Translation Validation of System Abstractions

Jan Olaf Blech, Ina Schaefer, Arnd Poetzsch-Heffter

Software Technology Group
University of Kaiserslautern
Germany

Abstraction is a technique intensively used in the verification of large, complex or infinite-state systems. Due to inherent limitations model checkers are unable to deal with such systems directly, instead abstraction can reduce or discretize the state space. During the past decade a significant amount of research has focussed on finding abstraction methods reducing the state space sufficiently while preserving necessary precision. With abstraction algorithms getting more and more complex it is often difficult to see whether valid abstractions are generated. However, for using abstraction in model checking it has to be ensured that properties verified for the abstract system also hold in the concrete. In principle, there are two ways to guarantee correctness of abstractions: Abstraction algorithms (and their implementations!) can be verified once and for all or a tool can be build that verifies the abstraction results for each distinct run of the algorithm's implementation.

In this work, we will show how to use the second variant. For verifying a system abstraction, we set up a tool that is given a concrete system and a property to be checked. As output it produces an abstract system, a corresponding abstract property and furthermore a proof script that the abstraction is property preserving. The abstraction is correct if the proof script succesfully passes a theorem prover. Thus, the abstraction algorithm's implementation is runtime verified.

Our work towards runtime verification of system description abstractions is inspired by a translation validation [8] based approach for compilers [5]. In the area of compiler verification it has turned out that runtime verification of compilers is often the method of choice for achieving guaranteed correct compilation results.

While previously correctness of abstractions was established by showing soundness for all possible systems [2, 3], in our approach the abstraction is proved correct for a specific system and properties to be verified. Runtime verification of abstractions allows to view the tool generating abstractions as black box, although this black box may still provide basic hints on the performed abstraction. If the abstraction algorithm is replaced it is not always necessary to change the generation mechanism of the correctness proofs. Correctness proofs for distinct abstractions are usually less complex and easier to establish than proofs for a general abstraction algorithm. Also note, that in this approach the correctness of abstractions is proved formally using a theorem prover instead of a paper-and-pencil-proof.

We formalise concrete and abstract system semantics in the theorem prover Isabelle/HOL[7]. Additionally, we formulate a correctness criterion based on sim-

ulation between original and abstract system. The correctness criteria are based on property preservation of temporal logic fragments under simulation, for instance the universal fragment of CTL* is preserved under simulation [1] which can further be extended to fragments of the mu-calculus [6, 4]. The actual proof of simulation consists of two main tasks. First, a concrete simulation relation satisfying the correctness criterion has to be found. Each class of abstractions has its own requirements on the selection of this relation. Second, proof scripts to be run in Isabelle/HOL have to be generated. These are highly dependant on the original system and performed abstractions. Hence, for both steps hints provided by the abstraction algorithm are desirable. It is furthermore crucial for the verification process that the produced proof scripts do not only allow the derivation of a correct proof but can also be checked in adequate time.

The proposed technique is applied for the verification of embedded adaptive systems in the automotive sector [9]. Beside potentially unbounded data domains the size of the considered systems is huge. For model checking, these systems can in a first step be abstracted by mapping unbounded data domains to finite abstract domains to reduce and discretise the state space. We have successfully applied runtime verification of this kind of data abstraction.

For future work, we aim at extending our approach to a broader class of abstraction techniques and plan to combine abstraction with modular reasoning applying runtime verification for correctness of resulting system transformations.

References

1. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM TOPLAS*, 16(5):1512–1542, September 1994.
2. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Proceedings of POPL*, pages 238–252. ACM Press, January 1977.
3. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. pages 269–282. ACM Press, January 1979.
4. Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.
5. Marek Jezry Gawkowski, Jan Olaf Blech, and Arnd Poetzsch-Heffter. Certifying Compilers based on Formal Translation Contracts. Technical Report 355-06, Technische Universität Kaiserslautern, November 2006.
6. Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
7. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.
8. A. Pnueli, M. Siegel, and E. Singerman. Translation validation. *Lecture Notes in Computer Science*, 1384, 1998.
9. I. Schaefer and A. Poetzsch-Heffter. Using Abstraction in Modular Verification of Synchronous Adaptive Systems. In *Proc. of "Workshop on Trustworthy Software", Saarbrücken, Germany, May 18-19*, 2006.