CPAchecker for Reachability, Memory Safety, Overflows, Concurrency, and Termination

(Competition Contribution for SVCOMP'17)

Matthias Dangl and Karlheinz Friedberger

SoSy-Lab Software Systems

LMU Ludwig-Maximilians-Universität München

Overview

CPAchecker is a modern framework for software verification and is based on well-known concepts like

- CEGAR
- Configurable program analysis (CPA) [2]
- Interpolation
- Predicate abstraction [3]
- Explicit-state model checking [4]
- k-induction [1]
- Lazy abstraction
- Abstract interpretation
- Block-abstraction memoization [7]

CPAchecker has support for several abstract domains, such as values, intervals, octagons, BDDs, predicates, and memory graphs, which can all be used to build an analysis that matches the user's requirements.

Setup and Configuration

Download CPAchecker from

`https://cpachecker.sosy-lab.org`

and execute

```
scripts/cpa.sh -sv-comp17
    -disable-java-assertions -heap 10000m
    -spec property.prp program.i
```

The configuration `sv-comp17` is

- optimized for checking a wide range of properties,
- an effective approach for solving a heterogeneous set of verification tasks, and
- based on several verification approaches, from reachability analysis to synthesized ranking functions.

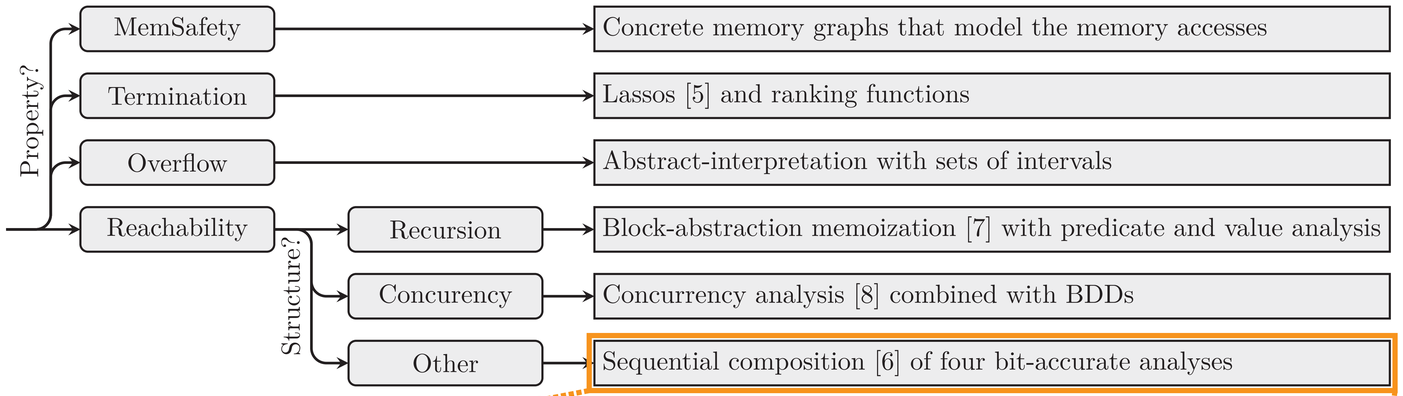
Contributors

CPAchecker is an open-source project, developed by members of Dirk Beyer's Software Systems Lab at LMU Munich, and is used and extended by associates from

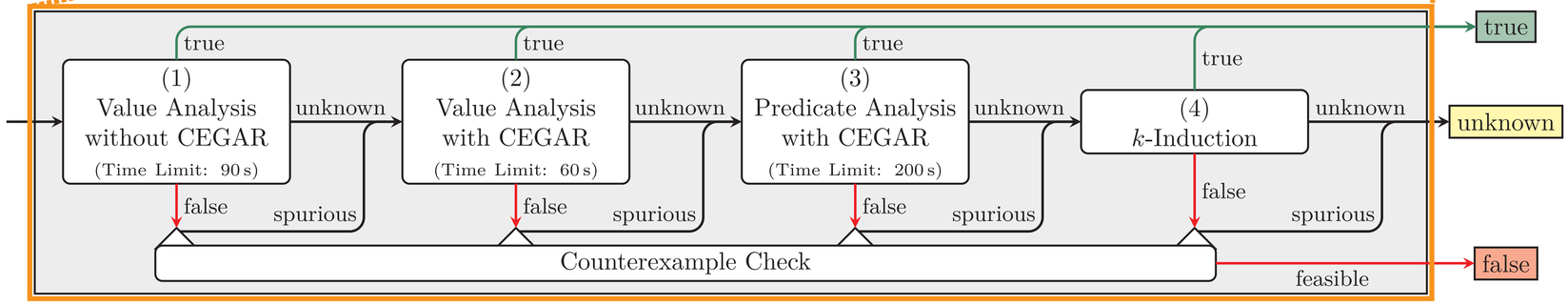
- the Institute for System Programming of the Russian Academy of Sciences,
- Universities of Darmstadt, Paderborn, Passau, and Vienna,
- VERIMAG in Grenoble, and
- several other universities and institutes.

We thank all contributors for their work on CPAchecker.

Verification Strategy for SV-COMP'17



Sequential Composition



Up to four bit-accurate analyses are executed in sequence. After a time limit is exceeded the next analysis is started, if no result is available yet.

(1,2) The value analysis [4] tracks values of integer variables explicitly. It is efficient but imprecise for non-deterministic variables.

(3) The predicate analysis [3] uses interpolation and predicate abstraction.

(4) k-induction [1] uses auxiliary invariants and extends bounded model checking from falsification to verification.

Counterexamples are cross-checked with a bit-accurate counterexample check, i.e., with predicate analysis for (1,2) and CBMC for (3,4).

References

[1] D. Beyer, M. Dangl, and P. Wendler. Boosting k-induction with continuously-refined invariants. In *Proc. CAV*, LNCS 9206, pages 622–640. Springer, 2015.

[2] D. Beyer, T. A. Henzinger, and G. Théoduloz. Configurable software verification: Concretizing the convergence of model checking and program analysis. In *Proc. CAV*, LNCS 4590, pages 504–518. Springer, 2007.

[3] D. Beyer, M. E. Keremoglu, and P. Wendler. Predicate abstraction with adjustable-block encoding. In *Proc. FMCAD*, pages 189–197. FMCAD, 2010.

[4] 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.

[5] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *Proc. CAV*, LNCS 4144, pages 415–418. Springer, 2006.

[6] M. Dangl, S. Löwe, and P. Wendler. CPAchecker with support for recursive programs and floating-point arithmetic (competition contribution). In *Proc. TACAS*, LNCS 9035, pages 423–425. Springer, 2015.

[7] K. Friedberger. CPA-BAM: Block-abstraction memoization with value analysis and predicate analysis (competition contribution). In *Proc. TACAS*, LNCS 9636, pages 912–915. Springer, 2016.

[8] K. Friedberger and D. Beyer. A light-weight approach for verifying multi-threaded programs with CPAchecker. In *Proc. MEMICS*, EPTCS 233, pages 61–71. EPTCS, 2016.