Counterexample-Guided Abstraction Refinement*

Edmund Clarke
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
edmund.clarke@cs.cmu.edu

Abstract

The main practical problem in model checking is the combinatorial explosion of system states commonly known as the state explosion problem. Abstraction methods attempt to reduce the size of the state space by employing knowledge about the system and the specification in order to model only relevant features in the Kripke structure. Counterexample-guided abstraction refinement is an automatic abstraction method where, starting with a relatively small skeletal representation of the system to be verified, increasingly precise abstract representations of the system are computed. The key step is to extract information from false negatives ("spurious counterexamples") due to over-approximation.

The methods for alleviating the state explosion problem in model checking can be classified coarsely into *symbolic methods* and *abstraction methods* [6]. By symbolic methods we understand the use of succinct data structures and symbolic algorithms which help keep state explosion under control by compressing information, using, e.g., binary decision diagrams or efficient SAT procedures.

Abstraction methods in contrast attempt to reduce the size of the state space by employing knowledge about the system and the specification in order to model only relevant features in the Kripke structure. An abstraction function h associates a Kripke structure K with an abstract Kripke structure $\widehat{K}$ such that two properties hold:

- **Feasibility.** $\widehat{K}$ is significantly smaller than K.
- **Preservation.** $\widehat{K}$ preserves all behaviors of K.

Preservation ensures that every universal specification which is true in $\widehat{K}$ is also true in K. The converse implication, however, will not hold in general: a universal property which is false in $\widehat{K}$ may still be true in K. In this case, the counterexample obtained over $\widehat{K}$ cannot be reconstructed for the concrete Kripke structure K, and is called a *spurious counterexample* [10], or a false negative.

An important example of abstraction is *existential abstraction* [11] where the abstract states are essentially taken to be equivalence classes of concrete states; a transition between two abstract states holds if there was a transition between any two concrete member states in the corresponding equivalence classes.

In certain cases, the user knowledge about the system will be sufficient to allow manual determination of a good abstraction function. In general, however, finding abstraction functions gives rise to the following dichotomy:

- If $\widehat{K}$ is too small, then spurious counterexamples are likely to occur.
- If $\widehat{K}$ is too large, then verification remains infeasible.

Counterexample-Guided Abstraction Refinement (CEGAR) is a natural approach to resolve this situation by using an adaptive algorithm which gradually improves an abstraction function by analysing spurious counterexamples.

(i) **Initialization.** Generate an initial abstraction function.

(ii) **Model Checking.** Verify the abstract model. If verification is successful, the specification is correct, and the algorithm terminates successfully. Otherwise, generate a counterexample $\widehat{T}$ on the abstract model.

(iii) **Sanity Check.** Determine, if the abstract counterexample $\widehat{T}$ is spurious. If a concrete counterexample

*This research was sponsored by the Semiconductor Research Corporation (SRC) under contract no. 99-TJ-684, the National Science Foundation (NSF) under grant no. CCR-9803774, the Office of Naval Research (ONR), the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796, and by the Defense Advanced Research Projects Agency, and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485, and the General Motors Collaborative Research Lab at CMU. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of SRC, NSF, ONR, NRL, DOD, ARO, or the U.S. government.

Proceedings of the 10th International Symposium on Temporal Representation and Reasoning and
Fourth International Conference on Temporal Logic (TIME-ICTL'03) 1530-1311/03 $17.00 © 2003 IEEE

T can be generated, the algorithm outputs this counterexample and terminates.

(iv) **Refinement.** Refine the abstraction function in such a way that the spurious counterexample $\widehat{T}$ is avoided, and return to step (ii).

Using counterexamples to refine abstract models has been investigated by several researchers beginning with the *localization reduction* of Kurshan [19] where the model is abstracted/refined by removing/adding variables from the system description. A similar approach has been described by Balarin and Sangiovanni-Vincentelli in [1].

A systematic account of counterexample guided abstraction refinement for CTL model checking was given in [10, 8]. Here, the initial abstraction is obtained using predicate abstraction [17] in combination with a simple static analysis of the system description; all other steps use BDD-based techniques. The use of tree-like counterexamples guarantees that the method is complete for ACTL.

During the last few years, the CEGAR paradigm has been adapted to different projects and verification frameworks, both for hardware and software verification [20, 16, 14, 13, 3, 4, 2, 9, 7, 18, 5]. The major improvements to the method include, most notably, the integration of SAT solvers for both verification and refinement, and the use of multiple spurious counterexamples.

It is well-known that most abstraction methodologies can be paraphrased in the framework of abstract interpretation by Cousot and Cousot [12]. Giacobazzi and Quintarelli [15] have shown that, not surprisingly, this holds true for counterexample-guided abstraction refinement as well. The practical and computational significance of such embeddings for verifying real-life systems however remains controversial.

References

[1] F. Balarin and A. L. Sangiovanni-Vincentelli. An iterative approach to language containment. In *Computer-Aided Verification*, 1993.

[2] T. Ball and S. K. Rajamani. Getting abstract explanations of spurious counterexamples in C programs, 2002. Microsoft Technical Report MSR-TR-2002-09.

[3] S. Barner, D. Geist, , and A. Gringauze. Symbolic localization reducation with reconstruction layering and backtracking. In *CAV 2002*, volume 2404 of *LNCS*, pages 65–77, 2002.

[4] S. Chaki, J. Ouaknine, K. Yorav, and E. M. Clarke. Multi-level abstraction refinement for concurrent C programs. 2002. Submitted for Publication.

[5] E. Clarke, S. Chaki, S. Jha, and H. Veith. Strategy guided abstraction refinement, 2003. Manuscript.

[6] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Progress on the state explosion problem in model checking. In *Informatics, 10 Years Back, 10 Years Ahead*, volume 2000 of *LNCS*, pages 176–194, 2001.

[7] E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction - refinement using ILP and machine learning techniques. volume 2404 of *LNCS*, pages 265–279, Copenhagen, Denmark, July 2002.

[8] E. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Proc. Logic in Computer Science (LICS)*, 2002.

[9] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *TACAS'03*, pages 192–207, 2003.

[10] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, 2000.

[11] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[12] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. *ACM Symposium of Programming Language*, pages 238–252, 1977.

[13] S. Das and D. Dill. Successive approximation of abstract transition relations. In *LICS*, pages 51–60, 2001.

[14] S. Das and D. Dill. Counter-example based predicate discovery in predicate abstraction. In *Formal Methods in Computer-Aided Design*, pages 19–32, 2002.

[15] R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model checking. In *SAS'01*, pages 356–373, 2001.

[16] M. Glusman, G. Kamhi, S. Mador-Haim, R. Fraer, and M. Vardi. Multiple-counterexample guided iterative abstraction refinement: An industrial evaluation. In *TACAS'03*, pages 176–191, 2003.

[17] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Computer-Aided Verification*, June 1997.

[18] T. A. Henzinger, R. Jhala, and R. Majumdar. Counterexample guided control. In *ICALP*, 2003. To appear.

[19] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, 1994.

[20] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. Incremental verification by abstraction. pages 98–112, 2001.