A Composite Model Checking Toolset for Analyzing Software Systems

Tevfik Bultan
Department of Computer Science
University of California
Santa Barbara, CA 93106
e-mail: bultan@cs.ucsb.edu

Model checking has proved to be a successful technique for verifying hardware systems. Given a transition system and a temporal property, model checking procedures exhaustively search the state space of the input transition system to find out if it satisfies the given temporal property. Recently, model checking has been used for analyzing software specifications with encouraging results [CAB+98]. The state-space of a software specification can be explored using model checking procedures to verify or falsify (by generating counter-example behaviors) its properties.

The success of model checking has been partially due to Binary Decision Diagrams (BDDs) – a data structure that can encode boolean functions in a highly compact format. The main idea in BDD based model checking is to represent sets of system states and transitions as boolean formulas, and manipulate them efficiently using BDDs [McM93]. BDD data structure supports the operations required for model checking: intersection, union, complement, equivalence checking and existential quantifier elimination (used to compute pre- and post-conditions). This type of model checking is called *symbolic* since the system states are represented implicitly by BDDs during the state space search.

In recent years new symbolic representations have been proposed. For example, HyTech, a symbolic model checker for hybrid systems, encodes real domains using linear constraints on reals [AHH96]. Recently, we proposed a model checker for integer based systems, which uses Presburger arithmetic (integer arithmetic without multiplication) constraints as its underlying state representation [BGP97]. Using constraint representations one can verify systems with infinite variable domains (which is not possible using finite representations such as BDDs).

Our goal in this project is to develop a toolset which combines various symbolic representations in a single composite model checker. In the composite model checking approach each variable in the input system is mapped to a symbolic representation type [BGL98]. (For example, boolean and enumerated variables can be mapped to BDD representation, and integers can be mapped to Presburger constraint representation.) Then, each atomic event in the input system is conjunctively partitioned where each conjunct specifies the effect of the event on the variables mapped to a single symbolic representation. Conjunctive partitioning of the atomic events allows pre- and post-condition computations to distribute over different symbolic representations.

We plan to structure the composite model checking toolset using a layered class hierarchy. The lowest layer will contain libraries for manipulating various symbolic representations such as BDDs and arithmetic constraints. We plan to develop an API which will be shared by different symbolic representations. At the next level of the hierarchy we will have the composite-model library to handle operations over mixed-type expressions (e.g., equivalence check, intersection, etc.); in turn, these operations will invoke their relevant type-specific counterparts in the lower level to help carry out the desired effect. At the top level, the model checker will implement the fixpoint computations using the composite-model library. We already implemented a prototype toolset based on this structure which combines BDD and Presburger constraint representations [BGL98]. We plan to expand our composite model checker by adding other symbolic representations which will allow us to encode variable types such as reals and queues. We would also like to compare performances of different symbolic representations.

We plan to investigate techniques for generating efficient symbolic representations for software specifications. Particularly, we would like to investigate automated or semi-automated techniques for abstraction, partitioning, and compositional analysis. Our goal is to use the composite model checking toolset to investigate effectiveness of symbolic analysis techniques in verification of software systems.

References

[AHH96] R. Alur, T. A. Henzinger, and P. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[BGL98] T. Bultan, R. Gerber, and C. League. Verifying systems with integer constraints and boolean predicates: A composite approach. In *Proceedings of the 1998 International Symposium on Software Testing and Analysis*, pages 113–123, 1998.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using Presburger arithmetic. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *LNCS*, pages 400–411. Springer, 1997.

[CAB+98] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, 1998.

[McM93] K. L. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, Massachusetts, 1993.