LIRA: Handling Constraints of Linear Arithmetics over the Integers and the Reals*

Bernd Becker[1], Christian Dax[2], Jochen Eisinger[1], and Felix Klaedtke[2]

[1] Albert-Ludwigs-Universität Freiburg, Germany
[2] ETH Zurich, Switzerland

1 Introduction

The mechanization of many verification tasks relies on efficient implementations of decision procedures for fragments of first-order logic. Interactive theorem provers like PVS also make use of such decision procedures to increase the level of automation. Our tool LIRA[3] implements decision procedures based on automata-theoretic techniques for first-order logics with linear arithmetic, namely, for $\mathrm{FO}(\mathbb{N}, +)$, $\mathrm{FO}(\mathbb{Z}, +, <)$, and $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$.

The theoretical foundations for using automata to decide logics like Presburger arithmetic, i.e., $\mathrm{FO}(\mathbb{N}, +)$ were laid in the 1960s [4]: For Presburger arithmetic, the elements of the domain are represented by finite words, and for a given formula, one constructs recursively over the formula structure an automaton that accepts precisely the words that represent the natural numbers that satisfy the formula. Automata constructions handle the logical connectives and quantifiers. A similar approach works for $\mathrm{FO}(\mathbb{Z}, +, <)$ and $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. To represent reals, one uses infinite words. In [2], it is shown that weak deterministic Büchi automata (WDBAs) suffice to decide $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. WDBAs are a restricted class of Büchi automata, which can be handled algorithmically almost as efficiently as deterministic finite automata (DFAs).

LIRA also provides an automata library that efficiently represents and manipulates DFAs and WDBAs. LIRA's automata library can be compared to a BDD library for representing and manipulating finite sets encoded by booleans. Instead of BDDs, LIRA uses DFAs to represent and manipulate sets that are definable in $\mathrm{FO}(\mathbb{N}, +)$ and $\mathrm{FO}(\mathbb{Z}, +, <)$, and uses WDBAs for sets definable in $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$. Efficiently representing and manipulating such definable sets has applications beyond deciding these logics efficiently. For instance, in the safety verification of integer-counter systems and hybrid systems one has to cope with such sets. Furthermore,

* This work was supported by the German Research Foundation (DFG) and the Swiss National Science Foundation (SNF).

[3] LIRA is available at `http://lira.gforge.avacs.org/` under the GNU public licence.

approaches like regular model checking rely on manipulating automata efficiently. LIRA's automata library can be used in all these applications.

Closely related to LIRA are LASH [13], PRESTAF [5], and MONA [12]. Like LIRA's automata library, LASH provides operations for automata over finite and infinite words. LIRA outperforms LASH by several orders of magnitude. One reason for the speedup are novel automata constructions. PRESTAF's and MONA's automata libraries only support automata over finite words and can only handle Presburger definable sets. Moreover, MONA's automata library is not tailored to the representation and manipulation of Presburger definable sets. The OMEGA library [14] is related to LIRA since it allows one to represent and manipulate sets definable in $\mathrm{FO}(\mathbb{Z}, +, <)$. In contrast to LIRA, it does not support the reals and uses a formula-based set representation, which does not have a canonical form. Heuristics are used to simplify the set representations. SMT solvers like MATHSAT [3] and YICES [9] are also related to LIRA since they provide decision procedures for fragments of linear arithmetic over the integers and reals. However, these solvers do not handle quantifiers at all or only in a limited way. Note that most current SMT solvers also handle other fragments of first-order theories and combinations thereof.

In the following, in §2, we give implementation details and list some features of LIRA, and in §3, we report on applications and performance.

2 Implementation Details and Features

LIRA is implemented in C++. Given a formula, LIRA's decision procedures construct the minimal DFA or WDBA according to the selected logic. By analyzing the automaton, LIRA determines whether the formula is satisfiable. Additionally, it can output a satisfying assignment and a counterexample if they exist, or it can output the constructed automaton.

LIRA defines a flexible high-level API for the decision procedures. A formula is represented as a tree structure and generic functions implement syntactic transformations on such a tree representation. LIRA's decision procedures use the high-level API to generate from such a tree representation a sequence of abstract operations. The decision procedures can easily be modified and extended to generate sequences that exploit domain specific information or include certain heuristics. The sequence of operations is then executed by using LIRA's automata library to check whether the given formula is satisfiable. LIRA's automata library provides efficient implementations of standard automata constructions for DFAs and WDBAs and specific automata constructions for the supported logics, like for equations and inequations. The automata library is accessible through a low-level API.

LIRA uses a similar automata representation as MONA [12], where shared multi-terminal binary decision diagrams (MTBDDs) are used to compactly represent the transition function of an automaton. In our implementation we use CUDD [6] to represent and to manipulate these MTBDDs. We benefit here from CUDD's cache-optimized algorithms. Similar to MONA, our automata representation supports boolean variables. Our automata representation also utilizes don't care states, which were introduced in [11] for DFAs and can also be used for WDBAs. The advantage of don't care states is that automata constructions usually become conceptually cleaner and more efficient.

To reduce the number of states of a WDBA, we use don't care words as described in [10]. We use an automaton construction that handles quantifiers in $\mathrm{FO}(\mathbb{R}, \mathbb{Z}, +, <)$ more efficiently than previous proposed constructions: it copes with don't care words efficiently and is based on the powerset construction for DFAs instead of the more involved breakpoint construction for determinizing co-Büchi automata (see [8] for details).

3 Applications and Performance

We carried out the following case studies to evaluate LIRA's applicability and performance.[4]

(1) We ran LIRA and the frontends of PRESTAF and LASH on randomly generated formulas with a quantifier prefix $\exists$ and $\forall\exists$. LIRA outperformed PRESTAF and LASH. Moreover, LIRA succeeded to build the automaton for all given formulas whereas PRESTAF and LASH sometimes exceeded the time limit or ran out of memory.

(2) We tested LIRA on formulas that arise in the verification of hybrid systems. The test formulas have one quantifier alternation and are generated by a model checker when accelerating the reachability computation [7]. Although some of the formulas are large (the formula sizes range from under 1 Mbyte up to 39 Mbytes), the constructed WDBAs remain rather small and LIRA handles the quantifiers quickly. Note that in [7] another data-structure, based on and-inverter graphs (AIGs), is used to represent and manipulate the formulas. In contrast to DFAs and WDBAs, this data-structure does not have a canonical form and heuristics are applied for minimization. Their representations usually grow with the number of applied operations and contain redundancies.

(3) We wrote a plugin for the model checker FAST [1] that uses LIRA's automata library. We used FAST's benchmark suite to compare

[4] Experimental results are available at `http://lira.gforge.avacs.org/toolpaper/`.

the running times of our plugin with other FAST plugins based on MONA, PRESTAF, LASH, and the OMEGA library. The plugins based on MONA, PRESTAF, and LIRA have competitive performance, LIRA is in most cases the fastest, whereas the LASH plugin is on all examples significantly slower. The OMEGA plugin has, on few examples, competitive running times. However, on most examples it is either outperformed, exceeds the time limit, or crashes.

Acknowledgements. We thank Stefan Disch, Florian Pigorsch, and Viorica Sofronie-Stokkermans for providing benchmark formulas from the domain of hybrid system verification. We also thank Jérôme Leroux and Gérald Point for assisting us with PRESTAF and FAST related issues.

References

1. S. Bardin, J. Leroux, and G. Point, *FAST extended release*, in Proc. of the 18th Int. Conf. on Computer Aided Verification, vol. 4144 of LNCS, 2006, pp. 63–66.
2. B. Boigelot, S. Jodogne, and P. Wolper, *An effective decision procedure for linear arithmetic over the integers and reals*, ACM Trans. Comput. Log., 6 (2005), pp. 614–633.
3. M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani, *The MathSAT 3 system*, in Proc. of the 20th Int. Conf. on Automated Deduction, vol. 3632 of LNCS, 2005, pp. 315–321.
4. J. Büchi, *Weak second-order arithmetic and finite automata*, Zeitschrift der mathematischen Logik und Grundlagen der Mathematik, 6 (1960), pp. 66–92.
5. J.-M. Couvreur, *A BDD-like implementation of an automata package*, in Proc. of the 9th Int. Conf. on Implementation and Application of Automata, vol. 3317 of LNCS, 2004, pp. 310–311.
6. cudd, *Colorado university Decision Diagram package.* `http://vlsi.colorado.edu/~fabio/CUDD/`.
7. W. Damm, S. Disch, H. Hungar, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz, *Automatic verification of hybrid systems with large discrete state space*, in Proc. of the 4th Int. Symp. on Automated Technology for Verification and Analysis, vol. 4218 of LNCS, 2006, pp. 276–291.
8. C. Dax, J. Eisinger, and F. Klaedtke, *Mechanizing the powerset construction for restricted classes of ω-automata*, Tech. Rep. 228, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, 2007.
9. B. Dutertre and L. de Moura, *The Yices SMT solver*, 2006. Available at `http://yices.csl.sri.com/tool-paper.pdf`.
10. J. Eisinger and F. Klaedtke, *Don't care words with an application to the automata-based approach for real addition*, in Proc. of the 18th Int. Conf. on Computer Aided Verification, vol. 4144 of LNCS, 2006, pp. 67–80.
11. N. Klarlund, *A theory of restrictions for logics and automata*, in Proc. of the 11th Int. Conf. on Computer Aided Verification, vol. 1633 of LNCS, 1999, pp. 406–417.
12. N. Klarlund, A. Møller, and M. I. Schwartzbach, *MONA implementation secrets*, Int. J. Found. Comput. Sci., 13 (2002), pp. 571–586.
13. lash, *The Liège Automata-based Symbolic Handler.* `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
14. omega, *The Omega project.* `http://www.cs.umd.edu/projects/omega/`.