# Cooperative Test-Case Generation with Verifiers

Dirk Beyer[1] Marie-Christine Jakobs[2]

**Abstract:** Software testing is widely applied in software quality assurance. Often, test suites fulfilling a certain coverage measure must be constructed. Manually constructing them is laborious. However, numerous automatic test-generation approaches exist. Due to various strengths and weaknesses of individual approaches, hybrid approaches, which combine different approaches, construct test suites that achieve higher coverage values than test suites generated by individual approaches.

We propose the hybrid test-generation approach CoVeriTest. CoVeriTest is flexible, cooperative, and based on verification technology. It iteratively executes a sequence of verifiers that may exchange analysis information between each other and output a test case whenever they reach a test goal. The verifiers, their individual time limits, and which analysis information is exchanged between them is configurable. We experimented with different CoVeriTest configurations. The best configuration participated in the 1st International Competition on Software Testing (Test-Comp'19) and won the third place. This proves the value of our CoVeriTest approach.

**Keywords:** Test-case generation; Software testing; Test coverage; Conditional model checking; Cooperative verification; Model checking

## 1 Overview

Often, testing is an integral part of the software-development process, in which it is used to evaluate the software quality. Thereby, coverage criteria are applied to assess the adequacy of a generated test suite. Manually constructing adequate test suites is typically too laborious. Hence, automatic approaches are used. However, different strengths and weaknesses of existing approaches make it necessary to combine different approaches to achieve good coverage values. Inspired by abstraction-driven concolic testing [DGH16] and recent advances of software verifiers, we therefore propose the flexible, cooperative hybrid test-generation approach CoVeriTest [BJ19].

CoVeriTest is based on verifiers, which construct a test case whenever they reach a test goal (e. g., a branch). Figure 1 shows the workflow of CoVeriTest. CoVeriTest iteratively executes a sequence of $n$ verifiers. In each iteration, verifier $i$ is run for $t_i$ time units and adds its generated test cases to the global test suite. Before a verifier is run, the `init` procedure sets up its context, e. g., the open test goals, the program, and the information from other verifiers. At the end of a verification run, the verifier provides all analysis information in form of the

[1] LMU Munich, Institute of Informatics, Oettingenstraße 67, 80538 Munich, Germany
[2] TU Darmstadt, Department of Computer Science, Hochschulstraße 10, 64289 Darmstadt, Germany
This is a summary of an article that is published in Proc. FASE'19 [BJ19].

doi:10.18420/SE2020_31

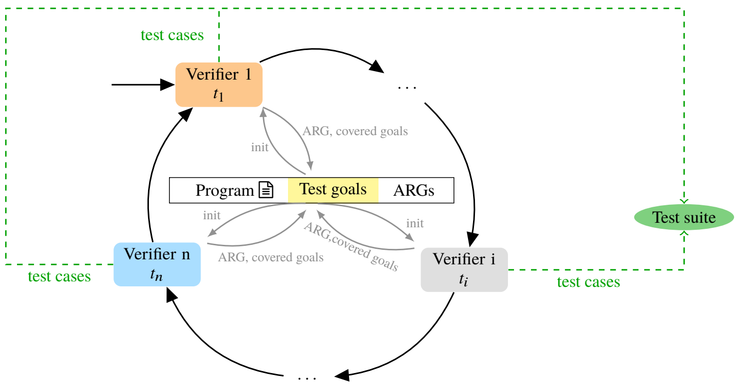

Fig. 1: Workflow of the CoVeriTest approach

reached state space (ARG). Additionally, it reports the goals it covered and that are no longer considered. The number $n$ of verifiers as well as the verifiers $i$ and their time limits $t_i$ can be configured. Also, the cooperation, i.e., the information exchange between verification runs, can be configured. Next to no cooperation, CoVeriTest supports (1) cooperation between different runs of the same verifier, e. g., reuse the abstraction level or continue the exploration of the state space (ARG), (2) cooperation between different verifiers, e. g., ignore already explored paths, and (3) combinations of the previous cooperation types.

For our experiments, CoVeriTest combines value and predicate analysis[3], uses either time unit pair (10 s, 10 s), (50 s, 50 s), (100 s, 100 s), (250 s, 250 s), (80 s, 20 s), or (20 s, 80 s) and one of the different cooperation settings mentioned above. Our evaluation with the programs from the software verification benchmark[4] revealed that the instance using value and predicate analysis with time units (20 s, 80 s) and that lets the analyses continue their exploration works best. Also, this CoVeriTest configuration often achieves higher coverage values than (1) either value or predicate analysis alone, (2) their parallel combination, and (3) their sequential combination (20 % value analysis followed by 80 % predicate analysis). Furthermore, CoVeriTest's 3rd place in Test-Comp'19 [Be19] confirms its value.

## Bibliography

[Be19] Beyer, D.: International Competition on Software Testing (Test-Comp). In: Proc. TACAS. LNCS 11429. Springer, pp. 167–175, 2019.

[BJ19] Beyer, D.; Jakobs, M.-C.: CoVeriTest: Cooperative Verifier-Based Testing. In: Proc. FASE. LNCS 11424. Springer, pp. 389–408, 2019.

[DGH16] Daca, P.; Gupta, A.; Henzinger, T. A.: Abstraction-Driven Concolic Testing. In: Proc. VMCAI. LNCS 9583. Springer, pp. 328–347, 2016.

[3] Meanwhile we also experimented with combinations of bounded model checking and symbolic execution.
[4] https://github.com/sosy-lab/sv-benchmarks