Competition on Software Verification and Witness Validation: SV-COMP 2023

Dirk Beyer ✉

LMU Munich, Munich, Germany

Abstract. The 12th edition of the Competition on Software Verification (SV-COMP 2023) is again the largest overview of tools for software verification, evaluating 52 verification systems from 34 teams from 10 countries. Besides providing an overview of the state of the art in automatic software verification, the goal of the competition is to establish standards, provide a platform for exchange to developers of such tools, educate PhD students on reproducibility approaches and benchmarking, and provide computing resources to developers that do not have access to compute clusters. The competition consisted of 23 805 verification tasks for C programs and 586 verification tasks for Java programs. The specifications include reachability, memory safety, overflows, and termination. This year, the competition introduced a new competition track on witness validation, where validators for verification witnesses are evaluated with respect to their quality.

Keywords: Formal Verification · Program Analysis · Competition · Software Verification · Verification Tasks · Benchmark · C Language · Java Language · SV-Benchmarks · BenchExec · CoVeriTeam

1 Introduction

This report extends the series of competition reports (see footnote) by describing the results of the 2023 edition, but also explaining the process and rules, giving insights into some aspects of the competition (this time the focus is on the added validation track). The 12th Competition on Software Verification (SV-COMP, https://sv-comp.sosy-lab.org/2023) is the largest comparative evaluation ever in this area. The objectives of the competitions were discussed earlier (1-4 [16]) and extended over the years (5-6 [17]):

1. provide an overview of the state of the art in software-verification technology and increase visibility of the most recent software verifiers,
2. establish a repository of software-verification tasks that is publicly available for free use as standard benchmark suite for evaluating verification software,

This report extends previous reports on SV-COMP [10, 11, 12, 13, 14, 15, 16, 17, 18, 20].
Reproduction packages are available on Zenodo (see Table 3).

✉ dirk.beyer@sosy-lab.org

S. Sankaranarayanan and N. Sharygina (Eds.): TACAS 2023, LNCS 13994, pp. 495–522, 2023.
https://doi.org/10.1007/978-3-031-30820-8_29

3. establish standards that make it possible to compare different verification tools, including a property language and formats for the results,
4. accelerate the transfer of new verification technology to industrial practice by identifying the strengths of the various verifiers on a diverse set of tasks,
5. educate PhD students and others on performing reproducible benchmarking, packaging tools, and running robust and accurate research experiments, and
6. provide research teams that do not have sufficient computing resources with the opportunity to obtain experimental results on large benchmark sets.

The SV-COMP 2020 report [17] discusses the achievements of the SV-COMP competition so far with respect to these objectives.

Related Competitions. There are many competitions in the area of formal methods [9], because it is well-understood that competitions are a fair and accurate means to execute a comparative evaluation with involvement of the developing teams. We refer to a previous report [17] for a more detailed discussion and give here only the references to the most related competitions [22, 58, 67, 74].

Quick Summary of Changes. While we try to keep the setup of the competition stable, there are always improvements and developments. For the 2023 edition, the following changes were made:

- The category for data-race detection was added (last year as demonstration, this year as regular category).
- New verification tasks were added, with an increase in C from 15 648 in 2022 to 23 805 in 2023.
- A new track was added that evaluates all validators for verification witnesses, which was discussed and approved by the jury in the 2022 community meeting in Munich, based on a proposal by two community members [37].

2 Organization, Definitions, Formats, and Rules

Procedure. The overall organization of the competition did not change in comparison to the earlier editions [10, 11, 12, 13, 14, 15, 16, 17, 18]. SV-COMP is an open competition (also known as comparative evaluation), where all verification tasks are known before the submission of the participating verifiers, which is necessary due to the complexity of the C language. The procedure is partitioned into the *benchmark submission* phase, the *training* phase, and the *evaluation* phase. The participants received the results of their verifier continuously via e-mail (for preruns and the final competition run), and the results were publicly announced on the competition web site after the teams inspected them.

Competition Jury. Traditionally, the competition jury consists of the chair and one member of each participating team; the team-representing members circulate every year after the candidate-submission deadline. This committee reviews the competition contribution papers and helps the organizer with resolving any disputes that might occur (cf. competition report of SV-COMP 2013 [11]). The

Table 1: Scoring schema for SV-COMP 2023 (unchanged from 2021 [18])

Reported result	Points	Description
Unknown	0	Failure to compute verification result
False correct	+1	Violation of property in program was correctly found and a validator confirmed the result based on a witness
False incorrect	−16	Violation reported but property holds (false alarm)
True correct	+2	Program correctly reported to satisfy property and a validator confirmed the result based on a witness
True incorrect	−32	Incorrect program reported as correct (wrong proof)

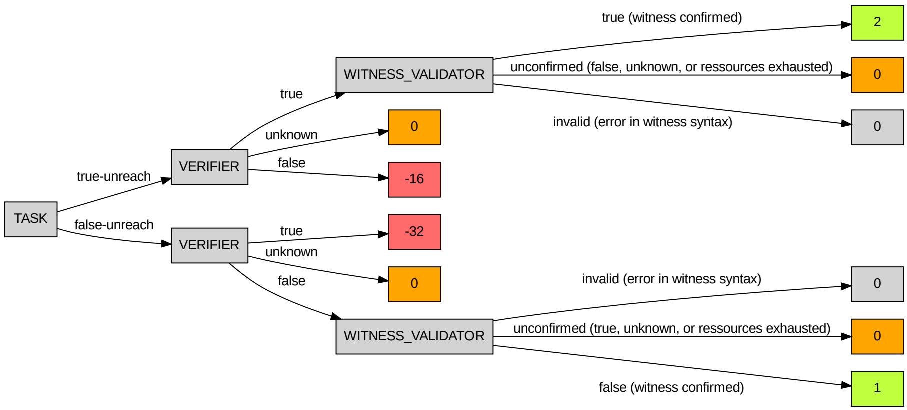

Fig. 1: Visualization of the scoring schema for the reachability property (unchanged from 2021 [18])

tasks of the jury were described in more detail in the report of SV-COMP 2022 [20]. The team representatives of the competition jury are listed in Table 5.

Scoring Schema and Ranking. The scoring schema of SV-COMP 2023 was the same as for SV-COMP 2021. Table 1 provides an overview and Fig. 1 visually illustrates the score assignment for the reachability property as an example. As before, the rank of a verifier was decided based on the sum of points (normalized for meta categories). In case of a tie, the rank was decided based on success run time, which is the total CPU time over all verification tasks for which the verifier reported a correct verification result. *Opt-out from Categories* and *Score Normalization for Meta Categories* was done as described previously [11, page 597].

License Requirements. Starting 2018, SV-COMP required that the verifier must be publicly available for download and has a license that

(i) allows reproduction and evaluation by anybody (incl. results publication),
(ii) does not restrict the usage of the verifier output (log files, witnesses), and
(iii) allows (re-)distribution of the unmodified verifier archive via SV-COMP repositories and archives.

Table 2: Publicly available components for reproducing SV-COMP 2023

Component	Fig. 3	Repository	Version
Verification Tasks	(a)	gitlab.com/sosy-lab/benchmarking/sv-benchmarks	svcomp23
Benchmark Definitions	(b)	gitlab.com/sosy-lab/sv-comp/bench-defs	svcomp23
Tool-Info Modules	(c)	github.com/sosy-lab/benchexec	3.16
Verifier Archives	(d)	gitlab.com/sosy-lab/sv-comp/archives-2023	svcomp23
Benchmarking	(e)	github.com/sosy-lab/benchexec	3.16
Witness Format	(f)	gitlab.com/sosy-lab/benchmarking/sv-witnesses	svcomp23
Continuous Integration	(f)	gitlab.com/sosy-lab/software/coveriteam	1.0

Table 3: Artifacts published for SV-COMP 2023

Content	DOI	Reference
Verification Tasks	10.5281/zenodo.7627783	[23]
Competition Results	10.5281/zenodo.7627787	[21]
Verifiers and Validators	10.5281/zenodo.7627829	[25]
Verification Witnesses	10.5281/zenodo.7627791	[24]
BenchExec	10.5281/zenodo.7612021	[112]
CoVeriTeam	10.5281/zenodo.7635975	[32]

Task-Definition Format 2.0. SV-COMP 2023 used the task-definition format in version 2.0. More details can be found in the report for Test-Comp 2021 [19].

Properties. Please see the 2015 competition report [13] for the definition of the properties and the property format. All specifications used in SV-COMP 2023 are available in the directory `c/properties/` of the benchmark repository.

Categories. The (updated) category structure of SV-COMP 2023 is illustrated by Fig. 2. Category *C-FalsificationOverall* contains all verification tasks of *C-Overall* without *Termination* and *Java-Overall* contains all Java verification tasks. Compared to SV-COMP 2022, we added one new sub-category *ReachSafety-Hardware* to main category *ReachSafety*, sub-categories *ConcurrencySafety-MemSafety*, *ConcurrencySafety-NoOverflows*, and *ConcurrencySafety-NoDataRace-Main* (was demo in 2022) to main category *ConcurrencySafety*, main category *NoOverflows* was restructured, and finally we added *SoftwareSystems-DeviceDriversLinux64-MemSafety* to main category *SoftwareSystems*. The categories are also listed in Tables 8, 9, and 10, and described in detail on the competition web site (`https://sv-comp.sosy-lab.org/2023/benchmarks.php`).

Reproducibility. SV-COMP results must be reproducible, and consequently, all major components are maintained in public version-control repositories. The overview of the components is provided in Fig. 3, and the details are given in Table 2. We refer to the SV-COMP 2016 report [14] for a description of all components of the SV-COMP organization. There are competition artifacts at Zenodo (see Table 3) to guarantee their long-term availability and immutability.

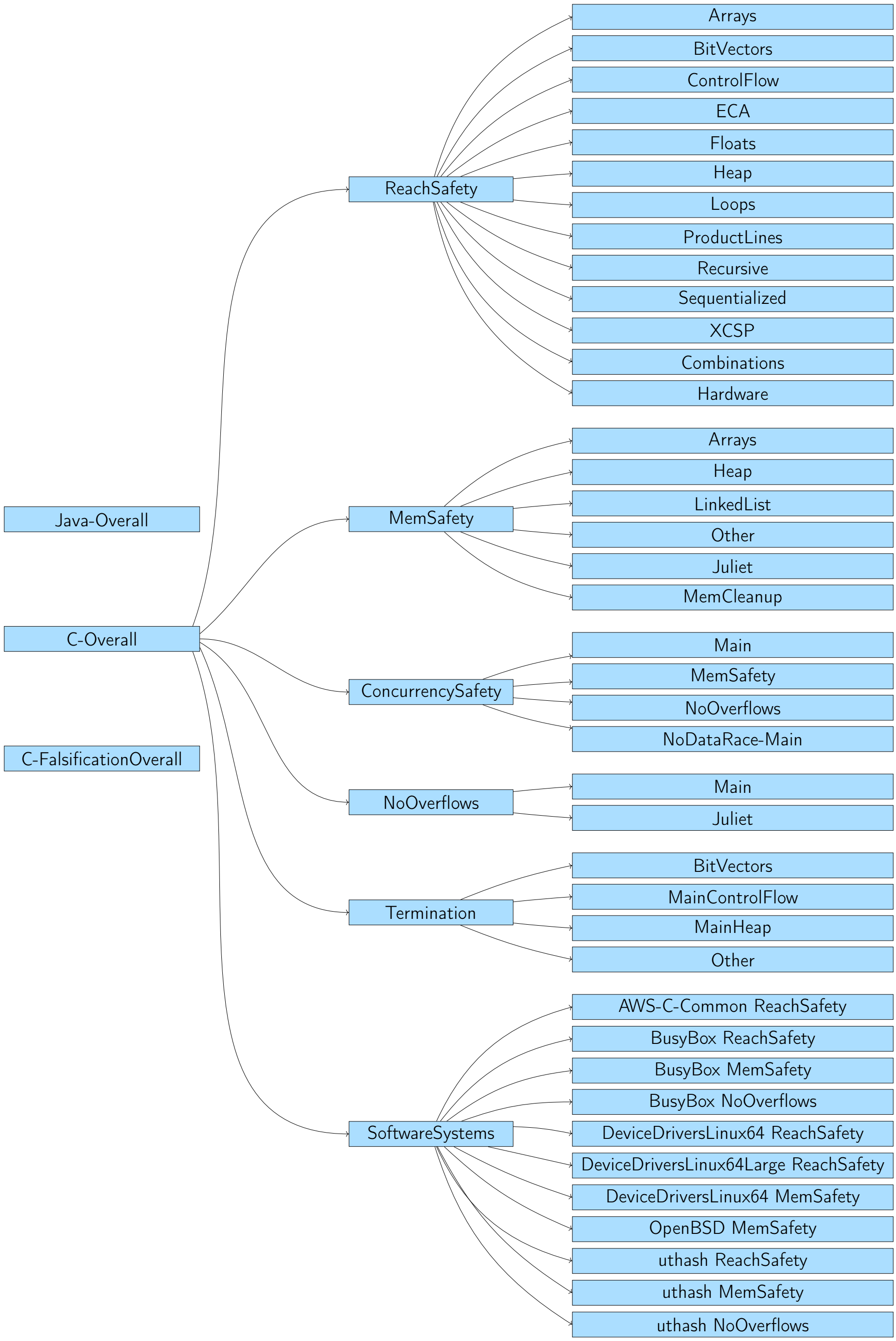

Fig. 2: Category structure for SV-COMP 2023

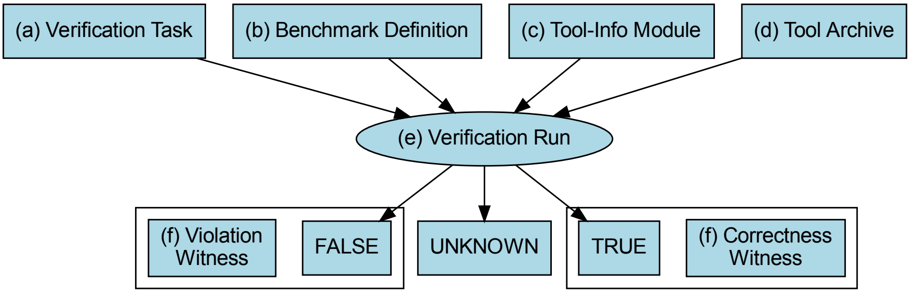

Fig. 3: Benchmarking components of SV-COMP and competition's execution flow (same as for SV-COMP 2020)

Table 4: Validation: Witness validators and witness linter

Validator	**Reference**	**Jury Member**	**Affiliation**
CPAchecker	[26, 27, 29]	Henrik Wachowitz	LMU Munich, Germany
CPA-w2t	[28]	Henrik Wachowitz	LMU Munich, Germany
Dartagnan	[98]	Hernán Ponce de León	Huawei Dresden, Germany
CProver-w2t	[28]	Michael Tautschnig	Queen Mary U. of London, UK
GWIT	[75]	Falk Howar	TU Dortmund U., Germany
MetaVal	[35]	Martin Spiessl	LMU Munich, Germany
NITWIT	[115]	Jana (Philipp) Berger	RWTH Aachen, Germany
Symbiotic-Witch	[7]	Paulína Ayaziová	Masaryk U., Brno, Czechia
UAutomizer	[26, 27]	Daniel Dietsch	U. of Freiburg, Germany
WIT4JAVA	[113]	Tong Wu	U. of Manchester, UK
WitnessLint		Martin Spiessl	LMU Munich, Germany

Competition Workflow. The workflow of the competition is described in the report for Test-Comp 2021 [19] (SV-COMP and Test-Comp use a similar workflow). For a description of how to reproduce single verification runs and a trouble-shooting guide, we refer to the previous report [20, Sect. 3].

3 Participating Verifiers and Validators

The participating verification systems are listed in Table 5. The table contains the verifier name (with hyperlink), references to papers that describe the systems, the representing jury member and the affiliation. The listing is also available on the competition web site at `https://sv-comp.sosy-lab.org/2023/systems.php`. Table 6 lists the algorithms and techniques that are used by the verification tools, and Table 7 gives an overview of commonly used solver libraries and frameworks.

Validation of Verification Results. The validation of the verification results was done by eleven validation tools (ten proper witness validators, and one

Table 5: Verification: Participating verifiers with tool references and representing jury members; [new] for first-time participants, ∅ for hors-concours participation

Participant	Ref.	Jury member	Affiliation
2LS	[39, 88]	Viktor Malík	BUT, Brno, Czechia
BRICK	[40]	Lei Bu	Nanjing U., China
BUBAAK [new]	[42]	Marek Chalupa	ISTA, Austria
CBMC	[46, 84]	Michael Tautschnig	Queen Mary U. London, UK
COASTAL∅	[109]	(hors concours)	–
CPA-BAM-BNB∅	[4, 111]	(hors concours)	–
CPA-BAM-SMG∅		(hors concours)	–
CPACHECKER	[33, 53]	Henrik Wachowitz	LMU Munich, Germany
CPALOCKATOR∅	[5, 6]	(hors concours)	–
CRUX∅	[57, 104]	(hors concours)	–
CSEQ∅	[51, 79]	(hors concours)	–
CVT-ALGOSEL∅	[30, 31]	(hors concours)	–
CVT-PARPORT∅	[30, 31]	(hors concours)	–
DARTAGNAN	[65, 97]	Hernán Ponce de León	Huawei Dresden, Germany
DEAGLE	[70]	Fei He	Tsinghua U., China
DIVINE∅	[8, 85]	(hors concours)	–
EBF	[3]	Fatimah Aljaafari	U. of Manchester, UK
ESBMC-INCR∅	[47, 50]	(hors concours)	–
ESBMC-KIND	[63, 64]	Rafael Sá Menezes	U. of Manchester, UK
FRAMA-C-SV	[36, 52]	Martin Spiessl	LMU Munich, Germany
GAZER-THETA∅	[1, 69]	(hors concours)	–
GDART	[93]	Falk Howar	TU Dortmund, Germany
GDART-LLVM [new]		Falk Howar	TU Dortmund, Germany
GOBLINT	[103, 110]	Simmo Saan	U. of Tartu, Estonia
GRAVES-CPA	[86]	Will Leeson	U. of Virginia, USA
GRAVES-PAR [new]		Hors Concurs	U. of Virginia, USA
INFER∅	[41, 82]	(hors concours)	–
JAVA-RANGER	[76, 106]	Soha Hussein	U. of Minnesota, USA
JAYHORN∅	[81, 105]	(hors concours)	–
JBMC	[48, 49]	Peter Schrammel	U. of Sussex / Diffblue, UK
JDART∅	[87, 92]	(hors concours)	–
KORN	[60, 61]	Gidon Ernst	LMU Munich, Germany
LAZY-CSEQ∅	[77, 78]	(hors concours)	–
LF-CHECKER [new]		Tong Wu	U. of Manchester, UK
LOCKSMITH	[99]	Vesal Vojdani	U. of Tartu, Estonia
MLB [new]		Lei Bu	Nanjing U., China
MOPSA [new]	[80, 91]	Raphaël Monat	Inria and U. of Lille, France
PESCO-CPA	[101, 102]	Cedric Richter	U. of Oldenburg, Germany
PICHECKER [new]	[107]	Jie Su	Xidian U., China
PINAKA∅	[45]	(hors concours)	–
PREDATORHP∅	[73, 96]	(hors concours)	–

(continues on next page)

Table 5: Competition candidates (continued)

Participant	Ref.	Jury member	Affiliation
Spf∅	[94, 100]	(hors concours)	–
Symbiotic	[43, 44]	Marek Trtík	Masaryk U., Brno, Czechia
Theta	[108, 114]	Levente Bajczi	BME Budapest, Hungary
UAutomizer	[71, 72]	Matthias Heizmann	U. of Freiburg, Germany
UGemCutter	[62, 83]	Dominik Klumpp	U. of Freiburg, Germany
UKojak	[59, 95]	Frank Schüssele	U. of Freiburg, Germany
UTaipan	[56, 68]	Daniel Dietsch	U. of Freiburg, Germany
VeriAbs	[2, 54]	Priyanka Darke	TCS, India
VeriAbsL new	[55]	Priyanka Darke	TCS, India
VeriFuzz	[89, 90]	Raveendra Kumar M.	TCS, India
VeriOOver new		HaiPeng Qu	Ocean U. of China, China

Table 6: Algorithms and techniques that the participating verification systems used; new for first-time participants, ∅ for hors-concours participation

Verifier	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reach.	Explicit-Value Analysis	Numeric. Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions	Evolutionary Algorithms	Algorithm Selection	Portfolio
2ls				✓	✓			✓	✓		✓						✓			
Brick	✓		✓	✓				✓								✓				
Bubaak new			✓								✓					✓	✓			✓
Cbmc				✓							✓					✓				
Coastal∅			✓																	
CPA-BAM-BnB∅	✓	✓					✓				✓	✓	✓	✓						
CPA-BAM-SMG∅																				
CPAchecker	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
CPALockator∅	✓	✓					✓				✓	✓	✓	✓		✓				
Crux∅			✓																	
CSeq∅				✓							✓					✓				
CVT-AlgoSel∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CVT-ParPort∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Dartagnan				✓							✓					✓				
Deagle				✓												✓				
Divine∅			✓				✓				✓					✓			✓	✓
Ebf				✓																
Esbmc-incr∅				✓	✓						✓					✓				
Esbmc-kind				✓	✓			✓			✓					✓				

(continues on next page)

Table 6: Algorithms and techniques (continued)

Verifier	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reach.	Explicit-Value Analysis	Numeric. Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions	Evolutionary Algorithms	Algorithm Selection	Portfolio
Frama-C-SV								✓												
Gazer-Theta[∅]	✓	✓		✓			✓				✓	✓	✓	✓						✓
GDart			✓								✓									✓
GDart-LLVM [new]			✓								✓									
Goblint								✓								✓			✓	
Graves-CPA	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Graves-Par [new]																				
Infer[∅]								✓	✓	✓										✓
Java-Ranger			✓								✓									
JayHorn[∅]	✓	✓				✓		✓					✓	✓						
Jbmc				✓							✓					✓				
JDart[∅]			✓								✓									✓
Korn		✓	✓				✓													✓
Lazy-CSeq[∅]				✓							✓					✓				
LF-checker [new]																				
Locksmith																✓				
Mlb [new]																				
Mopsa [new]								✓												
PeSCo-CPA	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
PIChecker [new]	✓	✓									✓	✓		✓		✓			✓	
Pinaka[∅]			✓	✓							✓									
PredatorHP[∅]									✓											
Spf[∅]			✓						✓							✓				
Symbiotic			✓		✓			✓	✓		✓					✓				✓
Theta	✓	✓					✓				✓	✓		✓		✓			✓	✓
UAutomizer	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓
UGemCutter	✓	✓									✓		✓	✓	✓	✓			✓	✓
UKojak	✓	✓									✓		✓	✓						
UTaipan	✓	✓					✓	✓			✓		✓	✓	✓	✓			✓	✓
VeriAbs	✓			✓	✓		✓	✓										✓	✓	✓
VeriAbsL [new]	✓			✓	✓		✓	✓										✓	✓	✓
VeriFuzz				✓				✓										✓		
VeriOover [new]																				

Table 7: Solver libraries and frameworks that are used as components in the participating verification systems (component is mentioned if used more than three times; new for first-time participants, ∅ for hors-concours participation)

Verifier	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc4	SMTInterpol	z3	MiniSAT	Apron
2ls		✓									✓	
Brick										✓	✓	
Bubaak new										✓		
Cbmc		✓									✓	
Coastal∅				✓								
CPA-BAM-BnB∅	✓					✓	✓					
CPA-BAM-SMG∅	✓					✓	✓					
CPAchecker	✓					✓	✓					✓
CPALockator∅	✓					✓	✓					
Crux∅										✓		
CSeq∅		✓									✓	
CVT-AlgoSel∅	✓	✓	✓		✓	✓	✓				✓	
CVT-ParPort∅	✓	✓	✓		✓	✓	✓				✓	
Dartagnan						✓						
Deagle											✓	
Divine∅												
Ebf			✓				✓					
Esbmc-incr∅			✓				✓					
Esbmc-kind			✓				✓					
Frama-C-SV												
Gazer-Theta∅												
GDart								✓		✓		
GDart-Llvm new										✓		
Goblint												✓
Graves-CPA	✓					✓	✓					
Graves-Par new												
Infer∅												
Java-Ranger				✓								
JayHorn∅												
Jbmc		✓									✓	
JDart∅				✓				✓		✓		
Korn										✓		
Lazy-CSeq∅		✓									✓	
LF-checker new												
Locksmith												

(continues on next page)

Table 7: Solver libraries and frameworks (continued)

Verifier	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc4	SMTInterpol	z3	MiniSAT	Apron
Mlb [new]												
Mopsa [new]												✓
PeSCo-CPA	✓					✓	✓					
PIChecker [new]	✓					✓	✓		✓			
Pinaka ∅												
PredatorHP ∅												
Spf ∅				✓								
Symbiotic										✓		
Theta												
UAutomizer					✓		✓	✓	✓	✓		
UGemCutter					✓		✓	✓	✓	✓		
UKojak					✓				✓	✓		
UTaipan					✓		✓	✓	✓	✓		
VeriAbs	✓	✓								✓	✓	
VeriAbsL [new]	✓	✓								✓	✓	
VeriFuzz										✓		
VeriOover [new]												

witness linter for syntax checks), which are listed in Table 4, including references to literature. The ten witness validators are evaluated based on all verification witnesses that were produced in the verification track of the competition.

Hors-Concours Participation. As in previous years, we also included verifiers to the evaluation that did not actively compete or that should not occur in the rankings for some reasons (e.g., meta verifiers based on other competing tools, or tools for which the submitting teams were not sure if they show the full potential of the tool). These participations are called *hors concours*, as they cannot participate in rankings and cannot "win" the competition. Those verifiers are marked as 'hors concours' in Table 5 and others, and the names are annotated with a symbol (∅).

4 Results of the Verification Track

The results of the competition represent the the state of the art of what can be achieved with fully automatic software-verification tools on the given benchmark set. We report the effectiveness (number of verification tasks that can be solved and correctness of the results, as accumulated in the score) and the efficiency (resource consumption in terms of CPU time and CPU energy). The results are presented in the same way as in last years, such that the improvements compared

Table 8: Verification: Quantitative overview over all regular results;

Participant	**ReachSafety** 9814 points 6138 tasks	**MemSafety** 4543 points 3202 tasks	**ConcurrencySafety** 5295 points 2865 tasks	**NoOverflows** 10200 points 6618 tasks	**Termination** 3103 points 1809 tasks	**SoftwareSystems** 5132 points 3173 tasks	**FalsificationOverall** 8689 points 21996 tasks	**Overall** 38644 points 23805 tasks	**JavaOverall** 827 points 586 tasks
2LS	3617	611	0	6570	**1183**	75	1884	9722	
Brick									
Bubaak [new]	4278	-4655	9	5005	195	**1589**	**4313**	2426	
Cbmc	3497	1798	1185	6267	862	-711	2110	10886	
CPAchecker	5535	**2612**	1744	4079	883	758	**4254**	**14559**	
Dartagnan			1268						
Deagle			**4744**						
Divine ∅	2698	-354	-2	0	0	101	-573	1429	
Ebf			-317						
Esbmc-incr ∅			480						
Esbmc-kind	5183	2000	1162	6342	782	275	3507	13299	
Frama-C-SV				1522					
GDart-Llvm [new]									
Goblint	874		1591	5306		358		6397	
Graves-CPA	4868					-1186	3447	5258	
Graves-Par [new]	-64	2179	118	-17785	590	-1337	-3063	-8217	
Korn									
LF-checker [new]			1023						
Locksmith									
Mopsa [new]	699	556		5671		**815**			
PeSCo-CPA	**5576**					812	**4258**	**14652**	
PIChecker [new]			552						
Symbiotic	4786	**2620**	194	2407	930	**1604**	4026	12097	
Theta	1076		1286						
UAutomizer	3997	2301	**2717**	**8639**	**2105**	476	4173	**19589**	
UGemCutter			**2710**						
UKojak	2306	1526	0	**7305**	0	274	3092	8102	
UTaipan	3672	**2354**	2612	**8492**	0	412	4100	14514	
VeriAbs	**6628**								
VeriAbsL [new]	**6478**								
VeriFuzz	1704			-500	**2305**				
VeriOover [new]									
GDart									**652**
Java-Ranger									400
Jbmc									**667**
Mlb [new]									**495**

Table 9: Verification: Quantitative overview over all hors-concours results; empty cells represent opt-outs, [new] for first-time participants, ∅ for hors-concours participation

Verifier	**ReachSafety** 8631 points 5400 tasks	**MemSafety** 5003 points 3321 tasks	**ConcurrencySafety** 1160 points 763 tasks	**NoOverflows** 685 points 454 tasks	**Termination** 4144 points 2293 tasks	**SoftwareSystems** 5898 points 3417 tasks	**FalsificationOverall** 5718 points 13355 tasks	**Overall** 25209 points 15648 tasks	**JavaOverall** 828 points 586 tasks
CVT-AlgoSel ∅	-507		59						
CVT-ParPort ∅	2033	2539	847	-3793	947	1421	3734	7212	
CPA-BAM-BnB ∅						458			
CPA-BAM-SMG ∅		2587				804			
CPALockator ∅			-2720						
Crux ∅	879			1316					
CSeq ∅			-11702						
Divine ∅	2698	-354	-2	0	0	101	-573	1429	
Esbmc-incr ∅			480						
Gazer-Theta ∅									
Infer ∅	-56129		-5737	-77220		-25556			
Lazy-CSeq ∅			-13840						
Pinaka ∅	3387			-879	631				
PredatorHP ∅		1926							
Coastal ∅									-2816
JayHorn ∅									220
JDart ∅									382
Spf ∅									182

to the last years are easy to identify. The results presented in this report were inspected and approved by the participating teams.

Quantitative Results. Tables 8 and 9 present the quantitative overview of all tools and all categories. Due to the large number of tools, we need to split the presentation into two tables, one for the verifiers that participate in the rankings (Table 8), and one for the hors-concours verifiers (Table 9). The head row mentions the category, the maximal score for the category, and the number of verification tasks. The tools are listed in alphabetical order; every table row lists the scores of one verifier. We indicate the top three candidates by formatting their scores in bold face and in larger font size. An empty table cell means that the verifier opted-out from the respective main category (perhaps participating in subcategories only, restricting the evaluation to a specific topic). More information (including interactive tables, quantile plots for every category, and also the raw data in XML format) is available on the competition web site (`https://sv-comp.sosy-lab.org/2023/results`) and in the results artifact (see Table 3).

Table 10: Verification: Overview of the top-three verifiers for each category; new for first-time participants, values for CPU time and energy rounded to two significant digits

Rank	Verifier	Score	CPU Time (in h)	CPU Energy (in kWh)	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
ReachSafety								
1	**VeriAbs**	**6628**	150	1.6	3 509	431		
2	VeriAbsL new	6478	120	1.1	3 600	567		**8**
3	PeSCo-CPA	5576	79	0.84	3 294	330	3	**8**
MemSafety								
1	**Symbiotic**	**2620**	1.4	0.018	304	0	2	
2	CPAchecker	2612	6.1	0.053	3 053	0		
3	UTaipan	2354	34	0.33	1 945	29		
ConcurrencySafety								
1	**Deagle**	**4744**	1.1	0.014	2 545	27	1	
2	UAutomizer	2717	34	0.37	1 498	18		
3	UGemCutter	2710	36	0.37	1 495	13		
NoOverflows								
1	**UAutomizer**	**8639**	53	0.48	5 407	62		
2	UTaipan	8492	55	0.51	5 296	107		
3	UKojak	7305	32	0.26	4 275	60		
Termination								
1	**VeriFuzz**	**2305**	21	0.26	1 216	141		**3**
2	UAutomizer	2105	13	0.12	1 196	9		
3	2ls	1183	3.7	0.029	1 005	205		
SoftwareSystems								
1	**Symbiotic**	**1604**	0.80	0.011	1 026	189		**1**
2	Bubaak new	1589	0.32	0.0036	432	206	1	
3	Mopsa new	815	12	0.16	1 610	94		
FalsificationOverall								
1	**Bubaak** new	**4313**	36	0.39	5 258	219	10	
2	PeSCo-CPA	4258	46	0.49	3 800	150	7	
3	CPAchecker	4254	90	1.0	3 677	99	4	
Overall								
1	**UAutomizer**	**19589**	250	2.5	13 367	337		
2	PeSCo-CPA	14652	160	1.7	10 372	497	9	**8**
3	CPAchecker	14559	220	2.5	10 200	539	6	
JavaOverall								
1	**Jbmc**	**667**	0.34	0.0032	473	29		
2	GDart	652	3.0	0.026	477	9		
3	Mlb new	495	0.38	0.0036	336	95		

Table 10 reports the top three verifiers for each category. The run time (column ‘CPU Time’) and energy (column ‘CPU Energy’) refer to successfully solved verification tasks (column ‘Solved Tasks’). We also report the number of tasks for which no witness validator was able to confirm the result (column ‘Unconf. Tasks’). The columns ‘False Alarms’ and ‘Wrong Proofs’ report the number of verification

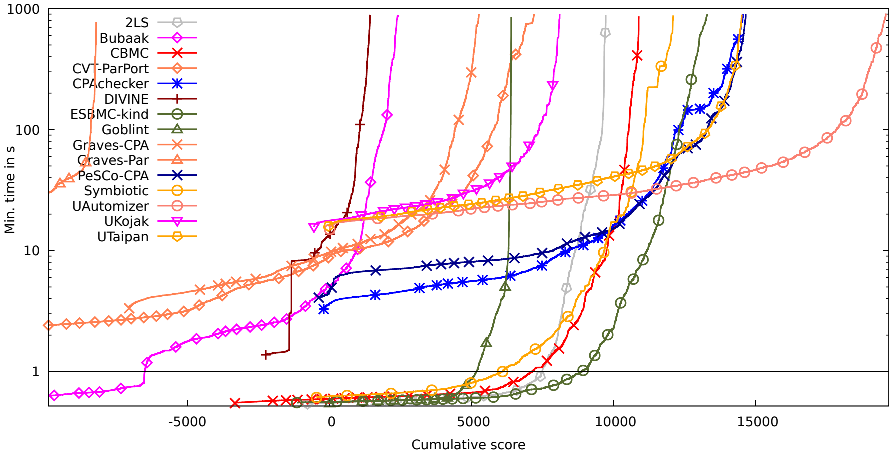

Fig. 4: Quantile functions for category *C-Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by correct verification runs below a certain run time (y-coordinate). More details were given previously [11]. A logarithmic scale is used for the time range from 1 s to 1000 s, and a linear scale is used for the time range between 0 s and 1 s.

tasks for which the verifier reported wrong results, i.e., reporting a counterexample when the property holds (incorrect FALSE) and claiming that the program fulfills the property although it actually contains a bug (incorrect TRUE), respectively.

Score-Based Quantile Functions for Quality Assessment. We use score-based quantile functions [11, 34] because these visualizations make it easier to understand the results of the comparative evaluation. The results archive (see Table 3) and the web site (`https://sv-comp.sosy-lab.org/2023/results`) include such a plot for each (sub-)category. As an example, we show the plot for category *C-Overall* (all verification tasks) in Fig. 4. A total of 13 verifiers participated in category *C-Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [11]). A more detailed discussion of score-based quantile plots, including examples of what insights one can obtain from the plots, is provided in previous competition reports [11, 14].

The winner of the competition, UAUTOMIZER, achieves the best cumulative score (graph for UAUTOMIZER has the longest width from $x = 0$ to its right end). Verifiers whose graphs start with a negative cumulative score produced wrong results.

New Verifiers. To acknowledge the verification systems that participate for the first or second time in SV-COMP, Table 11 lists the new verifiers (in SV-COMP 2022 or SV-COMP 2023). It is remarkable to see that first-time participants can win or almost win large categories: BUBAAK[new] is the best verifier for category *FalsificationOverall*, and BUBAAK[new] is the second-best and MOPSA[new] third-best in category *SoftwareSystems*. Figure 5 shows the growing interest in the competition over the years.

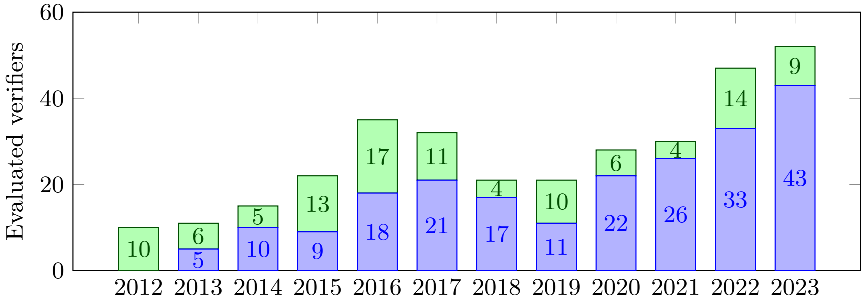

Fig. 5: Number of evaluated verifiers for each year (first-time participants on top)

Table 11: New verifiers in SV-COMP 2022 and SV-COMP 2023; column 'Sub-categories' gives the number of executed categories (including demo category *NoDataRace*), new for first-time participants, ∅ for hors-concours participation

Verifier	Language	First Year	Sub-categories
Bubaak new	C	2023	40
GDart-Llvm new	C	2023	1
Graves-Par new	C	2023	40
LF-checker new	C	2023	3
Mopsa new	C	2023	32
PIChecker new	C	2023	1
VeriAbsL new	C	2023	13
VeriOover new	C	2023	1
Mlb new	Java	2023	1
CVT-AlgoSel ∅	C	2022	18
CVT-ParPort ∅	C	2022	35
CPA-BAM-SMG ∅	C	2022	16
Crux ∅	C	2022	20
Deagle	C	2022	1
Ebf	C	2022	1
Graves-CPA	C	2022	35
Infer ∅	C	2022	25
Lart	C	2022	22
Locksmith	C	2022	1
Sesl	C	2022	6
Theta	C	2022	13
UGemCutter	C	2022	2
GDart	Java	2022	1

Computing Resources. The resource limits were the same as in the previous competitions [14], except for the upgraded operating system: Each verification run was limited to 8 processing units (cores), 15 GB of memory, and 15 min of CPU time. Witness validation was limited to 2 processing units, 7 GB of memory, and 1.5 min of CPU time for violation witnesses and 15 min of CPU time for correctness witnesses. The machines for running the experiments are part of a

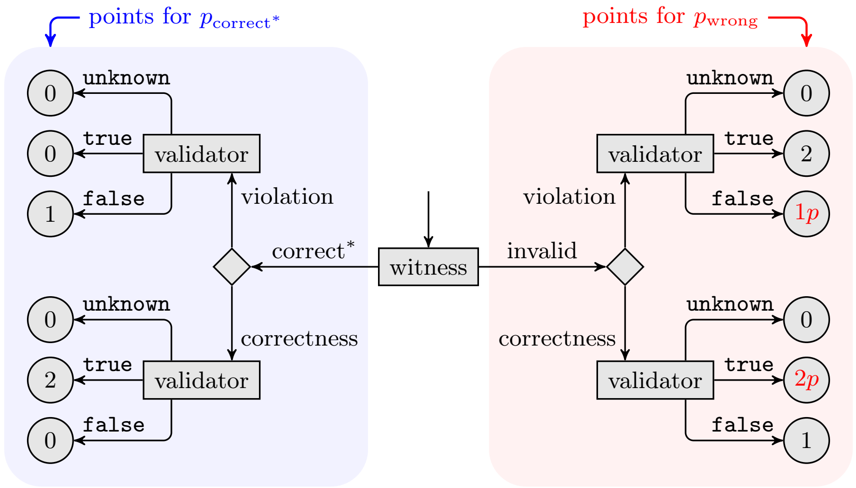

Fig. 6: Scoring schema for evaluation of validators; $p = -16$ for SV-COMP 2023; figure adopted from [37]

compute cluster that consists of 168 machines; each verification run was executed on an otherwise completely unloaded, dedicated machine, in order to achieve precise measurements. Each machine had one Intel Xeon E3-1230 v5 CPU, with 8 processing units each, a frequency of 3.4 GHz, 33 GB of RAM, and a GNU/Linux operating system (x86_64-linux, Ubuntu 22.04 with Linux kernel 5.15). We used BenchExec [34] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud to distribute, install, run, and clean-up verification runs, and to collect the results. The values for time and energy are accumulated over all cores of the CPU. To measure the CPU energy, we used CPU Energy Meter [38] (integrated in BenchExec [34]).

One complete verification execution of the competition consisted of 490 858 verification runs in 91 run sets (each verifier on each verification task of the selected categories according to the opt-outs), consuming 1 114 days of CPU time and 299 kWh of CPU energy (without validation). Witness-based result validation required 4.59 million validation runs in 1 527 run sets (each validator on each verification task for categories with witness validation, and for each verifier), consuming 877 days of CPU time. Each tool was executed several times, in order to make sure no installation issues occur during the execution. Including these preruns, the infrastructure managed a total of 2.78 million verification runs in 560 run sets (verifier × property) consuming 13.8 years of CPU time, and 35.9 million validation runs in 11 532 run sets (validator × verifier × property) consuming 17.8 years of CPU time. This means that also the load of the experiment infrastructure increased and was larger than ever before.

Table 12: Validation of violation witnesses: Overview of the top-three verifiers for each category; values for CPU time and energy rounded to two significant digits

Rank	Validator	Score	CPU Time (in h)	Solved Tasks	False Alarms	Wrong Proofs
ReachSafety						
1	**UAutomizer**	**62966**	99	12 196		
2	CProver-w2t	49545	16	18 903	2	
3	CPAchecker	33938	92	17 770	12	
MemSafety						
1	**UAutomizer**	**31156**	49	5 680		
2	CPAchecker	9013	40	16 881	7	
3	CPA-w2t	1241	0.76	327		
ConcurrencySafety						
1	**Dartagnan**	**9777**	44	6 520	14	
2	CPAchecker	2658	14	3 466	28	
3	UAutomizer	912	1.2	263		
NoOverflows						
1	**UAutomizer**	**74933**	150	23 142		
2	CProver-w2t	61848	6.3	13 450		
3	CPAchecker	28600	5.0	2 747		
Termination						
1	**UAutomizer**	**3017**	7.6	1 052		
2	CPAchecker	423	19	3 113		
3	MetaVal	0	0	0		
SoftwareSystems						
1	**Symbiotic-Witch**	**3304**	0.55	846	1	
2	UAutomizer	2468	29	3 579		
3	CPAchecker	1620	14	2 475		
Overall						
1	**UAutomizer**	**127030**	330	45 912		
2	CPAchecker	52851	180	46 452	47	
3	Symbiotic-Witch	35851	65	38 644	10	

5 Results of the Witness-Validation Track

The validation of verification results, in particular, verification witnesses, becomes more and more important for various reasons: verification witnesses justify and help to understand and interpret a verification result, they serve as exchange object for intermediate results, and they allow to make use of imprecise verification techniques (e.g., via machine learning). A case study on the quality of the results of witness validators [37] suggested that validators for verification results should also undergo a periodical comparative evaluation and proposed a scoring schema for witness-validation results. SV-COMP 2023 evaluated 10 validators on more than 100 000 verification witnesses.

Table 13: Validation of correctness witnesses: Overview of the top-three verifiers for each category; values for CPU time and energy rounded to two significant digits

Rank	**Validator**	**Score**	**CPU Time** (in h)	**Solved Tasks**	**False Alarms**	**Wrong Proofs**
ReachSafety						
1	**UAutomizer**	**21499**	350	16 768		
2	CPAchecker	17816	220	16 437		
3	MetaVal	-89088	320	14 217		**16**
MemSafety						
1	**UAutomizer**	**18219**	710	16 247		
2	MetaVal	0	0	0		
3	missing validator	0	0	0		
ConcurrencySafety						
1	**UAutomizer**	**12994**	140	10 232		
2	missing validator	0	0	0		
3	missing validator	0	0	0		
NoOverflows						
1	**UAutomizer**	**65478**	390	37 419		
2	CPAchecker	27151	14	3 082		
3	MetaVal	0	0	0		
Termination						
1	**missing validator**	**0**	0	0		
2	missing validator	0	0	0		
3	missing validator	0	0	0		
SoftwareSystems						
1	**CPAchecker**	**3147**	36	6 124		
2	UAutomizer	3027	300	17 385		
3	MetaVal	-121312	600	18 148		**232**
Overall						
1	**UAutomizer**	**93049**	1 900	98 051		
2	CPAchecker	30076	280	25 643		
3	MetaVal	-165166	910	32 365		**248**

Scoring Schema for Validation Track. The score of a validator in a sub-category is computed as

$$score = \left(\frac{p_{\text{correct}^*}}{|\text{correct}^*|} + q \cdot \frac{p_{\text{wrong}}}{|\text{wrong}|} \right) \cdot \frac{|\text{correct}^*| + |\text{wrong}|}{2}$$

where the points in p_{correct^*} and p_{wrong} are determined according to the schema in Fig. 6 and then normalized using the normalization schema that SV-COMP uses for meta categories [11, page 597], except for the factor q, which gives a higher weight to wrong witnesses. Wrong witnesses are witnesses that do not agree with the expected verification verdict. Witnesses that agree with the expected verification verdict cannot be automatically treated as correct because we do not yet have an established way to determine this. Therefore, we call this class of witnesses correct*. Further details are given in the proposal [37]. This schema relates to each base category from the verification track a meta category that consists of two sub-categories, one with the correct* and one with the wrong witnesses.

Tables 12 and 13 show the rankings of the validators. False alarms in Table 12 are claims of a validator that the program contains a bug described by a given violation witness although the program is correct (the validator confirms a wrong violation witness). Wrong proofs in Table 13 are claims of a validator that the program is correct according to invariants in a given correctness witness although the program contains a bug (the validator confirms a wrong correctness witness). The scoring schema significantly punishes results that confirm a wrong verification witness, as visible for validator MetaVal in Table 13.

Table 13 shows that there are categories that are supported by less than three validators ('missing validators'). This reveals a remarkable gap in software-verification research:

> There are verification results that cannot be independently confirmed, according to the state of the art in software verification.

6 Conclusion

The 12th edition of the Competition on Software Verification (SV-COMP 2023) again increased the number of participating systems and gave the largest ever overview over software-verification tools, with 52 participating verification systems (incl. 9 new verifiers and 18 hors-concours; see Fig. 5 for the participation numbers and Table 5 for the details). For the first time, a thorough comparative evaluation of 10 validation tools was performed; the validation tools were assessed in a similar manner as in the verification track, using a community-agreed scoring schema [37] which is derived from the scoring schema of the verification track. The number of verification tasks in SV-COMP 2023 was significantly increased to 23 805 in the C category. The high quality standards of the TACAS conference are ensured by a competition jury, with a member from each actively participating team. We hope that the broad overview of verification tools stimulates the further advancements of software verification, and in particular, the validation track showed some open problems that should be addressed.

Data-Availability Statement. The verification tasks and results of the competition are published at Zenodo, as described in Table 3. All components and data that are necessary for reproducing the competition are available in public version repositories, as specified in Table 2. For easy access, the results are presented also online on the competition web site `https://sv-comp.sosy-lab.org/2023/results`. The main results were reproduced in an independent reproduction study [66].

Funding Statement. This project was funded in part by the Deutsche Forschungsgemeinschaft (DFG) — 418257054 (Coop).

Acknowledgements. We thank Marcus Gerhold and Arnd Hartmanns for their reproduction study [66] on SV-COMP 2023.

References

1. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: Proc. TACAS (2). pp. 433–437. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_27
2. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141 (2019). https://doi.org/10.1109/ASE.2019.00121
3. Aljaafari, F., Shmarov, F., Manino, E., Menezes, R., Cordeiro, L.: EBF 4.2: Black-Box cooperative verification for concurrent programs (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
4. Andrianov, P., Friedberger, K., Mandrykin, M.U., Mutilin, V.S., Volkov, A.: CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions (competition contribution). In: Proc. TACAS. pp. 355–359. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_22
5. Andrianov, P., Mutilin, V., Khoroshilov, A.: CPALockator: Thread-modular approach with projections (competition contribution). In: Proc. TACAS (2). pp. 423–427. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_25
6. Andrianov, P.S.: Analysis of correct synchronization of operating system components. Program. Comput. Softw. **46**, 712–730 (2020). https://doi.org/10.1134/S0361768820080022
7. Ayaziová, P., Strejček, J.: Symbiotic-Witch 2: More efficient algorithm and witness refutation (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
8. Baranová, Z., Barnat, J., Kejstová, K., Kučera, T., Lauko, H., Mrázek, J., Ročkai, P., Štill, V.: Model checking of C and C++ with Divine 4. In: Proc. ATVA. pp. 201–207. LNCS 10482, Springer (2017). https://doi.org/10.1007/978-3-319-68167-2_14
9. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: TOOLympics 2019: An overview of competitions in formal methods. In: Proc. TACAS (3). pp. 3–24. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
10. Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_38
11. Beyer, D.: Second competition on software verification (Summary of SV-COMP 2013). In: Proc. TACAS. pp. 594–609. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_43
12. Beyer, D.: Status report on software verification (Competition summary SV-COMP 2014). In: Proc. TACAS. pp. 373–388. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_25
13. Beyer, D.: Software verification and verifiable witnesses (Report on SV-COMP 2015). In: Proc. TACAS. pp. 401–416. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_31
14. Beyer, D.: Reliable and reproducible competition results with BenchExec and witnesses (Report on SV-COMP 2016). In: Proc. TACAS. pp. 887–904. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_55

15. Beyer, D.: Software verification with validation of results (Report on SV-COMP 2017). In: Proc. TACAS. pp. 331–349. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_20
16. Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Proc. TACAS (3). pp. 133–155. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_9
17. Beyer, D.: Advances in automatic software verification: SV-COMP 2020. In: Proc. TACAS (2). pp. 347–367. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_21
18. Beyer, D.: Software verification: 10th comparative evaluation (SV-COMP 2021). In: Proc. TACAS (2). pp. 401–422. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_24
19. Beyer, D.: Status report on software testing: Test-Comp 2021. In: Proc. FASE. pp. 341–357. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_17
20. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20
21. Beyer, D.: Results of the 12th Intl. Competition on Software Verification (SV-COMP 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7627787
22. Beyer, D.: Software testing: 5th comparative evaluation: Test-Comp 2023. In: Proc. FASE. LNCS , Springer (2023)
23. Beyer, D.: SV-Benchmarks: Benchmark set for software verification and testing (SV-COMP 2023 and Test-Comp 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7627783
24. Beyer, D.: Verification witnesses from verification tools (SV-COMP 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7627791
25. Beyer, D.: Verifiers and validators of the 12th Intl. Competition on Software Verification (SV-COMP 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7627829
26. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). https://doi.org/10.1145/2950290.2950351
27. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). https://doi.org/10.1145/2786805.2786867
28. Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses: Execution-based validation of verification results. In: Proc. TAP. pp. 3–23. LNCS 10889, Springer (2018). https://doi.org/10.1007/978-3-319-92994-1_1
29. Beyer, D., Friedberger, K.: Violation witnesses and result validation for multi-threaded programs. In: Proc. ISoLA (1). pp. 449–470. LNCS 12476, Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_26
30. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31
31. Beyer, D., Kanav, S., Richter, C.: Construction of verifier combinations based on off-the-shelf verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_3
32. Beyer, D., Kanav, S., Wachowitz, H.: Coveriteam Release 1.0. Zenodo (2023). https://doi.org/10.5281/zenodo.7635975

33. Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16
34. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Tools Technol. Transfer **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y
35. Beyer, D., Spiessl, M.: MetaVal: Witness validation via verification. In: Proc. CAV. pp. 165–177. LNCS 12225, Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_10
36. Beyer, D., Spiessl, M.: The static analyzer Frama-C in SV-COMP (competition contribution). In: Proc. TACAS (2). pp. 429–434. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_26
37. Beyer, D., Strejček, J.: Case study on verification-witness validators: Where we are and where we go. In: Proc. SAS. pp. 160–174. LNCS 13790, Springer (2022). https://doi.org/10.1007/978-3-031-22308-2_8
38. Beyer, D., Wendler, P.: CPU Energy Meter: A tool for energy-aware algorithms engineering. In: Proc. TACAS (2). pp. 126–133. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_8
39. Brain, M., Joshi, S., Kröning, D., Schrammel, P.: Safety verification and refutation by k-invariants and k-induction. In: Proc. SAS. pp. 145–161. LNCS 9291, Springer (2015). https://doi.org/10.1007/978-3-662-48288-9_9
40. Bu, L., Xie, Z., Lyu, L., Li, Y., Guo, X., Zhao, J., Li, X.: Brick: Path enumeration-based bounded reachability checking of C programs (competition contribution). In: Proc. TACAS (2). pp. 408–412. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_22
41. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. ACM **58**(6), 26:1–26:66 (2011). https://doi.org/10.1145/2049697.2049700
42. Chalupa, M., Henzinger, T.: Bubaak: Runtime monitoring of program verifiers (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
43. Chalupa, M., Strejček, J., Vitovská, M.: Joint forces for memory safety checking. In: Proc. SPIN. pp. 115–132. Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_7
44. Chalupa, M., Řechtáčková, A., Mihalkovič, V., Zaoral, L., Strejček, J.: Symbiotic 9: String analysis and backward symbolic execution with loop folding (competition contribution). In: Proc. TACAS (2). pp. 462–467. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_32
45. Chaudhary, E., Joshi, S.: Pinaka: Symbolic execution meets incremental solving (competition contribution). In: Proc. TACAS (3). pp. 234–238. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_20
46. Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15
47. Cordeiro, L.C., Fischer, B.: Verifying multi-threaded software using SMT-based context-bounded model checking. In: Proc. ICSE. pp. 331–340. ACM (2011). https://doi.org/10.1145/1985793.1985839
48. Cordeiro, L.C., Kesseli, P., Kröning, D., Schrammel, P., Trtík, M.: JBmc: A bounded model checking tool for verifying Java bytecode. In: Proc. CAV. pp. 183–190. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_10

49. Cordeiro, L.C., Kröning, D., Schrammel, P.: Jbmc: Bounded model checking for Java bytecode (competition contribution). In: Proc. TACAS (3). pp. 219–223. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_17
50. Cordeiro, L.C., Morse, J., Nicole, D., Fischer, B.: Context-bounded model checking with Esbmc 1.17 (competition contribution). In: Proc. TACAS. pp. 534–537. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_42
51. Coto, A., Inverso, O., Sales, E., Tuosto, E.: A prototype for data race detection in CSeq 3 (competition contribution). In: Proc. TACAS (2). pp. 413–417. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_23
52. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C. In: Proc. SEFM. pp. 233–247. Springer (2012). https://doi.org/10.1007/978-3-642-33826-7_16
53. Dangl, M., Löwe, S., Wendler, P.: CPAchecker with support for recursive programs and floating-point arithmetic (competition contribution). In: Proc. TACAS. pp. 423–425. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_34
54. Darke, P., Agrawal, S., Venkatesh, R.: VeriAbs: A tool for scalable verification by abstraction (competition contribution). In: Proc. TACAS (2). pp. 458–462. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_32
55. Darke, P., Chimdyalwar, B., Agrawal, S., Venkatesh, R., Chakraborty, S., Kumar, S.: VeriAbsL: Scalable verification by abstraction and strategy prediction (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
56. Dietsch, D., Heizmann, M., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate Taipan 2023 (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
57. Dockins, R., Foltzer, A., Hendrix, J., Huffman, B., McNamee, D., Tomb, A.: Constructing semantic models of programs with the software analysis workbench. In: Proc. VSTTE. pp. 56–72. LNCS 9971, Springer (2016). https://doi.org/10.1007/978-3-319-48869-1_5
58. Dross, C., Furia, C.A., Huisman, M., Monahan, R., Müller, P.: Verifythis 2019: A program-verification competition. Int. J. Softw. Tools Technol. Transf. **23**(6), 883–893 (2021). https://doi.org/10.1007/s10009-021-00619-x
59. Ermis, E., Hoenicke, J., Podelski, A.: Splitting via interpolants. In: Proc. VMCAI. pp. 186–201. LNCS 7148, Springer (2012). https://doi.org/10.1007/978-3-642-27940-9_13
60. Ernst, G.: A complete approach to loop verification with invariants and summaries. Tech. Rep. arXiv:2010.05812v2, arXiv (January 2020). https://doi.org/10.48550/arXiv.2010.05812
61. Ernst, G.: Korn: Horn clause based verification of C programs (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
62. Farzan, A., Klumpp, D., Podelski, A.: Sound sequentialization for concurrent program verification. In: Proc. PLDI. pp. 506–521. ACM (2022). https://doi.org/10.1145/3519939.3523727
63. Gadelha, M.Y.R., Monteiro, F.R., Cordeiro, L.C., Nicole, D.A.: Esbmc v6.0: Verifying C programs using k-induction and invariant inference (competition contribution). In: Proc. TACAS (3). pp. 209–213. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_15
64. Gadelha, M.Y., Ismail, H.I., Cordeiro, L.C.: Handling loops in bounded model checking of C programs via k-induction. Int. J. Softw. Tools Technol. Transf. **19**(1), 97–114 (February 2017). https://doi.org/10.1007/s10009-015-0407-9

65. Gavrilenko, N., Ponce de León, H., Furbach, F., Heljanko, K., Meyer, R.: BMC for weak memory models: Relation analysis for compact SMT encodings. In: Proc. CAV. pp. 355–365. LNCS 11561, Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_19
66. Gerhold, M., Hartmanns, A.: Reproduction report for SV-COMP 2023. Tech. rep., University of Twente (2023). https://doi.org/10.48550/arXiv.2303.06477
67. Giesl, J., Mesnard, F., Rubio, A., Thiemann, R., Waldmann, J.: Termination competition (termCOMP 2015). In: Proc. CADE. pp. 105–108. LNCS 9195, Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_6
68. Greitschus, M., Dietsch, D., Podelski, A.: Loop invariants from counterexamples. In: Proc. SAS. pp. 128–147. LNCS 10422, Springer (2017). https://doi.org/10.1007/978-3-319-66706-5_7
69. Hajdu, Á., Micskei, Z.: Efficient strategies for CEGAR-based model checking. J. Autom. Reasoning **64**(6), 1051–1091 (2020). https://doi.org/10.1007/s10817-019-09535-x
70. He, F., Sun, Z., Fan, H.: Deagle: An SMT-based verifier for multi-threaded programs (competition contribution). In: Proc. TACAS (2). pp. 424–428. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_25
71. Heizmann, M., Barth, M., Dietsch, D., Fichtner, L., Hoenicke, J., Klumpp, D., Naouar, M., Schindler, T., Schüssele, F., Podelski, A.: Ultimate Automizer 2023 (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
72. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Proc. CAV. pp. 36–52. LNCS 8044, Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_2
73. Holík, L., Kotoun, M., Peringer, P., Šoková, V., Trtík, M., Vojnar, T.: Predator shape analysis tool suite. In: Hardware and Software: Verification and Testing. pp. 202–209. LNCS 10028, Springer (2016). https://doi.org/10.1007/978-3-319-49052-6
74. Howar, F., Jasper, M., Mues, M., Schmidt, D.A., Steffen, B.: The RERS challenge: Towards controllable and scalable benchmark synthesis. Int. J. Softw. Tools Technol. Transf. **23**(6), 917–930 (2021). https://doi.org/10.1007/s10009-021-00617-z
75. Howar, F., Mues, M.: GWit (competition contribution). In: Proc. TACAS (2). pp. 446–450. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_29
76. Hussein, S., Yan, Q., McCamant, S., Sharma, V., Whalen, M.: Java Ranger: Supporting string and array operations (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
77. Inverso, O., Tomasco, E., Fischer, B., La Torre, S., Parlato, G.: Lazy-CSeq: A lazy sequentialization tool for C (competition contribution). In: Proc. TACAS. pp. 398–401. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_29
78. Inverso, O., Tomasco, E., Fischer, B., Torre, S.L., Parlato, G.: Bounded verification of multi-threaded programs via lazy sequentialization. ACM Trans. Program. Lang. Syst. **44**(1), 1:1–1:50 (2022). https://doi.org/10.1145/3478536
79. Inverso, O., Trubiani, C.: Parallel and distributed bounded model checking of multi-threaded programs. In: Proc. PPoPP. pp. 202–216. ACM (2020). https://doi.org/10.1145/3332466.3374529
80. Journault, M., Miné, A., Monat, R., Ouadjaout, A.: Combinations of reusable abstract domains for a multilingual static analyzer. In: Proc. VSTTE. pp. 1–18. LNCS 12031, Springer (2019)

81. Kahsai, T., Rümmer, P., Sanchez, H., Schäf, M.: JayHorn: A framework for verifying Java programs. In: Proc. CAV. pp. 352–358. LNCS 9779, Springer (2016). https://doi.org/10.1007/978-3-319-41528-4_19
82. Kettl, M., Lemberger, T.: The static analyzer Infer in SV-COMP (competition contribution). In: Proc. TACAS (2). pp. 451–456. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_30
83. Klumpp, D., Dietsch, D., Heizmann, M., Schüssele, F., Ebbinghaus, M., Farzan, A., Podelski, A.: Ultimate GemCutter and the axes of generalization (competition contribution). In: Proc. TACAS (2). pp. 479–483. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_35
84. Kröning, D., Tautschnig, M.: Cbmc: C bounded model checker (competition contribution). In: Proc. TACAS. pp. 389–391. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_26
85. Lauko, H., Ročkai, P., Barnat, J.: Symbolic computation via program transformation. In: Proc. ICTAC. pp. 313–332. Springer (2018). https://doi.org/10.1007/978-3-030-02508-3_17
86. Leeson, W., Dwyer, M.: Graves-CPA: A graph-attention verifier selector (competition contribution). In: Proc. TACAS (2). pp. 440–445. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_28
87. Luckow, K.S., Dimjasevic, M., Giannakopoulou, D., Howar, F., Isberner, M., Kahsai, T., Rakamaric, Z., Raman, V.: JDart: A dynamic symbolic analysis framework. In: Proc. TACAS. pp. 442–459. LNCSS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_26
88. Malík, V., Schrammel, P., Vojnar, T., Nečas, F.: 2LS: Arrays and loop unwinding (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
89. Metta, R., Medicherla, R.K., Chakraborty, S.: BMC+Fuzz: Efficient and effective test generation. In: Proc. DATE. pp. 1419–1424. IEEE (2022). https://doi.org/10.23919/DATE54114.2022.9774672
90. Metta, R., Yeduru, P., Karmarkar, H., Medicherla, R.K.: VeriFuzz 1.4: Checking for (non-)termination (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
91. Monat, R., Ouadjaout, A., Miné, A.: Mopsa-C: Modular domains and relational abstract interpretation for C programs (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
92. Mues, M., Howar, F.: JDart: Portfolio solving, breadth-first search and SMT-Lib strings (competition contribution). In: Proc. TACAS (2). pp. 448–452. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_30
93. Mues, M., Howar, F.: GDart (competition contribution). In: Proc. TACAS (2). pp. 435–439. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_27
94. Noller, Y., Păsăreanu, C.S., Le, X.B.D., Visser, W., Fromherz, A.: Symbolic Pathfinder for SV-COMP (competition contribution). In: Proc. TACAS (3). pp. 239–243. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_21
95. Nutz, A., Dietsch, D., Mohamed, M.M., Podelski, A.: Ultimate Kojak with memory safety checks (competition contribution). In: Proc. TACAS. pp. 458–460. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_44
96. Peringer, P., Šoková, V., Vojnar, T.: PredatorHP revamped (not only) for interval-sized memory regions and memory reallocation (competition contribution). In: Proc. TACAS (2). pp. 408–412. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_30

97. Ponce-De-Leon, H., Haas, T., Meyer, R.: Dartagnan: Leveraging compiler optimizations and the price of precision (competition contribution). In: Proc. TACAS (2). pp. 428–432. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_26
98. Ponce-De-Leon, H., Haas, T., Meyer, R.: Dartagnan: Smt-based violation witness validation (competition contribution). In: Proc. TACAS (2). pp. 418–423. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_24
99. Pratikakis, P., Foster, J.S., Hicks, M.: Locksmith: Practical static race detection for C. ACM Trans. Program. Lang. Syst. **33**(1) (January 2011). https://doi.org/10.1145/1889997.1890000
100. Păsăreanu, C.S., Visser, W., Bushnell, D.H., Geldenhuys, J., Mehlitz, P.C., Rungta, N.: Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Autom. Software Eng. **20**(3), 391–425 (2013). https://doi.org/10.1007/s10515-013-0122-2
101. Richter, C., Hüllermeier, E., Jakobs, M.C., Wehrheim, H.: Algorithm selection for software validation based on graph kernels. Autom. Softw. Eng. **27**(1), 153–186 (2020). https://doi.org/10.1007/s10515-020-00270-x
102. Richter, C., Wehrheim, H.: PeSCo: Predicting sequential combinations of verifiers (competition contribution). In: Proc. TACAS (3). pp. 229–233. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_19
103. Saan, S., Schwarz, M., Erhard, J., Pietsch, M., Seidl, H., Tilscher, S., Vojdani, V.: Goblint: Autotuning thread-modular abstract interpretation (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
104. Scott, R., Dockins, R., Ravitch, T., Tomb, A.: Crux: Symbolic execution meets SMT-based verification (competition contribution). Zenodo (February 2022). https://doi.org/10.5281/zenodo.6147218
105. Shamakhi, A., Hojjat, H., Rümmer, P.: Towards string support in JayHorn (competition contribution). In: Proc. TACAS (2). pp. 443–447. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_29
106. Sharma, V., Hussein, S., Whalen, M.W., McCamant, S.A., Visser, W.: Java Ranger: Statically summarizing regions for efficient symbolic execution of Java. In: Proc. ESEC/FSE. pp. 123–134. ACM (2020). https://doi.org/10.1145/3368089.3409734
107. Su, J., Yang, Z., Xing, H., Yang, J., Tian, C., Duan, Z.: PIChecker: A POR and interpolation-based verifier for concurrent programs (competition contribution). In: Proc. TACAS (2). LNCS 13994, Springer (2023)
108. Tóth, T., Hajdu, A., Vörös, A., Micskei, Z., Majzik, I.: Theta: A framework for abstraction refinement-based model checking. In: Proc. FMCAD. pp. 176–179 (2017). https://doi.org/10.23919/FMCAD.2017.8102257
109. Visser, W., Geldenhuys, J.: Coastal: Combining concolic and fuzzing for Java (competition contribution). In: Proc. TACAS (2). pp. 373–377. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_23
110. Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: Static race detection for device drivers: The Goblint approach. In: Proc. ASE. pp. 391–402. ACM (2016). https://doi.org/10.1145/2970276.2970337
111. Volkov, A.R., Mandrykin, M.U.: Predicate abstractions memory modeling method with separation into disjoint regions. Proceedings of the Institute for System Programming (ISPRAS) **29**, 203–216 (2017). https://doi.org/10.15514/ISPRAS-2017-29(4)-13
112. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.16. Zenodo (2023). https://doi.org/10.5281/zenodo.7612021

113. Wu, T., Schrammel, P., Cordeiro, L.: Wit4Java: A violation-witness validator for Java verifiers (competition contribution). In: Proc. TACAS (2). pp. 484–489. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_36
114. Ádám, Z., Bajczi, L., Dobos-Kovács, M., Hajdu, A., Molnár, V.: Theta: Portfolio of cegar-based analyses with dynamic algorithm selection (competition contribution). In: Proc. TACAS (2). pp. 474–478. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_34
115. J. Švejda, Berger, P., Katoen, J.P.: Interpretation-based violation witness validation for C: NitWit. In: Proc. TACAS. pp. 40–57. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_3