
# Progress on Software Verification: SV-COMP 2022

Dirk Beyer

LMU Munich, Munich, Germany

**Abstract.** The 11th edition of the Competition on Software Verification (SV-COMP 2022) provides the largest ever overview of tools for software verification. The competition is an annual comparative evaluation of fully automatic software verifiers for C and Java programs. The objective is to provide an overview of the state of the art in terms of effectiveness and efficiency of software verification, establish standards, provide a platform for exchange to developers of such tools, educate PhD students on reproducibility approaches and benchmarking, and provide computing resources to developers that do not have access to compute clusters. The competition consisted of 15 648 verification tasks for C programs and 586 verification tasks for Java programs. Each verification task consisted of a program and a property (reachability, memory safety, overflows, termination). The new category on data-race detection was introduced as demonstration category. SV-COMP 2022 had 47 participating verification systems from 33 teams from 11 countries.

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

## 1 Introduction

This report is the 2022 edition of the series of competition reports (see footnote) that accompanies the competition, by explaining the process and rules, giving insights into some aspects of the competition (this time the focus is on trouble shooting and reproducing results on a small scale), and, most importantly, reporting the results of the comparative evaluation. The 11th Competition on Software Verification (SV-COMP, https://sv-comp.sosy-lab.org/2022) is the largest comparative evaluation ever in this area. The objectives of the competitions were discussed earlier (1-4 [16]) and extended over the years (5-6 [17]):

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

This report extends previous reports on SV-COMP [10, 11, 12, 13, 14, 15, 16, 17, 18]. Reproduction packages are available on Zenodo (see Table 4).

✉ dirk.beyer@sosy-lab.org

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

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

**Related Competitions.** There are many competitions in the area of formal methods [9], because it is well-understood that competitions are a fair and accurate means to execute a comparative evaluation with involvement of the developing teams. We refer to a previous report [17] for a more detailed discussion and give here only the references to the most related competitions [20, 53, 67].

**Quick Summary of Changes.** While we try to keep the setup of the competition stable, there are always improvements and developments. For the 2022 edition, the following changes were made:

- A demonstration category on data-race detection was added. Due to several participating verification tools, this category will become a normal main category in SV-COMP 2023. The results are outlined in Sect. 5.
- New verification tasks were added, with an increase in C from 15 201 in 2021 to 15 648 in 2022 and in Java from 473 in 2021 to 586 in 2022, combined with ongoing efforts on quality improvement.

## 2 Organization, Definitions, Formats, and Rules

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

**Competition Jury.** Traditionally, the competition jury consists of the chair and one member of each participating team; the team-representing members circulate every year after the candidate-submission deadline. This committee reviews

the competition contribution papers and helps the organizer with resolving any disputes that might occur (from competition report of SV-COMP 2013 [11]).

In more detail, the tasks of the jury consist of the following:

- The jury oversees the process and ensures transparency, fairness, and community involvement.
- Each jury member who participates in the competition is assigned a number of (3 or 4) submissions (papers and systems) to review.
- Participating systems are reviewed to determine whether they fulfill the requirements for verifier archives, based on the archives submitted to the repository.
- Teams and paper submissions are reviewed to verify the requirements for qualification, based on the submission data and paper in EasyChair and the results of the qualification runs.
- Some qualified competition candidates are selected to publish (in the LNCS proceedings of TACAS) a contribution paper that gives an overview of the participating system.
- The jury helps the organizer with discussing and resolving any disputes that might occur.
- Jury members adhere to the deadlines with all the duties.

The team representatives of the competition jury are listed in Table 5.

**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.

Two exceptions were made to allow minor incompatibilities for commercial participants: The jury felt that the rule "allows any kind of (re-)distribution of the unmodified verifier archive" is too broad. The idea of the rule was to maximize the possibilities for reproduction. Starting with SV-COMP 2023, this license requirement shall be changed to "allows (re-)distribution of the unmodified verifier archive via SV-COMP repositories and archives".

**Validation of Results.** The validation of the verification results was done by eleven validation tools, which are listed in Table 1, including references to literature. Four new validators support the competition:

- There are two new validators for the C language: Dartagnan [new] supports result validation for violation witnesses in category *ConcurrencySafety-Main*. Symbiotic-Witch [new] supports result validation for violation witnesses in categories *ReachSafety*, *MemSafety*, and *NoOverflows*.
- For the first time, there are validators for the Java language: GWit [new] and Wit4Java [new] support result validation for violation witnesses in category *ReachSafety-Java*.

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

| Validator | Reference | Representative | Affiliation |
|---|---|---|---|
| CPAchecker | [25, 26, 28] | Thomas Bunk | LMU Munich, Germany |
| CPA-w2t | [27] | Thomas Lemberger | LMU Munich, Germany |
| Dartagnan new | [89] | Hernán Ponce de León | Bundeswehr U., Germany |
| CProver-w2t | [27] | Michael Tautschnig | Queen Mary U. London, UK |
| GWit new | [68] | Falk Howar | TU Dortmund U., Germany |
| MetaVal | [33] | Martin Spiessl | LMU Munich, Germany |
| NitWit | [105] | Jana (Philipp) Berger | RWTH Aachen, Germany |
| Symbiotic-Witch new | [6] | Paulína Ayaziová | Masaryk U., Brno, Czechia |
| UAutomizer | [25, 26] | Daniel Dietsch | U. of Freiburg, Germany |
| Wit4Java new | [108] | Tong Wu | U. of Manchester, UK |
| WitnessLint | | Sven Umbricht | LMU Munich, Germany |

Table 2: Scoring schema for SV-COMP 2022 (unchanged from 2021 [18])

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

**Task-Definition Format 2.0.** SV-COMP 2022 used the task-definition format in version 2.0. More details can be found in the report for Test-Comp 2021 [19].

**Properties.** Please see the 2015 competition report [13] for the definition of the properties and the property format. All specifications used in SV-COMP 2022 are available in the directory `c/properties/` of the benchmark repository.

**Categories.** The (updated) category structure of SV-COMP 2022 is illustrated by Fig. 1. The categories are also listed in Tables 8, 9, and 10, and described in detail on the competition web site (`https://sv-comp.sosy-lab.org/2022/benchmarks.php`). Compared to the category structure for SV-COMP 2021, we added the sub-category *Termination-BitVectors* to category *Termination* and the sub-category *SoftwareSystems-BusyBox-ReachSafety* to category *SoftwareSystems*.

**Scoring Schema and Ranking.** The scoring schema of SV-COMP 2022 was the same as for SV-COMP 2021. Table 2 provides an overview and Fig. 2 visually illustrates the score assignment for the reachability property as an example. As before, the rank of a verifier was decided based on the sum of points (normalized for meta categories). In case of a tie, the rank was decided based on success run time, which is the total CPU time over all verification tasks for which the verifier reported

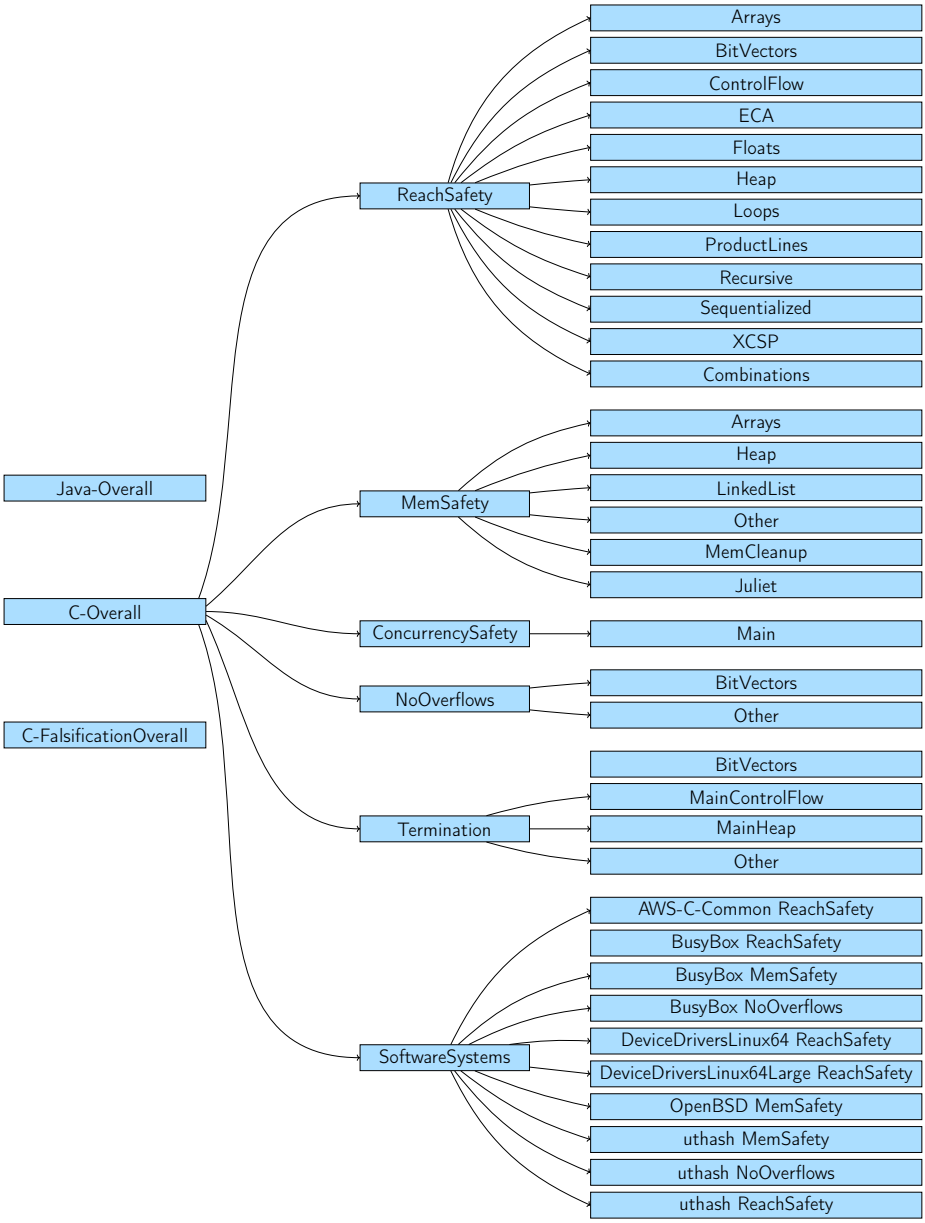


Fig. 1: Category structure for SV-COMP 2022; category *C-FalsificationOverall* contains all verification tasks of *C-Overall* without *Termination*; *Java-Overall* contains all Java verification tasks; compared to SV-COMP 2021, there is one new sub-category in *Termination* and one new sub-categories in *SoftwareSystems*

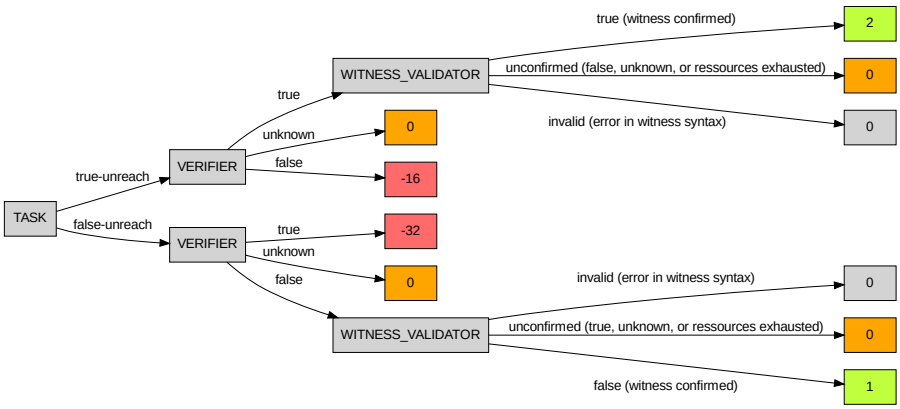


Fig. 2: Visualization of the scoring schema for the reachability property (unchanged from 2021 [18])

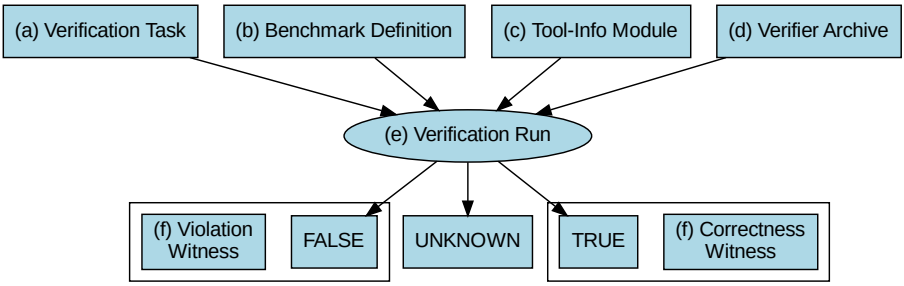


Fig. 3: Benchmarking components of SV-COMP and competition's execution flow (same as for SV-COMP 2020)

a correct verification result. *Opt-out from Categories* and *Score Normalization for Meta Categories* was done as described previously [11] (page 597).

**Reproducibility.** SV-COMP results must be reproducible, and consequently, all major components are maintained in public version-control repositories. The overview of the components is provided in Fig. 3, and the details are given in Table 3. We refer to the SV-COMP 2016 report [14] for a description of all components of the SV-COMP organization. There are competition artifacts at Zenodo (see Table 4) to guarantee their long-term availability and immutability.

**Competition Workflow.** The workflow of the competition is described in the report for Test-Comp 2021 [19] (SV-COMP and Test-Comp use a similar workflow).

Table 3: Publicly available components for reproducing SV-COMP 2022

| Component | Fig. 3 | Repository | Version |
|---|---|---|---|
| Verification Tasks | (a) | gitlab.com/sosy-lab/benchmarking/sv-benchmarks | svcomp22 |
| Benchmark Definitions | (b) | gitlab.com/sosy-lab/sv-comp/bench-defs | svcomp22 |
| Tool-Info Modules | (c) | github.com/sosy-lab/benchexec | 3.10 |
| Verifier Archives | (d) | gitlab.com/sosy-lab/sv-comp/archives-2022 | svcomp22 |
| Benchmarking | (e) | github.com/sosy-lab/benchexec | 3.10 |
| Witness Format | (f) | github.com/sosy-lab/sv-witnesses | svcomp22 |

Table 4: Artifacts published for SV-COMP 2022

| Content | DOI | Reference |
|---|---|---|
| Verification Tasks | 10.5281/zenodo.5831003 | [22] |
| Competition Results | 10.5281/zenodo.5831008 | [21] |
| Verifiers and Validators | 10.5281/zenodo.5959149 | [24] |
| Verification Witnesses | 10.5281/zenodo.5838498 | [23] |
| BenchExec | 10.5281/zenodo.5720267 | [106] |

## 3 Reproducing a Verification Run and Trouble-Shooting Guide

In the following we explain a few steps that are useful to reproduce individual results and for trouble shooting. It is written from the perspective of a participant.

**Step 1: Make Verifier Archive Available.** The first action item for a participant is to submit a merge request to the repository that contains all the verifier archives (see list of merge requests at GitLab). Typical problems include:

- The fork is not public. This means that the continuous integration (CI) pipeline results are not visible and the merge request cannot be merged.
- The shared runners are not enabled. This means that the CI pipeline cannot run and no results will be available.
- Verifier does not provide a version string (and this should not include the verifier name itself). This means that it is not possible to later determine which version of the verifier was used for the experiments. Therefore, version strings are mandatory and are checked by the CI.
- The interface between the execution (with BenchExec) and the verification tool can be checked using the procedure decribed in the BenchExec documentation.[1]

**Step 2: Ensure That Verifier Works on Competition Machines.** Once the CI checks passed and the archive is merged into the official competition repository, the verifier can be executed on the competition machines on a few verification

[1] https://github.com/sosy-lab/benchexec/blob/3.10/doc/tool-integration.md

tasks. The competition uses the infrastructure VerifierCloud, and remote execution in this compute cloud is possible using CoVeriTeam [29]. CoVeriTeam is a tool for constructing cooperative verification tools from existing components, and the competition is supported by this project since SV-COMP 2021. 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. A description and example invokation for each participating verifier is available in the CoVeriTeam documentation (see file doc/competition-help.md in the CoVeriTeam repository). Competition participants are asked to execute their tool locally using CoVeriTeam and then remotely on the competition machines. Typical problems include:

- Verifiers sometimes have insufficient log output, such that it is not possible to observe what the verifier was executing. The first step towards trouble shooting is always to ensure some minimal log output.
- The verifier assumes software that is not installed yet. Each verifier states its dependencies in its documentation. For example, the verifier Cbmc specifies under `required-ubuntu-packages` that is relies on the Ubuntu packages `gcc` and `libc6-dev-i386` in file benchmark-defs/category-structure.yml in the repository with the benchmark definitions. This is easy to fix by adding the dependency in the definition file and get it installed.
- The verifier makes assumptions about the hardware of the machine, e.g., selecting a specific processing unit. This can be investigated by running the verifier in the Docker container and remotely on the competition machines.
- For the above-mentioned purpose, the competition offers a Docker image that can be used to try out if all required dependencies are available.[2]
- The competition also provides a list of installed packages, which is important for ensuring reproducibility.

**Step 3: Check Prerun Results.** So far, we considered executing individual verification runs in the Docker container or remotely on the competition machines. As a service to the participating teams, the competition offers training runs and provides the results to the teams. Typical checks that teams perform on the prerun results include:

- Inspect the verification results (solution to the verification task, like True, False, Unknown, etc.) and log files.
- Inspect the validation results (was the verification result confirmed by a validator) and the produced verification witnesses.
- Inspect the result of the witness linter. All witnesses should be syntactically correct according to the witness specification.
- In case the verification result does not match the expected result, investigate the verifier and the verification task; in case of problems with the verification task, report to the jury by creating a merge request with a fix or an issue for discussion in the SV-Benchmarks repository.

[2] https://gitlab.com/sosy-lab/benchmarking/competition-scripts/-/tree/svcomp22

Table 5: Competition candidates with tool references and representing jury members; new for first-time participants, ∅ for hors-concours participation

| Participant | Ref. | Jury member | Affiliation |
|---|---|---|---|
| 2LS | [36, 81] | Viktor Malík | BUT, Brno, Czechia |
| APROVE | [65, 100] | Jera Hensel | RWTH Aachen, Germany |
| BRICK | [37] | Lei Bu | Nanjing U., China |
| CBMC | [75] | Michael Tautschnig | Queen Mary U. of London, UK |
| COASTAL∅ | [102] | (hors concours) | – |
| CVT-ALGOSEL new∅ | [29, 30] | (hors concours) | – |
| CVT-PARPORT new∅ | [29, 30] | (hors concours) | – |
| CPA-BAM-BNB∅ | [3, 104] | (hors concours) | – |
| CPA-BAM-SMG new | | Anton Vasilyev | ISP RAS, Russia |
| CPACHECKER | [31, 49] | Thomas Bunk | LMU Munich, Germany |
| CPALOCKATOR∅ | [4, 5] | (hors concours) | – |
| CRUX new | [52, 96] | Ryan Scott | Galois, USA |
| CSEQ | [47, 71] | Emerson Sales | Gran Sasso Science Institute, Italy |
| DARTAGNAN | [58, 88] | Hernán Ponce de León | U. Bundeswehr Munich, Germany |
| DEAGLE new | [62] | Fei He | Tsinghua U., China |
| DIVINE∅ | [8, 76] | (hors concours) | – |
| EBF new | | Fatimah Aljaafari | U. of Manchester, UK |
| ESBMC-INCR∅ | [43, 46] | (hors concours) | – |
| ESBMC-KIND | [56, 57] | Rafael Sá Menezes | U. of Manchester, UK |
| FRAMA-C-SV | [34, 48] | Martin Spiessl | LMU Munich, Germany |
| GAZER-THETA∅ | [1, 60] | (hors concours) | – |
| GDART new | [84] | Falk Howar | TU Dortmund, Germany |
| GOBLINT | [95, 103] | Simmo Saan | U. of Tartu, Estonia |
| GRAVES-CPA new | [79] | Will Leeson | U. of Virginia, USA |
| INFER new∅ | [38, 73] | (hors concours) | – |
| JAVA-RANGER | [98, 99] | Soha Hussein | U. of Minnesota, USA |
| JAYHORN | [72, 97] | Ali Shamakhi | Tehran Inst. Adv. Studies, Iran |
| JBMC | [44, 45] | Peter Schrammel | U. of Sussex / Diffblue, UK |
| JDART | [80, 83] | Falk Howar | TU Dortmund, Germany |
| KORN | [55] | Gidon Ernst | LMU Munich, Germany |
| LART new | [77, 78] | Henrich Lauko | Masaryk U., Brno, Czechia |
| LAZY-CSEQ∅ | [69, 70] | (hors concours) | – |
| LOCKSMITH new | [90] | Vesal Vojdani | U. of Tartu, Estonia |
| PESCO | [93, 94] | Cedric Richter | U. of Oldenburg, Germany |
| PINAKA∅ | [41] | (hors concours) | – |
| PREDATORHP∅ | [66, 87] | (hors concours) | – |
| SESL new | | Xie Li | Academy of Sciences, China |
| SMACK∅ | [61, 92] | (hors concours) | – |
| SPF∅ | [85, 91] | (hors concours) | – |
| SYMBIOTIC | [39, 40] | Marek Chalupa | Masaryk U., Brno, Czechia |
| THETA new | [101, 109] | Vince Molnár | BME Budapest, Hungary |
| UAUTOMIZER | [63, 64] | Matthias Heizmann | U. of Freiburg, Germany |
| UGEMCUTTER new | [74] | Dominik Klumpp | U. of Freiburg, Germany |
| UKOJAK | [54, 86] | Frank Schüssele | U. of Freiburg, Germany |
| UTAIPAN | [51, 59] | Daniel Dietsch | U. of Freiburg, Germany |
| VERIABS | [2, 50] | Priyanka Darke | Tata Consultancy Services, India |
| VERIFUZZ | [42, 82] | Raveendra Kumar M. | Tata Consultancy Services, India |

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

| Verifier | CEGAR | Predicate Abstraction | Symbolic Execution | Bounded Model Checking | k-Induction | Property-Directed Reach. | Explicit-Value Analysis | Numeric. Interval Analysis | Shape Analysis | Separation Logic | Bit-Precise Analysis | ARG-Based Analysis | Lazy Abstraction | Interpolation | Automata-Based Analysis | Concurrency Support | Ranking Functions | Evolutionary Algorithms | Algorithm Selection | Portfolio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2LS | | | | ✓ | ✓ | | | ✓ | ✓ | | ✓ | | | | | | ✓ | | | |
| APROVE | | | ✓ | | | | ✓ | ✓ | | ✓ | ✓ | | | | | | ✓ | | | |
| BRICK | ✓ | | ✓ | ✓ | | | | ✓ | | | | | | | | ✓ | | | | |
| CBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| COASTAL∅ | | | ✓ | | | | | | | | | | | | | | | | | |
| CVT-ALGOSEL new∅ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| CVT-PARPORT new∅ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| CPA-BAM-BNB∅ | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | | | |
| CPA-BAM-SMG new | | | | | | | | | | | | | | | | | | | | |
| CPALOCKATOR∅ | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | | | | |
| CPACHECKER | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| CRUX new | | | ✓ | | | | | | | | | | | | | | | | | |
| CSEQ | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| DARTAGNAN | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| DEAGLE new | | | | | | | | | | | | | | | | | | | | |
| DIVINE∅ | | | ✓ | | | | ✓ | | | | ✓ | | | | | ✓ | | | ✓ | ✓ |
| EBF new | | | | ✓ | | | | | | | | | | | | | | | | |
| ESBMC-INCR∅ | | | | ✓ | ✓ | | | | | | ✓ | | | | | ✓ | | | | |
| ESBMC-KIND | | | | ✓ | ✓ | | | ✓ | | | ✓ | | | | | ✓ | | | | |
| FRAMA-C-SV | | | | | | | | ✓ | | | | | | | | | | | | |
| GAZER-THETA∅ | ✓ | ✓ | | ✓ | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ |
| GDART new | | | ✓ | | | | | | | | ✓ | | | | | | | | | ✓ |
| GOBLINT | | | | | | | | ✓ | | | | | | | | ✓ | | | | |
| GRAVES-CPA new | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| INFER new∅ | | | | | | | | ✓ | ✓ | ✓ | | | | | | | | | | ✓ |
| JAVA-RANGER | | | ✓ | | | | | | | | ✓ | | | | | | | | | |
| JAYHORN | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | ✓ | | | | | | |
| JBMC | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |
| JDART | | | ✓ | | | | | | | | ✓ | | | | | | | | | ✓ |
| KORN | | ✓ | ✓ | | | | ✓ | | | | | | | | | | | | | ✓ |
| LART new | | | ✓ | | | | ✓ | | | | ✓ | | | | | | | | | ✓ |
| LAZY-CSEQ∅ | | | | ✓ | | | | | | | ✓ | | | | | ✓ | | | | |

(continues on next page)

| Verifier | CEGAR | Predicate Abstraction | Symbolic Execution | Bounded Model Checking | k-Induction | Property-Directed Reach. | Explicit-Value Analysis | Numeric. Interval Analysis | Shape Analysis | Separation Logic | Bit-Precise Analysis | ARG-Based Analysis | Lazy Abstraction | Interpolation | Automata-Based Analysis | Concurrency Support | Ranking Functions | Evolutionary Algorithms | Algorithm Selection | Portfolio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Locksmith [new] | | | | | | | | | | | | | | | | ✓ | | | | |
| PeSCo | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| Pinaka[∅] | | | ✓ | ✓ | | | | | | | ✓ | | | | | | | | | |
| PredatorHP[∅] | | | | | | | | | ✓ | | | | | | | | | | | |
| Sesl [new] | | | ✓ | | | | | | | ✓ | | | | | | | | | | |
| Smack[∅] | | | | ✓ | | | | | | | ✓ | | ✓ | | | ✓ | | | | |
| Spf[∅] | | | ✓ | | | | | | ✓ | | | | | | | ✓ | | | | |
| Symbiotic | | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | | | | | ✓ | | | | ✓ |
| Theta [new] | ✓ | ✓ | | | | | ✓ | | | | ✓ | ✓ | | ✓ | | ✓ | | | ✓ | ✓ |
| UAutomizer | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| UGemCutter [new] | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| UKojak | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | | | | |
| UTaipan | ✓ | ✓ | | | | | ✓ | ✓ | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| VeriAbs | ✓ | | | ✓ | ✓ | | ✓ | ✓ | | | | | | | | | | ✓ | ✓ | ✓ |
| VeriFuzz | | | | ✓ | | | | ✓ | | | | | | | | | | ✓ | | |

## 4 Participating Verifiers

The participating verification systems are listed in Table 5. The table contains the verifier name (with hyperlink), references to papers that describe the systems, the representing jury member and the affiliation. The listing is also available on the competition web site at `https://sv-comp.sosy-lab.org/2022/systems.php`. Table 6 lists the algorithms and techniques that are used by the verification tools, and Table 7 gives an overview of commonly used solver libraries and frameworks.

**Hors-Concours Participation.** There are verification tools that participated in the comparative evaluation, but did not participate in the rankings. We call this kind of participation *hors concours* as these participants cannot participate in rankings and cannot "win" the competition. Those are either passive or active participants. Passive participation means that the tools are taken from previous years of the competition, in order to show progress and compare new tools against them (Coastal[∅], CPA-BAM-BnB[∅], CPALockator[∅], Divine[∅], Esbmc-incr[∅], Gazer-Theta[∅], Lazy-CSeq[∅], Pinaka[∅], PredatorHP[∅], Smack[∅], Spf[∅]). Active paritication means that there are teams actively developing the tools, but there are reasons why those tools should not occur in the rankings. For example, a

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

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

(continues on next page)

| Verifier | CPAchecker | CProver | Esbmc | Jpf | Ultimate | JavaSMT | MathSAT | Cvc4 | SMTInterpol | z3 | MiniSAT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Pinaka ∅ | | | | | | | | | | | |
| PredatorHP ∅ | | | | | | | | | | | |
| Sesl new | | | | | | | | | | | |
| Smack ∅ | | | | | | | | | | | |
| Spf ∅ | | | | ✓ | | | | | | | |
| Symbiotic | | | | | | | | | | ✓ | |
| Theta new | | | | | | | | | | | |
| UAutomizer | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| UGemCutter new | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| UKojak | | | | | ✓ | | | | ✓ | ✓ | |
| UTaipan | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| VeriAbs | ✓ | ✓ | | | | | | | | ✓ | ✓ |
| VeriFuzz | | | | | | | | | | ✓ | |

tool might use other tools that participate in the competition on their own, and comparing such a tool in the ranking could be considered unfair (CVT-AlgoSel new ∅, CVT-ParPort new ∅). Also, a tool might produce uncertain results and the team was not sure if the full potential of the tool can be shown in the SV-COMP experiments (Infer new ∅). Those participations are marked as 'hors concours' in Table 5 and others, and the names are annotated with a symbol (∅).

## 5 Results and Discussion

The results of the competition represent the the state of the art of what can be achieved with fully automatic software-verification tools on the given benchmark set. We report the effectiveness (number of verification tasks that can be solved and correctness of the results, as accumulated in the score) and the efficiency (resource consumption in terms of CPU time and CPU energy). The results are presented in the same way as in last years, such that the improvements compared to last year are easy to identify, except that due to the number of tools, we have to split the table and put the hors-concours verifiers into a second results table. The results presented in this report were inspected and approved by the participating teams.

**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

Table 8: Quantitative overview over all regular results; empty cells are used for opt-outs, new for first-time participants

| Verifier | ReachSafety 8631 points 5400 tasks | MemSafety 5003 points 3321 tasks | ConcurrencySafety 1160 points 763 tasks | NoOverflows 685 points 454 tasks | Termination 4144 points 2293 tasks | SoftwareSystems 5898 points 3417 tasks | FalsificationOverall 5718 points 13355 tasks | Overall 25209 points 15648 tasks | JavaOverall 828 points 586 tasks |
|---|---|---|---|---|---|---|---|---|---|
| 2LS | 3585 | 810 | 0 | 428 | **2178** | 83 | 1462 | 7366 | |
| APROVE | | | | | **2305** | | | | |
| BRICK | | | | | | | | | |
| CBMC | 3808 | -262 | 460 | 284 | 1800 | -198 | 2024 | 6733 | |
| CPA-BAM-SMG new | | **3101** | | | | 776 | | | |
| CPACHECKER | **5572** | **3057** | 498 | **531** | 1270 | **809** | **3835** | **11904** | |
| CRUX new | 1408 | | | 290 | | | | | |
| CSEQ | | | **655** | | | | | | |
| DARTAGNAN | | | 481 | | | | | | |
| DEAGLE new | | | **757** | | | | | | |
| EBF new | | | 496 | | | | | | |
| ESBMC-KIND | 4959 | 2162 | -74 | 318 | 1389 | 633 | 1841 | 7727 | |
| FRAMA-C-SV | | | | 213 | | | | | |
| GOBLINT | 858 | | 106 | 159 | | 340 | | 1951 | |
| GRAVES-CPA new | 4520 | | | | | **802** | 2400 | 9218 | |
| KORN | | | | | | | | | |
| LART new | 3034 | | | | | 573 | | | |
| LOCKSMITH new | | | | | | | | | |
| PESCO | **5080** | | | | | -273 | **3683** | 10515 | |
| SESL new | | 345 | | | | | | | |
| SYMBIOTIC | 4571 | **4051** | 105 | 370 | 2030 | **2704** | **3274** | **12249** | |
| THETA new | 1132 | | -14 | | | | | | |
| UAUTOMIZER | 3969 | 2350 | 493 | **506** | **2552** | 712 | 3089 | **11802** | |
| UGEMCUTTER new | | | **612** | | | | | | |
| UKOJAK | 2058 | 1573 | 0 | 445 | 0 | 382 | 2056 | 5078 | |
| UTAIPAN | 3634 | 2336 | 535 | **501** | 0 | 486 | 3049 | 8666 | |
| VERIABS | **6923** | | | | | | | | |
| VERIFUZZ | 1518 | -777 | -32 | 136 | -129 | 0 | 817 | | |
| GDART new | | | | | | | | | 640 |
| JAVA-RANGER | | | | | | | | | **670** |
| JAYHORN | | | | | | | | | 376 |
| JBMC | | | | | | | | | **700** |
| JDART | | | | | | | | | **714** |

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

| Verifier | ReachSafety 8631 points 5400 tasks | MemSafety 5003 points 3321 tasks | ConcurrencySafety 1160 points 763 tasks | NoOverflows 685 points 454 tasks | Termination 4144 points 2293 tasks | SoftwareSystems 5898 points 3417 tasks | FalsificationOverall 5718 points 13355 tasks | Overall 25209 points 15648 tasks | JavaOverall 828 points 586 tasks |
|---|---|---|---|---|---|---|---|---|---|
| CVT-AlgoSel new ∅ | 5438 | | 314 | | | | | | |
| CVT-ParPort new ∅ | 5904 | 3700 | -551 | 553 | 2351 | 1282 | 1087 | 10704 | |
| CPA-BAM-BnB ∅ | | | | | | 504 | | | |
| CPALockator ∅ | | | -1154 | | | | | | |
| Divine ∅ | 110 | 99 | -136 | 0 | 0 | 112 | -1253 | -248 | |
| Esbmc-incr ∅ | | | -74 | | | | | | |
| Gazer-Theta ∅ | | | | | | | | | |
| Infer new ∅ | -50415 | | -5890 | -5982 | | -29566 | | | |
| Lazy-CSeq ∅ | | | 571 | | | | | | |
| Pinaka ∅ | 3710 | | | -200 | 1259 | | | | |
| PredatorHP ∅ | | 2205 | | | | | | | |
| Smack ∅ | | | | | | 1181 | | | |
| Coastal ∅ | | | | | | | | | -2541 |
| Spf ∅ | | | | | | | | | 430 |

for running the experiments are part of a compute cluster that consists of 167 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 [32] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud to distribute, install, run, and clean-up verification runs, and to collect the results. The values for time and energy are accumulated over all cores of the CPU. To measure the CPU energy, we used CPU Energy Meter [35] (integrated in BenchExec [32]).

One complete verification execution of the competition consisted of 309 081 verification runs (each verifier on each verification task of the selected categories according to the opt-outs), consuming 937 days of CPU time and 249 kWh of CPU energy (without validation). Witness-based result validation required 1.43 million validation runs (each validator on each verification task for categories with witness validation, and for each verifier), consuming 708 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

Table 10: Overview of the top-three verifiers for each category; new for first-time participants, measurements for CPU time and energy rounded to two significant digits ('–' indicates a missing energy value due to a configuration bug)

| Rank | Verifier | Score | CPU Time (in h) | CPU Energy (in kWh) | Solved Tasks | Unconf. Tasks | False Alarms | Wrong Proofs |
|---|---|---|---|---|---|---|---|---|
| ***ReachSafety*** | | | | | | | | |
| 1 | **VeriAbs** | **6923** | 170 | 1.8 | 4 117 | 359 | | |
| 2 | CPAchecker | 5572 | 130 | 1.5 | 3 245 | 228 | 4 | |
| 3 | PeSCo | 5080 | 63 | 0.57 | 3 033 | 314 | 7 | |
| ***MemSafety*** | | | | | | | | |
| 1 | **Symbiotic** | **4051** | 2.6 | 0.034 | 2 167 | 1 097 | | |
| 2 | CPA-BAM-SMG new | 3101 | 7.3 | 0.064 | 2 975 | 17 | | |
| 3 | CPAchecker | 3057 | 7.8 | 0.069 | 3 119 | 0 | | |
| ***ConcurrencySafety*** | | | | | | | | |
| 1 | **Deagle** new | **757** | 0.50 | 0.0059 | 517 | 42 | | |
| 2 | CSeq | 655 | 5.1 | 0.059 | 454 | 50 | | |
| 3 | UGemCutter new | 612 | 4.9 | – | 445 | 21 | | |
| ***NoOverflows*** | | | | | | | | |
| 1 | **CPAchecker** | **531** | 1.2 | 0.012 | 366 | 3 | | |
| 2 | UAutomizer | 506 | 2.0 | 0.019 | 356 | 2 | | |
| 3 | UTaipan | 501 | 2.2 | 0.023 | 355 | 1 | | |
| ***Termination*** | | | | | | | | |
| 1 | **UAutomizer** | **2552** | 13 | 0.12 | 1 581 | 8 | | |
| 2 | AProVE | 2305 | 38 | 0.43 | 1 114 | 37 | | |
| 3 | 2ls | 2178 | 2.9 | 0.025 | 1 163 | 203 | | |
| ***SoftwareSystems*** | | | | | | | | |
| 1 | **Symbiotic** | **2704** | 1.2 | 0.016 | 1 188 | 73 | | |
| 2 | CPAchecker | 809 | 52 | 0.60 | 1 660 | 169 | 1 | |
| 3 | Graves-CPA new | 802 | 19 | 0.17 | 1 582 | 95 | 2 | **3** |
| ***FalsificationOverall*** | | | | | | | | |
| 1 | **CPAchecker** | **3835** | 81 | 0.90 | 3 626 | 95 | 5 | |
| 2 | PeSCo | 3683 | 45 | 0.41 | 3 552 | 110 | 9 | |
| 3 | Symbiotic | 3274 | 14 | 0.18 | 2 295 | 1 191 | 3 | |
| ***Overall*** | | | | | | | | |
| 1 | **Symbiotic** | **12249** | 34 | 0.44 | 7 430 | 1 529 | 3 | |
| 2 | CPAchecker | 11904 | 210 | 2.3 | 9 773 | 408 | 14 | |
| 3 | UAutomizer | 11802 | 170 | 1.7 | 7 948 | 311 | 2 | **2** |
| ***JavaOverall*** | | | | | | | | |
| 1 | **JDart** | **714** | 1.2 | 0.012 | 522 | 0 | | |
| 2 | Jbmc | 700 | 0.42 | 0.0039 | 506 | 0 | | |
| 3 | Java-Ranger | 670 | 4.4 | 0.052 | 466 | 0 | | |

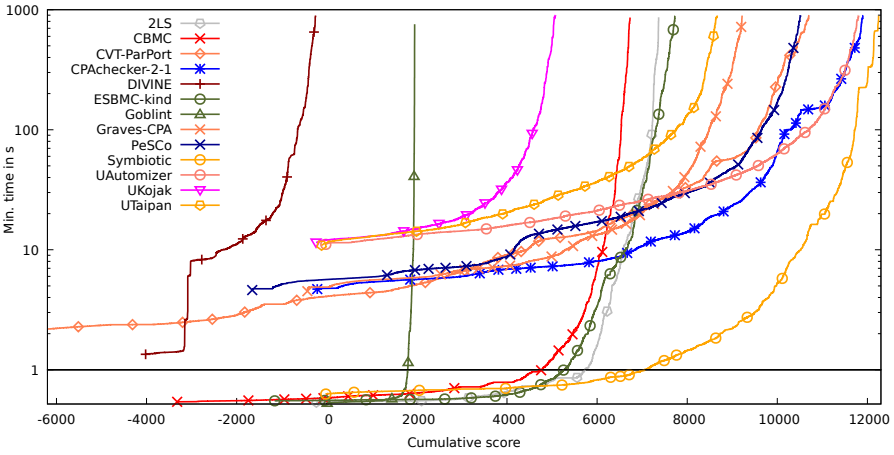


Fig. 4: Quantile functions for category *C-Overall*. Each quantile function illustrates the quantile ($x$-coordinate) of the scores obtained by correct verification runs below a certain run time ($y$-coordinate). More details were given previously [11]. A logarithmic scale is used for the time range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

total of 2.85 million verification runs consuming 19 years of CPU time, and 16.3 million validation runs consuming 11 years of CPU time.

**Quantitative Results.** Tables 8 and 9 present the quantitative overview of all tools and all categories. Due to the large number of tools, we need to split the presentation into two tables, one for the verifiers that participate in the rankings (Table 8), and one for the hors-concours verifiers (Table 9). The head row mentions the category, the maximal score for the category, and the number of verification tasks. The tools are listed in alphabetical order; every table row lists the scores of one verifier. We indicate the top three candidates by formatting their scores in bold face and in larger font size. An empty table cell means that the verifier opted-out from the respective main category (perhaps participating in subcategories only, restricting the evaluation to a specific topic). More information (including interactive tables, quantile plots for every category, and also the raw data in XML format) is available on the competition web site (`https://sv-comp.sosy-lab.org/2022/results`) and in the results artifact (see Table 4).

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

Table 11: Results of verifiers in demonstration category *NoDataRace*

| Verifier | Score | Correct *true* | Correct *false* | Incorrect *true* | Incorrect *false* |
|---|---|---|---|---|---|
| CSeq | 39 | 37 | 61 | 0 | 6 |
| Dartagnan | −299 | 47 | 23 | 13 | 0 |
| Goblint | 124 | **62** | 0 | 0 | 0 |
| Locksmith new | 34 | 17 | 0 | 0 | 0 |
| UAutomizer | 120 | 49 | 54 | 1 | 0 |
| UGemCutter new | **151** | 57 | **69** | 1 | 0 |
| UKojak | 0 | 0 | 0 | 0 | 0 |
| UTaipan | 139 | 56 | 59 | 1 | 0 |

**Score-Based Quantile Functions for Quality Assessment.** We use score-based quantile functions [11, 32] because these visualizations make it easier to understand the results of the comparative evaluation. The results archive (see Table 4) and the web site (`https://sv-comp.sosy-lab.org/2022/results`) include such a plot for each (sub-)category. As an example, we show the plot for category *C-Overall* (all verification tasks) in Fig. 4. A total of 13 verifiers participated in category *C-Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [11]). A more detailed discussion of score-based quantile plots, including examples of what insights one can obtain from the plots, is provided in previous competition reports [11, 14].

The winner of the competition, Symbiotic, not only achieves the best cummulative score (graph for Symbiotic has the longest width from $x = 0$ to its right end), but is also extremely efficient (area below the graph is very small). Verifiers whose graphs start with a negative commulative score produced wrong results. Several verifiers whose graphs start with a minimal CPU time larger than 3 s are based on Java and the time is consumed by starting the JVM.

**Demo Category *NoDataRace*.** SV-COMP 2022 had a new category on data-race detection and we report the results in Table 11. The benchmark set contained a total of 162 verification tasks. The category was defined as a demonstration category because it was not clear how many verifiers would participate. Eight verifiers specified the execution for this sub-category in their benchmark definition [3] and participated in this demonstration. A detailed table was generated by BenchExec's table-generator together with all other results as well and is available on the competition web site and in the artifact (see Table 4).

The results are presented as a means to show that such a category is useful; the results do not represent the full potential of the verifiers, as they were not fully tuned by their developers but handed in for demonstrating abilities only.

**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 12 is similar to Table 10, but contains

[3] `https://gitlab.com/sosy-lab/sv-comp/bench-defs/-/tree/svcomp22/benchmark-defs`

Table 12: 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 | GOBLINT | 1 951 | 4.9 | 0.070 | 1 574 | 0 | **0** |
| 2 | UKOJAK | 5 078 | 66 | 0.71 | 3 988 | 1 | .00020 |
| 3 | SYMBIOTIC | 12 249 | 34 | 0.44 | 7 430 | 3 | .00024 |
| worst (with pos. score) | | | | | | 282 | .042 |
| ***Green Verifiers*** | | | | | | | (J/sp) |
| 1 | GOBLINT | 1 951 | 4.9 | 0.070 | 1 574 | 0 | 120 |
| 2 | SYMBIOTIC | 12 249 | 34 | 0.44 | 7 430 | 3 | 130 |
| 3 | CBMC | 6 733 | 25 | 0.27 | 6 479 | 282 | 140 |
| worst (with pos. score) | | | | | | | 690 |

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 Results' 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 10 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 12 uses a failure rate as rank measure: $\frac{\text{number of incorrect results}}{\max(\text{total score},1)}$, 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 0.023 E/sp in SV-COMP 2021 and is now at 0.042 E/sp. GOBLINT is the best verifier regarding this measure.

*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 12 uses the energy consumption per score point as rank measure: $\frac{\text{total CPU energy}}{\max(\text{total score},1)}$, with the unit $J/sp$. The worst result from SV-COMP 2021 was 630 J/sp and is now at 690 J/sp. Also here, GOBLINT is the best verifier regarding this measure.

*New Verifiers.* To acknowledge the verification systems that participate for the first or second time in SV-COMP, Table 13 lists the new verifiers (in SV-COMP 2021 or SV-COMP 2022).

Table 13: New verifiers in SV-COMP 2021 and SV-COMP 2022; column 'Sub-categories' gives the number of executed categories (including demo category *NoDataRace*), new for first-time participants, ∅ for hors-concours participation

| Verifier | Language | First Year | Sub-categories |
|---|---|---|---|
| CVT-AlgoSel new ∅ | C | 2022 | 18 |
| CVT-ParPort new ∅ | C | 2022 | 35 |
| CPA-BAM-SMG new | C | 2022 | 16 |
| Crux new | C | 2022 | 20 |
| Deagle new | C | 2022 | 1 |
| Ebf new | C | 2022 | 1 |
| Graves-CPA new | C | 2022 | 35 |
| Infer new ∅ | C | 2022 | 25 |
| Lart new | C | 2022 | 22 |
| Locksmith new | C | 2022 | 1 |
| Sesl new | C | 2022 | 6 |
| Theta new | C | 2022 | 13 |
| UGemCutter new | C | 2022 | 2 |
| Frama-C-SV | C | 2021 | 4 |
| Gazer-Theta ∅ | C | 2021 | 9 |
| Goblint | C | 2021 | 25 |
| Korn | C | 2021 | 4 |
| GDart new | Java | 2022 | 1 |

Table 14: Confirmation rate of verification witnesses during the evaluation in SV-COMP 2022; new for first-time participants, ∅ for hors-concours participation

| Result | True | | | | False | | | |
|---|---|---|---|---|---|---|---|---|
| | Total | Confirmed | | Unconf. | Total | Confirmed | | Unconf. |
| **2ls** | 2 394 | 2 388 | **99.7 %** | **6** | 1 648 | 1 363 | 82.7 % | 285 |
| Cbmc | 3 837 | 3 493 | 91.0 % | 344 | 3 536 | 2 986 | 84.4 % | 550 |
| CVT-ParPort new ∅ | 7 440 | 7 083 | 95.2 % | 357 | 4 754 | 4 332 | 91.1 % | 422 |
| CPAchecker | 6 006 | 5 701 | 94.9 % | 305 | 4 175 | 4 072 | 97.5 % | 103 |
| Divine ∅ | 1 692 | 1 672 | 98.8 % | 20 | 1 040 | 870 | 83.7 % | 170 |
| Esbmc-kind | 5 542 | 5 483 | 98.9 % | 59 | 3 034 | 2 556 | 84.2 % | 478 |
| Goblint | 1 657 | 1 574 | 95.0 % | 83 | 0 | 0 | | |
| Graves-CPA new | 5 651 | 5 458 | 96.6 % | 193 | 3 723 | 3 576 | 96.1 % | 147 |
| PeSCo | 6 155 | 5 734 | 93.2 % | 421 | 4 116 | 3 934 | 95.6 % | 182 |
| Symbiotic | 4 878 | 4 798 | 98.4 % | 80 | 4 081 | 2 632 | 64.5 % | 1 449 |
| UAutomizer | 5 751 | 5 591 | 97.2 % | 160 | 2 508 | 2 357 | 94.0 % | 151 |
| **UKojak** | 2 875 | 2 863 | 99.6 % | 12 | 1 144 | 1 125 | **98.3 %** | **19** |
| UTaipan | 4 567 | 4 513 | 98.8 % | 54 | 1 719 | 1 576 | 91.7 % | 143 |

**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 result validator is available). We used ten independently developed witness-based result validators and one witness linter (see Table 1).

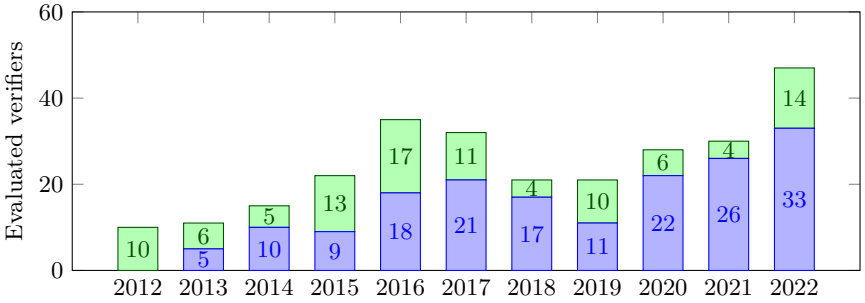


Fig. 5: Number of evaluated verifiers for each year (first-time participants on top)

Table 14 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 and 2021 [17, 18].

## 6 Conclusion

The 11th edition of the Competition on Software Verification (SV-COMP 2022) was the largest ever, with 47 participating verification systems (incl. 14 hors-concours and 14 new verifiers) (see Fig. 5 for the participation numbers and Table 5 for the details). The number of result validators was increased from 6 in 2021 to 11 in 2022, to validate the results (Table 1). The number of verification tasks was increased to 15 648 in the C category and to 586 in the Java category, and a new category on data-race detection was demonstrated. A new section in this report (Sect. 3) explains steps to reproduce verification results and to investigate problems during execution, and a new table tried to give an overview of the usage of common solver libraries and frameworks. 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. We hope that the broad overview of verification tools stimulates their further application by an ever growing user community of formal methods.

**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. For easy access, the results are presented also online on the competition web site `https://sv-comp.sosy-lab.org/2022/results`.

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

## References

1. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: Proc. TACAS (2). pp. 433–437. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_27
2. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141 (2019). https://doi.org/10.1109/ASE.2019.00121
3. 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
4. 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
5. Andrianov, P.S.: Analysis of correct synchronization of operating system components. Program. Comput. Softw. **46**, 712–730 (2020). https://doi.org/10.1134/S0361768820080022
6. Ayaziová, P., Chalupa, M., Strejček, J.: Symbiotic-Witch: A Klee-based violation witness checker (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
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.: Software verification: 10th comparative evaluation (SV-COMP 2021). In: Proc. TACAS (2). pp. 401–422. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_24
19. Beyer, D.: Status report on software testing: Test-Comp 2021. In: Proc. FASE. pp. 341–357. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_17
20. Beyer, D.: Advances in automatic software testing: Test-Comp 2022. In: Proc. FASE. LNCS 13241, Springer (2022)
21. Beyer, D.: Results of the 11th Intl. Competition on Software Verification (SV-COMP 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5831008
22. Beyer, D.: SV-Benchmarks: Benchmark set for software verification and testing (SV-COMP 2022 and Test-Comp 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5831003
23. Beyer, D.: Verification witnesses from verification tools (SV-COMP 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5838498
24. Beyer, D.: Verifiers and validators of the 11th Intl. Competition on Software Verification (SV-COMP 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5959149
25. 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
26. 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
27. 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
28. 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
29. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. Springer (2022)
30. Beyer, D., Kanav, S., Richter, C.: Construction of Verifier Combinations Based on Off-the-Shelf Verifiers. In: Proc. FASE. Springer (2022)
31. 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
32. 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
33. 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

34. Beyer, D., Spiessl, M.: The static analyzer Frama-C in SV-COMP (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
35. 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
36. 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
37. 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. LNCS 13244, Springer (2022)
38. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. ACM **58**(6), 26:1–26:66 (2011). https://doi.org/10.1145/2049697.2049700
39. 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
40. Chalupa, M., Řechtáčková, A., Mihalkovič, V., Zaoral, L., Strejček, J.: Symbiotic 9: String analysis and backward symbolic execution with loop folding (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
41. 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
42. Chowdhury, A.B., Medicherla, R.K., Venkatesh, R.: VeriFuzz: Program-aware fuzzing (competition contribution). In: Proc. TACAS (3). pp. 244–249. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_22
43. 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
44. 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
45. 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
46. 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
47. Coto, A., Inverso, O., Sales, E., Tuosto, E.: A prototype for data race detection in CSeq 3 (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
48. 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
49. 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
50. 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

51. 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
52. 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
53. 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
54. 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
55. Ernst, G.: A complete approach to loop verification with invariants and summaries. Tech. Rep. arXiv:2010.05812v2, arXiv (January 2020)
56. 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
57. 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
58. 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
59. 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
60. 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
61. 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
62. He, F., Sun, Z., Fan, H.: Deagle: An SMT-based verifier for multi-threaded programs (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
63. 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
64. 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
65. Hensel, J., Mensendiek, C., Giesl, J.: AProVE: Non-termination witnesses for C programs (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)

66. 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
67. 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
68. Howar, F., Mues, M.: GWit (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
69. 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
70. 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
71. 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
72. 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
73. Kettl, M., Lemberger, T.: The static analyzer Infer in SV-COMP (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
74. 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. LNCS 13244, Springer (2022)
75. 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
76. 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
77. Lauko, H., Ročkai, P.: Lart: Compiled abstract execution (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
78. Lauko, H., Ročkai, P., Barnat, J.: Symbolic computation via program transformation. In: Proc. ICTAC. pp. 313–332. LNCS 11187, Springer (2018). https://doi.org/10.1007/978-3-030-02508-3_17
79. Leeson, W., Dwyer, M.: Graves-CPA: A graph-attention verifier selector (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
80. 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
81. 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
82. Metta, R., Medicherla, R.K., Chakraborty, S.: BMC+Fuzz: Efficient and effective test generation. In: Proc. DATE. IEEE (2022)
83. 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

84. Mues, M., Howar, F.: GDart (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
85. 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
86. 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
87. 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
88. 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
89. Ponce-De-Leon, H., Haas, T., Meyer, R.: Dartagnan: Smt-based violation witness validation (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
90. 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
91. 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
92. 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
93. 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
94. 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
95. 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). pp. 438–442. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_28
96. 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
97. 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
98. 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

99. 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
100. Ströder, T., Giesl, J., Brockschmidt, M., Frohn, F., Fuhs, C., Hensel, J., Schneider-Kamp, P., Aschermann, C.: Automatically proving termination and memory safety for programs with pointer arithmetic. J. Autom. Reasoning **58**(1), 33–65 (2017). https://doi.org/10.1007/s10817-016-9389-x
101. 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
102. 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
103. 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
104. 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
105. Švejda, 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
106. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.10. Zenodo (2022). https://doi.org/10.5281/zenodo.5720267
107. 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
108. Wu, T., Schrammel, P., Cordeiro, L.: Wit4Java: A violation-witness validator for Java verifiers (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)
109. Ádám, Z., Bajczi, L., Dobos-Kovács, M., Hajdu, A., Molnár, V.: Theta: Portfolio of cegar-based analyses with dynamic algorithm selection (competition contribution). In: Proc. TACAS. LNCS 13244, Springer (2022)