Component-based CEGAR - Building Software Verifiers from Off-the-Shelf Components

Dirk Beyer[1] Jan Haltermann[2] Thomas Lemberger[3] Heike Wehrheim[4]

Abstract: Software verification tools typically consist of tighly coupled components, thereby precluding the easy integration of off-the-shelf components. We propose to decompose software verification into independent subtasks, each task being implemented by an own component communicating with other components via clearly defined interfaces. We apply this idea of decomposition to one of the most frequently used techniques in software verification: CEGAR. Our decomposition, called component-based CEGAR (C-CEGAR), comprises three components: An abstract model explorer, a feasibility checker and a precision refiner. It allows employing conceptually different components for each task within one instance. Our evaluation shows that C-CEGAR has, compared to a monolithic CEGAR-implementation, a similar efficiency and that the precision in solving verification tasks even increases.

Keywords: Software engineering, Software verification, Abstraction refinement, CEGAR, Decomposition, Cooperative verification

Component-based CEGAR

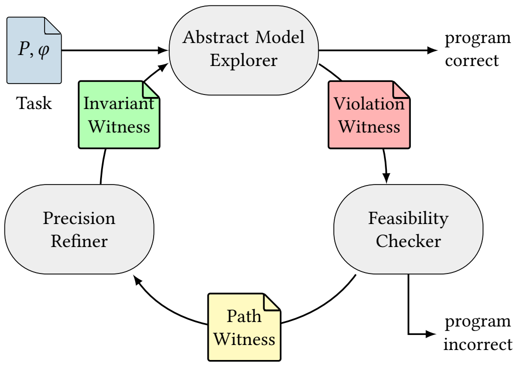

Fig. 1: Workflow of component-based CEGAR

The past years have seen enormous progress in software verification. Although there is an interest in standardizing verification artifacts (e.g., using witnesses), the verification task itself – though consisting of several subtasks – is predominantly realized using strongly cohesive software units with stateful components. This makes reusing components complicated, impacts scalability (e.g., parallelization) and hampers exchange and integration of new (off-the-shelf) components.

[1] LMU Munich, Munich, Germany, dirk.beyer@sosy-lab.org
[2] University of Oldenburg, Oldenburg, Germany, jan.haltermann@uol.de
[3] LMU Munich, Munich, Germany, thomas.lemberger@sosy.ifi.lmu.de
[4] University of Oldenburg, Oldenburg, Germany, heike.wehrheim@uol.de

We propose to *decompose* the construction of verifiers into independent components and employ *cooperative verification* to have the components solve verification tasks together. We demonstrate the feasibility of this idea by applying the decomposition to the *counterexample-guided abstraction refinement* (CEGAR) scheme. CEGAR's main concept during verification is to iteratively refine an abstract model of the system using infeasible counterexamples.

The result of the decomposition, called C-CEGAR, is depicted in Fig. 1. It comprises three components: *Abstract model explorer*, *feasibility checker* and *precision refiner*. The interfaces for the communication between them – *violation*, *path* and *invariant witnesses* – are defined using the existing and standardized witness format. The abstract model explorer builds an abstract model of the program and therein searches for property violations, given the task (program P and property φ) and an invariant witness (encoding the so-far discovered abstraction predicates). If no violation is found, the program is proven correct; otherwise, it constructs an error path leading to the violation, encoded as violation witness. The feasibility checker checks the counter-example encoded in the violation witness for feasibility. If it is feasible, the program violates the property. Otherwise, a path witness encoding the infeasible path is built and given to the precision refiner which computes additional abstraction predicates. These ensure that the infeasible counter-example is not rediscovered. To be able to employ off-the-shelf tools within C-CEGAR, we provide constructions for using a verifier as abstract model explorer or feasibility checker and an invariant generator as precision refiner [Be22].

We realized C-CEGAR with CoVeriTeam. First, we decomposed CPAchecker's CEGAR-based predicate abstraction and than employed it with FShell-witness2test and Ultimate Automizer as additional standalone-components within C-CEGAR. The evaluation on 8 347 verification tasks has shown that (1) the decomposition of an existing CEGAR implementation has (almost) no negative effects on the effectiveness, on the contrary increasing the precision through the use of more sophisticated components, and that the efficiency only decreases by a constant factor; (2) the cost of in addition using standardized formats for exchanging information leads to a reduction of the effectiveness by around 20% and (3) using different components within one C-CEGAR instance is simple and pays off.
In short: Building software verifiers from off-the-shelf components is doable and worth it.

Data Availability
Our implementation of C-CEGAR is open source and available online as part of CoVeriTeam; Our artifact (evaluated as available and reusable) containing the implementation and all experimental data is archived at Zenodo (`https://doi.org/10.5281/zenodo.5301636`).

Bibliography

[Be22] Beyer, D.; Haltermann, J.; Lemberger, T.; Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. ACM, pp. 536–548, 2022.