State of the Art in Software Verification and Witness Validation: SV-COMP 2024

Dirk Beyer

LMU Munich, Munich, Germany

Abstract. The 13th edition of the Competition on Software Verification (SV-COMP 2024) was the largest competition of its kind so far: A total of 76 tools for verification and witness validation were compared. The competition evaluated 59 verification systems and 17 validation systems from 34 teams from 12 countries. This yields a good overview of the state of the art in tools for software verification. The competition was executed on a benchmark set with 30 300 verification tasks for C programs and 587 verification tasks for Java programs. The specifications again included reachability, memory safety, overflows, and termination. This year was the second time that the competition had an extra competition track on witness validation. We introduced a new witness format 2.0, and a new scoring schema for the validation track. All meta data about the verification and validation tools are available in the FM-Tools repository.

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

1 Introduction

This report describes the results of the 2024 edition of SV-COMP, and is an extension of the series of competition reports (see footnote). We also list important processes and rules, and give insights into some aspects of the competition. The 13th Competition on Software Verification (`https://sv-comp.sosy-lab.org/2024`) is again the largest comparative evaluation ever in this area. The objectives of the competitions were discussed earlier (1-4 [22]) and extended over the years (5-6 [23]):

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,
3. establish standards that make it possible to compare different verification tools, including a property language and formats for the results,

This report extends previous reports on SV-COMP [16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27]. Reproduction packages are available on Zenodo (see Table 3).

✉ `dirk.beyer@sosy-lab.org`

B. Finkbeiner and L. Kovács (Eds.): TACAS 2024, LNCS 14572, pp. 299–329, 2024.
https://doi.org/10.1007/978-3-031-57256-2_15

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,
6. provide research teams that do not have sufficient computing resources with the opportunity to obtain experimental results on large benchmark sets, and
7. conserve tools for formal methods for later reuse by using a standardized format to announce archives (via DOIs), default options, contacts, competition participations, and other meta data in a central repository.

The SV-COMP 2020 report [23] discusses the achievements of the SV-COMP competition so far with respect to these objectives.

Related Competitions. SV-COMP is one of many competitions that measure progress of research in the are of formal methods [15]. Competitions can lead to fair and accurate comparative evaluations because of the involvement of the developing teams. The competitions most related to SV-COMP are RERS [80], VerifyThis [65], Test-Comp [28], and TermCOMP [73]. A previous report [23] provides a more detailed discussion.

Quick Summary of Changes. While we try to keep the setup of the competition stable, there are always improvements and developments. For the 2024 edition, the following changes were made:

- New verification tasks were added, with an increase in C from 23 805 in 2023 to 30 300 in 2024.
- Tool archives are now uploaded to Zenodo, instead of GitLab, and the meta data about the tools are hosted and maintained in the Repository for Formal-Methods Tools (`https://gitlab.com/sosy-lab/benchmarking/fm-tools`).
- The improved witness format version 2.0 [7] (which is based on YAML instead of GraphML) was used for the first time.
- The scoring schema for the witness validators [44] was changed based on the 2023 community meeting in Paris.

2 Organization, Definitions, Formats, and Rules

Procedure. The overall organization of the competition did not change in comparison to the earlier editions [16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27]. 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

Table 1: Scoring schema for SV-COMP 2024 (unchanged from 2021 [24])

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)

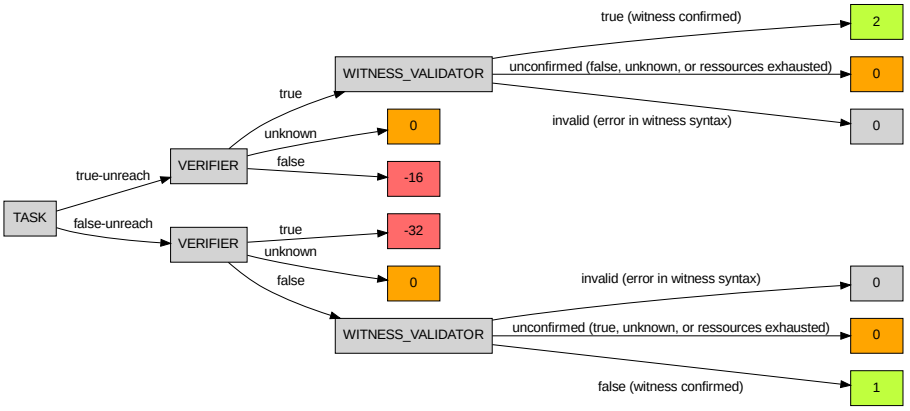

Fig. 1: Visualization of the scoring schema for the reachability property (unchanged from 2021 [24])

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 [17]). The tasks of the jury were described in more detail in the report of SV-COMP 2022 [26]. The team representatives of the competition jury are listed in Table 5.

Scoring Schema and Ranking. The scoring schema of SV-COMP 2024 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 [17, 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),

Table 2: Publicly available components for reproducing SV-COMP 2024

Component	Fig. 3	Repository	Version
Verification Tasks	(a)	gitlab.com/sosy-lab/benchmarking/sv-benchmarks	svcomp24
Benchmark Definitions	(b)	gitlab.com/sosy-lab/sv-comp/bench-defs	svcomp24
Tool-Info Modules	(c)	github.com/sosy-lab/benchexec	3.21
Verifiers	(d)	gitlab.com/sosy-lab/benchmarking/fm-tools	svcomp24
Benchmarking	(e)	github.com/sosy-lab/benchexec	3.21
Witness Format	(f)	gitlab.com/sosy-lab/benchmarking/sv-witnesses	2.0.2
Continuous Integration		gitlab.com/sosy-lab/software/coveriteam	1.1

Table 3: Artifacts published for SV-COMP 2024

Content	DOI	Reference
Verification Tasks	10.5281/zenodo.10669723	[31]
Competition Results	10.5281/zenodo.10669731	[30]
Verifiers and Validators	10.5281/zenodo.10669735	[29]
Verification Witnesses	10.5281/zenodo.10669737	[32]
BenchExec	10.5281/zenodo.10671136	[122]
CoVeriTeam	10.5281/zenodo.10843666	[45]

(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.

Task-Definition Format 2.0. SV-COMP 2024 used the task-definition format in version 2.0. More details can be found in the report for Test-Comp 2021 [25].

Properties. Please see the 2015 competition report [19] for the definition of the properties and the property format. All specifications used in SV-COMP 2024 are available in the directory `c/properties/` of the benchmark repository.

Categories. The community significantly extended the benchmark set for SV-COMP 2024. The (updated) category structure of SV-COMP 2024 is shown in Fig. 2. We refer to the previous reports for a description and mention only the changes here: Compared to SV-COMP 2023, we added two new sub-categories *ReachSafety-Hardness* and *ReachSafety-Fuzzle* to main category *ReachSafety*. We restructured main category *SoftwareSystems* as follows: We removed sub-categories *SoftwareSystems-BusyBox-ReachSafety*, *SoftwareSystems-BusyBox-MemSafety*, and *SoftwareSystems-OpenBSD-MemSafety*, and added sub-categories *SoftwareSystems-coreutils-MemSafety*, *SoftwareSystems-coreutils-NoOverflows*, *SoftwareSystems-Other-ReachSafety*, and *SoftwareSystems-Other-MemSafety*. 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/2024/benchmarks.php`).

Reproducibility. SV-COMP results must be reproducible, and consequently, all major components are maintained in public version-control repositories. The

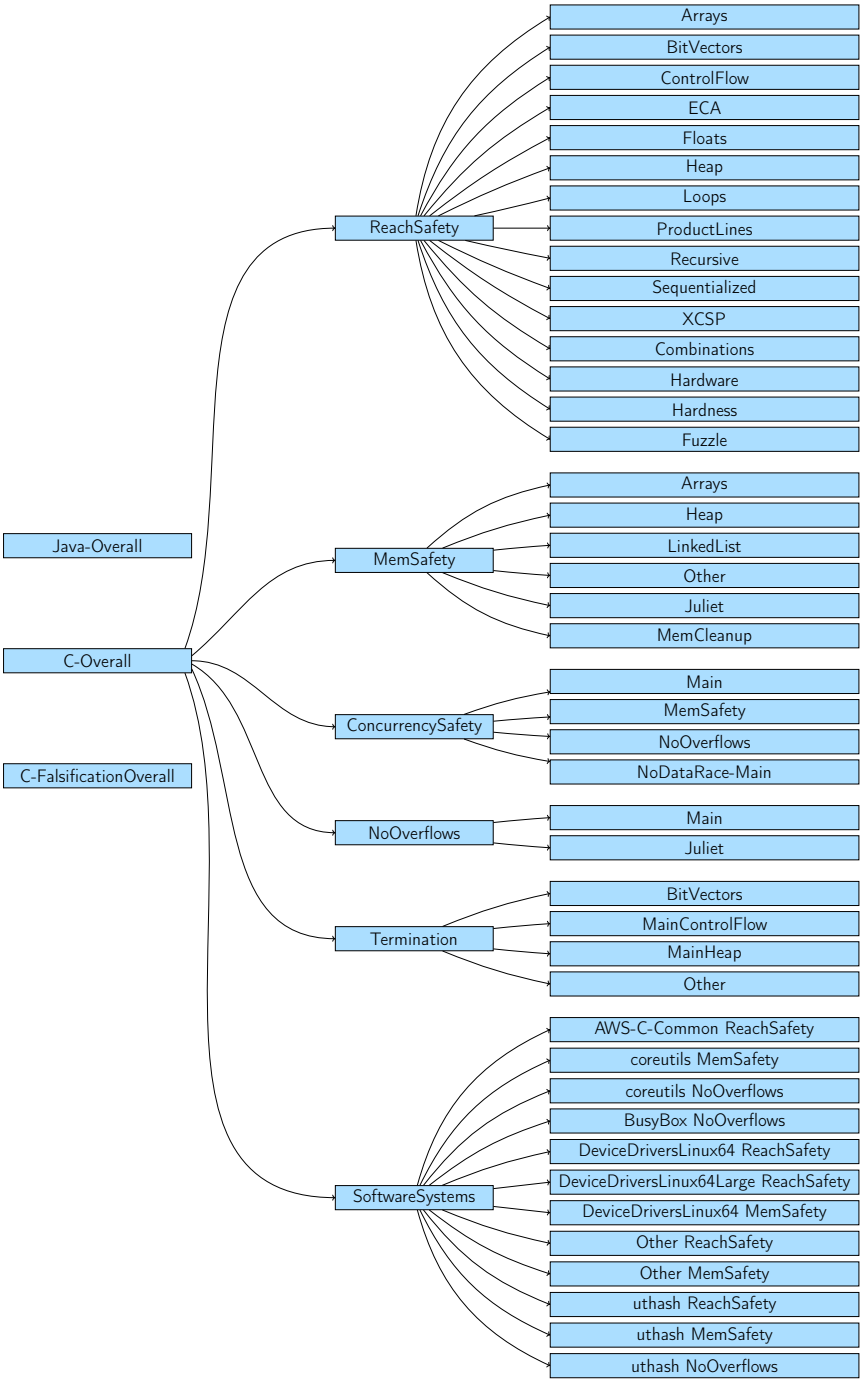

Fig. 2: Category structure for SV-COMP 2024 (changed from 2023)

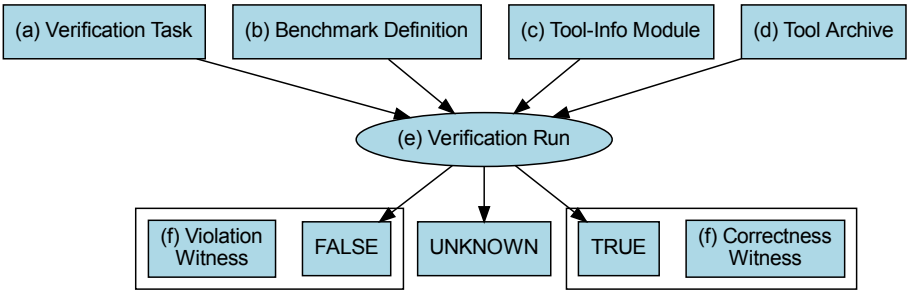

Fig. 3: Benchmarking components of SV-COMP and competition's execution flow (same as for SV-COMP 2020, except that we now download the tool archives from Zenodo instead of GitLab)

Table 4: Validation: Witness validators and witness linter

Validator	Reference	Jury Member	Affiliation
ConcurWitness2Test new	[13]	L. Bajczi	BME Budapest, Hungary
CPAchecker	[33, 34, 36]	D. Baier	LMU Munich, Germany
CPA-witness2test	[35]	T. Lemberger	LMU Munich, Germany
Dartagnan	[106]	H. Ponce de León	Huawei Dresden, Germany
CProver-witness2test ∅	[35]	(hors concours)	–
Goblint new	[112]	S. Saan	U. of Tartu, Estonia
GWIT	[81]	F. Howar	TU Dortmund, Germany
JCWIT new		Z. Cheng	U. of Manchester, UK
LIV new	[43]	M. Spiessl	LMU Munich, Germany
MetaVal	[41]	M. Spiessl	LMU Munich, Germany
Mopsa new	[99]	R. Monat	Inria and U. of Lille, France
NITWIT	[124]	J. (P.) Berger	RWTH Aachen, Germany
Symbiotic-Witch	[8]	P. Ayaziová	Masaryk U., Brno, Czechia
UAutomizer	[33, 34]	M. Heizmann	U. of Freiburg, Germany
WIT4JAVA ∅	[123]	(hors concours)	–
Witch new	[7, 9]	P. Ayaziová	Masaryk U., Brno, Czechia
WitnessLint	[7]	M. Lingsch-Rosenfeld	LMU Munich, Germany

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 [20] 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.

Competition Workflow. The workflow of the competition is described in the report for Test-Comp 2021 [25] (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 2022 report [26, Sect. 3].

Table 5: Verification: Participating verifiers with tool references and representing jury members; new for first-time, ∅ for hors-concours (RELAY-SV new was not able to qualify)

Participant	Ref.	Jury member	Affiliation
2LS	[46, 96]	V. Malík	BUT, Czechia
AISE new	[121]	Z. Chen	NUDT, China
BRICK	[47]	L. Bu	Nanjing U., China
BUBAAK	[49]	M. Chalupa	ISTA, Austria
BUBAAK-SPLIT new	[50]	M. Chalupa	ISTA, Austria
CBMC∅	[54, 91]	(h. c.)	–
COASTAL∅	[118]	(h. c.)	–
COVERITEAM-ALGSEL∅	[37, 38]	(h. c.)	–
COVERITEAM-PARPORT∅	[37, 38]	(h. c.)	–
CPACHECKER	[10, 39]	D. Baier	LMU Munich, Germany
CPALOCKATOR∅	[5, 6]	(h. c.)	–
CPA-BAM-BNB∅	[4, 120]	(h. c.)	–
CPA-BAM-SMG∅		(h. c.)	–
CPV new	[53]	P.-C. Chien	LMU Munich, Germany
CRUX∅	[64, 113]	(h. c.)	–
CSEQ∅	[59, 85]	(h. c.)	–
DARTAGNAN	[71, 105]	H. Ponce de León	Huawei Dresden, Germany
DEAGLE	[76]	F. He	Tsinghua U., China
DIVINE∅	[14, 92]	(h. c.)	–
EBF	[3]	F. Aljaafari	U. of Manchester, UK
EMERGENTHETA new	[11]	L. Bajczi	BME Budapest, Hungary
ESBMC-INCR∅	[55, 58]	(h. c.)	–
ESBMC-KIND	[70, 97]	F. Brauße	U. Manchester, UK
FRAMA-C-SV	[42, 60]	M. Spiessl	LMU Munich, Germany
GAZER-THETA∅	[1, 75]	(h. c.)	–
GDART	[101]	F. Howar	TU Dortmund, Germany
GDART-LLVM∅		(h. c.)	–
GOBLINT	[111, 119]	S. Saan	U. Tartu, Estonia
GRAVES-CPA∅	[93]	(h. c.)	–
GRAVES-PAR∅		(h. c.)	–
INFER∅	[48, 89]	(h. c.)	–
JAVA-RANGER∅	[82, 115]	(h. c.)	–
JAYHORN	[88, 114]	H. Mousavi	U. Tehran, TIAS, Iran
JBMC	[56, 57]	P. Schrammel	U. Sussex / Diffblue, UK
JDART∅	[95, 100]	(h. c.)	–
KORN	[67, 68]	G. Ernst	LMU Munich, Germany
LAZY-CSEQ∅	[83, 84]	(h. c.)	–
LF-CHECKER∅		(h. c.)	–
LOCKSMITH∅	[107]	(h. c.)	–
MLB		L. Bu	Nanjing U., China
MOPSA	[87, 99]	R. Monat	Inria and U. Lille, France

(continues on next page)

Table 5: Competition candidates (continued)

Participant	Ref.	Jury member	Affiliation
PeSCo-CPA ∅	[109, 110]	(h. c.)	–
PIChecker ∅	[116]	(h. c.)	–
Pinaka ∅	[52]	(h. c.)	–
PredatorHP	[79, 104]	V. Šoková	BUT, Czechia
Proton new	[98]	R. Metta	TCS, India
SPF ∅	[102, 108]	(h. c.)	–
sv-sanitizers new		S. Saan	U. of Tartu, Estonia
SWAT new	[94]	N. Loose	U. of Luebeck, Germany
Symbiotic	[51, 86]	M. Jonáš	Masaryk U., Czechia
Theta	[12, 117]	L. Bajczi	BME Budapest, Hungary
UAutomizer	[77, 78]	M. Heizmann	U. Freiburg, Germany
UGemCutter	[69, 90]	D. Klumpp	U. Freiburg, Germany
UKojak	[66, 103]	F. Schüssele	U. Freiburg, Germany
UTaipan	[63, 74]	D. Dietsch	U. Freiburg, Germany
VeriAbs	[2, 61]	P. Darke	TCS, India
VeriAbsL	[62]	P. Darke	TCS, India
VeriOover ∅		(h. c.)	–

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				✓	✓			✓	✓		✓						✓			
AISE new			✓																	
BRICK	✓		✓	✓				✓								✓				
Bubaak			✓								✓					✓	✓			✓
Bubaak-SpLit new			✓		✓						✓				✓	✓	✓		✓	✓
CBMC ∅				✓							✓					✓				
COASTAL ∅			✓																	
CVT-AlgoSel ∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CVT-ParPort ∅	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
CPAchecker	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CPALockator ∅	✓	✓					✓				✓	✓	✓	✓		✓				

(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
CPA-BAM-BnB[⌀]	✓	✓					✓				✓	✓	✓	✓						
CPA-BAM-SMG[⌀]																				
CPV[new]				✓	✓	✓								✓						
Crux[⌀]			✓																	
CSeq[⌀]				✓							✓					✓				
Dartagnan				✓							✓					✓				
Deagle				✓												✓				
DIVINE[⌀]			✓				✓				✓					✓			✓	✓
EBF				✓																
EmergenTheta[new]				✓	✓						✓	✓		✓					✓	✓
ESBMC-incr[⌀]				✓	✓						✓					✓				
ESBMC-kind				✓	✓		✓	✓			✓					✓				
Frama-C-SV								✓												
Gazer-Theta[⌀]	✓	✓		✓			✓				✓	✓	✓	✓						✓
GDart			✓								✓									✓
GDart-LLVM[⌀]			✓								✓									
Goblint								✓								✓			✓	
Graves-CPA[⌀]	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Graves-Par[⌀]																				
Infer[⌀]								✓	✓	✓										✓
Java-Ranger[⌀]			✓								✓									
JayHorn	✓	✓				✓		✓					✓	✓						
JBMC				✓							✓					✓				
JDart[⌀]			✓								✓									✓
Korn		✓	✓				✓													✓
Lazy-CSeq[⌀]				✓							✓					✓				
LF-checker[⌀]																				
Locksmith[⌀]																✓				
MLB			✓								✓									✓
Mopsa								✓												
PeSCo-CPA[⌀]	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
PIChecker[⌀]	✓	✓									✓	✓		✓		✓			✓	
Pinaka[⌀]			✓	✓							✓									
PredatorHP									✓											

(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
PROTON [new]				✓																
SPF [∅]			✓						✓							✓				
SV-SANITIZERS [new]																✓				
SWAT [new]			✓																	
SYMBIOTIC			✓		✓			✓	✓		✓					✓				✓
THETA	✓	✓					✓				✓	✓		✓		✓			✓	✓
UAUTOMIZER	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓
UGEMCUTTER	✓	✓									✓		✓	✓	✓	✓			✓	✓
UKOJAK	✓	✓									✓		✓	✓						
UTAIPAN	✓	✓					✓	✓			✓		✓	✓	✓	✓			✓	✓
VERIABS	✓			✓	✓		✓	✓										✓	✓	✓
VERIABSL	✓			✓	✓		✓	✓										✓	✓	✓
VERIOOVER [∅]																				

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		✓									✓	
AISE [new]												
BRICK										✓	✓	
BUBAAK										✓		
BUBAAK-SPLIT [new]												
CBMC [∅]		✓									✓	
COASTAL [∅]				✓								
CVT-ALGOSEL [∅]	✓	✓	✓		✓	✓	✓				✓	
CVT-PARPORT [∅]	✓	✓	✓		✓	✓	✓				✓	
CPACHECKER	✓					✓	✓					✓

(continues on next page)

Table 7: Solver libraries and frameworks (continued)

Verifier	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc4	SMTinterpol	z3	MiniSAT	Apron
CPALockator[∅]	✓					✓	✓					
CPA-BAM-BnB[∅]	✓					✓	✓					
CPA-BAM-SMG[∅]	✓					✓	✓					
Crux[∅]										✓		
CSeq[∅]		✓									✓	
Dartagnan						✓						
Deagle											✓	
DIVINE[∅]												
EBF			✓				✓					
EmergenTheta [new]												
ESBMC-incr[∅]			✓				✓					
ESBMC-kind			✓				✓					
Frama-C-SV												
Gazer-Theta[∅]												
GDart								✓		✓		
GDart-LLVM[∅]										✓		
Goblint												✓
Graves-CPA[∅]	✓					✓	✓					
Graves-Par[∅]												
Infer[∅]												
Java-Ranger[∅]				✓								
JayHorn												
JBMC		✓									✓	
JDart[∅]				✓				✓		✓		
Korn										✓		
Lazy-CSeq[∅]		✓									✓	
LF-checker[∅]												
Locksmith[∅]												
MLB												
Mopsa												✓
PeSCo-CPA[∅]	✓					✓	✓					
PIChecker[∅]	✓					✓	✓		✓			
Pinaka[∅]												
PredatorHP												
Proton [new]												
SPF[∅]				✓								
sv-sanitizers [new]												

(continues on next page)

Table 7: Solver libraries and frameworks (continued)

Verifier	CPAchecker	CPROVER	ESBMC	JPF	ULTIMATE	JAVASMT	MATHSAT	CVC4	SMTINTERPOL	Z3	MINISAT	APRON
SWAT [new]												
SYMBIOTIC										✓		
THETA												
UAUTOMIZER					✓		✓	✓	✓	✓		
UGEMCUTTER					✓		✓	✓	✓	✓		
UKOJAK					✓				✓	✓		
UTAIPAN					✓		✓	✓	✓	✓		
VERIABS	✓	✓								✓	✓	
VERIABSL	✓	✓								✓	✓	
VERIOOVER[⌀]												

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/2024/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 17 validation tools (16 proper witness validators, and one 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

Table 8: Verification: Quantitative overview over all regular results; empty cells are used for opt-outs; new for first-time participants; the number of tasks includes invalid tasks that were excluded from scoring by the jury (details available on web site or in artifact)

Participant	**ReachSafety** 17746 points 11305 tasks	**MemSafety** 3216 points 2135 tasks	**ConcurrencySafety** 5672 points 3259 tasks	**NoOverflows** 13044 points 8188 tasks	**Termination** 4000 points 2354 tasks	**SoftwareSystems** 5251 points 3813 tasks	**FalsificationOverall** 8817 points 28700 tasks	**Overall** 49097 points 31054 tasks	**JavaOverall** 828 points 587 tasks
2LS	6000	224	0	5976	**1584**	10	1311	10564	
AISE new									
BRICK									
BUBAAK	3788	1890	11	6465	1481	-1082	-617	12206	
BUBAAK-SPLIT new	4692	1312	7	-41374	661	**872**	1959	-18177	
CPACHECKER	**10084**	1897	2029	**8603**	1195	**784**	**4812**	**21568**	
CPV new	6330								
DARTAGNAN			**3547**						
DEAGLE									
EBF			636						
EMERGENTHETA new	1178								
ESBMC-KIND	8364	2077	1853	8272	1048	-1063	2394	17896	
FRAMA-C-SV				1098					
GDART									**616**
GOBLINT	2289	1304	2583	7059	890	536		15458	
JAYHORN									325
JBMC									**618**
KORN									
MLB									**676**
MOPSA	2241	1516		8063		**2197**			
PREDATORHP		**2321**							
PROTON new					**3526**				
SV-SANITIZERS new		290							
SWAT new									566
SYMBIOTIC	7052	**2156**	238	7370	1258	687	**4050**	17192	
THETA	2119		2354						
UAUTOMIZER	6320	**2110**	**3079**	**9497**	**3248**	261	3139	**26396**	
UGEMCUTTER			**3189**						
UKOJAK	4869	1400	0	7363	0	233	2291	10593	
UTAIPAN	5751	2014	2655	**9231**	0	351	**3157**	**18042**	
VERIABS	**10541**								
VERIABSL	**10735**								

Table 9: Verification: Quantitative overview over all hors-concours results; empty cells represent opt-outs, ∅ for hors-concours participation; the number of tasks includes invalid tasks that were excluded from scoring by the jury (details available on web site or in artifact)

Participant	**ReachSafety** 17746 points 11305 tasks	**MemSafety** 3216 points 2135 tasks	**ConcurrencySafety** 5672 points 3259 tasks	**NoOverflows** 13044 points 8188 tasks	**Termination** 4000 points 2354 tasks	**SoftwareSystems** 5251 points 3813 tasks	**FalsificationOverall** 8817 points 28700 tasks	**Overall** 49097 points 31054 tasks	**JavaOverall** 828 points 587 tasks
CBMC∅	1269	1330	1229	5771	1125	-2569	-3764	8391	
COASTAL∅									-2752
CVT-AlgoSel∅	2635		41						
CVT-ParPort∅	-6152	1655	911	-17812	1289	-1297	-9118	-7545	
CPA-BAM-BnB∅						-2439			
CPA-BAM-SMG∅		2039				-2804			
CPALockator∅			-4924						
Crux∅	2066			490					
CSeq∅			-12478						
DIVINE∅	4655	298	390	0	0	76	256	3576	
ESBMC-incr∅			542						
Gazer-Theta∅									
GDart-LLVM∅									
Graves-CPA∅	3831					-322	-1538	5470	
Graves-Par∅	876	1627	53	-17650	1256	-2037	-9024	-6731	
Infer∅	-99128		-8289	-73312		-24917			
Java-Ranger∅									398
JDart∅									382
Lazy-CSeq∅			-15024						
LF-checker∅			772						
Locksmith∅									
PeSCo-CPA∅	5814					-76	3247	17315	
PIChecker∅			521						
Pinaka∅	2418			1337	855				
SPF∅									182
VeriOover∅									

and correctness of the results, as accumulated in the score) and the efficiency (resource consumption in terms of CPU time). The results are presented in the same way as in last years, such that the improvements compared to the last years are easy to identify. The results presented in this report were inspected and approved by the participating teams.

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

Rank	Verifier	Score	CPU Time (in h)	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
ReachSafety							
1	**VeriAbsL**	**10735**	190	7 075	1 138		**2**
2	VeriAbs	10541	190	6 720	1 032		**1**
3	CPAchecker	10084	200	6 468	286	2	
MemSafety							
1	**PredatorHP**	**2321**	1.2	1 823	3	3	
2	Symbiotic	2156	0.77	1 855	0		**5**
3	UAutomizer	2110	62	1 637	4		
ConcurrencySafety							
1	**Dartagnan**	**3547**	14	2 086	0		**5**
2	UGemCutter	3189	32	1 851	4	1	
3	UAutomizer	3079	28	1 791	3		**1**
NoOverflows							
1	**UAutomizer**	**9497**	62	4 532	2		
2	UTaipan	9231	66	4 420	11		**1**
3	CPAchecker	8603	18	5 596	192		
Termination							
1	**Proton** new	**3526**	19	1 888	126	1	
2	UAutomizer	3248	18	1 631	11		
3	2ls	1584	4.2	1 167	201		
SoftwareSystems							
1	**Mopsa**	**2197**	15	2 030	0		
2	Bubaak-SpLit new	872	0.42	480	163	8	
3	CPAchecker	784	43	1 756	71		
FalsificationOverall							
1	**CPAchecker**	**4812**	91	4 920	218	10	
2	Symbiotic	4050	27	4 281	191	11	
3	UTaipan	3157	33	1 602	34	1	
Overall							
1	**UAutomizer**	**26396**	290	13 617	114	3	**7**
2	CPAchecker	21568	320	17 968	698	16	**1**
3	UTaipan	18042	240	11 524	71	1	**13**
JavaOverall							
1	**MLB**	**676**	0.93	484	34		
2	JBMC	618	0.44	424	80		
3	GDart	616	2.6	453	9		

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

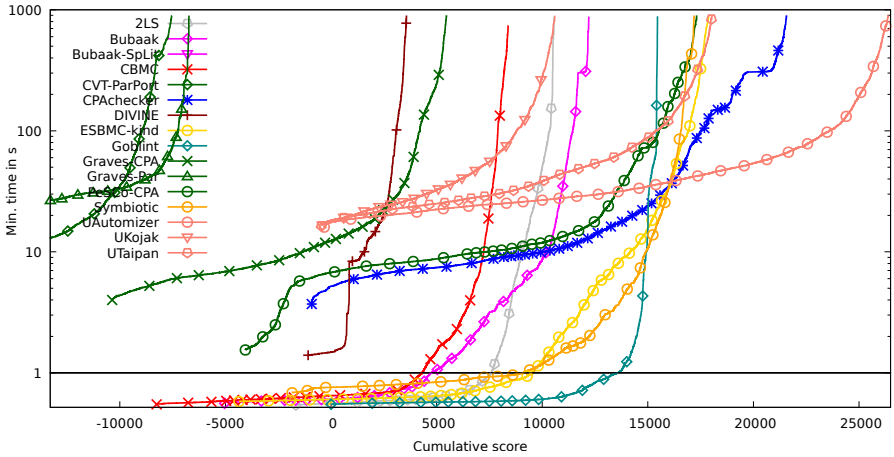

Fig. 4: Quantile functions for category *C-Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by correct verification runs below a certain run time (y-coordinate). More details were given previously [17]. A logarithmic scale is used for the time range from 1 s to 1000 s, and a linear scale is used for the time range between 0 s and 1 s.

tasks. The verification tasks consist of tasks with expected verdict TRUE, expected verdict FALSE, and tasks that are VOID (tasks that were excluded from scoring by the jury). 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; DEAGLE was disqualified by the jury, with details on the web site). 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/2024/results`) and in the results artifact (see Table 3).

Table 10 reports the top three verifiers for each category. The run time (column 'CPU Time') refers 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 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 [17, 40] 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/2024/results`) include such a plot for each (sub-)category. As an example, we show the plot for category

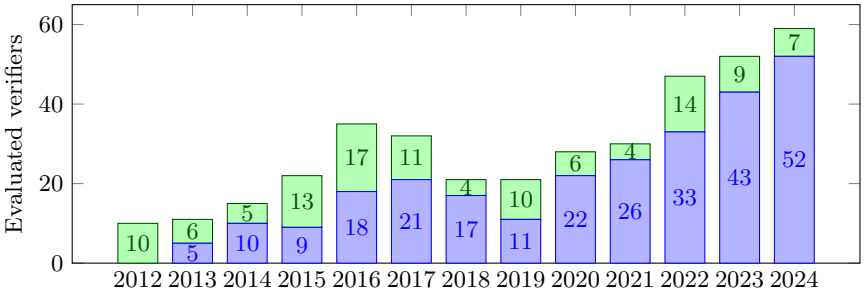

Fig. 5: Number of evaluated verifiers for each year (first-time participants on top)

Table 11: New verifiers in SV-COMP 2023 and SV-COMP 2024; column 'Sub-categories' gives the number of executed categories; new for first-time participants in 2024; ∅ for those that were hors-concours participants in 2024

Verifier	Language	First Year	Sub-categories
AISE new	C	2024	1
BUBAAK-SPLIT new	C	2024	45
CPV new	C	2024	20
EMERGENTHETA new	C	2024	15
PROTON new	C	2024	5
SV-SANITIZERS new	C	2024	13
SWAT new	Java	2024	1
BUBAAK	C	2023	40
GDART-LLVM ∅	C	2023	1
GRAVES-PAR ∅	C	2023	40
LF-CHECKER ∅	C	2023	3
MOPSA	C	2023	32
PICHECKER ∅	C	2023	1
VERIABSL	C	2023	13
VERIOOVER ∅	C	2023	1
MLB	Java	2023	1

C-Overall (all verification tasks) in Fig. 4. A total of 16 verifiers participated in category *C-Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [17]). 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 [17, 20].

The winner of the competition, UAUTOMIZER, achieves the best cumulative score (the graph for UAUTOMIZER has the longest width from its left to its right end; the graph starts left from $x = 0$ because the verifier produced 7 wrong proofs and 4 false alarms and therefore received some negative points). Also other 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 2023 or SV-COMP 2024). Figure 5 shows the growing interest in the competition over the years.

Computing Resources. The CPU time and memory limits were the same as in the previous competitions [20] (15 GB of memory and 15 min of CPU time), but we reduced the number of processing units per run from 8 to 4 processing units. This has the disadvantage that the measurements are more imprecise due to shared resources in the machine, but it roughly doubles the throughput. This change was necessary because of the ever increasing number of participating systems and the continuously increasing benchmark set. Witness validation was again 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 compute cluster at the SoSy-Lab at LMU that consists of 168 machines, where each machine has 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 [40] to measure and control computing resources (CPU time, memory) and VCloud to distribute, install, run, and clean-up verification runs, and to collect the results. The values for the time are accumulated over all cores of the CPU.

To give an impression of the overall computation work, we report some statistics: One complete verification execution of the competition consisted of 787 779 verification runs (each verifier on each verification task of the selected categories according to the opt-outs), consuming 2 104 days of CPU time (without validation). This is almost double the CPU time spent for the previous edition of SV-COMP. Witness-based result validation required 13.6 million validation runs in 21 243 run sets (each validator on each verification task for categories with witness validation, and for each verifier), consuming 2290 days of CPU time. Each tool was executed several times, in order to make sure no installation issues occur during the execution.

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 [44] suggested that validators for verification results should also undergo a periodical comparative evaluation and proposed a scoring schema for witness-validation results. SV-COMP 2024 evaluated a total of 17 validators on 100 998 correctness and 71 577 violation witnesses in format 1.0, and 45 614 correctness and 27 561 violation witnesses in format 2.0. Figure 6 shows the growing importance of evaluating witness validators.

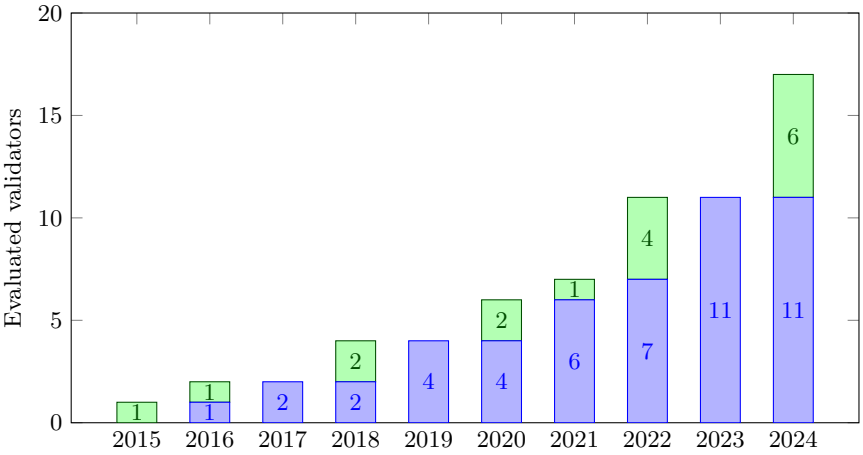

Fig. 6: Number of witness validators for each year (first-time participants on top)

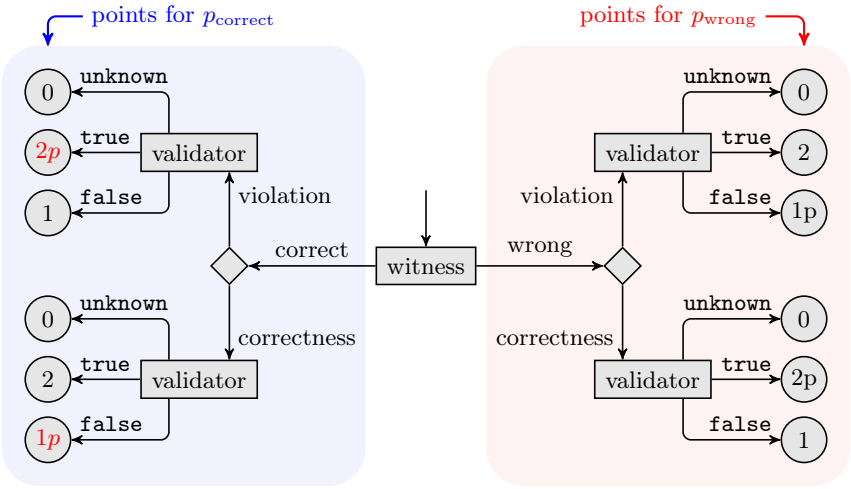

Fig. 7: Scoring schema for evaluation of validators; $p = -16$ for SV-COMP 2024; figure adopted from [44]; changed scores compared to 2023 are highlighted in red

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

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

where the points in p_{correct} and p_{wrong} are determined according to the schema in Fig. 7 and then normalized using the normalization schema that SV-COMP uses for meta categories [17, page 597] (note that the factor q is removed in comparison to last year [27, page 513] from the formula, because it is not necessary to give a higher

Table 12: Validation of correctness witnesses (version 2.0): Overview of the top-three validators for each category; values for CPU time rounded to two significant digits

Rank	Validator	Score	CPU Time (in h)	Solved Tasks	False Alarms	Wrong Proofs
ReachSafety						
1	**UAutomizer**	**4545**	69	3 830		
2	Mopsa [new]	3284	17	2 816		
3	CPAchecker	2872	23	2 784		
MemSafety						
1	**UAutomizer**	**5213**	190	5 701		
2	Mopsa [new]	5015	2.5	5 658		
3	Goblint [new]	4225	0.26	5 677		
ConcurrencySafety						
1	**Goblint** [new]	**0**	0	0		
2	missing validator	0	0	0		
3	missing validator	0	0	0		
NoOverflows						
1	**UAutomizer**	**25441**	220	17 913		
2	Mopsa [new]	23601	7.8	17 333		
3	Goblint [new]	17143	0.77	14 125		
Termination						
1	**Goblint** [new]	**0**	0	0		
2	missing validator	0	0	0		
3	missing validator	0	0	0		
SoftwareSystems						
1	**Mopsa** [new]	**3521**	23	6 102		
2	Goblint [new]	2793	9.2	4 636		
3	UAutomizer	1258	90	5 963		**14**
Overall						
1	**UAutomizer**	**20919**	570	33 407		**14**
2	Mopsa [new]	20889	50	31 909		
3	Goblint [new]	16186	11	26 224		

weight to wrong witnesses anymore). Witnesses that do not agree with the expected verification verdict are classified as *wrong*. Witnesses that agree with the expected verification verdict can be wrong although they agree with the expected version, for example, if a violation witness has a wrong path to the violation, or a correctness witness has an invariant that does not hold. Therefore, we use the information from the majority of the validators: a witness that agrees with the expected verification result is classified as *correct* if at least 75 % of the true/false results from validators confirm the result, and as *wrong* if at least 75 % of the true/false results from validators refute this result (and there must be at least 2 true/false results). Further details are given in the proposal [44]. This schema relates to

Table 13: Validation of correctness witnesses (version 1.0): Overview of the top-three validators for each category; values for CPU time rounded to two significant digits

Rank	Validator	Score	CPU Time (in h)	Solved Tasks	False Alarms	Wrong Proofs
ReachSafety						
1	**UAutomizer**	**28020**	540	21 331		
2	CPAchecker	25183	250	29 082		
3	LIV new	-44060	3.7	2 527		**31**
MemSafety						
1	**UAutomizer**	**259**	4.8	366		
2	LIV new	87	0.22	186		**6**
3	MetaVal	0	0	0		
ConcurrencySafety						
1	**UAutomizer**	**70**	2.6	120		
2	missing validator	0	0	0		
3	missing validator	0	0	0		
NoOverflows						
1	**CPAchecker**	**57309**	170	45 618		**9**
2	UAutomizer	56467	320	42 011		**2**
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**	**3275**	28	5 812		
2	UAutomizer	2211	240	13 916		**18**
3	LIV new	0	0	0		
Overall						
1	**UAutomizer**	**47571**	1 100	77 744		**20**
2	CPAchecker	35095	450	80 512		**9**
3	MetaVal	-38172	1 300	44 296		**504**

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, 13, and 14 show the rankings of the validators. Violation witnesses in format version 2.0 were not yet ranked, because the jury decided that in SV-COMP 2024, this is a demonstration track. The score results for all validators and all categories are available on the SV-COMP web site [1] and in the artifact [30]. Wrong proofs in Tables 12 and 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). False alarms in Table 14 are claims of a validator that the program contains

[1] https://sv-comp.sosy-lab.org/2024/results/results-validated/

Table 14: Validation of violation witnesses (version 1.0): Overview of the top-three validators for each category; values for CPU time rounded to two significant digits

Rank	Validator	Score	CPU Time (in h)	Solved Tasks	False Alarms	Wrong Proofs
ReachSafety						
1	UAutomizer	**24390**	120	15 932	4	
2	CProver-w2t	22251	18	16 970	2	
3	CPAchecker	15686	83	16 602	71	
MemSafety						
1	Symbiotic-Witch	**799**	0.59	1 723		
2	CPAchecker	626	3.7	1 570	6	
3	UAutomizer	472	5.2	809		
ConcurrencySafety						
1	Dartagnan	**9186**	37	8 674		
2	UAutomizer	6742	72	6 533		
3	CPAchecker	2110	14	3 061	28	
NoOverflows						
1	UAutomizer	**20030**	63	10 236	5	
2	CPAchecker	18892	81	14 323		
3	CProver-w2t	18400	7.7	13 679	18	
Termination						
1	UAutomizer	**692**	7.0	1 004		
2	MetaVal	0	0	0		
3	CPAchecker	-1496	5.3	993	26	
SoftwareSystems						
1	UAutomizer	**2633**	26	3 036	2	
2	Symbiotic-Witch	1696	0.59	1 113		
3	CPAchecker	1359	15	2 474		
Overall						
1	UAutomizer	**43235**	290	37 550	11	
2	Symbiotic-Witch	20980	42	27 484	4	
3	CProver-w2t	19651	27	32 936	178	

a bug described by a given violation witness although the program is correct (the validator confirms a wrong violation witness).

The adoption rate of the new witness format version 2.0 is discussed in the article that defines the format [7]. Tables 12 and 13 shows that there are categories that are supported still by less than three validators ('missing validators' for categories *ConcurrencySafety* and *Termination*).

> While there are six new validators in SV-COMP 2024 (Fig. 6), and while there is a great adoption rate of the new witness format 2.0 (Table 12), there is still a remarkable gap in software-verification research: There are verification results that can not yet be independently confirmed.

6 Conclusion

The 13th edition of the Competition on Software Verification (SV-COMP 2024) again compared automatic tools for software verification and the validation of the produced verification witnesses. SV-COMP again had a record number of 59 participating verification systems (incl. 7 new verifiers and 19 hors-concours; see Fig. 5 for the participation numbers and Table 5 for the details). Furthermore, the validation track compared 17 validation tools; the validation tools were assessed in a similar manner as in the verification track, using a community-agreed scoring schema. The number of verification tasks in SV-COMP 2024 was significantly increased to 30 300 in the C category. Table 10 shows that the top verification tools have an extremely low number of wrong results. However, there are still wrong results, and validation of the verification results is absolutely necessary. We hope that this overview and the competition leads to a broader adoption of software verification, and in particular, that more and better validation tools are developed in the near future.

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/2024. The main results of last year's competition were reproduced in an independent reproduction study [72].

Funding Statement. Some participants of this competition were funded in part by the Deutsche Forschungsgemeinschaft (DFG) — 378803395 (ConVeY).

Acknowledgments. We thank the verification community for contributing their tools to the evaluation, the jury for their work on improving the quality of the verification tasks and for their advice in refining and applying to the competition rules, Philipp Wendler for maintaining and improving BenchExec, Matthias Kettl for his help with the competition scripts, and the VCloud team for keeping the scheduling system up to speed.

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. IEEE (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). pp. 541–546. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_33
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., Beyer, D., Lingsch-Rosenfeld, M., Spiessl, M., Strejček, J.: Software verification witnesses 2.0. In: Proc. SPIN. Springer (2024)
8. Ayaziová, P., Strejček, J.: Symbiotic-Witch 2: More efficient algorithm and witness refutation (competition contribution). In: Proc. TACAS (2). pp. 523–528. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_30
9. Ayaziová, P., Strejček, J.: Witch 3: Validation of violation witnesses in the witness format 2.0 (competition contribution). In: Proc. TACAS. LNCS. pp. 341–346. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_18
10. Baier, D., Beyer, D., Chien, P.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Spiessl, M., Wachowitz, H., Wendler, P.: CPAchecker 2.3 with strategy selection (competition contribution). In: Proc. TACAS. LNCS. pp. 359–364. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_21
11. Bajczi, L., Szekeres, D., Mondok, M., Ádám, Z., Somorjai, M., Telbisz, C., Dobos-Kovács, M., Molnár, V.: EmergenTheta: Verification beyond abstraction refinement (competition contribution). In: Proc. TACAS. LNCS. pp. 371–375. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_23
12. Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Mondok, M., Molnár, V.: Theta: Abstraction based techniques for verifying concurrency (competition contribution). In: Proc. TACAS. LNCS. pp. 412–417. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_30
13. Bajczi, L., Ádám, Z., Micskei, Z.: ConcurrentWitness2Test: Test-harnessing the power of concurrency (competition contribution). In: Proc. TACAS. LNCS. pp. 330–334. LNCS 14752, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_16
14. 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
15. 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
16. 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
17. 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
18. 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
19. 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
20. 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

21. 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
22. 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
23. 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
24. 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
25. 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
26. 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
27. Beyer, D.: Competition on software verification and witness validation: SV-COMP 2023. In: Proc. TACAS (2). pp. 495–522. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_29
28. Beyer, D.: Software testing: 5th comparative evaluation: Test-Comp 2023. In: Proc. FASE. pp. 309–323. LNCS 13991, Springer (2023). https://doi.org/10.1007/978-3-031-30826-0_17
29. Beyer, D.: Fm-tools data set of metadata about verifiers and validators (SV-COMP 2024). Zenodo (2024). https://doi.org/10.5281/zenodo.10669735
30. Beyer, D.: Results of the 13th Intl. Competition on Software Verification (SV-COMP 2024). Zenodo (2024). https://doi.org/10.5281/zenodo.10669731
31. Beyer, D.: SV-Benchmarks: Benchmark set for software verification (SV-COMP 2024). Zenodo (2024). https://doi.org/10.5281/zenodo.10669723
32. Beyer, D.: Verification witnesses from verification tools (SV-COMP 2024). Zenodo (2024). https://doi.org/10.5281/zenodo.10669737
33. 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
34. 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
35. 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
36. 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
37. 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
38. 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

39. 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
40. 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
41. 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
42. 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
43. Beyer, D., Spiessl, M.: LIV: A loop-invariant validation using straight-line programs. In: Proc. ASE. pp. 2074–2077. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00214
44. 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
45. Beyer, D., Wachowitz, H.: Coveriteam Release 1.1. Zenodo (2024). https://doi.org/10.5281/zenodo.10843666
46. 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
47. 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
48. 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
49. Chalupa, M., Henzinger, T.: Bubaak: Runtime monitoring of program verifiers (competition contribution). In: Proc. TACAS (2). pp. 535–540. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_32
50. Chalupa, M., Richter, C.: Bubaak-SpLit: Split what you cannot verify (competition contribution). In: Proc. TACAS. LNCS. pp. 353–358. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_20
51. 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
52. 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
53. Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS. LNCS. pp. 365–370. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_22
54. 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
55. 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

56. 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
57. 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
58. 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
59. 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
60. 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
61. 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
62. 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). pp. 588–593. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_41
63. Dietsch, D., Heizmann, M., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate Taipan 2023 (competition contribution). In: Proc. TACAS (2). pp. 582–587. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_40
64. 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
65. 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
66. 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
67. 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
68. Ernst, G.: Korn: Horn clause based verification of C programs (competition contribution). In: Proc. TACAS (2). pp. 559–564. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_36
69. 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
70. 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
71. 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

72. Gerhold, M., Hartmanns, A.: Reproduction report for SV-COMP 2023. Tech. rep., University of Twente (2023). https://doi.org/10.48550/arXiv.2303.06477
73. 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
74. 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
75. 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
76. 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
77. Heizmann, M., Bentele, M., Dietsch, D., Jiang, X., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate automizer and the abstraction of bitwise operations (competition contribution). In: Proc. TACAS. LNCS. pp. 418–423. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_31
78. 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
79. 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
80. 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
81. 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
82. Hussein, S., Yan, Q., McCamant, S., Sharma, V., Whalen, M.: Java Ranger: Supporting string and array operations (competition contribution). In: Proc. TACAS (2). pp. 553–558. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_35
83. 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
84. 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
85. 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
86. Jonáš, M., Kumor, K., Novák, J., Sedláček, J., Trtík, M., Zaoral, L., Ayaziová, P., Strejček, J.: Symbiotic 10: Lazy memory initialization and compact symbolic execution (competition contribution). In: Proc. TACAS. LNCS. pp. 406–411. LNCS (2024). https://doi.org/10.1007/978-3-031-57256-2_29
87. 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)

88. 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
89. 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
90. 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
91. 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
92. 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
93. 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
94. Loose, N., Mächtle, F., Sieck, F., Eisenbarth, T.: SWAT: Modular dynamic symbolic execution for java applications using dynamic instrumentation (competition contribution). In: Proc. TACAS. LNCS. pp. 399–405. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_28
95. 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
96. Malík, V., Schrammel, P., Vojnar, T., Nečas, F.: 2LS: Arrays and loop unwinding (competition contribution). In: Proc. TACAS (2). pp. 529–534. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_31
97. Menezes, R., Aldughaim, M., Farias, B., Li, X., Manino, E., Shmarov, F., Song, K., Brauße, F., Gadelha, M.R., Tihanyi, N., Korovin, K., Cordeiro, L.: ESBMC v7.4: Harnessing the power of intervals (competition contribution). In: Proc. TACAS. LNCS. pp. 376–380. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_24
98. Metta, R., Karmarkar, H., Madhukar, K., Venkatesh, R., Chakraborty, S.: Proton: Probes for non-termination and termination (competition contribution). In: Proc. TACAS. LNCS. pp. 393–398. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_27
99. Monat, R., Milanese, M., Parolini, F., Boillot, J., Ouadjaout, A., Miné, A.: Mopsa-C: Improved verification for C programs, simple validation of correctness witnesses (competition contribution). In: Proc. TACAS. LNCS. pp. 387–392. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_26
100. 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
101. 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
102. 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

103. 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
104. 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
105. 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
106. 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
107. 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
108. 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
109. 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
110. 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
111. Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.: Goblint: Abstract interpretation for memory safety and termination (competition contribution). In: Proc. TACAS. LNCS. pp. 381–386. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_25
112. Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.: Goblint Validator: Correctness witness validation by abstract interpretation (competition contribution). In: Proc. TACAS. LNCS. pp. 335–340. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_17
113. 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
114. 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
115. 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
116. 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). pp. 571–576. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_38
117. 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

118. 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
119. 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
120. 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
121. Wang, Z., Chen, Z.: AISE: A symbolic verifier by synergizing abstract interpretation and symbolic execution (competition contribution). In: Proc. TACAS. LNCS. pp. 347–352. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_19
122. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.21. Zenodo (2024). https://doi.org/10.5281/zenodo.10671136
123. 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
124. 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