Improvements in Software Verification and Witness Validation: SV-COMP 2025

Dirk Beyer [1]✉ and Jan Strejček [2]

[1] LMU Munich, Munich, Germany
[2] Masaryk University, Brno, Czech Republic

Abstract. The 14th edition of the *Competition on Software Verification (SV-COMP 2025)* evaluated 62 verification tools and 18 witness validation tools, making it the largest comparison of its kind so far. Out of these, 35 verification and 13 validation tools participated with an active support of teams led by 33 different representatives from 12 countries. The verification track of the competition was executed on a benchmark set of 33 353 verification tasks with C programs and 6 different specifications (reachability, memory safety, memory cleanup, overflows, termination, and data races) and 674 verification tasks with Java programs checked for assertion validity. Additionally, we considered 673 verification tasks with Java programs checked for runtime exceptions as a demo category. The validation track analyzed the witnesses generated in the verification track and newly also 103 handcrafted witnesses. To handle the increasing complexity of the competition, the organization committee has been established.

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

1 Introduction

This report presents the objectives, processes, rules, participants, and results of SV-COMP 2025. It extends the series of competition reports (see footnote). This year, we focus on a more precise description of the current category structure, competition workflow, scoring schema, and changes in the competition done in 2025. The 14th Competition on Software Verification (`https://sv-comp.sosy-lab.org/2025`) is again the largest comparative evaluation ever in this area. The objectives of the competitions were discussed earlier (1–4 [23]) and extended over the years (5–6 [24] and 7 [30]):

1. provide an overview of the state of the art in software-verification technology and increase visibility of the most recent software verifiers,

This report extends previous reports on SV-COMP [17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 30]. Reproduction packages are available on Zenodo (see Table 5).

✉ `dirk.beyer@sosy-lab.org`

2. establish a repository of software-verification tasks that is publicly available for free use as a 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,
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 participation, and other meta data in a central repository.

The SV-COMP 2020 report [24] 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 area of formal methods [16]. 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 [84], VerifyThis [70], Test-Comp [29], and TermCOMP [77]. The SV-COMP 2020 report [24] provides a more detailed discussion.

Quick Summary of Changes. We aim to keep the setup of the competition stable. Still, there are always improvements and developments. For the 2025 edition, the following changes were made:

- The organization committee was established.
- The number of considered verification tasks again increased in both C and Java languages: the number of C tasks increased from 30 300 in 2024 to 33 353 and the number of Java tasks increased from 587 in 2024 to 674 (not counting the demo category mentioned below).
- We newly considered the property saying that a Java program does not cause a runtime exception. We used 673 tasks with this property as a demo category.
- In the validation track, we newly used 103 handcrafted witness validation tasks as a complement to the validation tasks generated by verifiers.
- The category structure was extended by three new base categories with C programs added to meta category *SoftwareSystems*, namely *SoftwareSystems-Intel-TDX-Module-ReachSafety*, *SoftwareSystems-uthash-MemCleanup*, and *SoftwareSystems-DeviceDriversLinux64-Termination*, one new base category *RuntimeException-Java* with Java programs (ran as a demo category due to late announcement) added to *JavaOverall*, and one new meta category *ValidationCrafted* of handcrafted validation tasks divided into two base categories *CorrectnessWitnesses-Loops* and *ViolationWitnesses-ControlFlow*.
- The definition of the *memory safety* subproperty expressing that all allocated memory is tracked was reformulated more precisely.

- The validation of violation witnesses in version 2.0 turned into a regular category due to a higher number of participating tools.
- The normalization formula computing the scores in meta categories was modified to ignore void tasks and empty categories.
- We now officially recognize two types of *hors-concours* participants: *meta verifiers* and *inactive* participants, which are tools without active team support.
- The medals can be assigned only for positive scores.

2 Organization and Processes

Organization. The competition was established and ran for the first 13 years by Dirk Beyer. The SV-COMP community supported the competition mostly by maintaining the collection of verification tasks. Executing the competition in its current form would not be possible without the benchmarking framework BenchExec [42] and the system for distributed benchmark execution BenchCloud [35], which are developed and maintained by his research group. Since the beginning, the numbers of benchmarks and participants are substantially growing. Moreover, completely new layers of complexity were added with witness validation, the whole validation track, and a second format of witnesses. To distribute the effort necessary to smoothly run of the competition, the organization committee was established before the SV-COMP 2025. The competition has now two chairs (Dirk Beyer and Jan Strejček) and members in charge of benchmark quality (Zsófia Ádám, Raphaël Monat, Simmo Saan, and Frank Schüssele), category structure (Thomas Lemberger), infrastructure development (Philipp Wendler, Po-Chun Chien, Marek Jankola, Henrik Wachowitz, Matthias Kettl, and Marian Lingsch-Rosenfeld), and reproducibility (Levente Bajczi).

Procedure. SV-COMP is an open competition (also known as comparative evaluation). Verification tasks and handcrafted validation tasks are publicly available in a repository (Table 4) where anyone can contribute. The competition has basically two phases: *training* and *evaluation*. During the training phase, participating teams can repeatedly submit new versions of their tool. The organizers run the tool on relevant tasks and provide the results to the whole community. The participants can inspect the results, fix bugs in their tools and submit a new version or report an issue with some tasks. The set of verification tasks and handcrafted validation tasks is frozen approximately two weeks before the the end of the training phase. In the evaluation phase, the tools are again executed on all relevant tasks, the participants are asked for an inspection of the results and they can challenge the validity of some tasks. After removing the invalid tasks (they are marked as *void*), the results are announced on the competition web site.

Competition Jury. The competition jury reviews the competition contribution papers and helps the organizer with resolving any disputes that might occur (cf. competition report of SV-COMP 2013 [18]). The tasks of the jury were described in more detail in the report of SV-COMP 2022 [27]. The jury consists of the SV-COMP chairs and one representative of each actively participating

tool. The representatives of participating tools circulate every year after the tool-submission deadline. The current representatives are listed in Tables 6 and 7. The jury includes one additional member, P. Darke (TCS, India), representing a tool that ultimately did not actively participate due to license issues. The current jury is also listed on the web site (`https://sv-comp.sosy-lab.org/2025/committee.php`).

3 Tasks, Workflow, and Scoring

Verification and Validation Tasks. A *verification task* consists of a program, a property to be verified, and an expected verification result. A *validation task* is a program, a property, and a witness of the program correctness or property violation to be validated. SV-COMP 2025 supports witnesses in two versions of the witness format, namely 1.0 [37] and 2.0 [7]. Some validation tasks (e.g., the handcrafted ones) contain also the expected validation result. SV-COMP 2025 used the task-definition format in version 2.1 to denote the verification and validation tasks.

Properties. In connection with C programs, we consider 6 different properties: *unreachability of a given function* (referenced as `unreach-call` in the competition), *memory safety* (`valid-memsafety`) composed of three subproperties saying that all pointer dereferences are valid (`valid-deref`), all memory deallocations are valid (`valid-free`), and all allocated memory is tracked (`valid-memtrack`), *memory cleanup* (`valid-memcleanup`) saying that all allocated memory is deallocated before the program terminates, *no overflow* (`no-overflow`) saying that operations on signed integers never overflow, *no data race* (`no-data-race`) saying that the program does not contain any data race, and *termination* (`termination`) saying that the program always terminates. Note that a program is considered memory safe only if it satisfies all three subproperties. The subproperties are used in particular to report what is violated if a program is not memory safe. For Java programs, we consider the property *assertion validity* (`assert_java`) and newly also *no runtime exception* (`runtime-exception`). We refer to the conference web page for precise definition of these properties (`https://sv-comp.sosy-lab.org/2025/rules.php`). We note that the precise definition of tracked memory was modified for SV-COMP 2025 as the previous definition was ambiguous.

Categories. The verification tasks are divided into base categories loosely reflecting the programming language, program features, the considered property, and the source of the benchmarks. Base categories are accumulated into meta categories. For the C language, there are two levels of meta categories. The top level contains the category *Overall* of all C verification tasks and the category *FalsificationOverall*, which is rather specific. The category *FalsificationOverall* was originally introduced to support bug-finding tools. It has basically the same structure as *Overall*, but it contains only the tasks with safety properties, which are all but *termination*. In other words, *FalsificationOverall* does not contain the whole meta category *Termination* and base category *SoftwareSystems-DeviceDriversLinux64-Termination*. It also uses a specific scoring schema as explained below. For Java language, there is only a single meta category called *JavaOverall*. The category

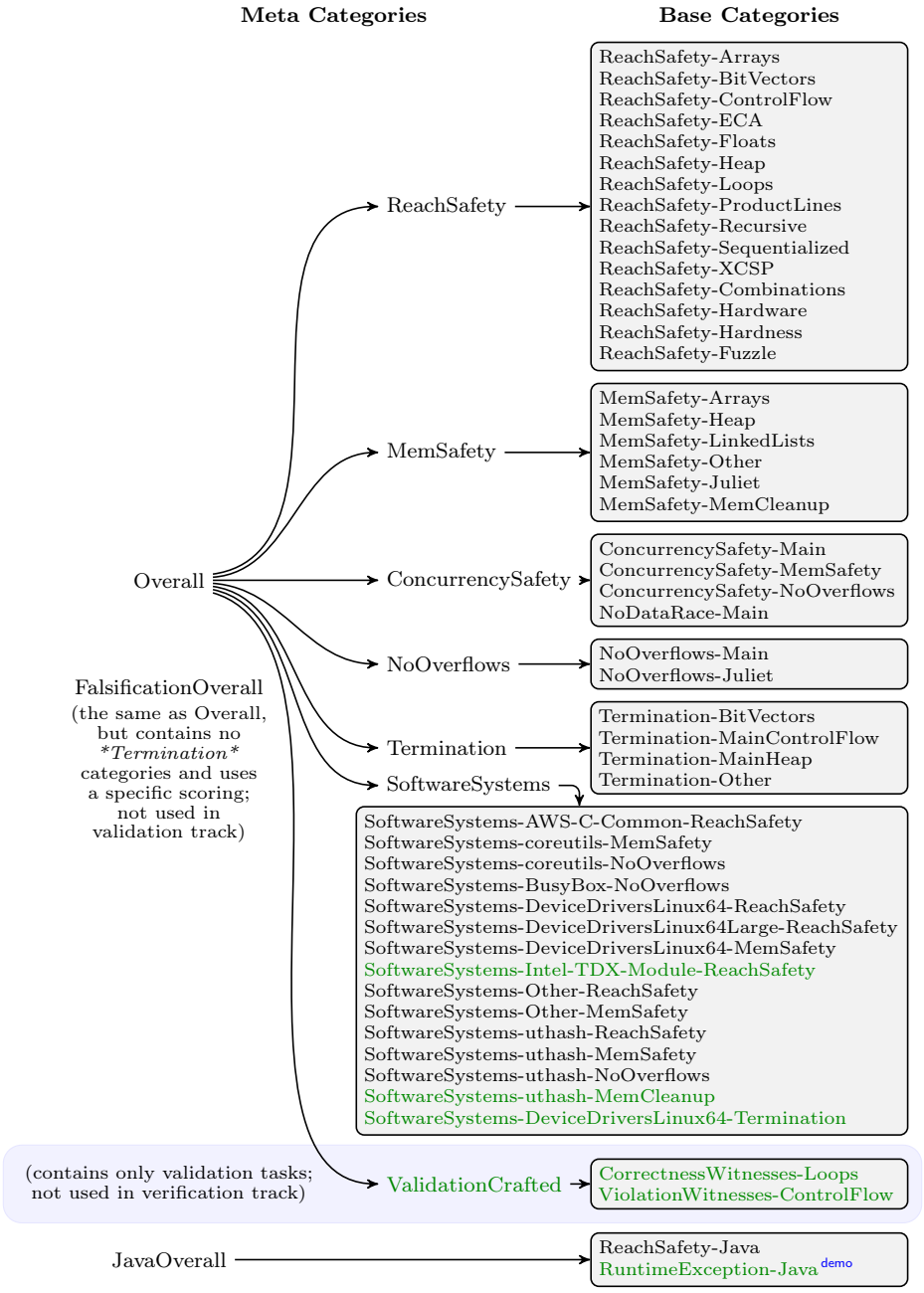

Fig. 1: Category structure for SV-COMP 2025; demo marks the demo category, new categories are green

structure was significantly updated for SV-COMP 2024. For SV-COMP 2025, we added 3 new base categories to the category *SoftwareSystems*. Further, the category *JavaOverall* was extended with the base category *RuntimeException-Java* of verification tasks with the property *no runtime exception*. This base category ran only as a demo category because it was announced shortly before the competition deadlines. As a consequence, this base category is not reflected in the score computations of *JavaOverall* The current category structure including the new categories is shown in Fig. 1. The meta categories and the number of verification tasks they contain are shown in Tables 10, 11, and 12 presenting the results of the verification track. Finally, the categories are described in detail on the competition web site (`https://sv-comp.sosy-lab.org/2025/benchmarks.php`).

All validation tasks are basically divided into 4 groups: the witnesses of program correctness are separated from the witnesses of property violation and each of these two groups is again divided according to the used witness format (version 1.0 or 2.0). In each of the four groups, the validation tasks are organized basically in the same category structure: each validation task generated by a verifier is in the base category determined by the corresponding verification task. The handcrafted validation tasks are stored in two new base categories united in a new meta category called *ValidationCrafted* as shown in Fig. 1. The meta category *FalsificationOverall* is not considered in the validation track. As the witness formats and current validators have certain limitations, SV-COMP 2025 supports witnesses in individual formats only for selected properties and categories, as shown in Table 1. Some categories in some groups are empty, for example because SV-COMP does not support the particular witness format for the category. All empty categories are removed from the category structures of the groups.

Competition Workflow. Roughly speaking, the inputs of the competition are (a) *verification tasks* and, for each verifier and witness validator, the participating teams provide (b) *benchmark definition* listing the base categories the tool is supposed to be applied on (i.e., the tool opts-out from the categories that are not listed there), (c) *tool-info module* that specifies the interface for running the tool and interpreting its results, and (d) *tool archive* on Zenodo from where the tool is downloaded. The inputs provided by participating teams are described more precisely in the report for Test-Comp 2021 [26] (SV-COMP and Test-Comp use similar workflow and components).

The verification track proceeds as follows. The competition scripts run each verifier on all relevant verification tasks given by its *benchmark definition*. If the verifier solves a task, it returns either TRUE meaning that the program satisfies the given property or FALSE meaning that the program violates the property. The output TRUE is accompanied with a *correctness witness* and the output FALSE with a *violation witnesses* in some of the formats supported by SV-COMP for the corresponding property and category (see Table 1). When SV-COMP supports both formats, the verifier can actually produce two witnesses, one in each format. If no format is supported, the witness is not required. This is the case of correctness witnesses in categories **-Arrays*, **-Floats*, **-Heap*, **MemSafety**, *ConcurrencySafety-**, **NoDataRace**, **Termination-**, and **-Java*. The generated

Table 1: Support of witnesses in individual formats based on property and category; '–' indicates that the property is not relevant for the witness type in the column

		Correctness Witnesses		Violation Witnesses	
Property	**Categories**	v1.0	v2.0	v1.0	v2.0
`unreach-call`	all except **-Arrays*, **-Floats*, **-Heap*, and *ConcurrencySafety-**	✓	✓	✓	✓
`unreach-call`	**-Arrays*, **-Floats*, and **-Heap*			✓	✓
`unreach-call`	*ConcurrencySafety-**			✓	
`valid-memsafety`	all			–	–
`valid-deref`	all except *ConcurrencySafety-**	–	–	✓	✓
`valid-deref`	*ConcurrencySafety-**	–	–	✓	
`valid-free`	all except *ConcurrencySafety-**	–	–	✓	✓
`valid-free`	*ConcurrencySafety-**	–	–	✓	
`valid-memtrack`	all	–	–	✓	
`valid-memcleanup`	all			✓	
`no-overflow`	all except *ConcurrencySafety-**	✓	✓	✓	✓
`no-overflow`	*ConcurrencySafety-**			✓	
`no-data-race`	all			✓	
`termination`	all			✓	
`assert-java`	all			✓	
`runtime-exception`	all			✓	

witnesses are turned into validation tasks. If the program of the validation task is written in C, we run WitnessLint to check that the witness adheres to the format. If the answer is negative, the witness is *syntactically invalid* and we never consider such a witness as confirmed. We do not apply this step for validation tasks with programs in Java as there is no witness linter for these witnesses. Competition scripts then run all suitable validators on each validation task, where suitability is determined by the *benchmark definition* of validators. Each validator can either confirm the task, refute it, or fail to solve it. A validator confirms a correctness witness by returning True and refutes it by returning False. A violation witness is confirmed by False and refuted by True.

The validation track uses the same inputs, only the verification tasks are replaced by *validation tasks*. These are the handcrafted validation tasks and all validation tasks generated by the verifiers in the verification track and successfully checked by WitnessLint. A validation task contains an expected result if it is handcrafted or if it was generated with an incorrect verification result (i.e., it contains a correctness witness for a verification task with property violation or a violation witness for a verification task with a correct program). In the latter case, the expected validation result corresponds to refutation. Each validator is executed on all validation tasks specified by the corresponding *benchmark definition*.

Computing Resources. The computing resources for each tool execution were the same as in SV-COMP 2024. Each verifier run was limited by 15 GB of memory

Table 2: Scores per individual verification results used since SV-COMP 2021

Result	Points	Description
True correct	+2	Program correctly reported to satisfy the property and the correctness witness confirmed (or not required)
True incorrect	−32	Incorrect program reported as correct (wrong proof)
False correct	+1	Property violation was correctly found and the violation witness was confirmed
False incorrect	−16	Violation reported but the property holds (false alarm)

and 15 min of cumulative CPU time on 4 processing units. Each run of a validator 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 computer cluster at the SoSy-Lab at LMU, which consists of 168 machines, where each machine has one Intel Xeon E3-1230 v5 CPU with 8 processing units, a frequency of 3.4 GHz, 33 GB of RAM, and a GNU/Linux operating system (x86_64-linux, Ubuntu 24.04 with Linux kernel 6.8). We used BenchExec [42] to measure and control computing resources (CPU time, memory) and BenchCloud [35] to distribute, install, run, and clean-up verification runs, and to collect the results.

Scoring Schema. For verification track, the scoring schema of SV-COMP 2025 in base categories was the same as for SV-COMP 2021. Table 2 lists all the cases when verification results are awarded with non-zero points, where a verification result is

- *incorrect* if it does not agree with the expected result of the verification task,
- *correct* if it agrees with the expected result and the produced witness (or at least one of them if there are two) is confirmed by some validator. In the categories where correctness witnesses are not required, a verification results is *correct* whenever it agrees with the expected result.

When a verification result agrees with the expected result but no validator confirms the witness (although it is required), we call the result *correct-unconfirmed*. The score of a verifier in a base category is simply the sum of the points for individual verification tasks. The score for a meta category is computed from the scores of all contained (meta or base) categories on the next level and the number of tasks in these categories. Formally, if a meta category contains k categories of the next level and the i-th contained category has the score s_i and consists of n_i verification tasks, then the meta category gets the score $(\Sigma_{i=1}^{k} s_i/n_i) \cdot (\Sigma_{i=1}^{k} n_i)/k$, i.e., the sum of scores in each category normalized by the number of tasks in the category multiplied by the average number of tasks in the contained categories. Note that in previous years, the numbers n_i included also void tasks that are technically in categories but are not used in the competition (a verification task is marked as *void* typically because it was changed for some serious reason after the task freezing deadline). Since SV-COMP 2025, these void tasks are not included in n_i.

Table 3: Scores per individual validation results used since SV-COMP 2024

Result	Points	Description for correctness witnesses Description for violation witnesses
TRUE correct	+2	The correctness witness was correctly confirmed The violation witness was correctly refuted
TRUE incorrect	−32	The correctness witness was confirmed but it is not correct The violation witness was refuted but it is correct
FALSE correct	+1	The correctness witness was correctly refuted The violation witness was correctly confirmed
FALSE incorrect	−16	The correctness witness was refuted but it is correct The violation witness was confirmed but it is not correct

The scoring in *FalsificationOverall* and its subcategories is slightly different. As this meta category was motivated by tools that can only find a bug, the scores in its base categories are sums of the points for results FALSE (both correct and incorrect), while the results TRUE are ignored. The computation of scores for meta categories remains unchanged.

For validation track, the scoring schema of SV-COMP 2025 in base categories was the same as for SV-COMP 2024. The biggest difference from the verification track comes from the fact that many validation tasks do not contain the expected validation result. In fact, it is contained only in handcrafted validation tasks and in the tasks generated by verifiers that solve some verification task incorrectly (and thus the witness they produce should be refuted). For the remaining tasks, the expected results are determined by voting. For each such a task, we collect the results from all validators that solved (i.e., confirmed or refuted) the task. If we have at least two such results and at least 75 % of them agree on their decision, then the expected result is set by the majority vote. In all other cases, the expected result is not determined and the validation task has no influence on the validation track results as it is considered *void*. Tool developers can inspect the results and convince the community that a voted expected results is in fact wrong. In such a case, the corresponding validation task is removed from the competition.

Table 3 lists all the cases when validation results are awarded with non-zero points. A validation result TRUE or FALSE is considered *correct* if it agrees with the expected result and *incorrect* otherwise. To compute the scores in base categories, we virtually divide each base category C into two subcategories C_c, C_w, where

- C_c contains the witnesses that are expected to be *correct* (i.e., correctness witnesses with the expected result TRUE and violation witnesses with the expected result FALSE), and
- C_w contains the witnesses that are expected to be *wrong* (i.e., correctness witnesses with the expected result FALSE and violation witnesses with the expected result TRUE).

The score of a validator in each subcategory C_c, C_w is the sum of the points for individual validation tasks in the subcategory. If some of the subcategories C_c, C_w

Table 4: Publicly available components for reproducing SV-COMP 2025

Component	Repository	Version
Verification Tasks	gitlab.com/sosy-lab/benchmarking/sv-benchmarks	svcomp25
Benchmark Definitions	gitlab.com/sosy-lab/sv-comp/bench-defs	svcomp25
Tool-Info Modules	github.com/sosy-lab/benchexec	3.29
Verifiers	gitlab.com/sosy-lab/benchmarking/fm-tools	svcomp25
BenchExec (Benchmarking)	github.com/sosy-lab/benchexec	3.29
BenchCloud (Distribution)	gitlab.com/sosy-lab/software/benchcloud	1.3.0
Witness Format	gitlab.com/sosy-lab/benchmarking/sv-witnesses	2.0.3
CoVeriTeam for CI	gitlab.com/sosy-lab/software/coveriteam	1.2.1
Processing Scripts	gitlab.com/sosy-lab/benchmarking/competition-scripts	svcomp25

Table 5: Artifacts published for SV-COMP 2025

Content	DOI	Reference
Verification Tasks	10.5281/zenodo.15012096	[33]
Competition Results	10.5281/zenodo.15012085	[32]
FM-Tools (Verifiers and Validators)	10.5281/zenodo.15055359	[31]
Verification Witnesses	10.5281/zenodo.15012077	[34]
BenchExec	10.5281/zenodo.15007216	[133]
CoVeriTeam	10.5281/zenodo.11193690	[47]

is empty, the score for C is directly the score for the non-empty subcategory. If both subcategories C_c, C_w are non-empty and have respective scores s_c, s_w, then the score for C is computed as for a meta category in the verification track, i.e., $(\frac{s_c}{|C_c|} + \frac{s_w}{|C_w|}) \cdot \frac{|C_c|+|C_w|}{2}$. The score of a validator in a meta category is then computed by the same process as in the verification track. Note that all the scores presented in this paper and on the competition web are rounded to integers before printing, but their computation is done with a higher precision.

Ranking. As before, the rank of a verifier in each category was decided based on the achieved score. In case of a tie, we used the *success run time* as the secondary criterion, which is the total CPU time of the verifier over all tasks in a given category for which the verifier reported a correct verification result. Ranking in validation track works in the same way. Recall that a tool participates only in the categories specified in its *benchmark definition*. In contrast to the previous year, we assign medals only to tools with a positive score.

Reproducibility. SV-COMP results must be reproducible. Hence, all major components are maintained in public version-control repositories. Table 4 lists these components with the links to the repositories and their versions used in SV-COMP 2025. Most of these components are described with more details in the SV-COMP 2016 report [21]. Later, BenchCloud was introduced to distribute the benchmarking jobs in an elastic cloud and collect results. Moreover, CoVeriTeam is used to continuously check (GitLab CI pipeline) whether tools can be executed in

the competition environment. The processing scripts to execute the experiments and post-process the data into tables, scores, and rankings are also publicly released. The competition artifacts are published at Zenodo (see Table 5) with the relevant tools and data to guarantee their long-term availability and immutability.

For the reproducibility reasons, SV-COMP requires since 2018 that the verifiers and validators must be publicly available for download and has a license that

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

The verification and handcrafted validation tasks used in the competition are also accompanied by a license. In this case, the stated license must allow to

- view, understand, investigate, and reverse engineer the algorithm or system,
- change the program (in particular, pre-process and adopt the programs to be useful for a verification task),
- (re-)distribute the (original and changed) program under the same terms (in particular, in replication packages for research projects or as regression tests),
- compile and execute the program (in particular, for the purpose of verifying that a specification violation exists),
- and commercially take advantage of the program (in particular, to not exclude developers of commercial verifiers).

4 Participating Verifiers and Validators

In total, SV-COMP 2025 evaluates 62 verification and 18 validation tools. Besides 35 verifiers and 13 validators registered to and supported in the competition by development teams, we also evaluated some tools participating in previous years but not actively registered and supported this year. These tools are called *inactive*, clearly marked with $^{\varnothing}$ in all tables, and they do not appear in rankings. Further, we clearly distinguish *meta verifiers* according to the following characterization approved by the community of SV-COMP 2023.

> A *meta verifier* is a combination of at least two existing verification components such that each result produced by the combination can be computed by some of its components alone. A verifier is the result of research and engineering in verification algorithms and approaches, while a typical meta verifier selects a verification component to run, sets up its parameters, and potentially post-processes its output.

Meta verifiers are annotated with $^{\text{meta}}$ in all tables and also excluded from rankings. Note that before SV-COMP 2025, both inactive tools and meta verifiers were marked as *hors-concours* participants and not properly distinguished.

Tables 6 and 7 list all validation and verification tools evaluated in SV-COMP 2025, respectively. The tables contain the tool name (with hyperlink),

Table 6: Participating validators with tool references, representing jury members, their affiliations, and indications of supported witnesses depending on format version and type; ∅ for inactive, new for first-time participants, *Cor.* for correctness witnesses, *Vio.* for violation witnesses, and ✓ for newly added support

				Witness Format			
				v1.0		v2.0	
Validator	**Ref.**	**Jury Member**	**Affiliation**	**Cor.**	**Vio.**	**Cor.**	**Vio.**
ConcurrentW2T	[13]	Z. Ádám	BME Budapest		✓		
CPAchecker	[40, 41]	M. Lingsch-Rosenfeld	LMU Munich	✓	✓	✓	✓
CPA-w2t∅	[37, 39]	–	–		✓		
CProver-w2t∅	[37, 39]	–	–		✓		
Dartagnan	[116]	H. Ponce de León	Huawei Dresden		✓		
Goblint	[122]	S. Saan	U. Tartu			✓	
GWIT∅	[85]	–	–		✓		
JCWIT∅	[56]	–	–	✓			
LIV	[45]	M. Lingsch-Rosenfeld	LMU Munich	✓		✓	
MetaVal	[43]	M. Lingsch-Rosenfeld	LMU Munich	✓	✓	✓	✓
MetaVal++new		M. Lingsch-Rosenfeld	LMU Munich			✓	
Mopsa	[106]	R. Monat	Inria & U. Lille			✓	
NITWIT∅	[136]	–	–		✓		
Symbiotic-Witch	[8]	P. Ayaziová	Masaryk U., Brno		✓		
UAutomizer	[36, 38]	M. Ebbinghaus	U. Freiburg	✓	✓	✓	✓
UReferee new		F. Schüssele	U. Freiburg	✓		✓	
Wit4Java	[135]	T. Wu	U. Manchester		✓		
Witch	[7, 9]	P. Ayaziová	Masaryk U., Brno				✓

Table 7: Participating verifiers with tool references, representing jury members and their affiliations; ∅ for inactive, meta for meta verifiers, and new for first-time participants

Verifier	**Ref.**	**Jury Member**	**Affiliation**
2LS	[48, 103]	V. Malík	BUT, Czechia
aise	[99, 132]	Z. Chen	NUDT, China
APROVE	[100]	N. Lommen	RWTH Aachen, Germany
BRICK	[49]	L. Bu	Nanjing U., China
Bubaak	[51, 53]	M. Chalupa	ISTA, Austria
Bubaak-SpLit	[52]	M. Chalupa	ISTA, Austria
CBMC∅	[58, 95]	–	–
COASTAL∅	[129]	–	–
CoOpeRace meta new		V. Vojdani	U. Tartu, Estonia
CPAchecker	[10, 11]	M. Lingsch-Rosenfeld	LMU Munich, Germany
CPALockator∅	[5, 6]	–	–

(continues on next page)

Table 7: Participating verifiers (continued)

Verifier	Ref.	Jury Member	Affiliation
CPA-bam-bnb∅	[4, 131]	–	–
CPA-bam-smg∅		–	–
CPV	[57]	P.-C. Chien	LMU Munich, Germany
Crux∅	[69, 123]	–	–
CSeq∅	[63, 89]	–	–
Dartagnan	[76, 115]	H. Ponce de León	Huawei Dresden, Germany
Deagle	[80]	F. He	Tsinghua U., China
DIVINE∅	[15, 96]	–	–
EBF∅	[3]	–	–
EmergenTheta	[12, 108]	L. Bajczi	BME Budapest, Hungary
ESBMC-incr	[59, 62]	T. Wu	U. Manchester, UK
ESBMC-kind	[75, 134]	T. Wu	U. Manchester, UK
Frama-C-SV∅	[44, 64]	–	–
Gazer-Theta∅	[1, 79]	–	–
GDart	[110]	F. Howar	TU Dortmund, Germany
GDart-LLVM∅		–	–
Goblint	[121, 130]	S. Saan	U. Tartu, Estonia
Graves-CPA∅ meta	[97]	–	–
Hornix new		M. Blicha	U. Lugano, Switzerland
Infer∅	[50, 93]	–	–
Java-Ranger	[86, 125]	S. Hussein	Ain Shams U., Egypt
JayHorn∅	[92, 124]	–	–
JBMC	[60, 61]	P. Schrammel	Diffblue, UK
JDart∅	[102, 109]	–	–
Korn	[72, 73]	G. Ernst	LMU Munich, Germany
Lazy-CSeq∅	[87, 88]	–	–
LF-checker∅		–	–
Locksmith∅	[117]	–	–
MLB		L. Bu	Nanjing U., China
Mopsa	[91, 107]	R. Monat	Inria & U. Lille, France
Nacpa meta new	[98]	H. Wachowitz	LMU Munich, Germany
PeSCo-CPA∅ meta	[119, 120]	–	–
PIChecker∅	[126]	–	–
Pinaka∅	[55]	–	–
PredatorHP∅	[83, 114]	–	–
Proton	[105, 111]	R. Metta	TCS, India
RacerF new	[65]	T. Dacík	BUT, Czechia
SPF∅	[112, 118]	–	–
SVF-SVC new	[104]	M. Richards	U. New South Wales, AU
sv-sanitizers		S. Saan	U. Tartu, Estonia
SWAT	[101]	N. Loose	U. Luebeck, Germany
Symbiotic	[54, 90]	M. Jonáš	Masaryk U., Czechia

(continues on next page)

Table 7: Participating verifiers (continued)

Verifier	Ref.	Jury Member	Affiliation
Theta	[127, 128]	L. Bajczi	BME Budapest, Hungary
Thorn new		L. Bajczi	BME Budapest, Hungary
UAutomizer	[81, 82]	M. Heizmann	U. Freiburg, Germany
UGemCutter	[74, 94]	D. Klumpp	U. Freiburg, Germany
UKojak	[71, 113]	M. Bentele	U. Freiburg, Germany
UTaipan	[68, 78]	D. Dietsch	U. Freiburg, Germany
VeriAbs ∅	[2, 66]	–	–
VeriAbsL ∅	[67]	–	–
VeriOover ∅		–	–

Table 8: Algorithms and techniques used by the participating tools;
∅ for inactive, meta for meta verifiers, and new for first-time participants

Tool	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	Task Translation
2LS				✓	✓			✓	✓		✓						✓				
aise			✓																		
AProVE																	✓				
BRICK	✓		✓	✓				✓								✓					
Bubaak			✓								✓					✓	✓			✓	
Bubaak-SpLit			✓		✓						✓				✓	✓	✓		✓	✓	
CBMC ∅				✓							✓					✓					
COASTAL ∅			✓																		
ConcurrentW2T																✓					
CoOpeRace meta new																✓			✓	✓	
CPAchecker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
CPALockator ∅	✓	✓					✓				✓	✓	✓	✓		✓					
CPA-bam-bnb ∅	✓	✓					✓				✓	✓	✓	✓							
CPA-bam-smg ∅																					
CPA-w2t ∅						✓						✓			✓						
CProver-w2t ∅				✓																	
CPV	✓	✓		✓	✓	✓					✓			✓					✓	✓	✓
Crux ∅			✓																		
CSeq ∅				✓							✓					✓					

(continues on next page)

Table 8: Algorithms and techniques (continued)

Tool	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	Task Translation
DARTAGNAN				✓							✓					✓					
DEAGLE				✓												✓					
DIVINE[∅]			✓				✓				✓					✓			✓	✓	
EBF[∅]				✓																	
EMERGENTHETA	✓	✓		✓	✓						✓			✓					✓	✓	✓
ESBMC-INCR				✓	✓						✓					✓					
ESBMC-KIND				✓	✓		✓	✓			✓					✓					
FRAMA-C-SV[∅]								✓													
GAZER-THETA[∅]	✓	✓		✓			✓				✓	✓	✓	✓						✓	
GDART			✓								✓									✓	
GDART-LLVM[∅]			✓								✓										
GOBLINT								✓								✓			✓		
GRAVES-CPA[∅] meta	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓	
GWIT[∅]			✓								✓									✓	
HORNIX new						✓															
INFER[∅]								✓	✓	✓										✓	
JAVA-RANGER			✓								✓										
JAYHORN[∅]	✓	✓				✓		✓					✓	✓							
JBMC				✓							✓					✓					
JCWIT[∅]				✓																	
JDART[∅]			✓								✓									✓	
KORN	✓	✓	✓				✓													✓	
LAZY-CSEQ[∅]				✓							✓					✓					
LF-CHECKER[∅]																					
LIV																					✓
LOCKSMITH[∅]																✓					
METAVAL																			✓		✓
METAVAL++ new																					✓
MLB			✓								✓									✓	
MOPSA								✓													
NACPA meta new																			✓	✓	
NITWIT[∅]							✓														
PESCO-CPA[∅] meta	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓	
PICHECKER[∅]	✓	✓									✓	✓		✓		✓			✓		

(continues on next page)

Table 8: Algorithms and techniques (continued)

Tool	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	Task Translation
PINAKA[∅]			✓	✓							✓										
PREDATORHP[∅]									✓												
PROTON				✓																	
RACERF[new]																✓					
SPF[∅]			✓						✓							✓					
SVF-SVC[new]			✓																		
SV-SANITIZERS																✓					
SWAT			✓																		
SYMBIOTIC			✓		✓			✓	✓		✓					✓				✓	
SYMBIOTIC-WITCH			✓																		
THETA	✓	✓		✓			✓				✓	✓		✓		✓			✓	✓	✓
THORN[new]		✓																	✓	✓	
UAUTOMIZER	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓	
UGEMCUTTER	✓	✓									✓		✓	✓	✓	✓			✓	✓	
UKOJAK	✓	✓									✓		✓	✓							
UREFEREE[new]	✓	✓									✓		✓	✓	✓	✓	✓		✓	✓	
UTAIPAN	✓	✓					✓	✓			✓		✓	✓	✓	✓			✓	✓	
VERIABS[∅]	✓			✓	✓		✓	✓										✓	✓	✓	
VERIABSL[∅]	✓			✓	✓		✓	✓										✓	✓	✓	
VERIOOVER[∅]																					
WIT4JAVA																					
WITCH			✓																		

references to papers that describe the tool, the representing jury member and the affiliation. The listings are also available on the competition web site at https://sv-comp.sosy-lab.org/2025/systems.php. Table 6 additionally indicates the witness formats and witness kinds supported by individual validators and whether they were supported in SV-COMP 2025 for the first time (✓) or not (✓).

Table 8 lists the algorithms and techniques used by the verification and validation tools (some techniques were omitted due to limited space). Further, Table 9 gives an overview of common solver libraries and frameworks used by these tools. Note that both tables are based on information provided by teams that registered individual tools to SV-COMP 2025 or to some previous editions in the case of currently inactive tools. The web site https://fm-tools.sosy-lab.org

Table 9: Solver libraries and frameworks used as components by at least three participating tools; tools that did not declare any used library or framework are omitted, ∅ for inactive, meta for meta verifiers, and new for first-time participants

Tool	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc	SMTinterpol	Z3	MiniSAT	Apron
2LS		✓									✓	
aise								✓		✓		
APRoVE										✓		
BRICK										✓	✓	
Bubaak										✓		
CBMC∅		✓									✓	
COASTAL∅				✓								
CPAchecker	✓					✓	✓					✓
CPALockator∅	✓					✓	✓					
CPA-bam-bnb∅	✓					✓	✓					
CPA-bam-smg∅	✓					✓	✓					
CPV											✓	
Crux∅										✓		
CSeq∅		✓									✓	
Dartagnan						✓						
Deagle											✓	
EBF∅			✓				✓					
ESBMC-incr			✓				✓					
ESBMC-kind			✓				✓					
GDart								✓		✓		
GDart-LLVM∅										✓		
Goblint												✓
Graves-CPA∅ meta	✓					✓	✓					
Hornix new										✓		
Java-Ranger				✓								
JBMC		✓									✓	
JDart∅				✓				✓		✓		
Korn										✓		
Lazy-CSeq∅		✓									✓	
Mopsa												✓
Nacpa meta new	✓											
PeSCo-CPA∅ meta	✓					✓	✓					
PIChecker∅	✓					✓	✓		✓			
SPF∅				✓								
SWAT										✓		

(continues on next page)

Table 9: Solver libraries and frameworks (continued)

Tool	CPAchecker	CProver	Esbmc	Jpf	Ultimate	JavaSMT	MathSAT	Cvc	SMTInterpol	Z3	MiniSAT	Apron
Symbiotic										✓		
UAutomizer					✓		✓	✓	✓	✓		
UGemCutter					✓		✓	✓	✓	✓		
UKojak					✓				✓	✓		
UReferee [new]					✓		✓	✓	✓	✓		
UTaipan					✓		✓	✓	✓	✓		
VeriAbs [∅]	✓	✓								✓	✓	
VeriAbsL [∅]	✓	✓								✓	✓	

provides more information about all tools evaluated in SV-COMP and Test-Comp since 2023 in a uniform way.

Figure 2 shows the evolution of the number of verifiers and validators participating in individual editions of SV-COMP. It seems that the number of actively participating tools, in particular verifiers, is now stable.

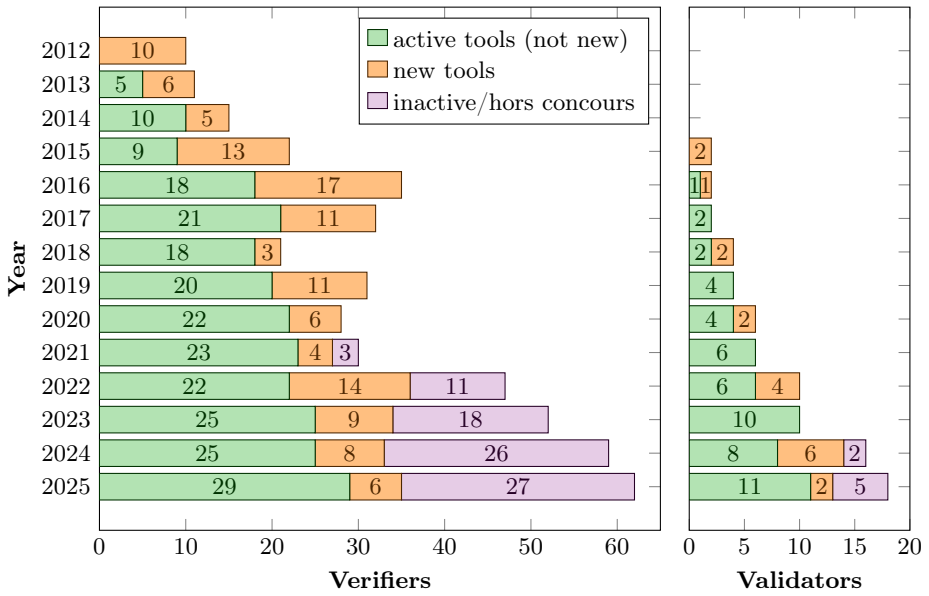

Fig. 2: Number of evaluated verifiers and validators in each year of SV-COMP; three new hors concours tools of 2022 counted only as new, not as hors concours

5 Results

The results of the competition represent 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* (the 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). 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 provided to the participants in advance and their objections have been settled.

Consumed Resources. Before we present the competition results, we report some statistics to give an impression of the overall computation work: One complete execution of all verifiers in the competition consisted of 942 284 verification runs (each verifier runs on each verification task of the categories listed in the verifier's *benchmark definition*), consuming 2 312 days of CPU time (without validation). Witness validation required 21.8 million validation runs (each validator runs on each validation task of the categories listed in the validator's *benchmark definition*) consuming 2 573 days of CPU time. Moreover, each tool was executed several times, in order to make sure no installation issues occur during the execution.

Verification track. Tables 10 and 11 present the quantitative overview of all tools and all meta categories except the meta categories included in *FalsificationOverall*. We split the presentation into two tables, one for the verifiers that participate with an active team support (Table 10), and one for the inactive verifiers (Table 11). The head row lists meta categories with the number of valid tasks and the maximal score for each category. The tools are listed in alphabetical order; every table row lists the scores of one verifier. An empty table cell means that the verifier did not participate in the corresponding meta category. In Table 10, we indicate the top three candidates in each category by formatting their scores in bold face and in larger font size. We recall that inactive tools and meta verifiers are excluded from rankings. 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/2025/results`) and in the results artifact (see Table 5). Note that the results for subcategories of *FalsificationOverall* are not explicitly presented neither in this report nor on the web due to their marginal significance. The results can be obtained from the detailed results of the corresponding categories in *Overall* presented on the web.

Table 12 shows the medalists for each meta category. The column 'Solved Tasks' shows the number of tasks that the respective verifier solved correctly and produced a witness that was confirmed (or not required). The cumulative run time of the verifier on these tasks is presented in the column 'CPU Time'. The column 'Unconf. Tasks' indicates the number of tasks for which the verifier returned a correct answer, but the corresponding witness was not confirmed by any validator. The columns 'False Alarms' and 'Wrong Proofs' provide the number of verification tasks for which the verifier reported wrong results, i.e., reporting a property

Table 10: Overview of the results of all actively participating verifiers; empty cells indicate opt-outs, meta for meta verifiers, new for first-time participants

Participant	**ReachSafety** 11268 tasks max. score 17860	**MemSafety** 4042 tasks max. score 6409	**ConcurrencySafety** 3175 tasks max. score 5733	**NoOverflows** 8211 tasks max. score 13297	**Termination** 2328 tasks max. score 4079	**SoftwareSystems** 4329 tasks max. score 7131	**FalsificationOverall** 30758 tasks max. score 10675	**Overall** 33353 tasks max. score 55561	**JavaOverall** 673 tasks max. score 926
2LS	6053	686	0	6887	1703	1	1938	12658	
AISE									
APROVE					**2219**				
BRICK									
BUBAAK	6684	3355	-190	6572	1491	1697	**5565**	17768	
BUBAAK-SPLIT	6053	3349	-189	6556	1122	1649	5562	16497	
COOPERACE meta new									
CPACHECKER	**10368**	**4892**	1770	8777	1301	**2178**	**6999**	**26786**	
CPV	**7755**								
DARTAGNAN			**3385**						
DEAGLE			**4604**						
EMERGENTHETA	2106			-361	620				
ESBMC-INCR			2155						
ESBMC-KIND	**8717**	3158	2155	8668	1115	-1948	3741	18444	
GOBLINT	2427	2198	2448	8486	969	545		17266	
HORNIX new									
KORN									
MOPSA	2807	2697	0	8491	0	**2086**		13521	
NACPA meta new	10270	4887	1453	8843	1290	2127	6997	26131	
PROTON					**3685**				
RACERF new									
SV-SANITIZERS		861		1723					
SVF-SVC new	-68717	-10965		-19469		0			
SYMBIOTIC	7097	**4479**	60	7704	1411	**1822**	**6459**	**20691**	
THETA	3277	600	2275	-170	712				
THORN new	-2519		298	-286	536				
UAUTOMIZER	5666	**3909**	2993	**11074**	**3334**	654	4762	**29710**	
UGEMCUTTER			**3144**						
UKOJAK	4935	2939	0	**8878**	0	300	3848	12872	
UTAIPAN	6007	3711	2593	**10736**	0	288	4695	20244	
GDART									**627**
JAVA-RANGER									**676**
JBMC									**628**
MLB									579
SWAT									508

Table 11: Overview of the results of all inactively participating verifiers; empty cells indicate opt-outs, ∅ for inactive, meta for meta verifiers

Verifier	**ReachSafety** 11268 tasks max. score 17860	**MemSafety** 4042 tasks max. score 6409	**ConcurrencySafety** 3175 tasks max. score 5733	**NoOverflows** 8211 tasks max. score 13297	**Termination** 2328 tasks max. score 4079	**SoftwareSystems** 4329 tasks max. score 7131	**FalsificationOverall** 30758 tasks max. score 10675	**Overall** 33353 tasks max. score 55561	**JavaOverall** 673 tasks max. score 926
CBMC ∅	1330	1885	819	7200	1199	-2586	-3581	9100	
CPA-bam-bnb ∅						-2370			
CPA-bam-smg ∅		3249				-4079			
CPALockator ∅			-4967						
Crux ∅	2133			608					
CSeq ∅			-12720						
DIVINE ∅	4629	502	351	0	0	72	352	3680	
EBF ∅			360						
Frama-C-SV ∅				1573					
Gazer-Theta ∅									
GDart-LLVM ∅									
Graves-CPA ∅ meta	4041					-670	-820	4508	
Infer ∅	-96489		-8970	-76213		-32987			
Lazy-CSeq ∅			-15153						
LF-checker ∅			396						
Locksmith ∅									
PeSCo-CPA ∅ meta	6269					-1598	2435	16328	
PIChecker ∅			459						
Pinaka ∅	2448			958	922				
PredatorHP ∅		4733							
VeriAbs ∅	11012								
VeriAbsL ∅	11224								
VeriOover ∅									
WitnessMap									
COASTAL ∅									-3960
JayHorn ∅									248
JDart ∅									-1224
SPF ∅									184

violation when the property holds (false alarm) and claiming that the program satisfies the property although it actually violates it (wrong proof), respectively.

Score-Based Quantile Functions. We use score-based quantile functions [18, 42] because these visualizations make it easier to understand the results of the

Table 12: Verification: Overview of the medalists in each meta category; values for CPU time in hours and rounded to two significant digits

Category Rank	Verifier	Score	CPU Time	Solved Tasks	Unconf. Tasks	False Alarms	Wrong Proofs
ReachSafety (11268 tasks, max. score 17860)							
1	**CPAchecker**	**10368**	150	6 653	230	2	
2	ESBMC-kind	8717	69	6 830	599		**14**
3	CPV	7755	160	6 235	438	27	
MemSafety (4042 tasks, max. score 6409)							
1	**CPAchecker**	**4892**	18	3 818	1		
2	Symbiotic	4479	2.3	3 671	0		**1**
3	UAutomizer	3909	37	2 280	2		**1**
ConcurrencySafety (3175 tasks, max. score 5733)							
1	**Deagle**	**4604**	3.1	2 500	38	1	**4**
2	Dartagnan	3385	17	2 012	30	3	**3**
3	UGemCutter	3144	50	1 805	48		
NoOverflows (8211 tasks, max. score 13297)							
1	**UAutomizer**	**11074**	68	6 724	13		
2	UTaipan	10736	74	6 622	11	1	**2**
3	UKojak	8878	54	5 910	2		
Termination (2328 tasks, max. score 4079)							
1	**Proton**	**3685**	24	1 942	159	1	
2	UAutomizer	3334	18	1 667	4		
3	AProVE	2219	32	1 006	43		
SoftwareSystems (4329 tasks, max. score 7131)							
1	**CPAchecker**	**2178**	30	2 022	55		
2	Mopsa	2086	20	2 212	0		
3	Symbiotic	1822	6.1	1 487	231		**1**
FalsificationOverall (30758 tasks, max. score 10675)							
1	**CPAchecker**	**6999**	100	7 100	67	2	
2	Symbiotic	6459	28	6 379	37		
3	Bubaak	5565	18	5 739	236	9	
Overall (33353 tasks, max. score 55561)							
1	**UAutomizer**	**29710**	270	16 677	196		**8**
2	CPAchecker	26786	240	20 506	372	6	**1**
3	Symbiotic	20691	63	14 324	628		**3**
JavaOverall (673 tasks, max. score 926)							
1	**Java-Ranger**	**676**	5.7	491	13		**1**
2	JBMC	628	0.32	430	91		
3	GDart	627	2.1	460	15		

comparative evaluation. The results archive (see Table 5) and the web site (`https://sv-comp.sosy-lab.org/2025/results`) include such a plot for each category. As an example, we show the plot for category *Overall* (all verification tasks in C) in Fig. 3. A total of 16 verifiers participated in category *Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [18]). A more detailed discussion of score-based quan-

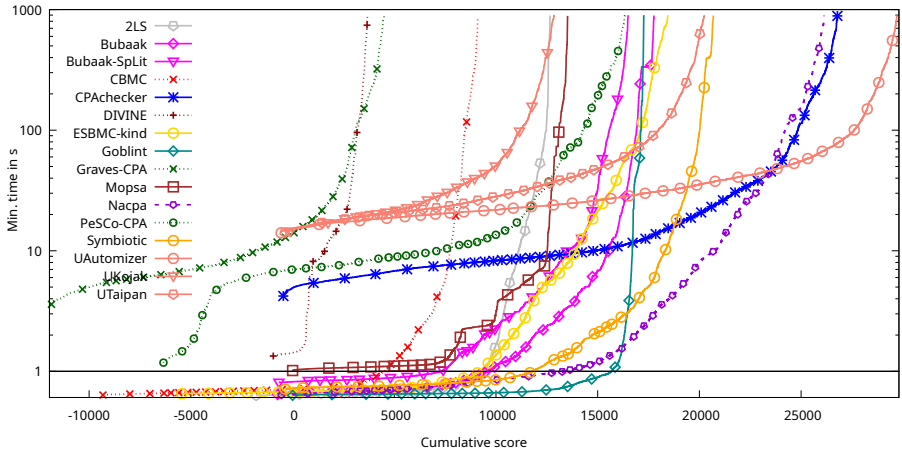

Fig. 3: Quantile functions for category *Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by correct verification runs below a certain run time (y-coordinate), minus the overall penalty for incorrect results. More details were given previously [18]. A logarithmic 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.

tile plots, including examples of what insights one can obtain from the plots, is provided in previous competition reports [18, 21]. Since this year we use different lines to distinguish regular active participants (solid line), active meta verifiers (dashed line), and inactive verifiers (dotted line).

The graph shows that the *Overall* winner is UAutomizer as its graph end most to the right. The graph for UAutomizer also starts left from $x = 0$ because the verifier produced 8 wrong proofs and therefore received some negative points. Also other verifiers whose graphs start with a negative cumulative score produced wrong results.

Validation Track. Validation of verification witnesses was pioneered by SV-COMP in 2015 (by the tools CPAchecker and CBMC$^{\varnothing}$, see the documentation for the early attempts: `https://sv-comp.sosy-lab.org/2015/witnesses/`). Shortly after that, verification witnesses become more and more important for various reasons: they do not only justify and help to understand and interpret verification results, but they also serve as exchange object for intermediate results and allow to make use of imprecise verification techniques (e.g., via machine learning). However, a case study on the quality of the results of witness validators [46] published in 2022 revealed a great potential for improvements. To stimulate further advances in verification witnesses and their verifiers, the study suggested that witness validators should also undergo a periodical comparative evaluation and proposed a scoring schema for witness-validation results. This materializes in the validation track run by SV-COMP since 2023.

Table 13: Validation of correctness witnesses (version 1.0): Overview of the medalists in each meta category; values for CPU time in hours and rounded to two significant digits, new for first-time participants

Rank	Validator	Score	CPU Time	Solved Tasks	Wrong Confirm.	Wrong Refut.
Category						
ReachSafety (39573 tasks, max. score 70152)						
1	UAutomizer	**52303**	390	16 576		4
2	MetaVal	49340	750	24 887		53
3	CPAchecker	40427	460	33 426		
NoOverflows (38846 tasks, max. score 77692)						
1	UAutomizer	**77578**	280	38 812		
2	CPAchecker	75470	140	38 181		
3	LIV	16438	5.6	4 920		
SoftwareSystems (13913 tasks, max. score 25507)						
1	UAutomizer	**20171**	230	13 572		
2	MetaVal	15487	290	13 783		5
3	UReferee new	14072	48	5 399		
Overall (92332 tasks, max. score 172540)						
1	UAutomizer	**146763**	900	68 960		4
2	CPAchecker	113930	620	76 991		
3	MetaVal	72634	1 000	38 670		58

Thanks to the development and adoption of the version 2.0 of the witness format [7], the validation track has now four regular subtracks:

1. validation of correctness witnesses in format version 1.0 (see Table 13),
2. validation of violation witnesses in format version 1.0 (see Table 14),
3. validation of correctness witnesses in format version 2.0 (see Table 15), and
4. validation of violation witnesses in format version 2.0 (see Table 16).

The subtrack with violation witnesses 2.0 ran for the first time as a regular part of the competition as it was only a demonstration subtrack in SV-COMP 2024. The tables present only the medalists for all non-empty meta categories of each subtrack. If some category has less than 3 medalists, it means that less than three validators reached a positive score. The column 'Wrong Confirm.' gives the number of cases when the respective validator confirmed a witness that was incorrect. The column 'Wrong Refut.' shows the number of cases when the respective validator refuted a correct witness. We recall that the correctness or incorrectness of many witnesses used in the validation track is determined by voting and thus the numbers of wrong confirmations and refutation do not have to be completely objective.

The complete results of all validators in all relevant categories of all subtracks are available in the results artifact (see Table 5) and on the SV-COMP web site (`https://sv-comp.sosy-lab.org/2025/results/results-validated/`).

The limited support of some properties and program features by the witness formats (see Table 1), missing medalists in some categories, and the negative scores

Table 14: Validation of violation witnesses (version 1.0): Overview of the medalists in each meta category; values for CPU time in hours and rounded to two significant digits

Category Rank	Validator	Score	CPU Time	Solved Tasks	Wrong Confirm.	Wrong Refut.
ReachSafety (20391 tasks, max. score 29907)						
1	ConcurrentW2T	**1931**	2.2	2 734	4	
MemSafety (8810 tasks, max. score 13215)						
1	CPAchecker	**4455**	19	8 638	22	27
ConcurrencySafety (755 tasks, max. score 1132)						
1	Dartagnan	**777**	2.9	693		
2	CPAchecker	511	1.6	531		3
3	UAutomizer	478	4.9	567		104
NoOverflows (17697 tasks, max. score 26546)						
1	Symbiotic-Witch	**3926**	14	13 326	185	2
Termination (2057 tasks, max. score 2314)						
1	CPAchecker	**2314**	9.5	2 057		
2	UAutomizer	2314	16	2 057		
SoftwareSystems (1334 tasks, max. score 2001)						
1	CPAchecker	**1356**	5.8	1 086		1
2	Symbiotic-Witch	586	0.38	653	1	3
3	MetaVal	535	5.6	633	9	
Overall (51044 tasks, max. score 73092)						
-	demo category (less than three participants)					

in the detailed results on the web site: all of these show the need of further research and development in the are of software verification witnesses and their validation.

6 Conclusion

The 14th edition of the Competition on Software Verification (SV-COMP 2025) compared 62 automatic tools for software verification (including 6 new ones and 27 tools without active team support) and 18 automatic tools for validation of verification witnesses (including 2 new and 5 tools without active team support). The overall numbers of evaluated verifiers and validators were historically the highest (see Fig. 2). The total number of verification tasks in SV-COMP 2025 for both C and Java programs was significantly increased to precisely 34 700 (including one demo category).

The results of the competition show a progress in both verification and witness validation area, especially in the adoption of the witness format 2.0. However, even the best verification and validation tools still produce some incorrect results. This motivates further improvements of verifiers, but also extensions of the witness format to support more properties and program features, and development of validators.

Table 15: Validation of correctness witnesses (version 2.0): Overview of the medalists in each meta category; values for CPU time in hours and rounded to two significant digits

Category Rank	Validator	Score	CPU Time	Solved Tasks	Wrong Confirm.	Wrong Refut.
ReachSafety (19365 tasks, max. score 28241)						
1	UAutomizer	**16085**	200	7 719	2	10
2	CPAchecker	10378	180	14 467	2	
3	LIV	6829	50	12 667	16	
NoOverflows (26686 tasks, max. score 40029)						
1	UAutomizer	**32637**	270	23 890		93
2	CPAchecker	26940	60	18 931		
3	Mopsa	25022	12	21 430		
SoftwareSystems (7581 tasks, max. score 11913)						
1	Mopsa	**7751**	29	6 948		
2	CPAchecker	5092	12	2 474		
3	MetaVal	3182	60	2 880	2	
ValidationCrafted (3 tasks, max. score 6)						
1	UAutomizer	**6**	0.043	3		
2	CPAchecker	6	0.16	3		
Overall (53635 tasks, max. score 87557)						
1	UAutomizer	**57804**	580	37 976	3	134
2	CPAchecker	56546	260	35 875	2	
3	Mopsa	30743	95	32 857		

Data-Availability Statement. The verification tasks and results of the competition are published at Zenodo, as described in Table 5. All components and data that are necessary for reproducing the competition are available in public version repositories, as specified in Table 4. For easy access, the results are presented also online on the competition web site `https://sv-comp.sosy-lab.org/2025`. The main results of SV-COMP 2025 were reproduced in an independent reproduction report [14].

Funding Statement. SV-COMP 2025 was supported by Huawei – Dresden Research Center, Germany. Some participants of this competition were funded in part by the Deutsche Forschungsgemeinschaft (DFG) — 378803395 (ConVeY). Jan Strejček has been supported by the Czech Science Foundation grant GA23-06506S.

Acknowledgments. We thank the organization committee for their help in running the competition, the jury for their work on improving the quality of the verification tasks and for their advice in refining and apply the competition rules, and the verification community for contributing their tools to the evaluation.

Table 16: Validation of violation witnesses (version 2.0): Overview of the medalists in each meta category; values for CPU time in hours and rounded to two significant digits

Category Rank	Validator	Score	CPU Time	Solved Tasks	Wrong Confirm.	Wrong Refut.
ReachSafety (3548 tasks, max. score 4494)						
1	Witch	**3259**	6.1	3 305		
2	CPAchecker	3081	15	3 083	8	
MemSafety (107 tasks, max. score 147)						
1	UAutomizer	**147**	0.53	107		
2	Witch	104	0.040	84		
3	CPAchecker	43	0.043	23		
NoOverflows (4706 tasks, max. score 5882)						
1	UAutomizer	**5147**	27	4 691	1	
2	CPAchecker	5010	11	4 475		
3	Witch	3900	4.7	4 584		
SoftwareSystems (148 tasks, max. score 222)						
1	Witch	**222**	0.12	148		
2	CPAchecker	74	0.70	140		
3	MetaVal	73	0.97	135		
ValidationCrafted (100 tasks, max. score 150)						
1	Witch	**150**	0.042	100		
Overall (8609 tasks, max. score 11866)						
1	Witch	**9850**	11	8 221		
2	MetaVal	162	11	1 846	2	51

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. pp. 184–203. LNCS 14624, Springer (2024). https://doi.org/10.1007/978-3-031-66149-5_11
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 (3). 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., Jakobs, M.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Wachowitz, H., Wendler, P.: Software verification with CPAchecker 3.0: Tutorial and user guide. In: Proc. FM. pp. 543–570. LNCS 14934, Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_30
11. 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 (3). pp. 359–364. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_21
12. 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 (3). pp. 371–375. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_23
13. Bajczi, L., Ádám, Z., Micskei, Z.: ConcurrentWitness2Test: Test-harnessing the power of concurrency (competition contribution). In: Proc. TACAS (3). pp. 330–334. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_16
14. Bajczi, L., Ádám, Z., Micskei, Z.: SV-COMP'25 reproduction report (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
15. 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
16. 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
17. 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
18. 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
19. 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
20. 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
21. 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

22. 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
23. 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
24. 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
25. 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
26. 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
27. 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
28. 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
29. 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
30. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15
31. Beyer, D.: FM-Tools Release 2.2: Data set of metadata about tools for formal methods (SV-COMP 2025, Test-Comp 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15055359
32. Beyer, D., Strejček, J.: Results of the 14th Intl. Competition on Software Verification (SV-COMP 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15012085
33. Beyer, D., Strejček, J.: SV-Benchmarks: Benchmark set for software verification (SV-COMP 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15012096
34. Beyer, D., Strejček, J.: Verification witnesses from verification tools (SV-COMP 2025). Zenodo (2025). https://doi.org/10.5281/zenodo.15012077
35. Beyer, D., Chien, P.C., Jankola, M.: BenchCloud: A platform for scalable performance benchmarking. In: Proc. ASE. pp. 2386–2389. ACM (2024). https://doi.org/10.1145/3691620.3695358
36. 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
37. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022). https://doi.org/10.1145/3477579
38. 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
39. 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

40. 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
41. Beyer, D., Lingsch-Rosenfeld, M.: CPAchecker Validator 4.0 (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
42. 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
43. 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
44. 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
45. 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
46. 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
47. Beyer, D., Wachowitz, H.: Coveriteam Release 1.2.1. Zenodo (2024). https://doi.org/10.5281/zenodo.11193690
48. 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
49. 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
50. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. J. ACM **58**(6), 26:1–26:66 (2011). https://doi.org/10.1145/2049697.2049700
51. 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
52. Chalupa, M., Richter, C.: Bubaak-SpLit: Split what you cannot verify (competition contribution). In: Proc. TACAS (3). pp. 353–358. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_20
53. Chalupa, M., Richter, C.: Bubaak: Dynamic cooperative verification (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
54. 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
55. 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
56. Cheng, Z., Wu, T., Schrammel, P., Tihanyi, N., de Lima Filho, E.B., Cordeiro, L.C.: JCWIT: A correctness-witness validator for Java programs based on bounded model checking. In: Proc. ISSTA. pp. 1831–1835. ACM (2024). https://doi.org/10.1145/3650212.3685303

57. Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_22
58. 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
59. 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
60. 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
61. 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
62. 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
63. 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
64. 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
65. Dacík, T., Vojnar, T.: RacerF: Data race detection with Frama-C (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
66. 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
67. 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
68. 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
69. 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
70. 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
71. 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
72. 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

73. 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
74. 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
75. 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
76. 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
77. 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
78. 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
79. 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
80. 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
81. 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 (3). pp. 418–423. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_31
82. 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
83. 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
84. 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
85. 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
86. 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
87. 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

88. 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
89. 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
90. 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 (3). pp. 406–411. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_29
91. 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)
92. 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
93. 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
94. 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
95. 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
96. 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
97. 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
98. Lemberger, T., Wachowitz, H.: Nacpa: Native checking with parallel-portfolio analyses (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
99. Lin, Y., Chen, Z., Wang, J.: AISE v2.0: Combining loop transformations (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
100. Lommen, N., Giesl, J.: AProVE (KoAT + LoAT) (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
101. Loose, N., Mächtle, F., Sieck, F., Eisenbarth, T.: SWAT: Modular dynamic symbolic execution for Java applications using dynamic instrumentation (competition contribution). In: Proc. TACAS (3). pp. 399–405. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_28
102. 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
103. 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

104. McGowan, C., Richards, M., Sui, Y.: SVF-SVC: Software verification using SVF (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
105. Metta, R., Karmarkar, H., Madhukar, K., Venkatesh, R., Chakraborty, S.: Proton: Probes for non-termination and termination (competition contribution). In: Proc. TACAS (3). pp. 393–398. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_27
106. 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 (3). pp. 387–392. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_26
107. Monat, R., Ouadjaout, A., Miné, A.: Mopsa-C with trace partitioning and autosuggestions (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
108. Mondok, M., Bajczi, L., Szekeres, D., Molnár, V.: EmergenTheta: Variations on symbolic transition systems (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
109. 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
110. 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
111. Mukhopadhyay, D., Metta, R., Karmarkar, H., Madhukar, K.: PROTON 2.1: Synthesizing ranking functions via fine-tuned locally hosted LLM (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
112. 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
113. 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
114. 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
115. 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
116. 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
117. 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
118. 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

119. 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
120. 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
121. 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 (3). pp. 381–386. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_25
122. 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 (3). pp. 335–340. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_17
123. 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
124. 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
125. 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
126. 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
127. Telbisz, C., Bajczi, L., Szekeres, D., Vörös, A.: Theta: Various approaches for concurrent program verification (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
128. 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
129. 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
130. 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
131. 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
132. Wang, Z., Chen, Z.: AISE: A symbolic verifier by synergizing abstract interpretation and symbolic execution (competition contribution). In: Proc. TACAS (3). pp. 347–352. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_19
133. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.29. Zenodo (2025). https://doi.org/10.5281/zenodo.15007216

134. Wu, T., Li, X., Manino, E., Menezes, R., Gadelha, M., Xiong, S., Tihanyi, N., Petoumenos, P., Cordeiro, L.: ESBMC v7.7: Efficient concurrent software verification with scheduling, incremental SMT and partial order reduction (competition contribution). In: Proc. TACAS (3). LNCS 15698, Springer (2025)
135. 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
136. 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

Open Access. This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.