CPAchecker with Sequential Combination of Explicit-State Analysis and Predicate Analysis (Competition Contribution)

Philipp Wendler

University of Passau, Germany

Abstract. CPAchecker is an open-source framework for software verification, based on the concepts of Configurable Program Analysis (CPA). We submit a CPAchecker configuration that uses a sequential combination of two approaches. It starts with an explicit-state analysis, and, if no answer can be found within some time, switches to a predicate analysis with adjustable-block encoding and CEGAR.

1 Verification Approach

CPAchecker [3] is an open software-verification framework that integrates several state-of-the-art approaches for software model checking. None of these approaches is a clear winner over the other in terms of successfully verified programs and performance. Instead, each technique has its own distinct advantages and might solve programs that other analyses perhaps cannot verify. Thus we use a sequential combination of two analyses in order to be able to verify more programs than the two analyses alone. The second analysis is started after the first analysis if the first terminated with no verification result, e.g., because of an exhaustion of the available resources. This is a simple instance of conditional model checking [1] without information passing between the subsequent analysis runs. An overview of the approach can be seen in Fig. 1.

We use as a first analysis a rather simple explicit analysis which tracks values of integer variables. It does not use concepts such as CEGAR or lazy abstraction. This analysis often finds counterexamples quickly, but fails in other cases due to state-space explosion. In particular, we have experienced that it is unlikely to produce a result if it is not successful in short time. Thus we limit this analysis to a runtime of 100 s. If the analysis has neither found a valid counterexample nor proved the program safe after this time, it will terminate gracefully. In this case, we use the predicate analysis from last year's competition submission [5] as the second analysis. It uses lazy predicate abstraction with adjustable-block encoding [4], CEGAR, and Craig interpolation.

In order to prevent false alarms, we dump each counterexample in form of a (loop-free) C program, and run the bit-precise bounded model checker CBMC[1] in version 4.2 on it. If CBMC refutes the reachability of the error in the generated program, we skip the counterexample and continue with the analysis.

[1] `http://www.cprover.org/cbmc`

N. Piterman and S. Smolka (Eds.): TACAS 2013, LNCS 7795, pp. 613–615, 2013.

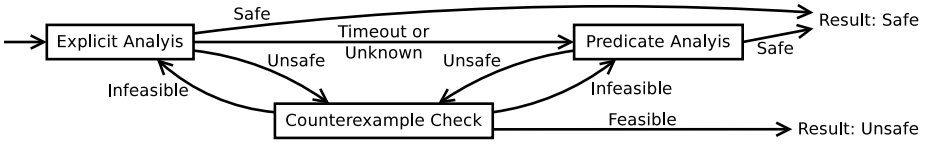

Fig. 1. Verification approach

2 Software Architecture

CPAchecker is written in Java and based on the Configurable Program Analysis (CPA) framework [2]. The explicit analysis, the predicate analysis, and "helper" analyses that are used in this configuration, such as tracking of the program counter, call stack, and function pointers, are implemented as CPAs. CPAs can be enabled as desired without changing other CPAs, and are used by a common algorithm for reachability analysis. Other algorithms, for example for CEGAR, counterexample checks, and analysis combinations, wrap the core algorithm depending on the configuration. The framework also uses the C parser from the Eclipse CDT project[2], and MathSAT4[3] as an SMT solver and interpolation engine.

3 Strengths and Weaknesses

The main advantage of the submitted configuration is the combination of two conceptually different analyses. This allows verifying a wide variety of programs. For example, most programs in the category "ProductLines" can be verified by the rather simple explicit analysis in short time, whereas the predicate analysis fails on many of them. However, in cases where the explicit analysis is not successful, this combination may lead to a decreased performance. For example, the predicate analysis alone needed 1000 s for the category "ControlFlowInteger" in the 2012 competition, whereas now 3400 s are needed (obtaining the same score).

The precise counterexample checks with CBMC make sure that CPAchecker never produces a wrong counterexample. Furthermore, CPAchecker only reports 4 programs erroneously as safe. No other tool in the competition that participated in all categories managed to achieve a non-negative score in all categories.

The implementation of CPAchecker sticks closely to the theoretical concepts and is primarily focused on re-usability and flexibility (witnessed by the existence of extensions contributed by other groups). The use of the CPA framework makes CPAchecker an easily extensible framework for software verification as it allows to re-use and combine analyses implemented as CPAs. However, this means that CPAchecker does not exploit all possible low-level optimizations that are available in a strongly coupled implementation.

[2] `http://www.eclipse.org/cdt/`
[3] `http://mathsat4.disi.unitn.it/`

Multi-threaded programs and the verification of memory-safety properties are currently not explicitly supported by CPAchecker.

4 Setup and Configuration

CPAchecker is available online at `http://cpachecker.sosy-lab.org` under the Apache 2.0 license. It requires Java 6 to run. We submitted CPAchecker in version `1.1.10-svcomp13` using the configuration `-sv-comp13--combinations` to the competition on software verification. The command line for running it is

```
./scripts/cpa.sh -sv-comp13--combinations -heap 12000m
                 -disable-java-assertions path/to/sourcefile.cil.c
```

For C programs that assume a 64-bit environment (i.e., those in the category "Linux Device Drivers 64-bit") the below parameter needs to be added:

```
-setprop cpa.predicate.machineModel=Linux64
```

For the category "Memory Safety", the property to verify is given by `-spec p` with `p` in {`valid-free`, `valid-deref`, `valid-memtrack`}. For machines with less RAM, the amount of memory given to the Java VM needs to be adjusted with the parameter `-heap`. CPAchecker will print the verification result and the name of the output directory to the console. Additional information (such as the error path) will be written to files in this directory.

5 Project and Contributors

CPAchecker is an open-source project lead by Dirk Beyer from the Software Systems Lab at the University of Passau. Several other research groups use and contribute to CPAchecker, e.g., the Russian Academy of Science, the University of Paderborn, and the Technical University of Vienna.

We would like to thank all contributors for their work on CPAchecker since 2007. The full list can be found online at `http://cpachecker.sosy-lab.org`.

References

1. Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012)
2. Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 504–518. Springer, Heidelberg (2007)
3. Beyer, D., Keremoglu, M.E.: CPAchecker: A Tool for Configurable Software Verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011)
4. Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD, pp. 189–197. FMCAD (2010)
5. Löwe, S., Wendler, P.: CPAchecker with Adjustable Predicate Analysis (Competition Contribution). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 528–530. Springer, Heidelberg (2012)