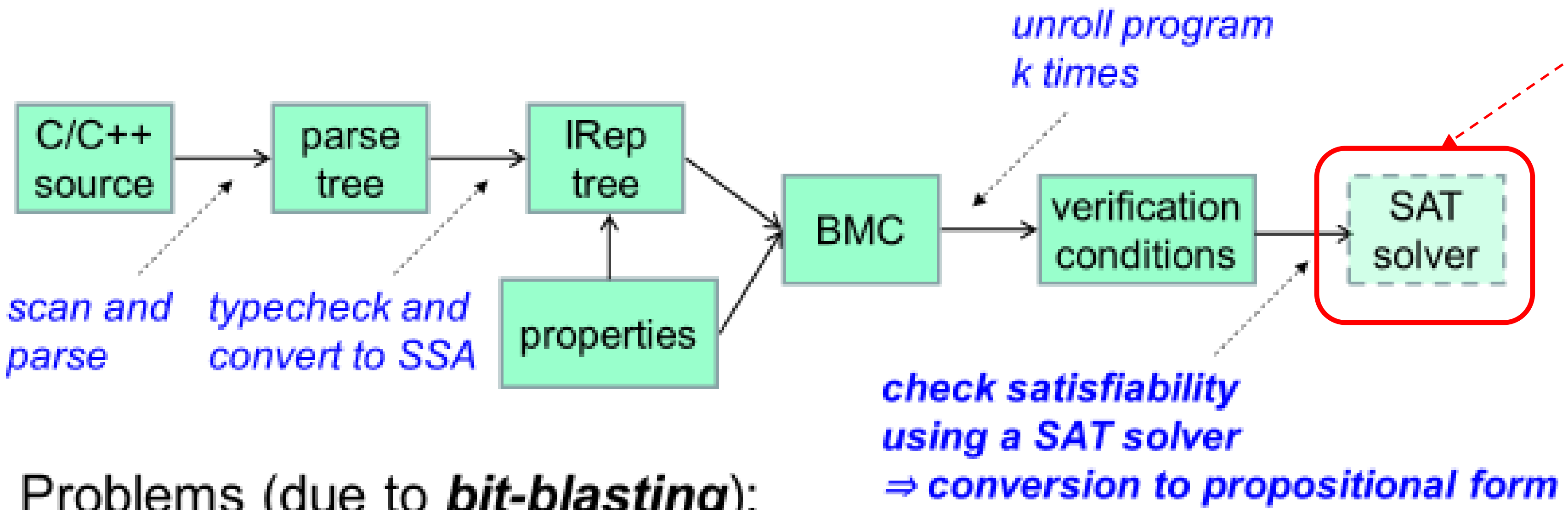
Exploit SMT to improve BMC of embedded software

- exploit background theories of SMT solvers
- provide suitable encodings for
  - pointers
  - bit operations
  - unions
  - arithmetic over- and underflow
- build an SMT-based BMC tool for full ANSI-C
  - build on top of CBMC front-end
  - use several third-party SMT solvers as back-ends
- evaluate ESBMC over embedded software applications

SAT-based CBMC [D. Kroening]

University of Southampton
School of Electronics and Computer Science

implements BMC for ANSI-C/C++ programs using SAT-solvers:



replace by SMT translation and solver

Problems (due to ***bit-blasting***):

- **complex expressions** lead to large propositional formulae
- **high-level information is lost**

Encoding of x == a + b

- *represent x, a, b by n independent propositional variables each*
- *represent addition by logical circuit*
- *represent equality by equivalences on propositional variables*

Encoding of Numeric Types

- SMT solvers typically provide different encodings for numbers:
  - abstract domains ($\mathbb{Z}$, $\mathbb{R}$)
  - fixed-width bit vectors (`unsigned int`, …)
    - ▹ "internalized bit-blasting"
- verification results can depend on encodings

$$(a > 0) \wedge (b > 0) \Rightarrow (a + b > 0)$$

valid in abstract domains such as $\mathbb{Z}$ or $\mathbb{R}$

doesn't hold for bitvectors, due to possible overflows

  - majority of VCs solved faster if numeric types are modelled by abstract domains but possible loss of precision
  - ESBMC supports both encodings

UNIVERSITY OF Southampton School of Electronics and Computer Science

Encoding of Structured Datatypes

- arrays and records / tuples typically handled directly by SMT-solver
- pointers modelled as tuples
  - p.o ≙ representation of underlying object
  - p.i ≙ index (if pointer used as array base)

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

⇒ C

$p_1 := store(p_0, 0, \&a[0])$
$\wedge\ p_2 := store(p_1, 1, 0)$
$\wedge\ g_2 := (x_2 ==$
$\wedge\ a_1 := store(a_0, i_0$
$(a_2, 1+ i_0, 1)$
$\wedge\ a_4 := ite($ $_1, a_1, a_3)$
$\wedge\ p_3 := store(p_2, 1, select(p_2, 1)+2)$

Store object at position 0

Store index at position 1

Update index

UNIVERSITY OF Southampton School of Electronics and Computer Science

Encoding Numeric Types as Bitvectors

Bitvector encodings need to handle

- type casts and implicit conversions
  - arithmetic conversions implemented using word-level functions (part of the bitvector theory: extractBits, …)
    - ▹ different conversions for every pair of types
    - ▹ uses type information provided by front-end
  - conversion to / from bool via if-then-else operator
- arithmetic over- / underflow
  - standard requires modulo-arithmetic for unsigned integers
  - define error literals to detect over- / underflow for other types

$$res\_ok \Leftrightarrow \neg\, overflow(x, y) \wedge \neg\, underflow(x, y)$$

    - ▹ similar to conversions
- floating-point numbers
  - approximated by fixed-point numbers, integral part only
  - represented by fixed-width bitvector

Comparison to SAT-CBMC [D. Kroening]

UNIVERSITY OF Southampton
School of Electronics and Computer Science

			SAT-CBMC				ESBMC			
			Time		#P		Time		#P	
Module	#L	#P	Enc.	**Solver**	Fail	Error	Enc.	**Solver**	Fail	Error
fft1	218	72	0.4	<0.1	0	0	0.4	<0.1	0	0
fft			MO	-	0	39	2337.8	<0.1	0	0
jf			1.2	<0.1	1	0	0.5	2.4	1	0
lr			MO	-	0	35	132.6	0.2	0	0
lu			4.5	TO	0	1	<0.1	1.44	0	0
q			18.8	TO	0	1	1.2	7.7	0	0
p			15.3	0.1	1	0	12.3	5.8	1	0
ad			71.3	3.5	0	0	45.7	9.2	0	0
laplace	110	76	30.8	TO	0	76	12.3	0.3	0	0
exStbKey	558	18	1.2	<0.1	0	0	1.2	<0.1	0	0
exStbHDMI	1045	25	167.9	78.9	0	0	164.4	33.5	0	0
exStbLED	430	6	195.9	130.0	0	0	165.6	44.5	0	0
exStbHwAcc	1432	113	0.7	<0.1	0	0	0.7	<0.1	0	0
exStbRes	353	40	271.8	319.0	0	0	269.3	1161.0	0	0

SMT-solver often significantly faster than SAT-solver

Comparison to SMT-CBMC [A. Armando et al.]

- SMT-based BMC for C, built on top of CVC3 (hard-coded)
  - limited coverage of language
- Goal: compare efficiency of encodings

Module	ESBMC		SMT-CBMC
	Z3	**CVC3**	**CVC3**
BubbleSort (n=35)	[illegible]	**28.7**	**94.5**
[illegible]	[illegible]	MO	*
[illegible]	[illegible]	**8.5**	**66.5**
[illegible]	[illegible]	MO	MO
BellmanFord	0.3	**0.5**	**13.6**
Prim	0.5	**16.9**	**18.4**
StrCmp	38.8	**9.9**	**TO**
SumArray	4.7	**1.2**	**113.8**
MinMax	6.2	MO	MO

ESBMC substantially faster, even with identical solvers
⇒ probably better encoding

Conclusions

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- SMT-based BMC is more efficient than SAT-based BMC
  - but not uniformly
- described and evaluated first SMT-based BMC for ANSI-C
  - provided encodings for typical ANSI-C constructs not directly supported by SMT-solvers
- available at `users.ecs.soton.ac.uk/lcc08r/esbmc/`

Future work:

- better handling of floating-point numbers
- concurrency (based on Pthread library)
- termination analysis

ESBMC – Post 2009: Building a better tool

- SMT-Based Bounded Model Checking for Embedded ANSI-C Software. [TSE 2012, 371 citations]
- ESBMC 5.0: an industrial-strength C model checker. [ASE 2018, 95 citations]
- Verifying Multi-Threaded Software using SMT-Based Context-Bounded Model Checking. [ICSE 2011, 202 citations]
- Context-Bounded Model Checking with ESBMC 1.17. [TACAS 2012, 69 citations]
- Model Checking LTL Properties over ANSI-C Programs with Bounded Traces. [Softw. Syst. Model. 2015, 38 citations]
- Handling Unbounded Loops with ESBMC 1.20. [TACAS 2013, 55 citations]
- ESBMC v6.0: Verifying C Programs Using k-Induction and Invariant Inference - (Competition Contribution). [TACAS 2019, 58 citations]

ESBMC – Post 2009: Building a *software engineering* tool

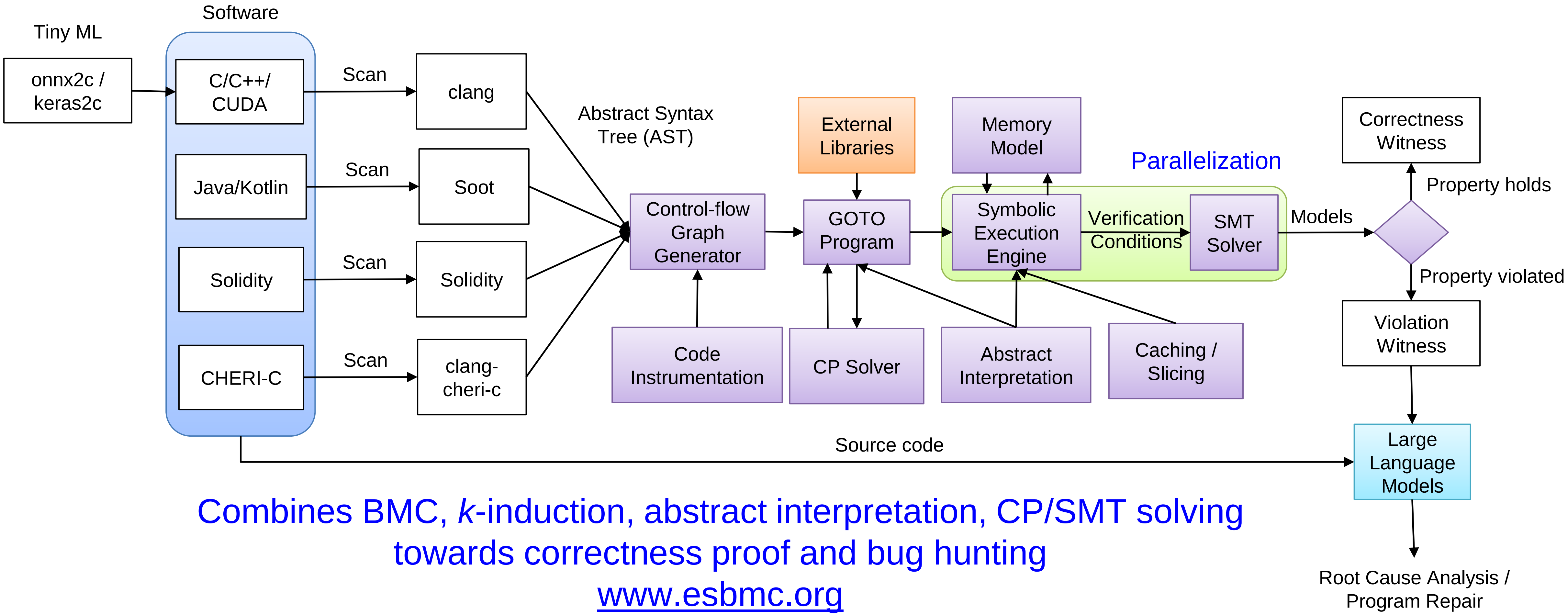
- Continuous Verification of Large Embedded Software Using SMT-Based Bounded Model Checking. [ECBS 2010, 19 citations]
- ESBMC: Scalable and Precise Test Generation based on the Floating-Point Theory. [FASE 2020, 13 citations]
- ESBMC 6.1: Automated Test Case Generation Using Bounded Model Checking. [STTT 2021, 13 citations]
- A Method to Localize Faults in Concurrent C Programs. [JSS 2017, 16 citations]
- A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. [Under Review ACM TOSEM 2023, 5 citations]
- The FormAI Dataset: Generative AI in Software Security Through the Lens of Formal Verification. [PROMISE 2023]

ESBMC – Post 2009: Supporting more languages

- SMT-Based Bounded Model Checking of C++ Programs [ECBS 2013, 49 citations]
- Bounded Model Checking of C++ Programs Based on the Qt Cross-Platform Framework. [STVR 2017, 21 citations]
- Verifying CUDA Programs using SMT-Based Context-Bounded Model Checking. [SAC 2016, 30 citations]
- ESBMC-Jimple: Verifying Kotlin Programs via Jimple Intermediate Representation. [ISSTA 2022]
- ESBMC-Solidity: An SMT-Based Model Checker for Solidity Smart Contracts. [ICSE 2022, 3 citations]
- ESBMC-CHERI: Towards Verification of C Programs for CHERI Platforms with ESBMC. [ISSTA 2022, 2 citations]

ESBMC today: integrated logic-based verification platform

Logic-based automated reasoning for checking the **safety** and **security** of **AI and software systems**



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

www.esbmc.org

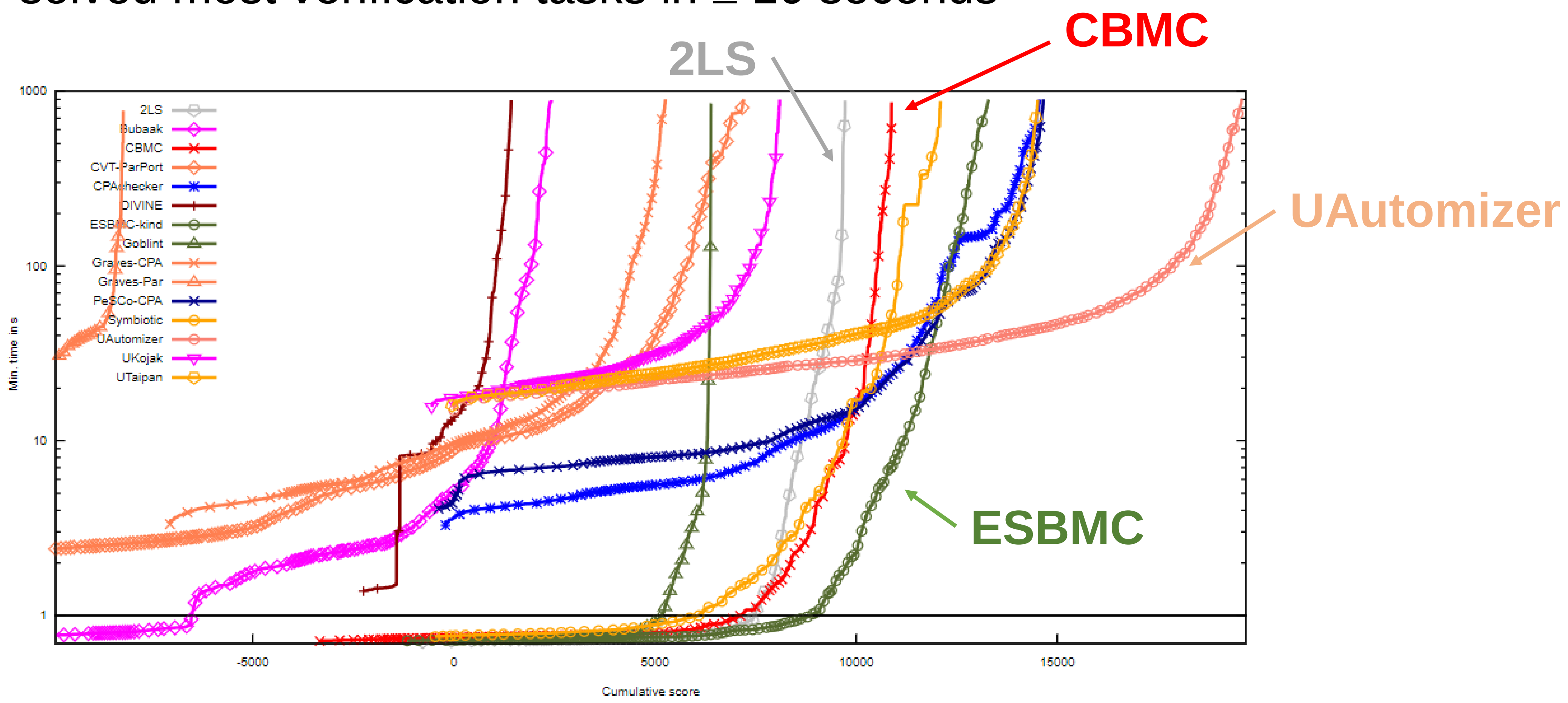
International Competitions

- Intl. Competition on Software Verification (TACAS 2012-2023)
  - 6 x Gold 🥇
  - 4 x Silver 🥈
  - 10 x Bronze 🥉
- Intl. Competition on Software Testing (FASE 2020-2023)
  - 7 x Gold 🥇
  - 1 x Silver 🥈
  - 1 x Bronze 🥉

13 x gold, 5 x silver and 11 bronze (29 medals)

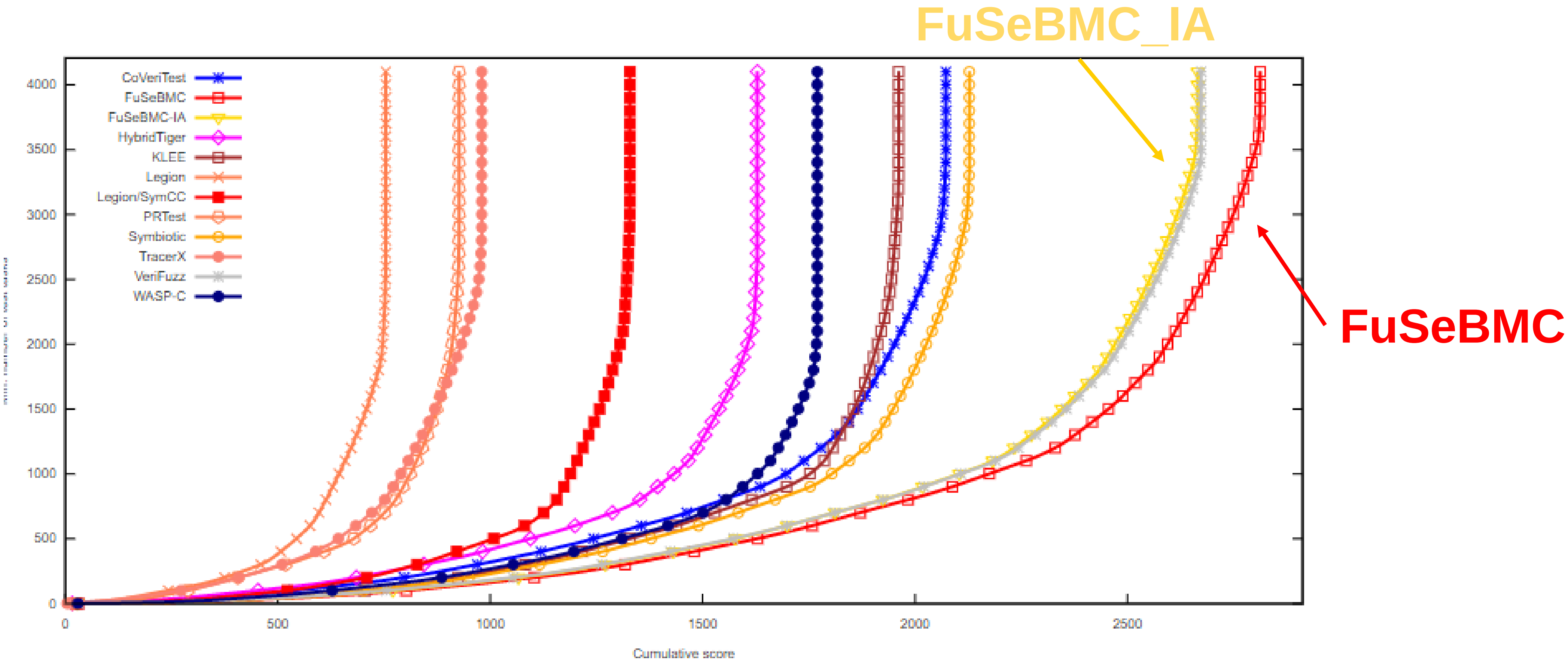
Intl. Software Verification Competitions (SV-Comp 2023)

- SV-COMP 2023, 23805 verification tasks, max. score: 38644
- ESBMC solved most verification tasks in ≤ 10 seconds



Verification of the `Overall` Category

Intl. Software Testing Competitions (Test-Comp 2023)



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/2023/

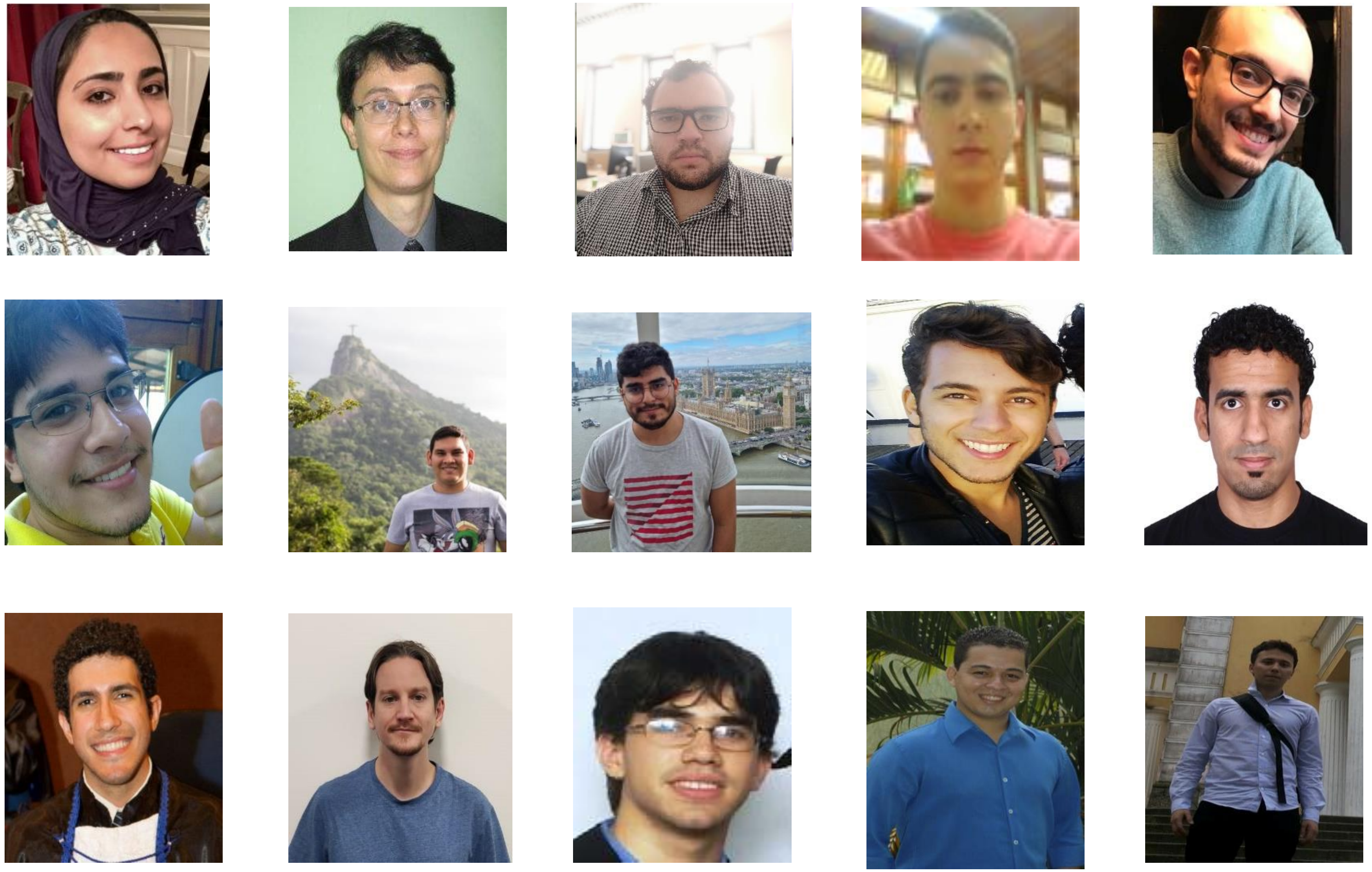
Impact: Awards and Industrial Deployment

- **Distinguished Paper Award** at ICSE’11
- **Best Paper Award** at SBESC’15
- **Most Influential Paper Award** at ASE’23
- **29 awards** from the international competitions on software verification (SV-COMP) and testing (Test-Comp) 2012-2023 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 government** (EPSRC, British Council, Royal Society, CAPES, CNPq, FAPEAM) and **industry** (Intel, Motorola, Samsung, Nokia, ARM)

(Real) Impact: Students and Contributors

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

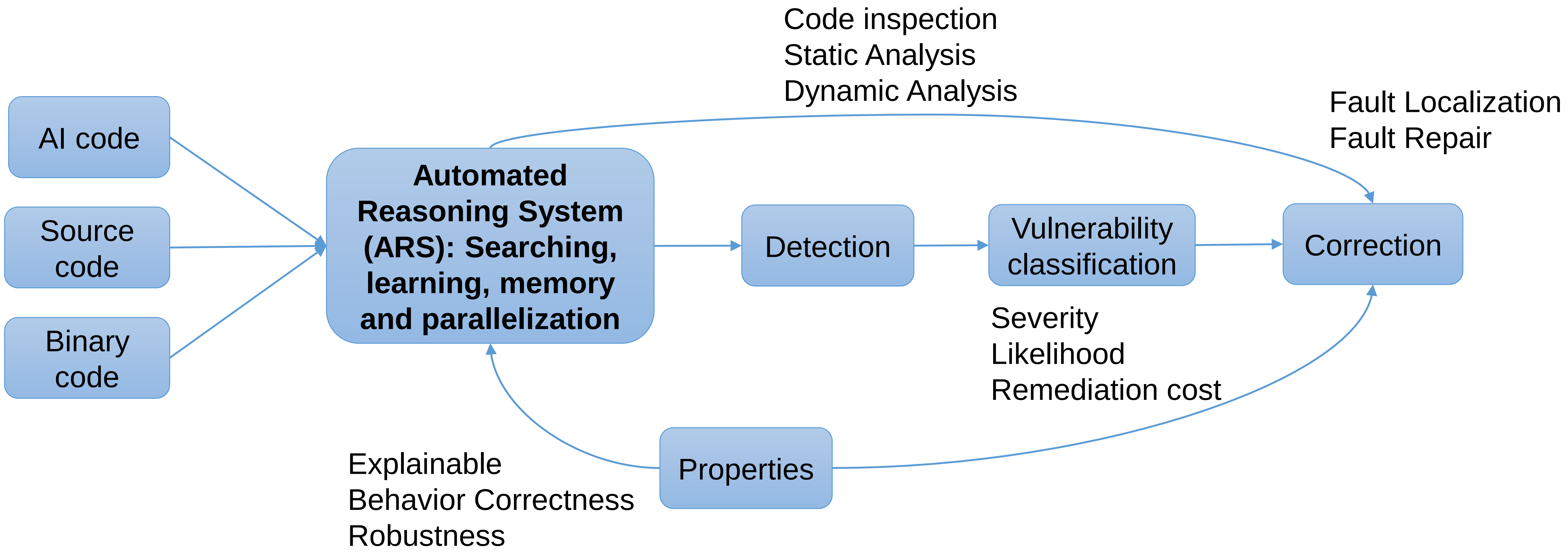
- GitHub:
  - 35 contributors
  - 21,580 commits
  - 195 stars
  - 81 forks



https://github.com/esbmc/esbmc

Vision: Automated Reasoning System for Secure SW and AI

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



Acknowledgements

EPSRC
Engineering and Physical Sciences Research Council

UKRI
UK Research and Innovation

CNPq

CAPES

BRITISH COUNCIL

intel

motorola

NOKIA

arm