Partial Order Reduction for Trace Abstraction Refinement

Dominik Klumpp

klumpp@informatik.uni-freiburg.de, University of Freiburg

It is well-known that concurrent programs suffer from the state explosion problem: Due to the non-deterministic scheduling of different threads, the number of reachable program states grows exponentially in the number of threads. Accordingly, the runtime of static analyses or verification algorithms also grows exponentially in the number of threads. In this research, we focus on improving the efficiency of *Trace Abstraction Refinement* (TAR) [3], a Software Model Checking technique, when applied to concurrent programs. We achieve this by integrating it with *Partial Order Reduction* (POR) [2], a technique from the field of (finite-state) model checking that combats the state explosion problem. This work is done in collaboration with Azadeh Farzan and Andreas Podelski.

Trace Abstraction Refinement (TAR) With TAR, we verify programs with infinite state spaces, in a completely automated manner. The input program is given in the form of a finite control flow automaton P, whose alphabet is composed of program statements, and whose accepting states are *error states*, that can only be reached if the verified (safety) property is violated. The program is proven correct by showing that all traces τ (words over the alphabet of statements) accepted by P, called *error traces*, are in fact *infeasible*: The Hoare triple $\{\top\}\,\tau\,\{\bot\}$ is valid, i.e., there exists no program execution corresponding to τ. A proof candidate A for this is iteratively constructed as a finite automaton, in particular a *Floyd-Hoare automaton*: The automaton states are given by assertions over the program variables, the alphabet is the same as for P, and each transition, going from an assertion φ to an assertion ψ and labeled with a statement st, forms a valid Hoare triple $\{\varphi\}\,st\,\{\psi\}$. As the initial state is given by the assertion $\top$ and the accepting state is $\bot$, any trace accepted by A is infeasible. In each round of the verification algorithm, a new Floyd-Hoare automaton is constructed and unioned with the previous A. The termination condition of the refinement loop and the proof criterion used to establish correctness is the inclusion $P \subseteq A$: Each error trace in P is proven infeasible by membership in A.

In TAR, the state explosion problem occurs in two forms: Firstly, the size of the control flow automaton P grows exponentially in the number of threads, and hence so does the runtime of the inclusion check $P \subseteq A$. Secondly, a Floyd-Hoare automaton for the program might require an exponential number of assertions and Hoare triples, the construction of which requires an exponential number of TAR iterations. Often, these different assertions are only needed for different intermediate states, while the general proof idea for different interleavings is the same. To tackle these issues, we integrate TAR with *Partial Order Reduction*.

Partial Order Reduction (POR) POR is based on a so-called *(partial) commutativity relation* between program statements. The idea is that a pair of

Thread 1	Thread 2
`a[i]:=0`	`a[j]:=1`
`assert a[i]==0`	

(a) no commutativity

Thread 1	Thread 2
	`j:=i+1`
`a[i]:=0`	`a[j]:=1`
`assert a[i]==0`	

(b) conditional comm.

Thread 1	Thread 2
`a[i]:=0`	`a[j]:=1`
`assert a[i]>=0`	

(c) abstract comm.

Fig. 1: Example programs manipulating an integer array `a` at positions `i` and `j`.

statements a and b *commute*, if the sequential compositions ab and ba exhibit the same behaviour (in some sense). Within some trace τ containing the subsequence ab, it is then allowed to commute the two, yielding some trace τ' containing the subsequence ba. This defines an equivalence relation between traces, e.g., $\tau \sim \tau'$. We then define the *closure* of a language L of traces as the union of all equivalence classes represented in L, i.e., $cl(L) = \bigcup_{\tau \in L} [\tau]$. In the opposite direction, there exist POR algorithms that compute a *reduction* of a given input system, i.e., an L' such that $cl(L') = L$. While L' has fewer traces and often fewer states, it still contains one representative of each class of equivalent traces. Hence analysis results on L' can under some conditions be transferred to L.

Integrating TAR with POR Cassez and Ziegler [1] have shown that it is possible to apply POR within the TAR setting. They compute a reduction P' of the input program P, before then verifying P' using the classical TAR algorithm. In this research, we aim to go beyond this and integrate POR directly in the TAR workflow. With the additional information thus available – in particular, the current proof candidate A –, we achieve increased reduction power, and thus further improve verification efficiency. To this end, we define a new proof criterion for TAR with integrated POR: Instead of demanding that $P \subseteq A$, we weaken it to $P \subseteq cl(A)$: Each error trace in P is equivalent to a trace that is proven infeasible by membership in A. This is equivalent to the existence of a reduction P' of P such that $P' \subseteq A$. While this criterion is known to be undecidable in general – $cl(A)$ is not necessarily regular, and there exist infinitely many reductions of P –, POR algorithms can be used to implement a sound test for this criterion: *Some* reduction $P' \subseteq P$ is computed and checked for inclusion in A. If it holds, the proof criterion must be satisfied. Otherwise, we cannot be sure that there does not exist another reduction that would be included in A. However, we can be certain that at least the old proof criterion $P \subseteq A$ would also fail. Therefore, the new proof criterion is a strict improvement.

We will discuss two main directions for achieving increased reduction power: *conditional commutativity* and *abstraction*. To illustrate, we will consider the programs in fig. 1. We begin with *conditional commutativity* [2] and the program in fig. 1a. Consider the question: Do the array assignments `a[i]:=0` and `a[j]:=1` commute, i.e., do the sequences `a[i]:=0; a[j]:=1` and `a[j]:=1; a[i]:=0` exhibit the same behaviour? In fact, they do not: One sequence is guaranteed to leave `a[i]` with the value 0, while the other may not (in the case where $i = j$). We cannot soundly declare these statements commutative. However, consider the situation in fig. 1b: It is easy to see that whenever both statements are enabled, $i \neq j$ is guaranteed to hold. Hence, we declare the assignments commutative. Formally, the partial commutativity relation is now parametrized in a

system state. In TAR, the states (i.e., assertions) of the Floyd-Hoare automaton A provide an excellent source of such information.

Let us now turn to the second research direction, *abstraction*, and the program in fig. 1c. Here we have the same statements as before. As in fig. 1a, the two orders can lead to different outcomes for the value of `a[i]`. However, it makes no difference for the property of interest, as specified by the assertion in thread 1. Therefore, there exist abstractions of the statements that commute and are still precise enough to guarantee property. In particular, these abstracted statements would set `a[i]` resp. `a[j]` to a non-deterministically chosen non-negative value. This corresponds to the well-known fact that abstraction changes the commutativity, and sometimes increases it. We thus aim to define commutativity relations based on abstraction. However, we must take into account two possible problems:

Firstly, the abstract commutativity may not be more general than concrete commutativity, but simply incomparable. In order not to lose the reduction power of the concrete commutativity, the two can be combined. Doing so without losing soundness turns out to be non-trivial. Given now a concrete and an abstract commutativity relation, we extend our proof criterion to the inclusion $P \subseteq cl_{concr}\big(cl_{abstr}(A)\big)$, corresponding to the relational composition of the respective trace equivalence relations. This is strictly weaker than either of the criteria $P \subseteq cl_{concr}(A)$ or $P \subseteq cl_{abstr}(A)$. We develop new POR algorithms in order to provide an effective (sufficient) test for the new criterion.

Secondly, automatically chosen abstractions may introduce incorrectness not present in the original program. In order to avoid the need for a second dimension of abstraction refinement, we bound our abstraction by the current proof candidate A: The abstraction of a trace already proven infeasible by A must be guaranteed to be infeasible as well. Again the integration into the TAR approach provides us with the necessary information. It is still non-trivial which abstraction should be chosen precisely. We aim here for a definition that not only provides good empirical results, but also a strong theoretical guarantee, allowing us to compare POR to other proof schemes for concurrent programs.

In conclusion, the additional information provided by integrating POR directly in the verification process opens up interesting new possibilities. There are still many open questions in this area, including POR algorithms for the combination of several commutativity relations, different notions of conditional commutativity and their properties, as well as an automatic choice of abstractions suitable for increased commutativity.

References

[1] F. Cassez et al. "Verification of Concurrent Programs Using Trace Abstraction Refinement". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. LNCS. Springer, 2015, pp. 233–248.

[2] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Springer, 1996.

[3] M. Heizmann et al. "Refinement of Trace Abstraction". In: *Static Analysis*. LNCS. Springer, 2009, pp. 69–85.