

Analysis of Agent-Based Systems Using Decision Procedures

In *Formal Approaches to Agent-Based Systems, Revised Papers*, Edited by
J.L. Rash et al. *Lecture Notes in Artificial Intelligence 1871*, Springer,
Heidelberg, Germany, July 2001

Ramesh Bharadwaj

Naval Research Laboratory
Washington, DC 20375-5320
ramesh@itd.nrl.navy.mil

In recent years, model checking has emerged as a remarkably effective technique for the automated analysis of descriptions of hardware systems and communication protocols. To analyze software system descriptions, however, a direct application of model checking rarely succeeds [1, 3], since these descriptions often have huge (often infinite) state spaces which are not amenable to the finite-state methods of model checking. More important, the computation of a fixpoint (the hallmark of the model checking approach) is not always needed in practice for the verification of an interesting class of properties, viz, properties that are invariantly true in all reachable states or transitions of the system. To establish a property as an invariant, an induction proof, suitably augmented with automatically generated lemmas, often suffices.

Salsa is an invariant checker for specifications in SAL (the **SCR Abstract Language**). To establish a formula as an invariant without any user guidance Salsa carries out an induction proof that utilizes tightly integrated decision procedures, currently a combination of BDD algorithms and a constraint solver for integer linear arithmetic, for discharging the verification conditions. The user interface of Salsa is designed to mimic the interfaces of model checkers; i.e., given a formula and a system description, Salsa either establishes the formula as an invariant of the system (but returns no proof) or provides a *counterexample*. In either case, the algorithm will terminate. Unlike model checkers, Salsa returns a state pair as a counterexample and not an execution sequence. Also, due to the incompleteness of induction, users must *validate* the counterexamples. The use of induction enables Salsa to combat the state explosion problem that plagues model checkers – it can handle specifications whose state spaces are too large for model checkers to analyze. Also, unlike general purpose theorem provers, Salsa concentrates on a single task and gains efficiency by employing a set of optimized heuristics.

The design of Salsa was motivated by the need within the SCR Toolset [4] for more automation during consistency checking and invariant checking [1, 3]. Salsa achieves complete automation of proofs by its reliance on *decision procedures*, i.e., algorithms that establish the logical truth or falsity of formulae of *decidable* sub-theories, such as the fragment of arithmetic involving only integer linear constraints called Presburger arithmetic. Salsa's invariant checker consists of a tightly integrated set of decision procedures, each optimized to work

within a particular domain. Currently, Salsa implements decision procedures for propositional logic, the theory of unordered enumerations, and integer linear arithmetic.

After some experimentation, we arrived at the following practical method for checking state and transition invariants using Salsa (see Figure 1): Initially apply Salsa. If Salsa returns *yes* then the property is an invariant of the system, and we are done. If Salsa returns *no*, then we examine the counterexample to determine whether the states corresponding to the counterexample are reachable in the system. If so, the property is false and we are done. However, if one concludes after this analysis that the counterexample states are unreachable, then one looks for *stronger invariants* to prove the property. Salsa currently includes a facility that allows users to include such auxiliary lemmas during invariant checking. There are promising algorithms for automatically deducing such invariants, although Salsa currently does not implement them.

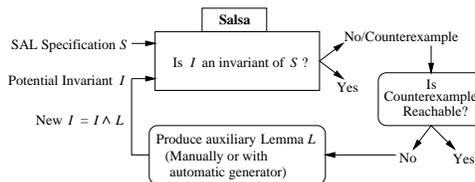


Fig. 1. Process for applying Salsa.

In my future work, I would like to focus on using Salsa technology for descriptions of distributed and concurrent systems, most notably agent-based systems. My hope is that my interactions with scientists and researchers from the formal methods and agents based systems community will provide the necessary impetus for this work to proceed. The First Goddard Workshop on Formal Approaches to Agent-Based Systems is an excellent forum for such interactions.

References

1. Ramesh Bharadwaj and Constance Heitmeyer. Model checking complete requirements specifications using abstraction. *Automated Software Engineering*, 6(1), January 1999.
2. Salsa: Combining constraint solvers with BDDs for automatic invariant checking. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2000)*, LNCS 1785, Springer-Verlag, March 2000.
3. C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Trans. on Softw. Eng.*, 24(11), November 1998.
4. Constance Heitmeyer, James Kirby, Jr., Bruce Labaw, and Ramesh Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *Proc. Computer-Aided Verification, 10th Annual Conf. (CAV'98)*, Vancouver, Canada, 1998.