# Software Verification: 10th Comparative Evaluation (SV-COMP 2021)

Dirk Beyer

LMU Munich, Munich, Germany


**Abstract.** SV-COMP 2021 is the 10th edition of the Competition on Software Verification (SV-COMP), which is an annual comparative evaluation of fully automatic software verifiers for C and Java programs. The competition provides a snapshot of the current state of the art in the area, and has a strong focus on reproducibility of its results. The competition was based on 15 201 verification tasks for C programs and 473 verification tasks for Java programs. Each verification task consisted of a program and a property (reachability, memory safety, overflows, termination). SV-COMP 2021 had 30 participating verification systems from 27 teams from 11 countries.

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

## 1 Introduction

Among several other objectives, the Competition on Software Verification (SV-COMP, https://sv-comp.sosy-lab.org/2021) showcases the state of the art in the area of automatic software verification. This edition of SV-COMP is already the 10th edition of the competition and presents again an overview of the currently achieved results by tool implementations that are based on the most recent ideas, concepts, and algorithms for fully automatic verification. This competition report describes the (updated) rules and definitions, presents the competition results, and discusses some interesting facts about the execution of the competition experiments. The objectives of the competitions were discussed earlier (1-4 [16]) and extended over the years (5-6 [17]):

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

This report extends previous reports on SV-COMP [10, 11, 12, 13, 14, 15, 16, 17].
Reproduction packages are available on Zenodo (see Table 4).
Funded in part by the Deutsche Forschungsgemeinschaft (DFG) – 378803395 (ConVeY).
✉ dirk.beyer@sosy-lab.org

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

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

**Related Competitions.** Competitions are an important evaluation method and there are many competitions in the field of formal methods. We refer to the previous report [17] for a more detailed discussion and give here only the references to the most related competitions [9, 19, 55, 56].

**Quick Summary of Changes.** We strive to continuously improve the competition, and this report describes the changes of the last year. In the following we list a brief summary of new items in SV-COMP 2021:

- SPDX identification of licenses in SV-Benchmarks collection
- WitnessLint: New checker for syntactical validity of verification witnesses
- Upgrade of the task-definition format to version 2.0
- Addition of several verification tasks and whole new sub-categories to the SV-Benchmarks collection
- Elimination of competition-specific functions `__VERIFIER_error` and `__VERIFIER_assume` from the verification tasks (and rules)
- Change in scoring schema: Unconfirmed results not counted anymore (when validation was applied)
- CoVeriTeam: New tool that can be used to remotely execute verification runs on the competition machines
- Automatic participation of previous verifiers

## 2 Organization, Definitions, Formats, and Rules

**Procedure.** The overall organization of the competition did not change in comparison to the earlier editions [10, 11, 12, 13, 14, 15, 16, 17]. 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 pre-runs and the final competition run), and the results were publicly announced on the competition web site after the teams inspected them. The *Competition Jury* oversees the process and consists of the competition chair and one member of each participating team. Team representatives of the jury are listed in Table 5.

Table 1: Tools for witness-based result validation (validators) and witness linter

| Validator | References | Represent./Developer | Affiliation |
|---|---|---|---|
| CPAchecker | [22, 23, 25] | Martin Spiessl | LMU Munich, Germany |
| UAutomizer | [22, 23] | Daniel Dietsch | Uni Freiburg, Germany |
| CPA-w2t | [24] | Thomas Lemberger | LMU Munich, Germany |
| FShell-w2t | [24] | Michael Tautschnig | Queen Mary U. of London, UK |
| Nitwit | [78] | Philipp Berger | RWTH Aachen, Germany |
| MetaVal | [29] | Martin Spiessl | LMU Munich, Germany |
| WitnessLint | | Sven Umbricht | LMU Munich, Germany |

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

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

During the qualification phase, when the jury members inspect the verifier archives, several issues with licenses (missing licenses, incompatibilities) were detected that the developers were able to address the issues on time.

With SV-COMP 2021, the community started the process of making the benchmark collection REUSE compliant (`https://reuse.software`) by adding SPDX license identifiers (`https://spdx.dev`). A few directories are properly labeled already, and continuous-integration checks with REUSE ensure that new contributions adhere to the standard.

**Validation of Results.** This time, the validation of the verification results was done by seven validation tools, which are listed in Table 1, including references to literature. The validators CPAchecker and UAutomizer support the competition since the beginning of its result validation in 2015. Execution-based validation was added in 2018 using CPA-w2t and FShell-w2t. Two new validators participated since the previous SV-COMP in 2020: Nitwit and MetaVal. A few categories were still excluded from validation because no validators were available for some types of programs or properties.

For SV-COMP 2021, the new validator WitnessLint was added for validating witnesses regarding their syntax. It checks the witnesses produced by the verification tools against the specification of the format for verification witnesses (`https://github.com/sosy-lab/sv-witnesses/tree/svcomp21`). For example, WitnessLint ensures that a verification witness is a proper XML/GraphML file and contains the required meta data. This means that the validators can focus on the validation of the verification result, assuming that the verification witness is syntactically valid. If the witness linter deems a verification witness as syntactically invalid, then the answers of the result validators are ignored and the result is not counted as confirmed.

**Task-Definition Format 2.0.** The format for the task definitions in the SV-Benchmarks repository was recently extended to include a set of

options that can carry information from the verification task to the verification tool. SV-COMP 2021 used the task-definition format in version 2.0 (https://gitlab.com/sosy-lab/benchmarking/task-definition-format/-/tree/2.0). More details can be found in the report for Test-Comp 2021 [19].

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

**Categories.** The updated category structure is illustrated by Fig. 1. The categories are also listed in Tables 7 and 8, and described in detail on the competition web site (https://sv-comp.sosy-lab.org/2021/benchmarks.php). Compared to the category structure for SV-COMP 2020, we added the sub-categories *XCSP* and *Combinations* to category *ReachSafety*, and the sub-categories *DeviceDriversLinux64Large ReachSafety*, *uthash MemSafety*, *uthash NoOverflows*, and *uthash ReachSafety* to category *SoftwareSystems*.

Another effort was to integrate some of the Juliet benchmark tasks [31] into the SV-Benchmarks collection. We requested a license for the Juliet programs that properly clarifies the license terms also outside the USA. We thank our colleagues from NIST for releasing their Juliet benchmark (which is declared as public domain) under the Creative Commons license CC0-1.0 (https://github.com/sosy-lab/sv-benchmarks/blob/svcomp21/LICENSES/CC0-1.0.txt). SV-COMP 2021 used many verification tasks from Juliet, in particular for the memory-safety properties CWE121 (stack-based buffer overflow), CWE401 (memory leak), CWE415 (double free), CWE476 (null-pointer dereference), and CWE590 (free memory that is not on the heap) (see https://github.com/sosy-lab/sv-benchmarks/blob/svcomp21/c/MemSafety-Juliet.set).

All those new contributions to the benchmark collection lead to the growth of the number of verification tasks from 11 052 in SV-COMP 2020 to 15 201 in SV-COMP 2021.

**Verification Tasks.** The previous verification tasks and competition rules used special definitions for the functions `__VERIFIER_error` and `__VERIFIER_assume`. These special definitions were found to be unintuitive and inconsistent with expectations in the verification community, and repeatedly caused confusion among participants. A call of function `__VERIFIER_error()` was defined to never return. A call of function `__VERIFIER_assume(p)` was defined such that if expression $p$ evaluates to false, then the function loops forever, otherwise the function returns without any side effects. This led to unintended interactions with other properties.

We eliminated these two functions in two steps. In the first step, each function call was replaced by a C-code implementation of the intended behavior. In most of the cases, `__VERIFIER_error();` was replaced by the C code `reach_error(); abort();`, where `reach_error` is a 'normal' function, i.e., one whose interpretation follows the C standard [3].

Eliminating `__VERIFIER_assume` was more complicated: In some tasks for property *memory-cleanup*, `__VERIFIER_assume(p);` was replaced by the C code `assume_cycle_if_not(p);`, which is implemented

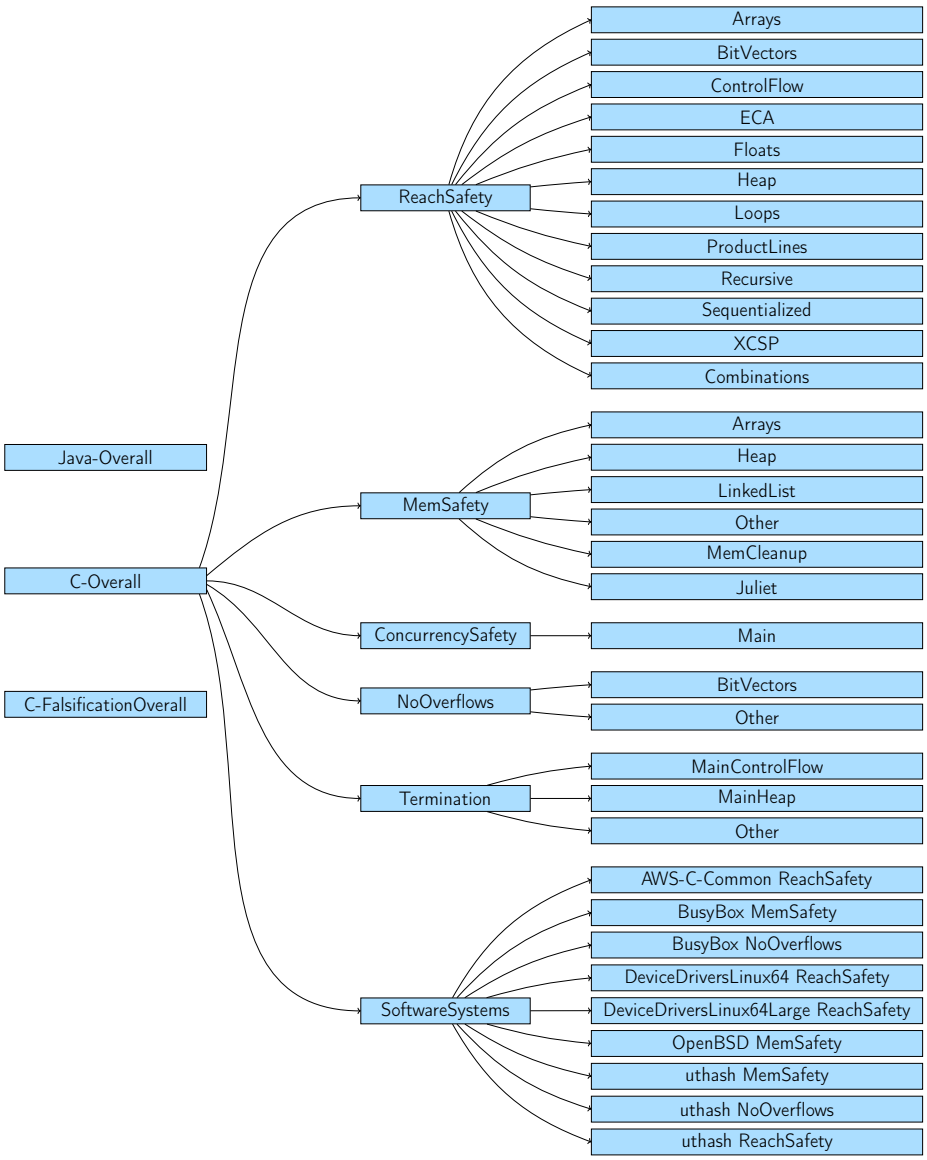


Fig. 1: Category structure for SV-COMP 2021; category *C-FalsificationOverall* contains all verification tasks of *C-Overall* without *Termination*; *Java-Overall* contains all Java verification tasks; compared to SV-COMP 2020, there are two new sub-categories in *ReachSafety* and four new sub-categories in *SoftwareSystems*

Table 2: Scoring schema for SV-COMP 2021 (new: no point for unconfirmed correct results anymore)

| 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) |

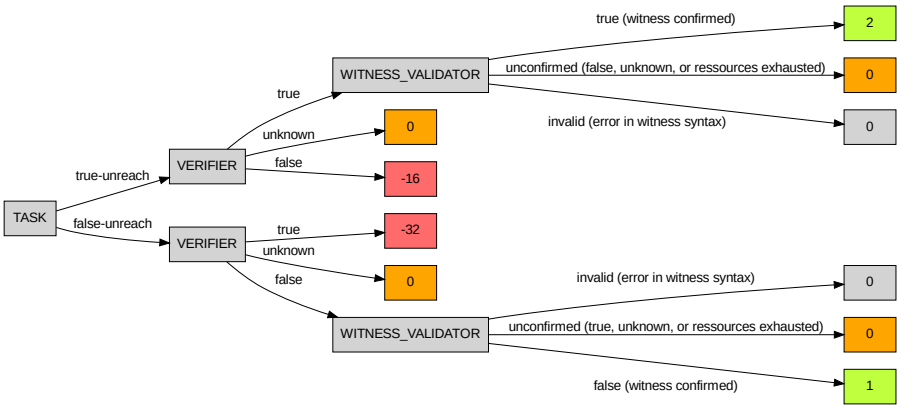


Fig. 2: Visualization of the scoring schema for the reachability property (adjusted from a previous report [15])

as `if (!p) while(1);`, while for other tasks, `__VERIFIER_assume(p);` was replaced by `assume_abort_if_not(p);`, which is implemented as `if (!p) abort();`. The solution nicely illustrates the problem of the special semantics: Consider property *memory-cleanup*, which requires that all allocated memory is deallocated before the program terminates. Here, the desired behavior of a failing assume statement would be that the program does not terminate (and does not unintendedly violate the *memory-cleanup* property). Now consider property *termination*, which requires that every path finally reaches the end of the program. Here, the desired behavior of a failing assume statement would be that the program terminates (and does not unintendedly violate the *termination* property).

In the second step, the specifications for functions `__VERIFIER_error` and `__VERIFIER_assume` were removed from the competition rules (because no such functions exist anymore in the SV-Benchmarks collection).

**Scoring Schema and Ranking.** Table 2 provides an overview and Fig. 2 visually illustrates the score assignment for the reachability property as an example.

The scoring schema was changed regarding the special rule for unconfirmed correct results for expected result TRUE. There was a rule during the transitioning phase to assign one point if the answer matches the expected result but the witness was not confirmed. Now score points are only assigned if the results got validated (or no validator was available).

As in the last years, the rank of a verifier was decided based on the sum of points (normalized for meta categories). In case of a tie, the rank was decided based on success run time, which is the total CPU time over all verification tasks for which the verifier reported a correct verification result. *Opt-out from Categories* and *Score Normalization for Meta Categories* was done as described previously [11] (page 597).

## 3 Reproducibility

To allow independent reproduction of the SV-COMP results, we made all major components that were used in the competition available in public version-control repositories. An overview of the components that contribute to the reproducible setup of SV-COMP is provided in Fig. 3, and the details are given in Table 3. We refer to the SV-COMP 2016 report [14] for a description of all components of the SV-COMP organization.

We have published the competition artifacts at Zenodo (see Table 4) to guarantee their long-term availability and immutability. These artifacts comprise the verification tasks, the competition results, the produced verification witnesses, and the BENCHEXEC package. The archive for the competition results includes the raw results in BENCHEXEC's XML exchange format, the log output of the verifiers and validators, and a mapping from file names to SHA-256 hashes. The hashes of the files are useful for validating the exact contents of a file, and accessing the files inside the archive that contains the verification witnesses.

**Competition Workflow.** The workflow of the competition is described in the report for Test-Comp 2021 [19].

**CoVeriTeam.** The competition was for the first time supported by COVERITEAM [26] (`https://gitlab.com/sosy-lab/software/coveriteam/`), which is a tool for cooperative verification. Among its many capabilities, it enables remote execution of verification runs directly on the competition machines, which was found to be a valuable service for trouble shooting.

## 4 Results and Discussion

The results of the competition experiments represent the state of the art in fully automatic software-verification tools. The report shows the results, in terms of effectiveness (number of verification tasks that can be solved and correctness of the results, as accumulated in the score) and efficiency (resource consumption in terms of CPU time and CPU energy). The results are presented in the same way as in last years, such that the improvements compared to last year are easy

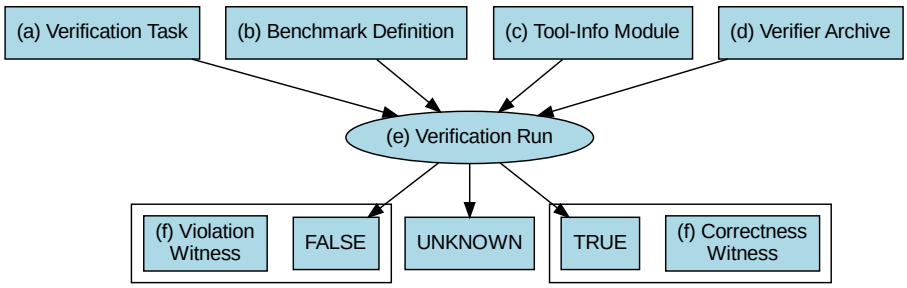


Fig. 3: Benchmarking components of SV-COMP and competition's execution flow (same as for SV-COMP 2020)

Table 3: Publicly available components for reproducing SV-COMP 2021

| Component | Fig. 3 | Repository | Version |
|---|---|---|---|
| Verification Tasks | (a) | github.com/sosy-lab/sv-benchmarks | svcomp21 |
| Benchmark Definitions | (b) | gitlab.com/sosy-lab/sv-comp/bench-defs | svcomp21 |
| Tool-Info Modules | (c) | github.com/sosy-lab/benchexec | 3.6 |
| Verifier Archives | (d) | gitlab.com/sosy-lab/sv-comp/archives-2021 | svcomp21 |
| Benchmarking | (e) | github.com/sosy-lab/benchexec | 3.6 |
| Witness Format | (f) | github.com/sosy-lab/sv-witnesses | svcomp21 |

Table 4: Artifacts published for SV-COMP 2021

| Content | DOI | Reference |
|---|---|---|
| Verification Tasks | 10.5281/zenodo.4459126 | [20] |
| Competition Results | 10.5281/zenodo.4458215 | [18] |
| Verification Witnesses | 10.5281/zenodo.4459196 | [21] |
| BenchExec | 10.5281/zenodo.4317433 | [82] |

to identify. The results presented in this report were inspected and approved by the participating teams. We now discuss the highlights of the results.

**Participating Verifiers.** Table 5 provides an overview of the participating verification systems (see also the listing on the competition web site at https://sv-comp.sosy-lab.org/2021/systems.php). Table 6 lists the algorithms and techniques that are used by the verification tools.

**Automatic Participation.** To ensure that the comparative evaluation continues to give an overview of the state of the art that is as broad as possible, a rule was introduced before SV-COMP 2020 which enables the option for the organizer to reuse systems that participated in previous years for the comparative evaluation. This option was used three times in SV-COMP 2021: for Coastal, PredatorHP, and SPF. Those participations are marked as 'hors concours' in Table 5.

Table 5: Competition candidates with tool references and representing jury members

| Participant | Ref. | Jury member | Affiliation |
|---|---|---|---|
| 2LS | [32, 63] | Viktor Malík | BUT, Brno, Czechia |
| Brick | | Lei Bu | Nanjing U., China |
| Cbmc | [60] | Michael Tautschnig | Queen Mary U. of London, UK |
| Coastal | [79] | (hors concours) | – |
| CPA-BAM-BnB | [4, 81] | Vadim Mutilin | ISP RAS, Russia |
| CPALockator | [5, 6] | Pavel Andrianov | ISP RAS, Russia |
| CPAchecker | [27, 41] | Stephan Holzner | LMU Munich, Germany |
| Dartagnan | [48, 68] | Hernán Ponce de León | U. Bundeswehr Munich, Germany |
| Divine | [8, 61] | Henrich Lauko | Masaryk U., Brno, Czechia |
| ESBMC-incr | [36, 39] | Felipe R. Monteiro | Amazon Web Services, USA |
| ESBMC-kind | [46, 47] | Lucas Cordeiro | U. of Manchester, UK |
| Frama-C | [40] | Martin Spiessl | LMU Munich, Germany |
| Gazer-Theta | [1, 74] | Ákos Hajdu | BME, Hungary |
| Goblint | [73, 80] | Simmo Saan | U. of Tartu, Estonia |
| Java Ranger | [76, 77] | Soha Hussein | U. of Minnesota, USA |
| JayHorn | [59, 75] | Hossein Hojjat | U. of Tehran, Iran |
| Jbmc | [37, 38] | Peter Schrammel | U. of Sussex / Diffblue, UK |
| JDart | [62, 64] | Falk Howar | TU Dortmund, Germany |
| Korn | [45] | Gidon Ernst | LMU Munich, Germany |
| Lazy-CSeq | [57, 58] | Omar Inverso | Gran Sasso Science Institute, Italy |
| PeSCo | [71, 72] | Cedric Richter | Paderborn U., Germany |
| Pinaka | [35] | Saurabh Joshi | IIT Hyderabad, India |
| PredatorHP | [54, 67] | (hors concours) | – |
| Smack | [51, 70] | Zvonimir Rakamaric | U. of Utah, USA |
| SPF | [65, 69] | (hors concours) | – |
| Symbiotic | [33, 34] | Marek Chalupa | Masaryk U., Brno, Czechia |
| UAutomizer | [52, 53] | Matthias Heizmann | U. of Freiburg, Germany |
| UKojak | [44, 66] | Dominik Klumpp | U. of Freiburg, Germany |
| UTaipan | [43, 49] | Daniel Dietsch | U. of Freiburg, Germany |
| VeriAbs | [2, 42] | Priyanka Darke | Tata Consultancy Services, India |

**Computing Resources.** The resource limits were the same as in the previous competitions [14]: Each verification run was limited to 8 processing units (cores), 15 GB of memory, and 15 min of CPU time. Witness validation was limited to 2 processing units, 7 GB of memory, and 1.5 min of CPU time for violation witnesses and 15 min of CPU time for correctness witnesses. The machines for running the experiments are part of a compute cluster that consists of 168 machines; each verification run was executed on an otherwise completely unloaded, dedicated machine, in order to achieve precise measurements. Each machine had one Intel Xeon E3-1230 v5 CPU, with 8 processing units each, a frequency of 3.4 GHz, 33 GB of RAM, and a GNU/Linux operating system (x86_64-linux, Ubuntu 20.04 with Linux kernel 5.4). We used BenchExec [28] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud (`https://vcloud.sosy-lab.org`) to distribute, install, run, and clean-up verification runs, and to collect the results. The values for time and

Table 6: Algorithms and techniques that the competition candidates used

| Participant | CEGAR | Predicate Abstraction | Symbolic Execution | Bounded Model Checking | k-Induction | Property-Directed Reach. | Explicit-Value Analysis | Numeric. Interval Analysis | Shape Analysis | Separation Logic | Bit-Precise Analysis | ARG-Based Analysis | Lazy Abstraction | Interpolation | Automata-Based Analysis | Concurrency Support | Ranking Functions | Evolutionary Algorithms | Algorithm Selection | Portfolio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2LS | | | | ✓ | ✓ | | | ✓ | ✓ | | ✓ | | | | | | ✓ | | | |
| BRICK | ✓ | | ✓ | ✓ | | | | ✓ | | | | | | | | ✓ | | | | |
| CBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| COASTAL | | | ✓ | | | | | | | | | | | | | | | | | |
| CPA-BAM-BNB | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | | | |
| CPALOCKATOR | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | | | | |
| CPACHECKER | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| DARTAGNAN | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| DIVINE | | | ✓ | | | | ✓ | | | | ✓ | | | | | ✓ | | | ✓ | ✓ |
| ESBMC-INCR | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | | | |
| ESBMC-KIND | | | | ✓ | ✓ | | | ✓ | | | ✓ | | | | | ✓ | | | | |
| FRAMA-C | | | | | | | | ✓ | | | | | | | | | | | | |
| GAZER-THETA | ✓ | ✓ | | ✓ | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ |
| GOBLINT | | | | | | | | ✓ | | | | | | | | ✓ | | | | |
| JAVA RANGER | | | ✓ | | | | | | | | ✓ | | | | | | | | | |
| JAYHORN | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | ✓ | | | | | | |
| JBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| JDART | | | ✓ | | | | | | | | ✓ | | | | | | | | | ✓ |
| KORN | | ✓ | ✓ | | | | ✓ | | | | | | | | | | | | | ✓ |
| LAZY-CSEQ | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| PESCO | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| PINAKA | | | ✓ | ✓ | | | | | | | ✓ | | | | | | | | | |
| PREDATORHP | | | | | | | | | ✓ | | | | | | | | | | | |
| SMACK | | | | ✓ | | | | | | | ✓ | | ✓ | | | ✓ | | | | |
| SPF | | | ✓ | | | | | | ✓ | | | | | | | ✓ | | | | |
| SYMBIOTIC | | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | | | | | | | | ✓ |
| UAUTOMIZER | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| UKOJAK | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | | | | |
| UTAIPAN | ✓ | ✓ | | | | | ✓ | ✓ | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| VERIABS | ✓ | | | ✓ | ✓ | | ✓ | ✓ | | | | | | | | | | ✓ | ✓ | ✓ |

energy are accumulated over all cores of the CPU. To measure the CPU energy, we used CPU Energy Meter [30] (integrated in BenchExec [28]).

One complete verification execution of the competition consisted of 163 177 verification runs (each verifier on each verification task of the selected categories according to the opt-outs), consuming 470 days of CPU time and 126 kWh of CPU energy (without validation). Witness-based result validation required 961 919 validation runs (each validator on each verification task for categories with witness validation, and for each verifier), consuming 274 days of CPU time. Each tool was executed several times, in order to make sure no installation issues occur during the execution. Including preruns, the infrastructure managed a total of 1.33 million verification runs consuming 4.16 years of CPU time, and 7.31 million validation runs consuming 3.84 years of CPU time.

**Quantitative Results.** Table 7 presents the quantitative overview of all tools and all categories. The head row mentions the category, the maximal score for the category, and the number of verification tasks. The tools are listed in alphabetical order; every table row lists the scores of one verifier. We indicate the top three candidates by formatting their scores in bold face and in larger font size. An empty table cell means that the verifier opted-out from the respective main category (perhaps participating in subcategories only, restricting the evaluation to a specific topic). More information (including interactive tables, quantile plots for every category, and also the raw data in XML format) is available on the competition web site (`https://sv-comp.sosy-lab.org/2021/results`) and in the results artifact (see Table 4).

Table 8 reports the top three verifiers for each category. The run time (column 'CPU Time') and energy (column 'CPU Energy') refer to successfully solved verification tasks (column 'Solved Tasks'). We also report the number of tasks for which no witness validator was able to confirm the result (column 'Unconf. Tasks'). The columns 'False Alarms' and 'Wrong Proofs' report the number of verification tasks for which the verifier reported wrong results, i.e., reporting a counterexample when the property holds (incorrect False) and claiming that the program fulfills the property although it actually contains a bug (incorrect True), respectively.

**Score-Based Quantile Functions for Quality Assessment.** We use score-based quantile functions [11, 28] because these visualizations make it easier to understand the results of the comparative evaluation. The web site (`https://sv-comp.sosy-lab.org/2021/results`) and the results archive (see Table 4) include such a plot for each (sub-)category. As an example, we show the plot for category *C-Overall* (all verification tasks) in Fig. 4. A total of 10 verifiers participated in category *C-Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [11]). A more detailed discussion of score-based quantile plots, including examples of what insights one can obtain from the plots, is provided in previous competition reports [11, 14].

**Alternative Rankings.** The community suggested to report a couple of alternative rankings that honor different aspects of the verification process as complement to the official SV-COMP ranking. Table 9 is similar to Table 8, but

Table 7: Quantitative overview over all results; empty cells represent opt-outs; an asterisk after the tool name marks hors-concours participation

| Participant | **ReachSafety** 7844 points 4927 tasks | **MemSafety** 4981 points 3296 tasks | **ConcurrencySafety** 1413 points 1130 tasks | **NoOverflows** 682 points 452 tasks | **Termination** 3897 points 2212 tasks | **SoftwareSystems** 5608 points 3184 tasks | **FalsificationOverall** 6173 points 12989 tasks | **Overall** 23778 points 15201 tasks | **JavaOverall** 693 points 473 tasks |
|---|---|---|---|---|---|---|---|---|---|
| 2LS | 3021 | 1100 | 0 | 414 | **1315** | -7 | 1436 | 6219 | |
| BRICK | | | | | | | | | |
| CBMC | 3395 | -725 | 486 | 279 | 872 | 565 | 2609 | 5289 | |
| CPA-BAM-BNB | | | | | | 491 | | | |
| CPALOCKATOR | | | -819 | | | | | | |
| CPACHECKER | **4764** | **2992** | **1050** | **531** | **1356** | 736 | **4356** | **12217** | |
| DARTAGNAN | | | 309 | | | | | | |
| DIVINE | 2012 | 95 | 391 | 0 | 0 | 124 | 306 | 2083 | |
| ESBMC-INCR | | | -134 | | | | | | |
| ESBMC-KIND | 4486 | 1281 | 37 | 317 | 832 | 694 | 2002 | 6656 | |
| FRAMA-C | | | | 172 | | | | | |
| GAZER-THETA | | | | | | | | | |
| GOBLINT | 777 | | 46 | 156 | | 331 | | | |
| KORN | | | | | | | | | |
| LAZY-CSEQ | | | **1206** | | | | | | |
| PESCO | **4526** | | | | | **878** | **4329** | **12208** | |
| PINAKA | 3408 | | | -200 | 669 | | | | |
| PREDATORHP* | | 2187 | | | | | | | |
| SMACK | | | | | | **894** | | | |
| SYMBIOTIC | 3864 | **3125** | 0 | 373 | 1043 | **2001** | 2947 | 9268 | |
| UAUTOMIZER | 3502 | **1615** | **943** | **512** | **3019** | 359 | **3432** | **11769** | |
| UKOJAK | 1768 | 925 | 0 | 441 | 0 | 298 | 1800 | 4332 | |
| UTAIPAN | 2743 | 1436 | 937 | **506** | 0 | 282 | 3336 | 7676 | |
| VERIABS | **5771** | | | | | | | | |
| COASTAL* | | | | | | | | | 298 |
| JAVA RANGER | | | | | | | | | **630** |
| JAYHORN | | | | | | | | | 369 |
| JBMC | | | | | | | | | **603** |
| JDART | | | | | | | | | **623** |
| SPF* | | | | | | | | | 409 |

Table 8: Overview of the top-three verifiers for each category (measurement values for CPU time and energy rounded to two significant digits)

| Rank | Verifier | Score | CPU Time (in h) | CPU Energy (in kWh) | Solved Tasks | Unconf. Tasks | False Alarms | Wrong Proofs |
|---|---|---|---|---|---|---|---|---|
| ***ReachSafety*** | | | | | | | | |
| 1 | **VeriAbs** | **5771** | 130 | 1.5 | 3 526 | 725 | | |
| 2 | CPAchecker | 4764 | 100 | 1.2 | 2 922 | 251 | 6 | |
| 3 | PeSCo | 4526 | 53 | 0.48 | 2 820 | 272 | 7 | |
| ***MemSafety*** | | | | | | | | |
| 1 | **Symbiotic** | **3125** | 1.6 | 0.021 | 370 | 8 | | |
| 2 | CPAchecker | 2992 | 7.8 | 0.069 | 3 092 | 0 | | |
| 3 | UAutomizer | 1615 | 4.1 | 0.046 | 160 | 2 | | |
| ***ConcurrencySafety*** | | | | | | | | |
| 1 | **Lazy-CSeq** | **1206** | 4.0 | 0.051 | 985 | 34 | | |
| 2 | CPAchecker | 1050 | 16 | 0.13 | 903 | 0 | 1 | |
| 3 | UAutomizer | 943 | 9.6 | 0.087 | 775 | 176 | | |
| ***NoOverflows*** | | | | | | | | |
| 1 | **CPAchecker** | **531** | 1.2 | 0.012 | 366 | 3 | | |
| 2 | UAutomizer | 512 | 1.7 | 0.015 | 358 | 0 | | |
| 3 | UTaipan | 506 | 1.9 | 0.018 | 355 | 0 | | |
| ***Termination*** | | | | | | | | |
| 1 | **UAutomizer** | **3019** | 22 | 0.24 | 1 581 | 9 | | |
| 2 | CPAchecker | 1356 | 17 | 0.20 | 1 078 | 70 | 10 | |
| 3 | 2ls | 1315 | 2.5 | 0.021 | 977 | 363 | 3 | |
| ***SoftwareSystems*** | | | | | | | | |
| 1 | **Symbiotic** | **2001** | 0.55 | 0.0075 | 1 024 | 128 | | |
| 2 | Smack | 894 | 14 | 0.14 | 1 362 | 58 | | **2** |
| 3 | PeSCo | 878 | 27 | 0.27 | 1 484 | 234 | 1 | |
| ***FalsificationOverall*** | | | | | | | | |
| 1 | **CPAchecker** | **4356** | 71 | 0.76 | 3 814 | 98 | 8 | |
| 2 | PeSCo | 4329 | 47 | 0.41 | 3 798 | 106 | 9 | |
| 3 | UAutomizer | 3432 | 30 | 0.30 | 1 585 | 215 | 1 | |
| ***Overall*** | | | | | | | | |
| 1 | **CPAchecker** | **12217** | 190 | 2.1 | 9 835 | 514 | 18 | |
| 2 | PeSCo | 12208 | 120 | 1.2 | 9 743 | 579 | 19 | |
| 3 | UAutomizer | 11769 | 99 | 1.0 | 5 980 | 489 | 1 | **1** |
| ***JavaOverall*** | | | | | | | | |
| 1 | **Java Ranger** | **630** | 4.9 | 0.056 | 427 | 0 | | |
| 2 | JDart | 623 | 0.93 | 0.0093 | 437 | 0 | | |
| 3 | Jbmc | 603 | 0.22 | 0.0022 | 423 | 0 | | |

contains the alternative ranking categories *Correct* and *Green Verifiers*. Column 'Quality' gives the score in score points, column 'CPU Time' the CPU usage of successful runs in hours, column 'CPU Energy' the CPU usage of successful runs in kWh, column 'Solved Tasks' the number of correct results, column 'Wrong Re-

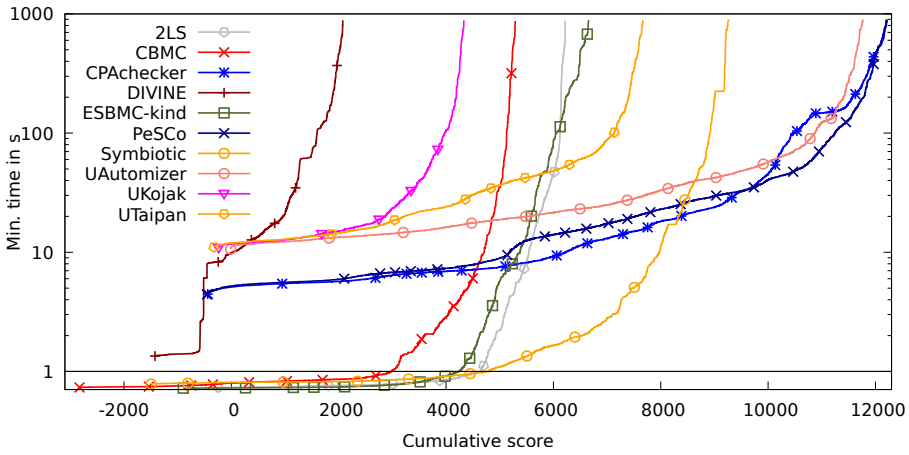


Fig. 4: Quantile functions for category *C-Overall*. Each quantile function illustrates the quantile ($x$-coordinate) of the scores obtained by correct verification runs below a certain run time ($y$-coordinate). More details were given previously [11]. A logarithmic scale is used for the time range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

Table 9: Alternative rankings for catagory *Overall*; quality is given in score points (sp), CPU time in hours (h), kilo-watt-hours (kWh), wrong results in errors (E), rank measures in errors per score point (E/sp), joule per score point (J/sp), and score points (sp)

| Rank | Verifier | Quality (sp) | CPU Time (h) | CPU Energy (kWh) | Solved Tasks | Wrong Results (E) | Rank Measure |
|---|---|---|---|---|---|---|---|
| ***Correct Verifiers*** | | | | | | | (E/sp) |
| 1 | UAutomizer | 11 769 | 99 | 1.0 | 5 980 | 2 | .00017 |
| 2 | UKojak | 4 332 | 46 | 0.48 | 2 476 | 1 | .00023 |
| 3 | CPAchecker | 12 217 | 190 | 2.1 | 9 835 | 18 | .0015 |
| worst | | | | | | 48 | .023 |
| ***Green Verifiers*** | | | | | | | (J/sp) |
| 1 | Symbiotic | 9 268 | 21 | 0.26 | 4 999 | 16 | 100 |
| 2 | 2ls | 6 219 | 26 | 0.24 | 3 372 | 12 | 140 |
| 3 | Cbmc | 5 289 | 26 | 0.31 | 5 596 | 52 | 210 |
| worst | | | | | | | 630 |

sults' the sum of false alarms and wrong proofs in number of errors, and column 'Rank Measure' gives the measure to determine the alternative rank.

*Correct Verifiers — Low Failure Rate.* The right-most columns of Table 8 report that the verifiers achieve a high degree of correctness (all top three verifiers in the *C-Overall* have less than 2 ‰ wrong results). The winners of category *Java-Overall* produced not a single wrong answer. The first category in

Table 10: New verifiers in SV-COMP 2020 and SV-COMP 2021

| Verifier | Language | First Year | Sub-categories |
|---|---|---|---|
| Frama-C | C | 2021 | 4 |
| Gazer-Theta | C | 2021 | 9 |
| Goblint | C | 2021 | 25 |
| Korn | C | 2021 | 13 |
| Brick | C | 2020 | 1 |
| Dartagnan | C | 2020 | 5 |
| Gacal | C | 2020 | 1 |
| Coastal | Java | 2020 | 1 |
| Java Ranger | Java | 2020 | 1 |
| JDart | Java | 2020 | 1 |

Table 11: Confirmation rate of verification witnesses in SV-COMP 2021

| Result | True | | | | False | | | |
|---|---|---|---|---|---|---|---|---|
| | Total | Confirmed | | Unconf. | Total | Confirmed | | Unconf. |
| **2ls** | 2 252 | 2 245 | **99.7 %** | **7** | 1 591 | 1 127 | 70.8 % | 464 |
| Cbmc | 3 875 | 3 498 | 90.3 % | 377 | 3 772 | 2 098 | 55.6 % | 1 674 |
| CPAchecker | 5 992 | 5 646 | 94.2 % | 346 | 4 357 | 4 189 | 96.1 % | 168 |
| Divine | 1 673 | 1 649 | 98.6 % | 24 | 1 317 | 986 | 74.9 % | 331 |
| ESBMC-kind | 4 954 | 4 901 | 98.9 % | 53 | 1 736 | 1 625 | 93.6 % | 111 |
| PeSCo | 5 973 | 5 570 | 93.3 % | 403 | 4 349 | 4 173 | 96.0 % | 176 |
| Symbiotic | 3 351 | 3 149 | 94.0 % | 202 | 2 166 | 1 850 | 85.4 % | 316 |
| UAutomizer | 4 121 | 3 856 | 93.6 % | 265 | 2 348 | 2 124 | 90.5 % | 224 |
| **UKojak** | 1 816 | 1 796 | 98.9 % | 20 | 690 | 680 | **98.6 %** | **10** |
| UTaipan | 2 602 | 2 542 | 97.7 % | 60 | 1 637 | 1 417 | 86.6 % | 220 |

Table 9 uses a failure rate as rank measure: $\frac{\text{number of incorrect results}}{\text{total score}}$, the number of errors per score point ($E/sp$). We use $E$ as unit for number of incorrect results and $sp$ as unit for total score. The worst result was 0.032 E/sp in SV-COMP 2020 and is now improved to 0.023 E/sp.

*Green Verifiers — Low Energy Consumption.* Since a large part of the cost of verification is given by the energy consumption, it might be important to also consider the energy efficiency. The second category in Table 9 uses the energy consumption per score point as rank measure: $\frac{\text{total CPU energy}}{\text{total score}}$, with the unit $J/sp$. The worst result from SV-COMP 2020 was 2 200 J/sp, now improved to 630 J/sp.

*New Verifiers.* To acknowledge the verification systems that participate for the first or second time in SV-COMP, Table 10 lists the new verifiers (in SV-COMP 2020 or SV-COMP 2021).

**Verifiable Witnesses.** Results validation is of primary importance in the competition. All SV-COMP verifiers are required to justify the result (True or False) by producing a verification witness (except for those categories for which no witness validator is available). We used six independently developed witness-based result validators and one witness linter (see Table 1).

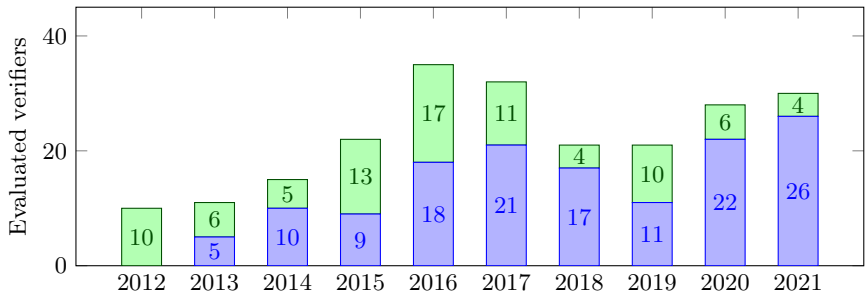


Fig. 5: Number of evaluated verifiers for each year (first-time participants on top)

Table 11 shows the confirmed versus unconfirmed results: the first column lists the verifiers of category *C-Overall*, the three columns for result TRUE reports the total, confirmed, and unconfirmed number of verification tasks for which the verifier answered with TRUE, respectively, and the three columns for result FALSE reports the total, confirmed, and unconfirmed number of verification tasks for which the verifier answered with FALSE, respectively. More information (for all verifiers) is given in the detailed tables on the competition web site and in the results artifact; all verification witnesses are also contained in the witnesses artifact (see Table 4). The verifiers 2LS and UKOJAK are the winners in terms of confirmed results for expected results TRUE and FALSE, respectively. The overall interpretation is similar to SV-COMP 2020 [17].

## 5 Conclusion

The 10th edition of the Competition on Software Verification (SV-COMP 2021) had 30 participating verification systems from 11 countries (see Fig. 5 for the participation numbers and Table 5 for the details). The competition does not only execute the verifiers and collect results, but also validates the verification results using verification witnesses. We used six independent validators to check the results and a witness linter to check if the verification witnesses are syntactically valid (Table 1). The number of verification tasks was increased to 15 201 in the C category and to 473 in the Java category. The high quality standards of the TACAS conference, in particular with respect to the important principles of fairness, community support, and transparency are ensured by a competition jury in which each participating team had a member. The results of our comparative evaluation provide a broad overview of the state of the art in automatic software verification. SV-COMP is instrumental in developing more reliable tools, as well as identifying and propagating successful techniques for software verification.

**Data Availability Statement.** The verification tasks and results of the competition are published at Zenodo, as described in Table 4. All components and data that are necessary for reproducing the competition are available in public version repositories, as specified in Table 3. Furthermore, the results are presented online on the competition web site for easy access: https://sv-comp.sosy-lab.org/2021/results/.

## References


1. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)
2. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141 (2019). https://doi.org/10.1109/ASE.2019.00121
3. American National Standards Institute: ANSI/ISO/IEC 9899-1999: Programming Languages — C. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA (1999)
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). LNCS 12652, Springer (2021)
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. Balyo, T., Heule, M.J.H., Järvisalo, M.: SAT Competition 2016: Recent developments. In: Proc. AAAI. pp. 5061–5063. AAAI Press (2017)
8. Baranová, Z., Barnat, J., Kejstová, K., Kučera, T., Lauko, H., Mrázek, J., Ročkai, P., Štill, V.: Model checking of C and C++ with Divine 4. In: Proc. ATVA. pp. 201–207. LNCS 10482, Springer (2017). https://doi.org/10.1007/978-3-319-68167-2_14
9. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: TOOLympics 2019: An overview of competitions in formal methods. In: Proc. TACAS (3). pp. 3–24. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
10. Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_38
11. Beyer, D.: Second competition on software verification (Summary of SV-COMP 2013). In: Proc. TACAS. pp. 594–609. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_43
12. Beyer, D.: Status report on software verification (Competition summary SV-COMP 2014). In: Proc. TACAS. pp. 373–388. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_25
13. Beyer, D.: Software verification and verifiable witnesses (Report on SV-COMP 2015). In: Proc. TACAS. pp. 401–416. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_31
14. Beyer, D.: Reliable and reproducible competition results with BenchExec and witnesses (Report on SV-COMP 2016). In: Proc. TACAS. pp. 887–904. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_55
15. Beyer, D.: Software verification with validation of results (Report on SV-COMP 2017). In: Proc. TACAS. pp. 331–349. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_20

16. Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Proc. TACAS (3). pp. 133–155. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_9
17. Beyer, D.: Advances in automatic software verification: SV-COMP 2020. In: Proc. TACAS (2). pp. 347–367. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_21
18. Beyer, D.: Results of the 10th Intl. Competition on Software Verification (SV-COMP 2021). Zenodo (2021). https://doi.org/10.5281/zenodo.4458215
19. Beyer, D.: Status report on software testing: Test-Comp 2021. In: Proc. FASE. LNCS 12649, Springer (2021), preprint available.
20. Beyer, D.: SV-Benchmarks: Benchmark set of 10th Intl. Competition on Software Verification (SV-COMP 2021). Zenodo (2021). https://doi.org/10.5281/zenodo.4459126
21. Beyer, D.: Verification witnesses from SV-COMP 2021 verification tools. Zenodo (2021). https://doi.org/10.5281/zenodo.4459196
22. 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
23. 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
24. 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
25. 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
26. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. unpublished manuscript (2021)
27. 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
28. 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
29. 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
30. Beyer, D., Wendler, P.: CPU Energy Meter: A tool for energy-aware algorithms engineering. In: Proc. TACAS (2). pp. 126–133. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_8
31. Black, P.E.: Juliet 1.3 Test Suite: Changes from 1.2. Tech. Rep. NIST TN - 1995, NIST (June 2018). https://doi.org/10.6028/NIST.TN.1995
32. 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
33. Chalupa, M., Jašek, T., Novák, J., Řechtáčková, A., Šoková, V., Strejček, J.: Symbiotic 8: Beyond symbolic execution (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)

34. 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
35. 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
36. 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
37. 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
38. 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
39. 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
40. 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
41. Dangl, M., Löwe, S., Wendler, P.: CPAchecker with support for recursive programs and floating-point arithmetic (competition contribution). In: Proc. TACAS. pp. 423–425. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_34
42. Darke, P., Agrawal, S., Venkatesh, R.: VeriAbs: A tool for scalable verification by abstraction (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)
43. Dietsch, D., Heizmann, M., Nutz, A., Schätzle, C., Schüssele, F.: Ultimate Taipan with symbolic interpretation and fluid abstractions (competition contribution). In: Proc. TACAS (2). pp. 418–422. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_32
44. 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
45. Ernst, G.: A complete approach to loop verification with invariants and summaries. Tech. Rep. arXiv:2010.05812v2, arXiv (January 2020)
46. Gadelha, M.Y.R., Monteiro, F.R., Cordeiro, L.C., Nicole, D.A.: Esbmc v6.0: Verifying C programs using $k$-induction and invariant inference (competition contribution). In: Proc. TACAS (3). pp. 209–213. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_15
47. 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 (Feb 2017). https://doi.org/10.1007/s10009-015-0407-9
48. 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
49. 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

50. 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
51. Haran, A., Carter, M., Emmi, M., Lal, A., Qadeer, S., Rakamarić, Z.: SMACK+Corral: A modular verifier (competition contribution). In: Proc. TACAS. pp. 451–454. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_42
52. Heizmann, M., Chen, Y.F., Dietsch, D., Greitschus, M., Hoenicke, J., Li, Y., Nutz, A., Musa, B., Schilling, C., Schindler, T., Podelski, A.: Ultimate Automizer and the search for perfect interpolants (competition contribution). In: Proc. TACAS (2). pp. 447–451. LNCS 10806, Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_30
53. 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
54. 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
55. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Proc. ISoLA. pp. 608–614. LNCS 7609, Springer (2012). https://doi.org/10.1007/978-3-642-34026-0_45
56. Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012: A program verification competition. STTT **17**(6), 647–657 (2015). https://doi.org/10.1007/s10009-015-0396-8
57. 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
58. Inverso, O., Tomasco, E., Fischer, B., La Torre, S., Parlato, G.: Bounded model checking of multi-threaded C programs via lazy sequentialization. In: Proc. CAV. pp. 585–602. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_39
59. 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
60. 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
61. 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
62. 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
63. Malík, V., Schrammel, P., Vojnar, T.: 2ls: Heap analysis and memory safety (competition contribution). In: Proc. TACAS (2). pp. 368–372. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_22
64. Mues, M., Howar, F.: JDart: Portfolio solving, breadth-first search and smt-lib strings (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)

65. 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
66. 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
67. 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
68. Ponce-De-Leon, H., Haas, T., Meyer, R.: Dartagnan: Leveraging compiler optimizations and the price of precision (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)
69. 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
70. Rakamarić, Z., Emmi, M.: SMACK: Decoupling source language details from verifier implementations. In: Proc. CAV. pp. 106–113. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_7
71. 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
72. 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
73. Saan, S., Schwarz, M., Apinis, K., Erhard, J., Seidl, H., Vogler, R., Vojdani, V.: Goblint: Thread-modular abstract interpretation using side-effecting constraints (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)
74. Sallai, Gy.: LLVM IR-based Transformations for Software Model Checking. Master's thesis, Budapest University of Technology and Economics (2019)
75. Shamakhi, A., Hojjat, H., Rümmer, P.: Towards string support in JayHorn (competition contribution). In: Proc. TACAS (2). LNCS 12652, Springer (2021)
76. Sharma, V., Hussein, S., Whalen, M.W., McCamant, S.A., Visser, W.: Java Ranger at SV-COMP 2020 (competition contribution). In: Proc. TACAS (2). pp. 393–397. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_27
77. 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
78. Svejda, J., 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
79. 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
80. 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

81. 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
82. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.6. Zenodo (2021). https://doi.org/10.5281/zenodo.4317433
83. Wetzler, N., Heule, M.J.H., Jr., W.A.H.: DRAT-TRIM: Efficient checking and trimming using expressive clausal proofs. In: Proc. SAT. pp. 422–429. LNCS 8561, Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_31

**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.