# Interpolation for Value Analysis *

Dirk Beyer and Stefan Löwe
University of Passau, Innstr. 33, 94032 Passau, Germany

**Abstract:** Abstraction, counterexample-guided refinement, and interpolation are techniques that are essential to the success of predicate-based program analysis. These techniques have not yet been applied together to value analysis. We present an approach that integrates abstraction and interpolation-based refinement into a value analysis, i.e., a program analysis that tracks values for a specified set of variables (the precision). The algorithm uses an abstract reachability graph as central data structure and a path-sensitive dynamic approach for precision adjustment. We evaluated our algorithm on the benchmark set of the Competition on Software Verification 2012 (SV-COMP'12) to show that our new approach is highly competitive. We also showed that combining our new approach with an auxiliary predicate analysis scores significantly higher than the SV-COMP'12 winner.

## Overview

Abstraction is one of the most successful techniques for enabling the verification of industrial-scale programs, because the abstract model omits details about the concrete semantics of a program that are unnecessary to prove the program's correctness.

Counterexample-guided abstraction refinement (CEGAR) [CGJ+03] is a successful technique that iteratively refines such an abstract model using counterexamples. The CEGAR loop starts with a coarse abstraction and tries to find a path to an error in the abstract model. If the analysis does not find an error path, the analysis reports that no violation exists. If the analysis finds an error path, this path is checked for feasibility, and if indeed feasible, the analysis reports this property violation as program bug. If this error path is infeasible, then the violation is due to a too coarse abstract model and the infeasible error path can be used to automatically derive information to refine the abstract model, such that the next iteration of the CEGAR loop continues the state-space exploration using a refined abstract model.

Interpolation is one such method to derive information from infeasible error paths. More precisely, Craig interpolation [Cra57], stemming from the field of logics, yields for two contradicting formulas an interpolant, also being a formula, that is weaker than the first formula, but is still strong enough to contradict the second formula.

The techniques of abstraction, CEGAR, and interpolation have been applied to software verification based on predicate abstraction [GS97] with great success [HJMM04], however, for the value domain [BHT08], these techniques were not yet used in combination.

Our new approach closed this gap: we integrate these three techniques into value analysis. Therefore, the notion of abstraction and the level of abstraction (i.e., the precision) are defined for the value domain. Furthermore, in order to automatically derive a parsimonious precision, we define interpolation for a new type of objects, namely constraint sequences.

*This is a summary of a full article on this topic that appeared in Proc. FASE 2013 [BL13].

This new interpolation technique is able to extract from infeasible error paths extremely small sets of program variables (precisions) that need to be tracked. Together with the CEGAR iterations, we obtain an algorithm that efficiently constructs an abstract model (using the parsimonious precision from interpolation) that omits enough detail to be able to efficiently verify large programs while containing the necessary detail to verify the property.

We implemented the algorithms for value analysis based on CEGAR and interpolation in the open-source verification framework CPACHECKER [1] [BK11] and evaluated our approach on the benchmark set of SV-COMP'12 [2] [Bey12]. We showed [BL13] that this approach is both more efficient and more effective when compared to the same analysis without CEGAR and interpolation, and is on a par with the world's leading symbolic model checkers, like BLAST and SATABS, which are based on SMT-based predicate analysis. When combined with an auxiliary predicate analysis [BKW10], our novel approach is able to score higher than the winner of SV-COMP'12. For the experimental data and download of the tool implementation, we refer to our supplementary web page [3] and to the full article [BL13].

**Summary.** The novel approach dramatically reduces the number of abstract states to be explored; the abstraction is chosen such that only variables that are relevant for the safety property are tracked within the value analysis, while we abstract from unimportant information. This helps to avoid state-space explosion in many cases in which a classic value analysis does not terminate within reasonable time and resource limits.

## References


[Bey12] D. Beyer. Competition on Software Verification (SV-COMP). In *Proc. TACAS*, LNCS 7214, pages 504–524. Springer, 2012.

[BHT08] D. Beyer, T. A. Henzinger, and G. Théoduloz. Program Analysis with Dynamic Precision Adjustment. In *Proc. ASE*, pages 29–38. IEEE, 2008.

[BK11] D. Beyer and M. E. Keremoglu. CPACHECKER: A Tool for Configurable Software Verification. In *Proc. CAV*, LNCS 6806, pages 184–190. Springer, 2011.

[BKW10] D. Beyer, M. E. Keremoglu, and P. Wendler. Predicate Abstraction with Adjustable-Block Encoding. In *Proc. FMCAD*, pages 189–197. FMCAD, 2010.

[BL13] D. Beyer and S. Löwe. Explicit-State Software Model Checking Based on CEGAR and Interpolation. In *Proc. FASE*, LNCS 7793, pages 146–162. Springer, 2013.

[CGJ+03] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[Cra57] W. Craig. Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem. *J. Symb. Log.*, 22(3):250–268, 1957.

[GS97] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. CAV*, LNCS 1254, pages 72–83. Springer, 1997.

[HJMM04] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. POPL*, pages 232–244. ACM, 2004.


[1] `http://cpachecker.sosy-lab.org`
[2] `http://sv-comp.sosy-lab.org/2012/`
[3] `http://www.sosy-lab.org/~dbeyer/cpa-explicit/`