Incremental LTL Model Checking

Gila Cohen and Orna Kupferman
The Hebrew University
{gilac,orna}@cs.huji.ac.il

March 12, 2003

Today's rapid development of complex and safety-critical systems requires formal-verification methods. One such popular method is model checking. In model checking, we check whether a system meets a desired behavior by executing an algorithm that checks whether a mathematical model of the system satisfies a formal specification that describes this behavior. The algorithmic nature of model checking makes it fully automatic, and thus attractive to practitioners. At the same time, model checking is very sensitive to the size of the mathematical model of the system. Commercial model-checking tools need to cope with the exceedingly large state spaces that are present in real-life designs, and often require intensive use of memory and CPU time [CGP99].

Almost all model-checking tools use a "start over" policy: suppose the system has to satisfy a list of 100 specifications, it passed successfully the check of specifications 1 to 63, and an error has been detected during the check for the 64th specification. The error is fixed, and the model-checking process starts all over again to verify the new system with respect to all the 100 specifications. Fixes are usually local and they involve only a small fraction of the system; e.g., a change of a single assignment. A natural question that arises is whether and how one can use the work already done when early versions of the system were checked in order to simplify the verification of the new version. The objective of our work is to answer these questions and to provide appropriate incremental model-checking techniques.

Incremental model-checking has the flavor of *dynamic graph algorithms*. Consider a problem P for graphs (e.g., connectivity) and assume we have a graph instance that undergoes an on-line sequence of updates (typically, edge insertion and removal). The goal of dynamic graph algorithms is to make use of previous computation to recompute the solution for P after each update more quickly [EGI97]. In model checking, we also reason about graphs, yet the setting is more challenging, as the problem P, namely the specification the graph should satisfy, is part of the input. We formalize the relation between incremental model checking and dynamic graph algorithms and prove that the problem of incremental LTL model checking and the problem of dynamic maintenance of strongly connected components in a graph are equivalent. Unfortunately, the latter problem (in fact, already dynamic graph connectivity) is one of the open problems in dynamic graph algorithms, thus one should not expect an easy solution to our problem.

In the automata-theoretic approach to model checking, we reduce problems about systems and specifications to problems about languages [VW94]. In particular, model checking of an LTL formula ψ is reduced to checking containment of the language of the system's computations in the language of all the computations that satisfy ψ. The first work we know to study the incremental model-checking problem is [Swa96], which follows this approach and describes[1] algorithms for incremental language containment between two Streett automata. The second work we know is [SS94], which describes an incremental model-checking algorithm for formulas in the alternation-free μ-calculus. Evaluation of μ-calculus formulas involves repeated calculation of fixed points. The algorithm in [SS94] assigns to each variable participating in the formula the set of states that satisfy it. The idea of [SS94] is to over-approximate the effect that changes in the graph has on the assignment to the variables. The variable assignment is raised in the assignment lattice for greatest fixed-point variables and lowered in the lattice for least fixed-point variables. The approximations are then checked for their correctness and assignment is updated accordingly.

Recall that LTL model checking is reduced to language containment. Technically, this is done by checking the nonemptiness of the product of an automaton that accepts exactly all the computations of the system and an automaton that accepts exactly all the computations that do not satisfy the LTL formula. Since LTL formulas describe nonterminating behaviors, the automata run on infinite words and use the Büchi acceptance condition, where a run is accepting iff it visits the set of accepting states infinitely often. Accordingly, checking the emptiness of the product automaton amounts to checking whether it contains a fair path – one that visits the set of accepting states infinitely often. Assuming that the accepting states are labeled *fair*, the above reduces LTL model checking into the model checking of the μ-calculus formula $\psi = \nu Y_1.\mu Y_2.\exists\bigcirc([fair \wedge Y_1] \vee Y_2)$, which asserts that the product contains a fair path (that is, ψ is equivalent to the $\exists$LTL formula $\exists\Box\Diamond fair$). The formula ψ involves an alternation between least and greatest fixed points, and hence it does not belong to the alternation-free fragment of the μ-calculus, and cannot be handled by the algorithm in [SS94]. The extension of [SS94] to the full μ-calculus is very complicated and seems impractical.

We follow the ideas that are used in [SS94] and develop an incremental algorithm that is tailored for the specific formula ψ. Given a set of changes to the system, where a change corresponds to insertion or removal of an edge, we first update the product automaton to reflect the changes to the system. For a new edge in the product automaton, we generate an over-approximated set of states that are candidates to satisfy ψ. For a deleted edge, we generate an over-approximated set of states that are candidates to violate ψ. This set is then reduced to contain only states that satisfy (or violate, in case of removal) ψ in the new product. The algorithm can handle insertion and removal of several edges simultaneously.

The formula ψ does not have an equivalent alternation-free μ-calculus formula. Consider the alternation-free μ-calculus formula $\psi' = \nu Y_1.(\mu Y_2.fair \vee \exists\bigcirc Y_2) \wedge \exists\bigcirc Y_1$. The formula ψ' asserts that the product contains a path such that all the locations in the path can reach a fair state

[1] As far as we were able to follow – we found it very hard to read Swamy's work as some of the notations are not well defined.

(that is, ψ' is equivalent to the $\exists$LTL formula $\exists\Box\exists\Diamond fair$). While ψ and ψ' are not equivalent, it is shown in [KV98, BRS99] how the evaluation of ψ' can often be helpful in the process of LTL model checking. We describe the algorithm of [SS94] when applied to the formula ψ' and show how the result can be used for an incremental valuation of ψ. Finally, while the algorithm in [SS94] is enumerative, we suggest both incremental and symbolic algorithms [2].

Our algorithms never perform worse than an LTL model-checking algorithm that starts from scratch, and it is easy to find examples where the complexity of our algorithms is proportional to the size of the change that the system undergoes rather than to the size of the entire system. Nevertheless, in typical settings, our algorithms do not do much better than an algorithm that applies some trivial considerations (e.g., restrict attention only to the parts of the system that are reachable from the places where updates occur). The ultimate goal of incremental model checking, of an algorithm with a complexity that is always proportional to the size of the change is still far. The fact that the corresponding problems in dynamic graph algorithms stand unchallenged suggests that the way to such an algorithm is long.

References

[BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. 11th CAV*, LNCS 1633, pages 222–235. Springer-Verlag, 1999.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[EGI97] D. Eppstein, Z. Galil, and G. Italiano. Dynamic graph algorithms. In *CRC Handbook of Algorithms and Theory of Computation*, chapter 22. CRC Press, 1997.

[KV98] O. Kupferman and M.Y. Vardi. Relating linear and branching model checking. In *IFIP Working Conference on Programming Concepts and Methods*, pages 304 – 326, New York, June 1998. Chapman & Hall.

[SS94] O.V. Sokolsky and S.A. Smolka. Incremental model checking in the modal μ-calculus. In *Proc. 6th CAV*, LNCS 818, pages 351–363, Springer-Verlag, 1994.

[Swa96] G. Swamy. *Incremental Methods for Formal Verification and Logic Synthesis.* PhD thesis, University of California at Berkeley, 1996.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[2] Our symbolic algorithm operates on the OBDDs of the system before the updates and ignores the problem of incremental update of OBDDs.