Software Testing: 5th Comparative Evaluation: Test-Comp 2023

Dirk Beyer(✉)

LMU Munich, Munich, Germany

Abstract. The 5th edition of the Competition on Software Testing (Test-Comp 2023) provides again an overview and comparative evaluation of automatic test-suite generators for C programs. The experiment was performed on a benchmark set of 4 106 test-generation tasks for C programs. Each test-generation task consisted of a program and a test specification (error coverage, branch coverage). There were 13 participating test-suite generators from 6 countries in Test-Comp 2023.

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

In its 5th edition, the International Competition on Software Testing (Test-Comp, `https://test-comp.sosy-lab.org`, [7,8,9,10,11]) again compares automatic test-suite generators for C programs, in order to showcase the state of the art in the area of automatic software testing. This competition report is an update of the previous reports, referring to the rules and definitions, presents the competition results, and give some interesting data about the execution of the competition experiments. We use BenchExec [24] to execute the benchmarks and the results are presented in tables and graphs on the competition web site (`https://test-comp.sosy-lab.org/2023/results`) and are available in the accompanying archives (see Table 3).

Competition Goals. In summary, the goals of Test-Comp are the following [8]:

- 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.

This report extends previous reports on Test-Comp [7,8,9,10,11].
Reproduction packages are available on Zenodo (see Table 3).
(✉) `dirk.beyer@sosy-lab.org`

L. Lambers and S. Uchitel (Eds.): FASE 2023, LNCS 13991, pp. 309–323, 2023.
https://doi.org/10.1007/978-3-031-30826-0_17

- 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 [5]. We refer to the report from Test-Comp 2020 [8] for a more detailed discussion and give here only the references to the most related competitions [5,13,46,48].

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 [7]. 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-suite generator on the benchmark suite. One test run for a test-suite generator gets 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

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

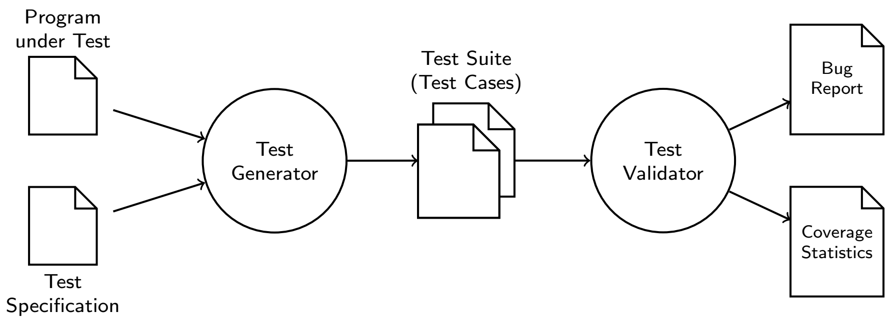

Fig. 1: Flow of the Test-Comp execution for one test generator (taken from [8])

Table 1: Coverage specifications used in Test-Comp 2023 (similar to 2019–2022)

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.

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 [23] [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 2023).

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 [36]) 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 2023; there was no change from 2020 (except that special function `__VERIFIER_error` does not exist anymore).

Task-Definition Format 2.0. Test-Comp 2023 used again the task-definition format in version 2.0.

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

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 [9].

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 [13]. As in 2020 and 2021, we selected all programs for which the following properties were satisfied (see issue on GitLab [4] and report [9]):

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 4 106 test-generation tasks, namely 1 173 tasks for category *Error Coverage* and 2 933 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. We produce 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 [35]. We produce for every tool and every test-generation task the coverage of branches of the program (as reported by TestCov [23]; 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, 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 [6], page 597).

[3] https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks
[4] https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/merge_requests/774
[5] https://test-comp.sosy-lab.org/2023/rules.php
[6] https://test-comp.sosy-lab.org/2023/benchmarks.php

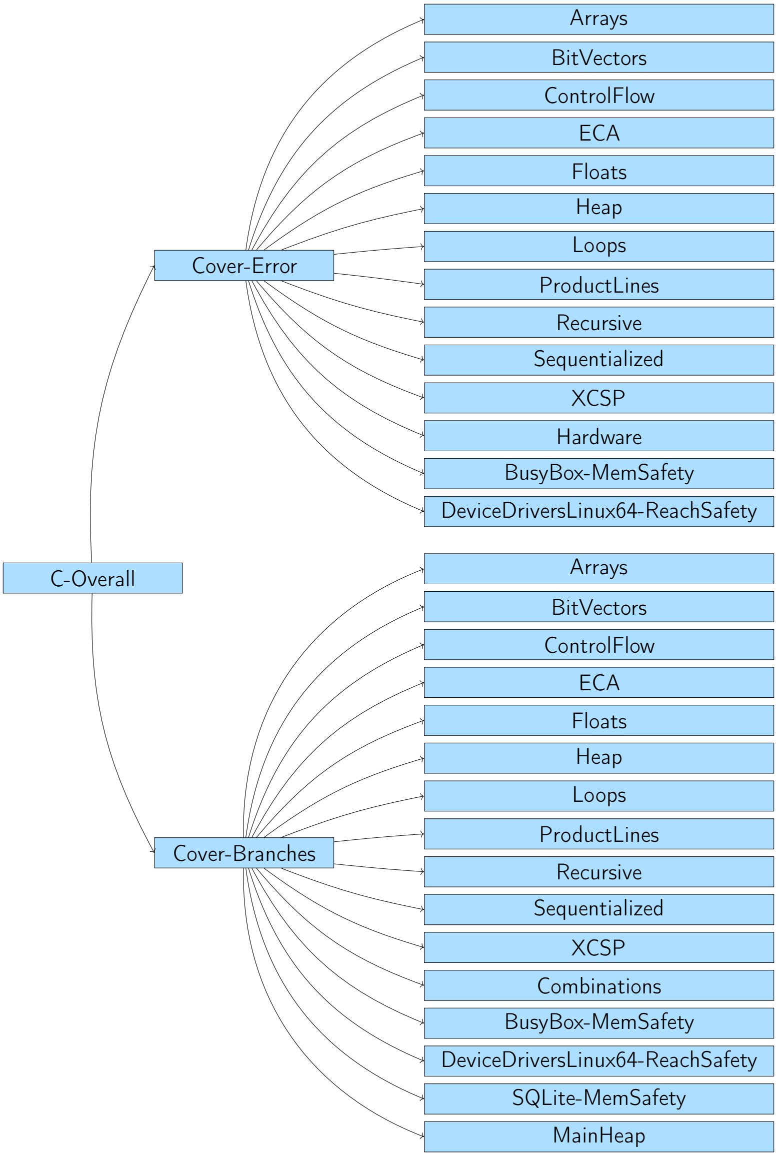

Fig. 2: Category structure for Test-Comp 2023; compared to Test-Comp 2022, sub-category *Hardware* was added to main category *Cover-Error*

4 Reproducibility

We followed the same competition workflow that was described in detail in the previous competition report (see Sect. 4, [10]). All major components that were used for the competition were made available in public version-control

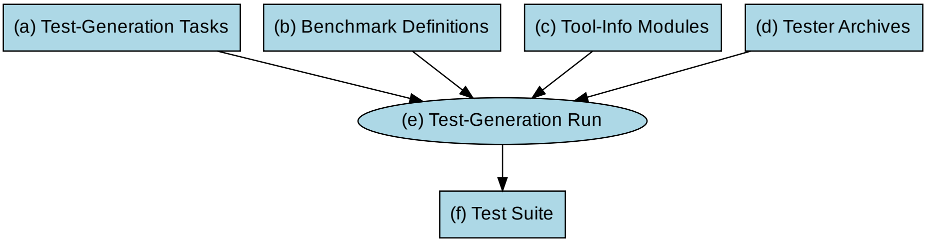

Fig. 3: Benchmarking components of Test-Comp and competition's execution flow (same as for Test-Comp 2020)

Table 2: Publicly available components for reproducing Test-Comp 2023

Component	Fig. 3	Repository	Version
Test-Generation Tasks	(a)	gitlab.com/sosy-lab/benchmarking/sv-benchmarks	testcomp23
Benchmark Definitions	(b)	gitlab.com/sosy-lab/test-comp/bench-defs	testcomp23
Tool-Info Modules	(c)	github.com/sosy-lab/benchexec	3.16
Test-Generator Archives	(d)	gitlab.com/sosy-lab/test-comp/archives-2023	testcomp23
Benchmarking	(e)	github.com/sosy-lab/benchexec	3.16
Test-Suite Format	(f)	gitlab.com/sosy-lab/software/test-format	testcomp23
Continuous Integration	(f)	gitlab.com/sosy-lab/software/coveriteam	1.0

Table 3: Artifacts published for Test-Comp 2023

Content	DOI	Reference
Test-Generation Tasks	10.5281/zenodo.7627783	[15]
Competition Results	10.5281/zenodo.7701122	[14]
Test-Suite Generators	10.5281/zenodo.7701118	[16]
Test Suites (Witnesses)	10.5281/zenodo.7701126	[17]
BenchExec	10.5281/zenodo.7612021	[52]
CoVeriTeam	10.5281/zenodo.7635975	[21]

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 [9] 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 [20] [7] again to provide participants access to execution machines that are similar to actual competition machines. The

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

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
CoVeriTest	[19,39]	Marie-Christine Jakobs	TU Darmstadt, Germany
ESBMC-kind new	[33,32]	Rafael Sá Menezes	U. of Manchester, UK
FuSeBMC	[3,4]	Kaled Alshmrany	U. of Manchester, UK
FuSeBMC_IA new	[1,2]	Mohannad Aldughaim	U. of Manchester, UK
HybridTiger	[26,47]	(hors concours)	–
KLEE	[27,28]	(hors concours)	–
Legion	[42,43]	(hors concours)	–
Legion/SymCC	[43]	Gidon Ernst	LMU Munich, Germany
PRTest	[22,41]	Thomas Lemberger	QAware GmbH, Germany
Symbiotic	[29,30]	Marek Trtík	Masaryk U., Brno, Czechia
TracerX	[37,38]	Joxan Jaffar	National U. of Singapore, Singapore
VeriFuzz	[45]	Raveendra Kumar M.	Tata Consultancy Services, India
WASP-C new	[44]	Filipe Marques	INESC-ID, Lisbon, Portugal

competition report of SV-COMP 2022 provides a description on reproducing individual results and on trouble-shooting (see Sect. 3, [12]).

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-Suite 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 2023. (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 (∅).

Computing Resources. The computing environment and the resource limits were the same as for Test-Comp 2020 [8], except for the upgraded operating system: 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,

[8] https://test-comp.sosy-lab.org/2023/systems.php

Table 5: Technologies and features that the test generators used

Tester	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
CoVeriTest		✓		✓	✓		✓					✓
ESBMC-kind new	✓			✓	✓							
FuSeBMC	✓				✓	✓				✓		✓
FuSeBMC_IA new	✓				✓	✓				✓		✓
HybridTiger		✓		✓	✓		✓					
KLEE					✓				✓	✓		
Legion				✓	✓	✓		✓	✓	✓		
Legion/SymCC				✓	✓	✓		✓	✓	✓		
PRTest					✓			✓				
Symbiotic					✓	✓			✓	✓		✓
TracerX	✓				✓				✓	✓		
VeriFuzz	✓		✓	✓	✓	✓		✓				
WASP-C new					✓			✓	✓			

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 168 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 22.04 with Linux kernel 5.15). We used BenchExec [24] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud[9] to distribute, install, 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 [25] (integrated in BenchExec [24]). Further technical parameters of the competition machines are available in the repository which also contains the benchmark definitions. [10]

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

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

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

Tester	**Cover-Error** 1173 tasks	**Cover-Branches** 2933 tasks	**Overall** 4106 tasks
CoVeriTest	581	1509	2073
ESBMC-kind new	289		
FuSeBMC	**936**	**1678**	**2813**
FuSeBMC_IA new	**908**	**1538**	**2666**
HybridTiger	463	1170	1629
KLEE	721	999	1961
Legion		838	
Legion/SymCC	349	1027	1329
PRTest	222	770	927
Symbiotic	644	1430	2128
TracerX		1400	
VeriFuzz	**909**	**1546**	**2673**
WASP-C new	570	1103	1770

One complete test-generation execution of the competition consisted of 50 445 single test-generation runs in 25 run sets (tester × property). The total CPU time was 315 days and the consumed energy 89.9 kWh for one complete competition run for test generation (without validation). Test-suite validation consisted of 53 378 single test-suite validation runs in 26 run sets (validator × property). The total consumed CPU time was 19 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 254 445 test-generation runs (consuming 3.0 years of CPU time). The prerun test-suite validation consisted of 338 710 single test-suite validation runs in 152 run sets (validator × property) (consuming 63 days of CPU time). The CPU energy was not measured during preruns.

New Test-Suite Generators. To acknowledge the test-suite generators that participated for the first time in Test-Comp, we list the test generators that participated for the first time. ESBMC-kind new, FuSeBMC_IA new, and WASP-C new participated for the first time in Test-Comp 2023, and Legion/SymCC participated first in Test-Comp 2022. Table 8 reports also the number of subcategories in which the tools participated.

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**	**936**	72	0.96
2	VeriFuzz	909	4.5	0.049
3	FuSeBMC_IA new	908	37	0.48
Cover-Branches				
1	**FuSeBMC**	**1678**	720	9.2
2	VeriFuzz	1546	730	9.1
3	FuSeBMC_IA new	1538	470	6.0
Overall				
1	**FuSeBMC**	**2813**	790	10
2	VeriFuzz	2673	730	9.2
3	FuSeBMC_IA new	2666	500	6.5

Table 8: New test-suite generators in Test-Comp 2022 and Test-Comp 2023; column 'Sub-categories' gives the number of executed categories

Tester	Language	First Year	Sub-categories
ESBMC-kind new	C	2023	14
FuSeBMC_IA new	C	2023	30
WASP-C new	C	2023	30
Legion/SymCC	C	2022	16

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 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.

[11] https://test-comp.sosy-lab.org/2023/results

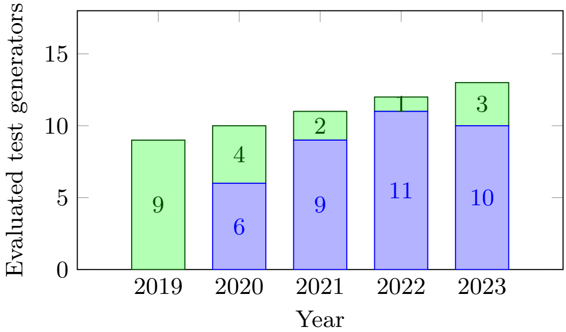

Fig. 4: Number of evaluated test generators for each year (top: number of first-time participants; bottom: previous year's participants)

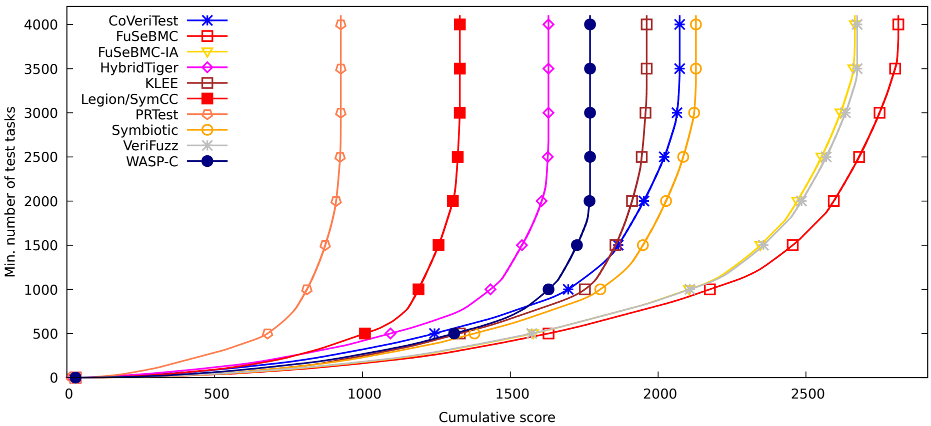

Fig. 5: Quantile functions for category *Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by test-generation runs below a certain number of test-generation tasks (y-coordinate). More details were given previously [9]. The graphs are decorated with symbols to make them better distinguishable without color.

Score-Based Quantile Functions for Quality Assessment. We use score-based quantile functions [24] 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. 5. 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 [6]). A more detailed discussion of score-based quantile plots for testing is provided in the Test-Comp 2019 competition report [9].

6 Conclusion

The Competition on Software Testing took place for the 5th time and provides an overview of fully-automatic test-generation tools for C programs. A total of 13 test-suite generators was compared (see Fig. 4 for the participation numbers and Table 4 for the details). This off-site competition uses a benchmark infrastructure that makes the execution of the experiments fully-automatic and reproducible. Transparency is ensured by making all components available in public repositories and have a jury (consisting of members from each team) that oversees the process. All test suites were validated by the test-suite validator TestCov [23] to measure the coverage. The results of the competition are presented at the 26th International Conference on Fundamental Approaches to Software Engineering at ETAPS 2023.

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/2023/results.

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

References

1. Aldughaim, M., Alshmrany, K.M., Gadelha, M.R., de Freitas, R., Cordeiro, L.C.: FuSeBMC_IA: Interval analysis and methods for test-case generation (competition contribution). In: Proc. FASE. LNCS 13991, Springer (2023)
2. Aldughaim, M., Alshmrany, K.M., Mustafa, M., Cordeiro, L.C., Stancu, A.: Bounded model checking of software using interval methods via contractors. arXiv/CoRR **2012**(11245) (December 2020). https://doi.org/10.48550/arXiv.2012.11245
3. Alshmrany, K., Aldughaim, M., Cordeiro, L., Bhayat, A.: FuSeBMC v.4: Smart seed generation for hybrid fuzzing (competition contribution). In: Proc. FASE. pp. 336–340. LNCS 13241, Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_19
4. 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
5. 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
6. 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
7. 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

8. 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
9. 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
10. 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
11. Beyer, D.: Advances in automatic software testing: Test-Comp 2022. In: Proc. FASE. pp. 321–335. LNCS 13241, Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_18
12. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20
13. Beyer, D.: Competition on software verification and witness validation: SV-COMP 2023. In: Proc. TACAS (2). LNCS , Springer (2023)
14. Beyer, D.: Results of the 5th Intl. Competition on Software Testing (Test-Comp 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7701122
15. Beyer, D.: SV-Benchmarks: Benchmark set for softwware verification and testing (SV-COMP 2023 and Test-Comp 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7627783
16. Beyer, D.: Test-suite generators and validator of the 5th Intl. Competition on Software Testing (Test-Comp 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7701118
17. Beyer, D.: Test suites from test-generation tools (Test-Comp 2023). Zenodo (2023). https://doi.org/10.5281/zenodo.7701126
18. 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
19. 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
20. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31
21. Beyer, D., Kanav, S., Wachowitz, H.: Coveriteam Release 1.0. Zenodo (2023). https://doi.org/10.5281/zenodo.7635975
22. 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
23. 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
24. 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
25. 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

26. 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
27. 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)
28. 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
29. 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
30. 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
31. Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)
32. 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
33. 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
34. 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
35. 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
36. 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
37. 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
38. 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
39. 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
40. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252
41. 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
42. 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

43. 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
44. Marques, F., Santos, J.F., Santos, N., Adão, P.: Concolic execution for webassembly (artifact). Dagstuhl Artifacts Series **8**(2), 20:1–20:3 (2022). https://doi.org/10.4230/DARTS.8.2.20
45. Metta, R., Medicherla, R.K., Karmarkar, H.: VeriFuzz: Fuzz centric test generation tool (competition contribution). In: Proc. FASE. pp. 341–346. LNCS 13241, Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_20
46. 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
47. 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
48. 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
49. 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
50. Sutcliffe, G.: The CADE ATP system competition: CASC. AI Magazine **37**(2), 99–101 (2016)
51. 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
52. Wendler, P., Beyer, D.: sosy-lab/benchexec: Release 3.16. Zenodo (2023). https://doi.org/10.5281/zenodo.7612021