Abstraction and refinement for model checking inconsistent systems

Michael Huth[1] and Shekhar Pradhan[2]

[1] Computing and Information Sciences, Kansas State University, huth@cis.ksu.edu,
WWW home page: http://www.cis.ksu.edu/~huth
[2] Department of Computer Science, University of Maryland, pradhan@cs.umd.edu,
WWW home page: http://www.cs.umd.edu/users/pradhan

Abstract. We stress the importance of refinement and abstraction notions and their restricted but well understood soundness in conventional model checking. We then explain why such notions should be central to the analysis of systems with multiple points of views. In particular, they can guide us in settling design questions such as the semantics of negation, and the production of valuable debugging information.

1 Established work

We applaud the established and recent work on modeling and analyzing software artifacts based on potentially inconsistent multiple points of view. Established frameworks typically allow the representation of such points of view as well as the detection of inherent inconsistencies; they then propose ways in which such conflicts could be resolved — we mention the work of A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh.

The recent work by S. Easterbrook, M. Chechik, et al. advances these modeling and detection frameworks in several critical ways. In the context of our position, the most important contributions of their work are

- to model individual points of view as multi-valued Kripke structures, where values range over a quasi-boolean lattice;
- to allow points of view to enjoy varying functionalities (e.g. novel features); and a tool-supported range of heuristics for composing points of view, resulting again in a multi-valued Kripke structure;
- to enable users to write specifications of behavior in a temporal logic, encompassing local and global behavior; and
- to give a semantics to temporal logic formulas over multi-valued Kripke structures, rendering an executable framework for reasoning about behavior of global systems or their particular points of view.

2 Our position

Among formal methods, these recent advances are classified as *property verification*, regardless of their multi-valued nature: a behavioral property is (i)

expressed in whatever suitable form (regular expression, temporal logic formula, tree automata, etc), (ii) analyzed on a given model, and (iii) debugging information is fed back to the user. Making this activity sound requires additional foundational machinery. Such machinery is fairly well understood in analyzing consistent systems, but needs to be redeveloped for inconsistent systems. We sketch below the two key issues present in the consistent case.

First, a positive verification of some property (ϕ) on a model ($\mathcal{M}$) may not secure that ϕ holds for all "correct" implementations. Thus, we require a formal notion of implementations and *refinement* and a proof that property verification is *sound* with respect to such refinements: if ϕ holds for $\mathcal{M}$, then ϕ holds for all of its refinements, including all implementations. We note that such formalisms are well developed in program analysis and concurrency theory.

Second, one often needs to analyze a complex software artifact whose specification may not even exist. Property verification of such a system can often be done only by *abstracting* it, performing the check on the abstraction, and transferring that result back to the actual artifact. Well established approaches are typically limited to checking *safety properties* ("nothing bad will happen") and their transfer is limited to *positive* checks only: if the model check refutes a property for the abstraction/specification, no insight is gained about the underlying artifact/refinement.

In the context of model checking systems with multiple, possibly inconsistent points of view, we therefore regard it important to develop similar foundations for the many-valued setting. Specifically, we claim that

1. sound semantics for refinement and abstraction of multi-valued Kripke structures should be developed, considerably extending the applicability of these frameworks to reasoning about systems with multiple points of view;
2. refinement and abstraction should be developed as dual notions, depending on the nature of the multi-valued logic;
3. the work on merging multiple points of view should be extended to the analysis of *real* implementations, as done in code optimizations in compilers;
4. the soundness of such notions not only directly depends on the particular multi-valued semantics of a temporal logic but, conversely, an intuitively justified notion of refinement or abstraction should provide important *guidance* in how to interpret temporal logic formulas on such models (e.g. an intuitionistic versus a classical treatment of negation);
5. the transfer of abstraction interpretation techniques to the many-valued setting should complement methods based on merging and collapsing many-valued truth lattices;
6. the presence of multiple values in the underlying logic can improve the transfer power of model-checking results on abstractions to real implementations by extending that capability to properties that go beyond mere safety aspects, e.g. *liveness properties* ("something good will happen"), or properties that combine safety and liveness features;
7. debugging techniques from the consistent setting should be modified to provide feedback as to why a model check fell short of attaining a desired truth value.