Advances in Automatic Software Testing: Test-Comp 2022

Dirk Beyer ✉

LMU Munich, Munich, Germany

Abstract. Test-Comp 2022 is the 4th edition of the Competition on Software Testing. Research competitions are a means to provide annual comparative evaluations. Test-Comp focusses on fully automatic software test generators for C programs. The results of the competition shall be reproducible and provide an overview of the current state of the art in the area of automatic test-generation. The competition was based on 4 236 test-generation tasks for C programs. Each test-generation task consisted of a program and a test specification (error coverage, branch coverage). Test-Comp 2022 had 12 participating test generators from 5 countries.

Keywords: Software Testing · Test-Case Generation · Competition · Program Analysis · Software Validation · Software Bugs · Test Validation · Test-Comp · Benchmarking · Test Coverage · Bug Finding · Test-Suites · SV-Benchmarks · BenchExec · TestCov · CoVeriTeam

1 Introduction

The Competition on Software Testing (Test-Comp, `https://test-comp.sosy-lab.org`, [5, 6, 7, 9]) showcases the state of the art in the area of automatic software testing. For the 4th time, the competition provides an overview of the results achieved by implementations of the most recent ideas, concepts, and algorithms for fully automatic test generation. 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. We use BenchExec [20] to execute the benchmarks and the results are presented in tables and graphs on the competition web site (`https://test-comp.sosy-lab.org/2022/results`) and are available in the accompanying archives (see Table 3).

This report extends previous reports on Test-Comp [5, 6, 7, 9].
Reproduction packages are available on Zenodo (see Table 3).
✉ dirk.beyer@sosy-lab.org

E. B. Johnsen and M. Wimmer (Eds.): FASE 2022, LNCS 13241, pp. 321–335, 2022.
https://doi.org/10.1007/978-3-030-99429-7_18

Competition Goals. In summary, the goals of Test-Comp are the following [6]:

- Establish *standards* for software test generation. This means, most prominently, to develop a standard for marking input values in programs, define an exchange format for test suites, agree on a specification language for test-coverage criteria, and define how to validate the resulting test suites.
- Establish a set of *benchmarks* for software testing in the community. This means to create and maintain a set of programs together with coverage criteria, and to make those publicly available for researchers to be used in performance comparisons when evaluating a new technique.
- Provide an overview of *available tools* for test-case generation and a snapshot of the state-of-the-art in software testing to the community. This means to compare, independently from particular paper projects and specific techniques, different test generators in terms of effectiveness and performance.
- Increase the visibility and credits that *tool developers* receive. This means to provide a forum for presentation of tools and discussion of the latest technologies, and to give the participants the opportunity to publish about the development work that they have done.
- Educate PhD students and other participants on how to set up performance experiments, package tools in a way that supports reproduction, and how to perform *robust and accurate research experiments.*
- Provide *resources* to development teams that do not have sufficient computing resources and give them the opportunity to obtain results from experiments on large benchmark sets.

Related Competitions. In the field of formal methods, competitions are respected as an important evaluation method and there are many competitions [3]. We refer to the report from Test-Comp 2020 [6] for a more detailed discussion and give here only the references to the most related competitions [3, 10, 41, 43].

2 Definitions, Formats, and Rules

Organizational aspects such as the classification (automatic, off-site, reproducible, jury, training) and the competition schedule is given in the initial competition definition [5]. In the following, we repeat some important definitions that are necessary to understand the results.

Test-Generation Task. A *test-generation task* is a pair of an input program (program under test) and a test specification. A *test-generation run* is a non-interactive execution of a test generator on a single test-generation task, in order to generate a test suite according to the test specification. A *test suite* is a sequence of test cases, given as a directory of files according to the format for exchangeable test-suites.[1]

Execution of a Test Generator. Figure 1 illustrates the process of executing one test generator on the benchmark suite. One test run for a test generator gets

[1] https://gitlab.com/sosy-lab/software/test-format

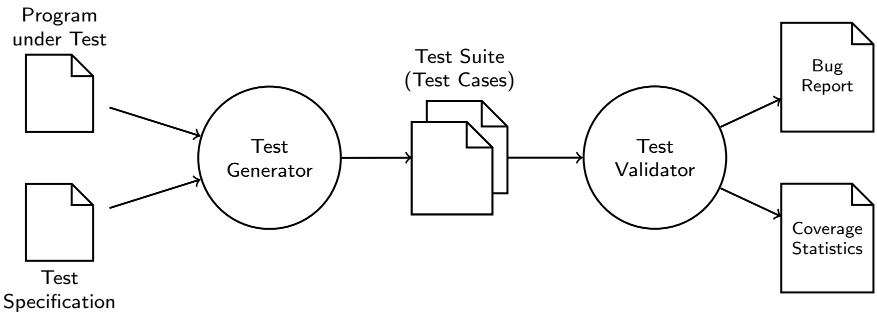

Fig. 1: Flow of the Test-Comp execution for one test generator (taken from [6])

as input (i) a program from the benchmark suite and (ii) a test specification (cover bug, or cover branches), and returns as output a test suite (i.e., a set of test cases). The test generator is contributed by a competition participant as a software archive in ZIP format. The test runs are executed centrally by the competition organizer. The test-suite validator takes as input the test suite from the test generator and validates it by executing the program on all test cases: for bug finding it checks if the bug is exposed and for coverage it reports the coverage. We use the tool TestCov [19] [2] as test-suite validator.

Test Specification. The specification for testing a program is given to the test generator as input file (either `properties/coverage-error-call.prp` or `properties/coverage-branches.prp` for Test-Comp 2022).

The definition `init(main())` is used to define the initial states of the program under test by a call of function `main` (with no parameters). The definition `FQL(f)` specifies that coverage definition `f` should be achieved. The FQL (FShell query language [30]) coverage definition `COVER EDGES(@DECISIONEDGE)` means that all branches should be covered (typically used to obtain a standard test suite for quality assurance) and `COVER EDGES(@CALL(foo))` means that a call (at least one) to function `foo` should be covered (typically used for bug finding). A complete specification looks like: `COVER(init(main()), FQL(COVER EDGES(@DECISIONEDGE)))`.

Table 1 lists the two FQL formulas that are used in test specifications of Test-Comp 2022; there was no change from 2020 (except that special function `__VERIFIER_error` does not exist anymore).

Task-Definition Format 2.0. Test-Comp 2022 used again the task-definition format in version 2.0.

License and Qualification. The license of each participating test generator must allow its free use for reproduction of the competition results. Details on qualification criteria can be found in the competition report of Test-Comp 2019 [7].

[2] `https://gitlab.com/sosy-lab/software/test-suite-validator`

Table 1: Coverage specifications used in Test-Comp 2022 (similar to 2019–2021)

Formula	Interpretation
`COVER EDGES(@CALL(reach_error))`	The test suite contains at least one test that executes function `reach_error`.
`COVER EDGES(@DECISIONEDGE)`	The test suite contains tests such that all branches of the program are executed.

3 Categories and Scoring Schema

Benchmark Programs. The input programs were taken from the largest and most diverse open-source repository of software-verification and test-generation tasks [3], which is also used by SV-COMP [8]. As in 2020 and 2021, we selected all programs for which the following properties were satisfied (see issue on GitHub [4] and report [7]):

1. compiles with `gcc`, if a harness for the special methods [5] is provided,
2. should contain at least one call to a nondeterministic function,
3. does not rely on nondeterministic pointers,
4. does not have expected result ‘false’ for property ‘termination’, and
5. has expected result ‘false’ for property ‘unreach-call’ (only for category *Error Coverage*).

This selection yielded a total of 4 236 test-generation tasks, namely 776 tasks for category *Error Coverage* and 3 460 tasks for category *Code Coverage*. The test-generation tasks are partitioned into categories, which are listed in Tables 6 and 7 and described in detail on the competition web site.[6] Figure 2 illustrates the category composition.

Category Error-Coverage. The first category is to show the abilities to discover bugs. The benchmark set consists of programs that contain a bug. Every run will be started by a batch script, which produces for every tool and every test-generation task one of the following scores: 1 point, if the validator succeeds in executing the program under test on a generated test case that explores the bug (i.e., the specified function was called), and 0 points, otherwise.

Category Branch-Coverage. The second category is to cover as many branches of the program as possible. The coverage criterion was chosen because many test generators support this standard criterion by default. Other coverage criteria can be reduced to branch coverage by transformation [29]. Every run will be started by a batch script, which produces for every tool and every

[3] https://github.com/sosy-lab/sv-benchmarks
[4] https://github.com/sosy-lab/sv-benchmarks/pull/774
[5] https://test-comp.sosy-lab.org/2022/rules.php
[6] https://test-comp.sosy-lab.org/2022/benchmarks.php

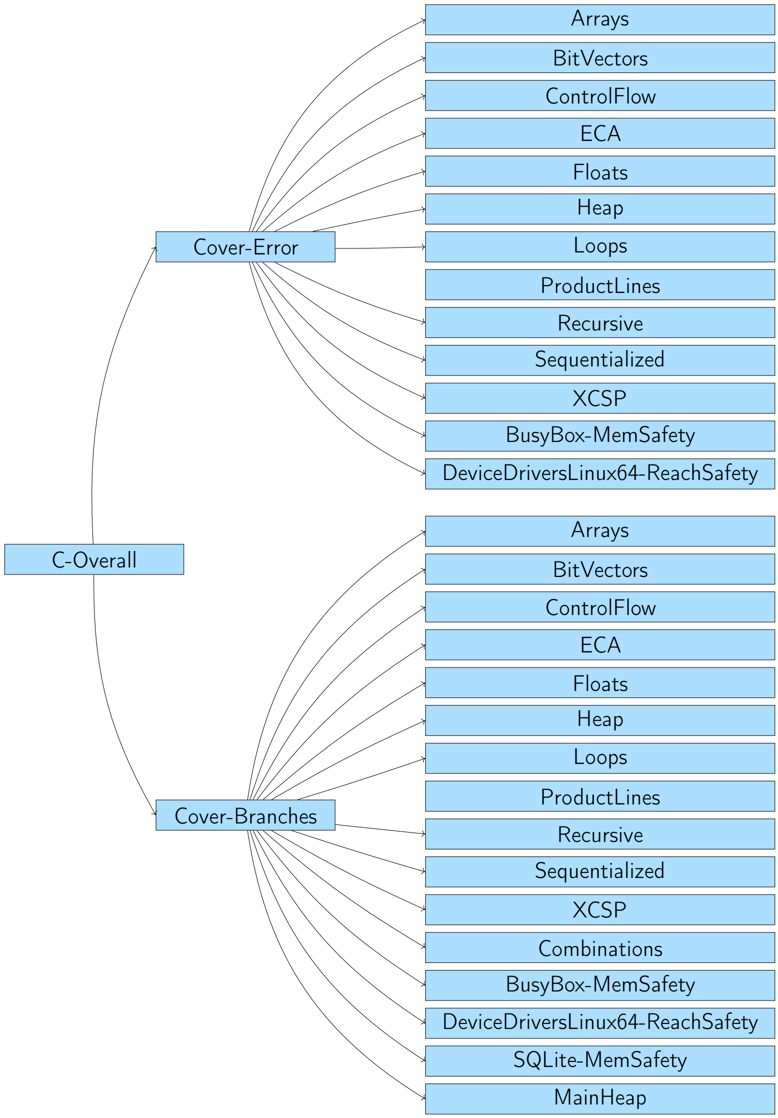

Fig. 2: Category structure for Test-Comp 2022; compared to Test-Comp 2021, sub-category *ProductLines* was added to both main categories *Cover-Error* and *Cover-Branches*

test-generation task the coverage of branches of the program (as reported by TestCov [19]; a value between 0 and 1) that are executed for the generated test cases. The score is the returned coverage.

Ranking. The ranking was decided based on the sum of points (normalized for meta categories). In case of a tie, the ranking was decided based on the run time,

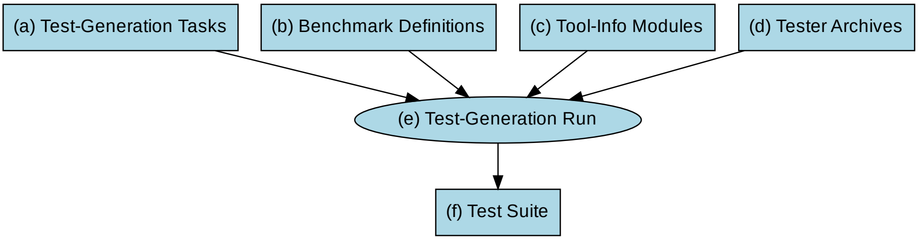

Fig. 3: Benchmarking components of Test-Comp and competition's execution flow (same as for Test-Comp 2020)

Table 2: Publicly available components for reproducing Test-Comp 2022

Component	Fig. 3	Repository	Version
Test-Generation Tasks	(a)	gitlab.com/sosy-lab/benchmarking/sv-benchmarks	testcomp22
Benchmark Definitions	(b)	gitlab.com/sosy-lab/test-comp/bench-defs	testcomp22
Tool-Info Modules	(c)	github.com/sosy-lab/benchexec	3.10
Test-Generator Archives	(d)	gitlab.com/sosy-lab/test-comp/archives-2022	testcomp22
Benchmarking	(e)	github.com/sosy-lab/benchexec	3.10
Test-Suite Format	(f)	gitlab.com/sosy-lab/software/test-format	testcomp22

which is the total CPU time over all test-generation tasks. Opt-out from categories was possible and scores for categories were normalized based on the number of tasks per category (see competition report of SV-COMP 2013 [4], page 597).

4 Reproducibility

We followed the same competition workflow that was described in detail in the previous competition report (see Sect. 4, [9]). All major components that were used for the competition were made available in public version-control repositories. An overview of the components that contribute to the reproducible setup of Test-Comp is provided in Fig. 3, and the details are given in Table 2. We refer to the report of Test-Comp 2019 [7] for a thorough description of all components of the Test-Comp organization and how we ensure that all parts are publicly available for maximal reproducibility.

In order to guarantee long-term availability and immutability of the test-generation tasks, the produced competition results, and the produced test suites, we also packaged the material and published it at Zenodo (see Table 3).

The competition used CoVeriTeam [17][7] again to provide participants access to the actual competition machines. The competition report of SV-COMP 2022 provides a description on reproducing individual results and on trouble-shooting (see Sect. 3, [10]).

[7] https://gitlab.com/sosy-lab/software/coveriteam

Table 3: Artifacts published for Test-Comp 2022

Content	DOI	Reference
Test-Generation Tasks	10.5281/zenodo.5831003	[12]
Competition Results	10.5281/zenodo.5831012	[11]
Test-Suite Generators	10.5281/zenodo.5959598	[13]
Test Suites (Witnesses)	10.5281/zenodo.5831010	[14]
BenchExec	10.5281/zenodo.5720267	[47]

Table 4: Competition candidates with tool references and representing jury members; new indicates first-time participants, ∅ indicates hors-concours participation

Tester	**Ref.**	**Jury member**	**Affiliation**
CMA-ES Fuzz∅	[34]	(hors concours)	–
CoVeriTest	[16, 33]	Marie-Christine Jakobs	TU Darmstadt, Germany
FuSeBMC	[1, 2]	Kaled Alshmrany	U. of Manchester, UK
HybridTiger∅	[22, 42]	(hors concours)	–
Klee∅	[23, 24]	(hors concours)	–
Legion	[38, 39]	Gidon Ernst	LMU Munich, Germany
Legion/SymCC new	[39]	Gidon Ernst	LMU Munich, Germany
LibKluzzer	[36]	Hoang M. Le	U. of Bremen, Germany
PRTest	[18, 37]	Thomas Lemberger	LMU Munich, Germany
Symbiotic	[25, 26]	Marek Chalupa	Masaryk U., Brno, Czechia
TracerX	[31, 32]	Joxan Jaffar	National U., Singapore
VeriFuzz	[40]	Raveendra Kumar M.	Tata Consultancy Services, India

5 Results and Discussion

This section represents the results of the competition experiments. The report shall help to understanding the state of the art and the advances in fully automatic test generation for whole C programs, in terms of effectiveness (test coverage, as accumulated in the score) and efficiency (resource consumption in terms of CPU time). All results mentioned in this article were inspected and approved by the participants.

Participating Test Generators. Table 4 provides an overview of the participating test generators and references to publications, as well as the team representatives of the jury of Test-Comp 2022. (The competition jury consists of the chair and one member of each participating team.) An online table with information about all participating systems is provided on the competition web site.[8] Table 5 lists the features and technologies that are used in the test generators.

There are test generators that did not actively participate (e.g., tester archives taken from last year) and that are not included in rankings. Those are called *hors-concours* participations and the tool names are labeled with a symbol (∅).

[8] https://test-comp.sosy-lab.org/2022/systems.php

Table 5: Technologies and features that the test generators used

Participant	Bounded Model Checking	CEGAR	Evolutionary Algorithms	Explicit-Value Analysis	Floating-Point Arithmetics	Guidance by Coverage Measures	Predicate Abstraction	Random Execution	Symbolic Execution	Targeted Input Generation	Algorithm Selection	Portfolio
CMA-ES Fuzz∅			✓	✓	✓	✓		✓				
CoVeriTest		✓		✓	✓		✓					✓
FuSeBMC	✓				✓	✓				✓		✓
HybridTiger∅		✓		✓	✓		✓					
Klee∅					✓				✓	✓		
Legion				✓	✓	✓		✓	✓	✓		
Legion/SymCC new				✓	✓	✓		✓	✓	✓		
LibKluzzer					✓	✓		✓	✓			
PRTest					✓			✓				
Symbiotic					✓	✓			✓	✓		✓
TracerX	✓				✓				✓	✓		
VeriFuzz	✓		✓	✓	✓	✓		✓				

Computing Resources. The computing environment and the resource limits were the same as for Test-Comp 2020 [6]: Each test run was limited to 8 processing units (cores), 15 GB of memory, and 15 min of CPU time. The test-suite validation was limited to 2 processing units, 7 GB of memory, and 5 min of CPU time. The machines for running the experiments are part of a compute cluster that consists of 167 machines; each test-generation 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 [20] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud[9] to distribute, install,

[9] https://vcloud.sosy-lab.org

Table 6: Quantitative overview over all results; empty cells mark opt-outs; new indicates first-time participants, ∅ indicates hors-concours participation

Tester	**Cover-Error** 776 tasks	**Cover-Branches** 3460 tasks	**Overall** 4236 tasks
CMA-ES Fuzz∅	0	624	382
CoVeriTest	423	1860	2293
FuSeBMC	**628**	**2104**	**3003**
HybridTiger∅	355	1406	1830
Klee∅	500	1242	2125
Legion	57	1033	787
Legion/SymCC new		1487	
LibKluzzer	**528**	**1990**	**2658**
PRTest	145	896	945
Symbiotic	463	1802	2367
TracerX	0	1746	1069
VeriFuzz	**623**	**2075**	**2971**

run, and clean-up test-case generation 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 use CPU Energy Meter [21] (integrated in BenchExec [20]). Further technical parameters of the competition machines are available in the repository which also contains the benchmark definitions. [10]

One complete test-generation execution of the competition consisted of 50 056 single test-generation runs. The total CPU time was 339 days and the consumed energy 88 kWh for one complete competition run for test generation (without validation). Test-suite validation consisted of 50 832 single test-suite validation runs. The total consumed CPU time was 15 days. 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 311 754 test-generation runs (consuming 4.9 years of CPU time). The CPU energy was not measured during preruns.

Quantitative Results. The quantitative results are presented in the same way as last year: Table 6 presents the quantitative overview of all tools and all categories. The head row mentions the category and the number of test-generation

[10] https://gitlab.com/sosy-lab/test-comp/bench-defs/tree/testcomp22

Table 7: Overview of the top-three test generators for each category (measurement values for CPU time and energy rounded to two significant digits)

Rank	Tester	Score	CPU Time (in h)	CPU Energy (in kWh)
Cover-Error				
1	FuSeBMC	**628**	22	0.28
2	VeriFuzz	623	3.5	0.039
3	LibKluzzer	528	140	1.5
Cover-Branches				
1	FuSeBMC	**2104**	850	11
2	VeriFuzz	2075	850	11
3	LibKluzzer	1990	760	8.3
Overall				
1	FuSeBMC	**3003**	870	11
2	VeriFuzz	2971	860	11
3	LibKluzzer	2658	900	9.8

tasks in that category. The tools are listed in alphabetical order; every table row lists the scores of one test generator. 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 test generator 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 [11] and in the results artifact (see Table 3). Table 7 reports the top three test generators for each category. The consumed run time (column 'CPU Time') is given in hours and the consumed energy (column 'Energy') is given in kWh.

Score-Based Quantile Functions for Quality Assessment. We use score-based quantile functions [20] because these visualizations make it easier to understand the results of the comparative evaluation. The web site [11] and the results artifact (Table 3) include such a plot for each category; as example, we show the plot for category *Overall* (all test-generation tasks) in Fig. 4. We had 11 test generators participating in category *Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [4]). A more detailed discussion of score-based quantile plots for testing is provided in the Test-Comp 2019 competition report [7].

Alternative Rankings. Table 8 is similar to Table 7, but contains the alternative ranking category *Green Testing*. Column 'Quality' gives the score in score points (sp), column 'CPU Time' the CPU usage in hours (h), column

[11] https://test-comp.sosy-lab.org/2022/results

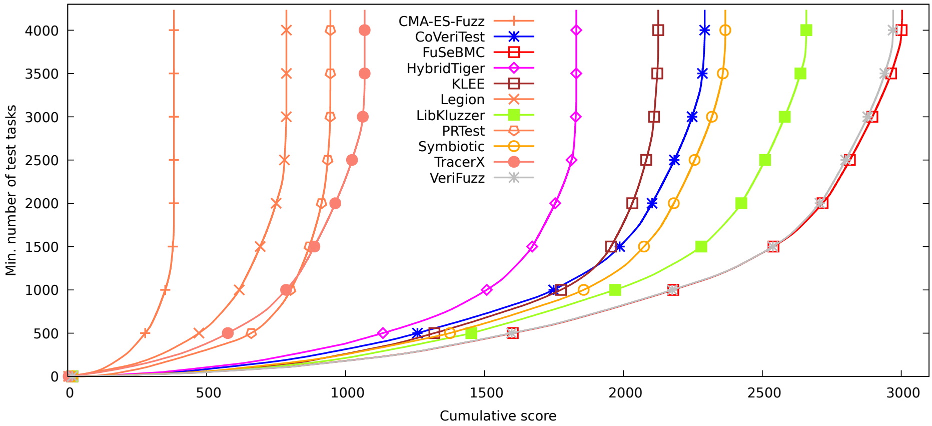

Fig. 4: Quantile functions for category *Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by test-generation runs below a certain number of test-generation tasks (y-coordinate). More details were given previously [7]. The graphs are decorated with symbols to make them better distinguishable without color.

Table 8: Alternative rankings; quality is given in score points (sp), CPU time in hours (h), energy in kilo-watt-hours (kWh), the first rank measure in kilo-joule per score point (kJ/sp), and the second rank measure in score points (sp); measurement values are rounded to 2 significant digits

Rank	**Test Generator**	**Quality (sp)**	**CPU Time (h)**	**CPU Energy (kWh)**	**Rank Measure**
Green Testing					**(kJ/sp)**
1	**TracerX**	1 069	120	1.4	4.8
2	Klee$^{\varnothing}$	2 125	310	3.5	6.0
3	Symbiotic	2 367	540	5.9	9.0
worst					41

'CPU Energy' the CPU usage in kilo-watt-hours (kWh), and column 'Rank Measure' reports the values for the rank measure.

Green Testing — Low Energy Consumption. Since a large part of the cost of test generation is caused by the energy consumption, it might be important to also consider the energy efficiency in rankings, as complement to the official Test-Comp ranking. This alternative ranking category uses the energy consumption per score point as rank measure: $\frac{\text{CPU Energy}}{\text{Quality}}$, with the unit kilo-joule per score point (kJ/sp).The energy is measured using CPU Energy Meter [21], which we use as part of BenchExec [20].

New Test Generators. To acknowledge the test generators that participated for the first time in Test-Comp, we list the test generators that participated for the first time. CMA-ES Fuzz$^{\varnothing}$ and FuSeBMC participated for the first time in

Table 9: New verifiers in Test-Comp 2021 and Test-Comp 2022; column 'Sub-categories' gives the number of executed categories

Verifier	Language	First Year	Sub-categories
Legion/SymCC new	C	2022	16
CMA-ES Fuzz ∅	C	2021	30
FuSeBMC	C	2021	30

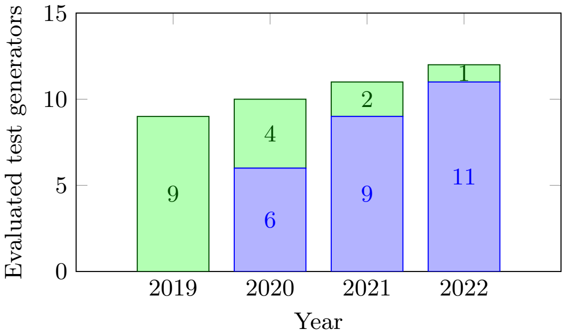

Fig. 5: Number of evaluated test generators for each year (top: number of first-time participants; bottom: previous year's participants)

Test-Comp 2021, and Legion/SymCC new participated first in Test-Comp 2022. Table 9 reports also the number of subcategories in which the tools participated.

6 Conclusion

For the 4th time, the Competition on Software Testing took place and provides an overview of test-generation tools for C programs. The competition event attracted 12 participating teams (see Fig. 5 for the participation numbers and Table 4 for the details). The competition is an off-site competition, the execution of the experiments is fully-automatatic and reproducible. To ensure transparency, all components are made available in public repositories and a jury (consisting of members from each team) oversees the process. The produced test suites are validated by the test-suite validator TestCov. The results of the competition are presented at the 25th International Conference on Fundamental Approaches to Software Engineering at ETAPS 2022.

Data-Availability Statement. The test-generation tasks and results of the competition are published at Zenodo, as described in Table 3. All components and data that are necessary for reproducing the competition are available in public version repositories, as specified in Table 2. For easy access, the results are presented also online on the competition web site `https://test-comp.sosy-lab.org/2022/results`.

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

References

1. Alshmrany, K., Aldughaim, M., Cordeiro, L., Bhayat, A.: FuSeBMC v.4: Smart seed generation for hybrid fuzzing (competition contribution). In: Proc. FASE. LNCS 13241, Springer (2022)
2. Alshmrany, K.M., Aldughaim, M., Bhayat, A., Cordeiro, L.C.: FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. In: Proc. TAP. pp. 85–105. Springer (2021). https://doi.org/10.1007/978-3-030-79379-1_6
3. 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
4. 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
5. Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_11
6. Beyer, D.: Second competition on software testing: Test-Comp 2020. In: Proc. FASE. pp. 505–519. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_25
7. Beyer, D.: First international competition on software testing (Test-Comp 2019). Int. J. Softw. Tools Technol. Transf. **23**(6), 833–846 (December 2021). https://doi.org/10.1007/s10009-021-00613-3
8. 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
9. 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
10. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS. LNCS 13244, Springer (2022)
11. Beyer, D.: Results of the 4th Intl. Competition on Software Testing (Test-Comp 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5831012
12. Beyer, D.: SV-Benchmarks: Benchmark set for softwware verification and testing (SV-COMP 2022 and Test-Comp 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5831003
13. Beyer, D.: Test-suite generators and validator of the 4th Intl. Competition on Software Testing (Test-Comp 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5959598
14. Beyer, D.: Test suites from test-generation tools (Test-Comp 2022). Zenodo (2022). https://doi.org/10.5281/zenodo.5831010
15. Beyer, D., Chlipala, A.J., Henzinger, T.A., Jhala, R., Majumdar, R.: Generating tests from counterexamples. In: Proc. ICSE. pp. 326–335. IEEE (2004). https://doi.org/10.1109/ICSE.2004.1317455
16. Beyer, D., Jakobs, M.C.: CoVeriTest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_23

17. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. Springer (2022)
18. Beyer, D., Lemberger, T.: Software verification: Testing vs. model checking. In: Proc. HVC. pp. 99–114. LNCS 10629, Springer (2017). https://doi.org/10.1007/978-3-319-70389-3_7
19. Beyer, D., Lemberger, T.: TestCov: Robust test-suite execution and coverage measurement. In: Proc. ASE. pp. 1074–1077. IEEE (2019). https://doi.org/10.1109/ASE.2019.00105
20. 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
21. 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
22. Bürdek, J., Lochau, M., Bauregger, S., Holzer, A., von Rhein, A., Apel, S., Beyer, D.: Facilitating reuse in multi-goal test-suite generation for software product lines. In: Proc. FASE. pp. 84–99. LNCS 9033, Springer (2015). https://doi.org/10.1007/978-3-662-46675-9_6
23. Cadar, C., Dunbar, D., Engler, D.R.: Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008)
24. Cadar, C., Nowack, M.: Klee symbolic execution engine in 2019 (competition contribution). Int. J. Softw. Tools Technol. Transf. **23**(6), 867 – 870 (December 2021). https://doi.org/10.1007/s10009-020-00570-3
25. Chalupa, M., Novák, J., Strejček, J.: Symbiotic 8: Parallel and targeted test generation (competition contribution). In: Proc. FASE. pp. 368–372. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_20
26. 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
27. Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)
28. Godefroid, P., Sen, K.: Combining model checking and testing. In: Handbook of Model Checking, pp. 613–649. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_19
29. Harman, M., Hu, L., Hierons, R.M., Wegener, J., Sthamer, H., Baresel, A., Roper, M.: Testability transformation. IEEE Trans. Software Eng. **30**(1), 3–16 (2004). https://doi.org/10.1109/TSE.2004.1265732
30. Holzer, A., Schallhart, C., Tautschnig, M., Veith, H.: How did you specify your test suite. In: Proc. ASE. pp. 407–416. ACM (2010). https://doi.org/10.1145/1858996.1859084
31. Jaffar, J., Maghareh, R., Godboley, S., Ha, X.L.: TracerX: Dynamic symbolic execution with interpolation (competition contribution). In: Proc. FASE. pp. 530–534. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_28
32. Jaffar, J., Murali, V., Navas, J.A., Santosa, A.E.: Tracer: A symbolic execution tool for verification. In: Proc. CAV. pp. 758–766. LNCS 7358, Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_61
33. Jakobs, M.C., Richter, C.: CoVeriTest with adaptive time scheduling (competition contribution). In: Proc. FASE. pp. 358–362. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_18

34. Kim, H.: Fuzzing with stochastic optimization (2020), Bachelor's Thesis, LMU Munich
35. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252
36. Le, H.M.: Llvm-based hybrid fuzzing with LibKluzzer (competition contribution). In: Proc. FASE. pp. 535–539. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_29
37. Lemberger, T.: Plain random test generation with PRTest (competition contribution). Int. J. Softw. Tools Technol. Transf. **23**(6), 871–873 (December 2021). https://doi.org/10.1007/s10009-020-00568-x
38. Liu, D., Ernst, G., Murray, T., Rubinstein, B.: Legion: Best-first concolic testing (competition contribution). In: Proc. FASE. pp. 545–549. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_31
39. Liu, D., Ernst, G., Murray, T., Rubinstein, B.I.P.: Legion: Best-first concolic testing. In: Proc. ASE. pp. 54–65. IEEE (2020). https://doi.org/10.1145/3324884.3416629
40. Metta, R., Kumar, M.R., Karmarkar, H.: VeriFuzz: Fuzz centric test generation tool (competition contribution). In: Proc. FASE. LNCS 13241, Springer (2022)
41. Panichella, S., Gambi, A., Zampetti, F., Riccio, V.: SBST tool competition 2021. In: Proc. SBST. pp. 20–27. IEEE (2021). https://doi.org/10.1109/SBST52555.2021.00011
42. Ruland, S., Lochau, M., Jakobs, M.C.: HybridTiger: Hybrid model checking and domination-based partitioning for efficient multi-goal test-suite generation (competition contribution). In: Proc. FASE. pp. 520–524. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_26
43. Song, J., Alves-Foss, J.: The DARPA cyber grand challenge: A competitor's perspective, part 2. IEEE Security and Privacy **14**(1), 76–81 (2016). https://doi.org/10.1109/MSP.2016.14
44. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Proc. IJCAR, pp. 367–373. LNCS 8562, Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_28
45. Sutcliffe, G.: The CADE ATP system competition: CASC. AI Magazine **37**(2), 99–101 (2016)
46. Visser, W., Păsăreanu, C.S., Khurshid, S.: Test-input generation with Java PathFinder. In: Proc. ISSTA. pp. 97–107. ACM (2004). https://doi.org/10.1145/1007512.1007526
47. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.10. Zenodo (2022). https://doi.org/10.5281/zenodo.5720267