LEVER: A Tool for Learning Based Verification (Tool Submission)

Abhay Vardhan, Mahesh Viswanathan,
Department of Computer Science,
University of Illinois at Urbana-Champaign, Urbana, IL, USA.
{vardhan,vmahesh}@cs.uiuc.edu

1 Introduction

Software systems are often modeled using infinite structures such as unbounded integers, infinite message queues and call stacks, and unbounded number of processes. This makes verification of these systems hard- in fact, for most common classes of infinite state systems, the verification problem can shown to be undecidable.

In the *Learning-to-Verify* [12, 13, 11, 10] project, we have developed a new paradigm for verification of systems (possibly infinite state) which is based on using techniques from *computational learning theory*. Verification of systems usually entails computing either the set of states reachable from the initial states or certain *fixpoints* associated with logical formulas. To see our main idea, consider the problem of identifying the set of reachable states which is needed for verifying safety properties. Instead of computing this set by iteratively applying the transition relation, we view it as a target set to be learned by answering certain queries (for example, membership and equivalence queries). In general, these queries cannot be answered for the reachable states directly. To solve this problem, instead of learning the reachable states, we learn a richer set of state-witness pairs where a pair consists of a reachable state and a witness which demonstrates how that state is reachable. We have shown that the additional information in the witness allows both membership and equivalence queries to be answered. Once the set of state-witness pairs is learned, the reachable states are easily computed which can in turn be used to check the safety property. We have also extended the learning based technique to verify liveness properties using either *Computational Tree Logic* with fairness or ω-regular languages (see [12, 13]).

The learning based verification method enjoys several nice properties. First, the running time of the verification algorithm depends not on the time taken to converge to the fixpoint (which may not even be achievable in a finite number of steps) but on the size of the symbolic representation of the fixpoint. Second, it avoids the space overhead of computing intermediate approximations to the fixpoint. Finally, the learning based verification method is sound (it never gives an incorrect answer) and if the fixpoint set is representable in the symbolic representation used by the learner, it is also complete (it is guaranteed to terminate).

In this paper, we present a tool called LEVER which implements some of the techniques developed in the *Learning-to-Verify* project. We give some details about the tool and discuss results of running LEVER on some interesting examples. We also compare LEVER with some other tools that are available for verification of infinite state systems.

Related Work. Independently of our work, Habermehl *et al.* [4] have also proposed a learning based method for verification of systems. However, their work assumes that the transition system is length-preserving which is a restrictive assumption in the case of verification of liveness properties. Learning has also been used for verification in other contexts such as learning assumptions in compositional reasoning and mining specifications. This is different than our approach because we are learning the fixpoints needed for verification. Apart from learning based verification, some other tools used for verification of infinite state systems are: FAST [3] which uses acceleration techniques to compute the effect of infinite iteration of certain loops, BRAIN [8] which does a backward search from the "unsafe states" and uses Hilbert's bases for symbolic representation of integer sets and ALV [1] which uses *widening* and can also employ acceleration techniques. For a more detailed treatment of the related work, the reader is referred to [14, 9].

2 Overview of Lever

LEVER is currently targeted towards systems with unbounded natural numbers and parameterized systems with unbounded number of processes. The input to LEVER consists of a description of the system model to be analyzed in terms of its variable declarations, transition guards and actions, initial states, the labeling of states with atomic propositions and the property to be verified. The syntax used for the input is similar to FAST [3]. If LEVER terminates on the given system, the output is simply whether the system satisfies the given property or not. In future, a negative answer may be extended to providing a counterexample trace demonstrating the property violation.

Motivated by the practical success demonstrated by *regular model checking*, the states of the system are represented as strings and *regular sets* (encoded as *Deterministic Finite Automata*) are used for the symbolic representation of the fixpoints to be computed for verification. More precisely, we encode a vector $(x_1, x_2, \ldots, x_n)$ of natural numbers as a string s over an alphabet $\Sigma = \{0,1\}^n$. The value of x_i is given in binary by a string formed by s by projecting each of its letters to its ith component. This representation is similar to Boigelot's Number Decision Diagrams and is known to be expressive enough to encode any Presburger set.

We use the automata library from MONA [6]. MONA keeps the states of the automata explicitly but the transition relation is encoded as a multi-terminal shared BDD. This allows a compact representation of the automata even when the alphabet is of large size. The transducer representing the transition relation is also encoded as an automaton in MONA with a set of new variables added to represent the values taken by system variables after application of a transition.

As mentioned earlier, LEVER uses learning to find the fixpoints needed for verification. We use a learning framework that allows the learner to make two kinds of queries: a membership query asking if a particular element is a member of the target set and an equivalence query asking if a proposed hypothesis is equal to the target set. Since regular sets are used as the underlying symbolic representation for sets of states, we need an algorithm for learning regular sets using membership and equivalence queries. For this, we use a modified version of the algorithm described in Kearns and Vazirani [5] which in turn is inspired by the classical Angluin's [2] algorithm for learning regular sets. One major change we have made in the Kearns-Vazirani algorithm is to use the idea of analyzing counterexamples in a binary-search manner as described in Rivest *et. al.* [7]. For efficiency, we also use a symbolic way of answering certain kinds of membership queries as described in [14].

3 Results

We have used LEVER to analyze over 30 different examples. These include various cache coherence protocols such as *Dragon, Firefly, Illinois, MESI, MOESI, Berkeley, Futurebus* and *Synapse*; mutual exclusion protocols such as *peterson, lamport, ticket* and *bakery*; broadcast protocols such as *consistency*, and *producer-consumer*; petri nets such as *lastinfirstserved protocol, Esparza-Finkel-Mayr Counter Machine, RTP* and *manufacturing*; and counter machines such as *lift* and *barber*. We have used LEVER to analyze some safety properties, some simple branching time properties and some more complicated liveness properties which also need fairness constraints. All these examples are available for download along with the LEVER tool.

In order to evaluate the performance of our tool, we have compared our LEVER tool with three other tools popular for verifying safety properties of infinite state systems: FAST [3], BRAIN [8] and ALV [1]. In Table 1, we present a few of the examples we have analyzed (space constraints prevent us from presenting the full results but these are available in [9]). All analysis was done on Intel Xeon based Linux machine running at 1.70GHz with 1GB memory. For some examples, the analysis could not be completed either because the tool did not terminate in two hours, or it exhausted available memory, or (in the case of ALV) it reported that it cannot provide an answer. For these cases, the table shows an entry of $\uparrow$.

The overall comparison of the performance of the various tools is mixed. No single tool is able to outperform all others for all the examples. There are also examples in which some tools are unable to give an answer within a given period of time while others are successful. However, the important observation is that the performance of LEVER is comparable to the other tools and for some examples it is significantly better. Another significant advantage of using LEVER is that, given enough time and memory, the learning based technique gives is guaranteed to terminate as long as the set being learned is regular.

We have also used LEVER to do a case study on the verification of a model of the *Read-Copy-Update* mechanism in the Linux kernel. The interested reader is referred to [9] for details.

	LEVER	FAST	BRAIN	ALV
noaccel	0.031s	↑	0.004s	0.025s
flatcounter	0.153s	↑	0.004s	0.052s
manufacturing	0.821s	2.422s	10.974s	↑
ticket2i	0.585s	0.679s	↑	↑
consistency	0.932s	142.814s	0.057s	571.473s
kanban	3.952s	7.081s	↑	↑

Table 1. Running times for safety properties. Due to space constraints only a few examples are presented; full results can be obtained from [9].

References

1. ALV. Action language verifier. http://www.cs.ucsb.edu/~bultan/composite/, 2004.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, Nov. 1987.
3. S. Bardin, A. Finkel, J. Leroux and L. Petrucci. FAST: Fast Accelereation of Symbolic Transition systems. In *Proc. 15th Conf. on Computer Aided Verification*, 2003.
4. P. Habermehl and T. Vojnar. Regular model checking using inference of regular languages. In *Proc. of Infinity'04, London, UK*, 2004.
5. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, Massachusetts, 1994.
6. N. Klarlund and A. Møller. Mona. http://www.brics.dk/mona/, 2004.
7. R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Inform. Comput.*, 103(2):299–347, Apr. 1993.
8. T. Rybina and A. Voronkov. Brain : Backward reachability analysis with integers. In *AMAST*, pages 489–494, 2002.
9. A. Vardhan. *Learning to Verify Systems*. PhD thesis, Dep. of Computer Science, University of Illinois at Urbana Champaign, 2006.
10. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Actively learning to verify safety for FIFO automata. In *LNCS 3328, Proc. of FSTTCS'04, Chennai, India*, pages 494–505, 2004.
11. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Learning to verify safety properties. In *LNCS 3308, Proc. of ICFEM'04, Seattle, USA*, pages 274–288, 2004.
12. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using language inference to verify omega-regular properties. In *Proceedings of the Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440, pages 45–60, Edinburgh, UK, April 2005. Springer.
13. A. Vardhan and M. Viswanathan. Learning to verify branching time properties. In *Proc. of the Twentieth IEEE/ACM International Conference on Automated Software Engineering, Long Beach, California, USA*, 2005.
14. A. Vardhan and M. Viswanathan. Learning to verify branching time properties. Technical Report UIUCDCS-R-2005-2630, ftp://ftp.cs.uiuc.edu/pub/dept/tech_reports/2005/UIUCDCS-R-2005-2630.pdf.gz, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.