CPA-RefSel: CPAchecker with Refinement Selection
(Competition Contribution)

Stefan Löwe(✉)

University of Passau, Passau, Germany
loewe@fim.uni-passau.de

Abstract. Our submission to SV-COMP'16 is based on the software verification framework CPAchecker. We suggest to combine the value and predicate analysis of the framework, both performing CEGAR based on interpolation. The novelty of our approach is that both analyses perform intra-analysis refinement selection, with a top-level refinement component additionally employing inter-analysis refinement selection. All in all, this allows for an efficient verification process, as intra-analysis refinement selection selects a suitable refinement for an analysis and inter-analysis refinement selection selects the analysis that is best to be refined.

1 Verification Approach

We built our verifier using the software verification framework CPAchecker. As framework, CPAchecker offers a wide range of analyses, and our approach combines the value analysis (VA) and the predicate analysis (PA) in a parallel composition. This compositional approach resides inside an extension of the counterexample-guided abstraction refinement (CEGAR) approach, which in both analyses is driven by interpolation [1]. Our extension of the CEGAR algorithm is depicted in Fig. 1 (taken from [2]), and a brief outline follows.

After a short pre-analysis, which already verifies a few programs successfully, our extended CEGAR algorithm is initiated, which first closely resembles the classic CEGAR approach. The composition of the value and the predicate analysis is started with empty precisions, $\pi^{\mathsf{VA}} = \emptyset, \pi^{\mathsf{PA}} = \emptyset$, i.e., during the initial state-space exploration no assignments (for VA) and no predicates (for PA) are tracked. If the resulting over-approximation of the state space is free of errors, then the CEGAR loop terminates with the verdict *true*, if a real counterexample is found, then the CEGAR loop terminates with the verdict *false*. If an inconclusive error path σ is found, here represented as sequence of pairs of an operation *op* and a program location l, then the standard CEGAR algorithm would compute a *single* refinement, e.g., by inferring interpolants from the single infeasible error path to exclude this infeasible error path in future state-space explorations.

M. Chechik and J.-F. Raskin (Eds.): TACAS 2016, LNCS 9636, pp. 916–919, 2016.
DOI: 10.1007/978-3-662-49674-9_59

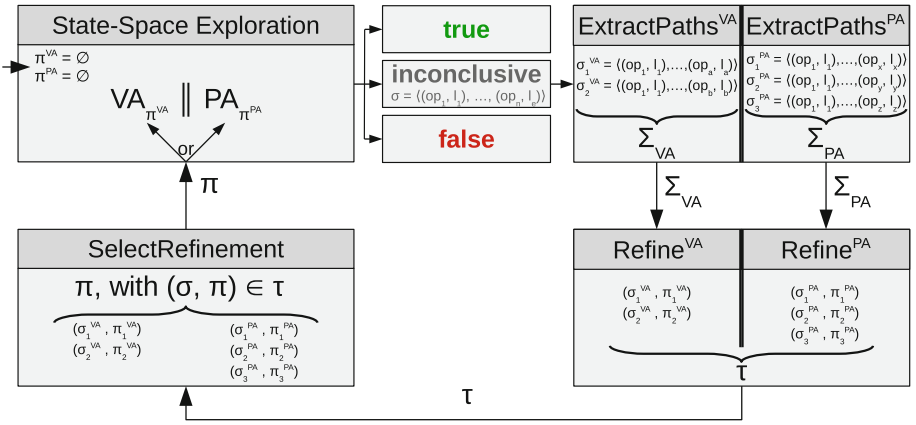

Fig. 1. Refinement selection for combining a value and a predicate analysis [2]

Exactly here we deviate from the standard CEGAR approach, and instead, we perform *intra-analysis refinement selection* by first calling procedure ExtractPaths to extract a set Σ of infeasible sliced paths from the original infeasible error path [3], and then, by calling procedure Refine, to compute a set τ of individual refinements, one for each of the available infeasible sliced paths [3].

Each of the refinements makes the precision of the analysis strong enough to exclude the original infeasible error path [3]. This allows the analysis to heuristically select from a pool of available refinements, and it may pick a refinement that seems like a good fit for the further course of the analysis, while at the same time, it can avoid unsuitable ones, e.g., those that might lead to loop unrollings.

The procedures ExtractPaths and Refine are available for both the value analysis and the predicate analysis, making intra-analysis refinement selection possible for both analyses [2]. Furthermore, we can leverage refinement selection to a higher level — with intra-analysis refinement selection we have multiple refinements to select from, and we have means available to distinguish between unsuited and well-suited refinements. So we can utilize these mechanisms to enable *inter-analysis refinement selection*, i.e., we do not only select the refinement that is best for a component analysis, but we also decide whether the composite analysis should perform its refinement for the value or the predicate analysis [2].

Specifically for the SV-COMP, we made refinement selection applicable together with adjustable-block encoding. Mind that with large blocks, e.g., with abstractions computed only at loops, the number of available refinements to select from tends to be lower compared to when having small blocks, e.g. with single-block encoding. Now, we compute abstraction whenever control flow joins, as the analysis performs best with medium-sized blocks, proving that selecting suitable refinements is as important as an appropriate block-encoding strategy.

2 Software Architecture

The CPAchecker framework is written in JAVA. For parsing C code we employ the C parser from the Eclipse CDT project. CPAchecker offers interfaces to a wide range of decision procedures, and for our submission we rely on MathSAT to solve SMT and interpolation queries issued by the bit-precise predicate analysis.

3 Strengths and Weaknesses

A combination of a value and a predicate analysis demonstrated its potential already in an earlier edition of SV-COMP [4], winning silver in the category `Overall` and in several sub-categories. However, the intent of this year's submission is to showcase the power of refinement selection in the, from our point of view, highly important category `DeviceDriversLinux64`, where refinement selection works particularly well, allowing us to win the *gold* medal. Still, we seek for a better understanding of heuristics for inter- and intra-analysis refinement selection. Despite the fact that the CPAchecker framework supports checking memory safety and overflows, our submission is not competitive there, while also lacking support for concurrency, termination, large arrays and explicit recursion.

4 Setup and Configuration

Our verifier is built from revision 18373 from the official CPAchecker repository, branch `refinementSelectionForABE`. It is also archived at http://sv-comp.sosy-lab.org/2016/systems.php. To run our tool please enter this command:

```
scripts/cpa.sh -sv-comp16--refsel -disable-java-assertions -heap 12500m -spec prop.prp task.i
```

For C programs that assume a 64-bit environment add the parameter `-64`. The tool prints to the console the verdict, the violated property, and the name of the output directory, the latter holding the witness file `witness.graphml` in case a property violation is found. To reproduce the results, use Java 7, the benchmark definition `cpa-refsel.xml` and the tool-info module `cpachecker.py`, both officially archived online at http://sv-comp.sosy-lab.org/2016/systems.php.

5 Project and Contributors

CPAchecker is a verification framework maintained by the Software Systems Lab at the University of Passau, made available under the Apache 2.0 license. It proofed successful in every edition of the SV-COMP, and it is used by practitioners and researchers at the Russian Academy of Science, the Universities of Darmstadt, Hamburg, Paderborn and Vienna, as well as at Verimag in Grenoble. We would like to thank all contributors for their efforts spent on CPAchecker.

References

1. Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 146–162. Springer, Heidelberg (2013)
2. Beyer, D., Löwe, S., Wendler, P.: Refinement selection. In: Fischer, B., Geldenhuys, J. (eds.) SPIN 2015. LNCS, vol. 9232, pp. 20–38. Springer, Heidelberg (2015)
3. Beyer, D., Löwe, S., Wendler, P.: Sliced path prefixes: an effective method to enable refinement selection. In: Graf, S., Viswanathan, M. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. LNCS, vol. 9039, pp. 228–243. Springer, Heidelberg (2015)
4. Löwe, S.: CPAchecker with Explicit-Value Analysis Based on CEGAR and Interpolation. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 610–612. Springer, Heidelberg (2013)