

Partial Order Reduction for Abstraction-Based Verification of Concurrent Software

Csanád Telbisz *, Vince Molnár 

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
csanadtelbisz@edu.bme.hu, molnarv@mit.bme.hu

Abstract—Concurrency introduces a new level of complexity into software verification due to the great number of possible thread interleavings. Partial order reduction is an effective approach to handle concurrency in model checking. In this work, we present ways to integrate a dynamic partial order reduction algorithm into an abstraction-based verification process. The focus of my ongoing research is to exploit information about the applied abstraction when building a dependency relation on operations of a program. If the source of dependency between certain operations is abstracted away, they need not be considered dependent. By decreasing the dependency in the model, we hope to increase the reducing effect of partial order reduction.

I. INTRODUCTION

Software verification suffers from the state space explosion problem since the size of the state space of a program often grows exponentially with the number of variables and the size of their domains [1]. Counterexample-Guided Abstraction Refinement (CEGAR) overcomes this issue by abstracting away information [2] such as not tracking the values of certain variables. If the abstraction is too coarse, it is refined until the algorithm can determine whether the safety requirement holds.

Multi-threading significantly increases the complexity of software verification. Concurrent programs have numerous executions, thus all possible thread interleavings must be checked in a verification process. The aim of *partial order reduction* (POR) is to efficiently handle this complexity by reducing the number of interleavings in executions while preserving the correctness of the model checking [3]. The basic idea of POR is that the order of consecutive independent transitions in an execution can be irrelevant from the perspective of model checking. Thus, we do not have to check every single execution. Having a dependency relation on the transitions allows us to use POR to reduce the state space.

While POR is an effective technique for handling concurrency, CEGAR is an efficient abstraction-based approach to handle data. The goal of this work is to develop a highly performant verification algorithm by combining these two model checking paradigms.

Successful tools (e.g. CPAchecker [4]) use *static* POR with CEGAR. In this work, we present a *dynamic* partial order reduction (DPOR) algorithm [5] integrated into a CEGAR-based verification process.

*This work was supported by IncQuery Group (incquery-group.com).

II. PRELIMINARIES

A. State Space of a Multi-Threaded Program

A common formalism for representing computer programs is the *Control Flow Automaton* (CFA) [6]. For multi-threaded programs, an extended CFA is used consisting of multiple processes, each process represented by a CFA. Global and thread-local variables are distinguished. The processes fire asynchronously (one process takes a transition at a time).

A program has a state space which can be easily formalized with the CFA. A state of a single-threaded program represents the control location and the values of the variables at a certain point during the operation of the program. On the other hand, a state of a multi-threaded program is the combined states of all of its processes. A transition in the state space is an operation of a process that the program executes.

B. Counterexample-Guided Abstraction Refinement (CEGAR)

CEGAR is an efficient abstraction-based technique for checking reachability [2]. It works on an abstract state space where abstract states may represent a set of concrete states. CEGAR iteratively refines an abstract model based on spurious abstract requirement violations until it finds a concretizable violation, or no abstract violation is found.

C. Concept of Partial Order Reduction

From a practical aspect, the main goal of partial order reduction is to find a (small) subset of outgoing transitions in a state and explore the state space further only through these selected transitions [3]. If these subsets are prudently chosen, the correctness of the model checking process (with regard to reachability criteria) can be preserved.

There are two main approaches to POR: *static* and *dynamic* POR [5]. In the former case, the model is analyzed and the reduced state space is generated prior to the model checking process. With the dynamic approach, the reduced state space is constructed during the model checking.

III. DYNAMIC PARTIAL ORDER REDUCTION IN CEGAR

A. Combining DPOR and CEGAR

In CEGAR, the reachability requirement is checked in the abstract state space, so POR is also applied in the abstract state space. Thus, a dependency relation is needed for the abstract transitions.

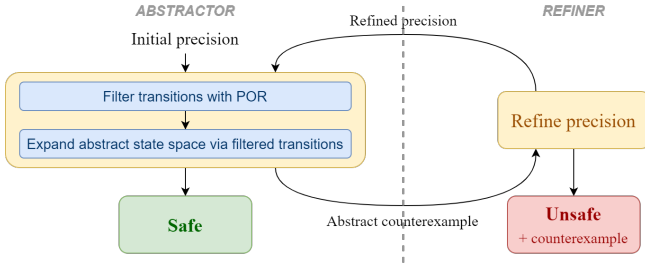


Fig. 1. CEGAR-loop combined with POR

Generally, two transitions t_1 and t_2 are considered independent if they can neither enable nor disable each other, and their execution is commutative from any state where they are both enabled [7]. A more practical sufficient condition for independence is the following: the process of t_1 is not the same as the process of t_2 and the set of objects (variables) accessed by t_1 is disjoint from the objects accessed by t_2 .

Transitions of the abstract state space are derived from operations of the program. Thus, a basic dependency relation can be defined with the program operations: two transitions in the state space are independent if their operations are not in the same process (thread), and they access only different global variables. This dependency model can be enhanced by distinguishing read and write operations: two read operations on the same variable may be independent.

This way, no extra information is used about the current abstraction when determining dependency. Considering the current abstraction in the CEGAR verification process can decrease the dependency in the transition system (that is the dependency relation becomes less dense) which increases the efficiency of POR. A possible abstraction in CEGAR is that the values of certain variables are not tracked: if we have two operations (from different processes) that use only untracked global variables, it is unnecessary to consider them dependent. Another type of abstraction is predicate abstraction where only certain predicates are tracked: operations on variables which are currently not present in any predicate may be considered independent.

B. Core of the DPOR algorithm

The core of the applied DPOR algorithm uses the notion of *persistent sets* from [7]. Persistent sets can be calculated in each state based on the dependency relation between transitions. A *persistent set selective search* is a search of the state space where we only proceed further from a state via transitions that are elements of the persistent set of that state. It is proven in [7] that such a persistent set selective search (with a minor modification) can be used to check reachability criteria without losing the correctness of the model checking.

In the context of a CEGAR-based verification process, using a persistent set selective search means filtering the outgoing transitions of abstract states: the abstract state space is explored further via a transition t by the abstractor, only if t is in the calculated persistent set of its source state. Figure 1

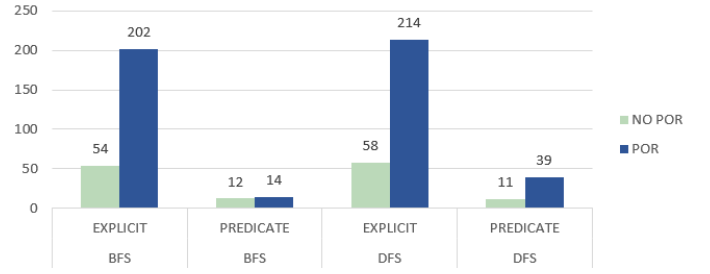


Fig. 2. Solved problems (from a total of 763 tasks) by configuration

illustrates the CEGAR-loop extended with POR in the state space expansion phase.

IV. PRELIMINARY RESULTS

The naive version of DPOR (not considering extra abstraction-based information) has been implemented with CEGAR in the open-source model checking framework THETA [8]. Benchmark tests performed on C programs provided by SoSy-Lab¹ with different configurations of THETA concerning the search strategy and the abstraction domain show promising results about combining POR and CEGAR.

Figure 2 shows the number of solved problems per configuration with the blue columns standing for the tasks solved with POR: it is considerably better than without applying POR. Based on the number of solved tasks, THETA can now be considered a competitive tool in the concurrency safety category of the Competition on Software Verification (according to results at SV-COMP 2022 [9]).

My ongoing work is focused around developing an abstraction-aware partial order reduction algorithm described in this paper: using information about the current abstraction allows creating a less dense dependency relation which hopefully improves the efficiency of the verification even further.

REFERENCES

- [1] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, “Model checking and the state explosion problem,” in *LASER Summer School on Software Engineering*. Springer, 2011, pp. 1–30.
- [2] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement for symbolic model checking,” *Journal of the ACM (JACM)*, vol. 50, no. 5, pp. 752–794, 2003.
- [3] D. Peled, “Ten years of partial order reduction,” in *International Conference on Computer Aided Verification*. Springer, 1998, pp. 17–28.
- [4] D. Beyer and K. Friedberger, “A light-weight approach for verifying multi-threaded programs with CPAchecker,” *arXiv preprint arXiv:1612.04983*, 2016.
- [5] C. Baier and J. Katoen, *Principles of Model Checking*. MIT Press, 2008.
- [6] D. Beyer and M. E. Keremoglu, “CPAchecker: A tool for configurable software verification,” in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 184–190.
- [7] P. Godefroid, *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*. Springer, 1996.
- [8] T. Tóth, Á. Hajdu, A. Vörös, Z. Micskei, and I. Majzik, “Theta: a framework for abstraction refinement-based model checking,” in *2017 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2017, pp. 176–179.
- [9] D. Beyer, “Progress on software verification: SV-COMP 2022,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022, pp. 375–402.

¹gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c