Information Societies Technology (IST) Programme

Future and Emerging Technologies - Open Domain

Project acronym: Advance

Project full title:

Advanced Validation Techniques
for Telecommunication Protocols

Contract no: IST-1999-29082

Deliverable 6

Tools for Efficient Analysis of Extended Automata

Year 2 : October 2001 - September 2002

Date of preparation: September 2002

Contributors: Paris, Liège, Uppsala, Weizmann, Edinburgh

1 Introduction

This document presents the tools developed by the Advance partners during the first 24 months of the project, dedicated to analyzing efficiently extended automata. Some research results aimed at improving the methods implemented by the tools, or at developing new tools, are also described in this document.

2 The TReX Tool

2.1 Overview

TReX (*Tool for the REachability analysis of compleX systems*, [BCAS01]) allows to analyze automatically automata-based models equipped with variables of different kinds of *infinite*-domain data structures and with *parameters* (i.e., uninstantiated constants). These models are, at the present time, parametric (continuous-time) timed automata, extended with integer counters and communicating via unbounded lossy FIFO queues.

The techniques used in TReX are based on *symbolic reachability analysis*. Symbolic representation structures are used to represent infinite sets of configurations, and forward/backward exploration procedures are used to generate a symbolic reachability graph.

The version 1.3 of TReX (which is freely available and can be downloaded at `http://www.liafa.jussieu.fr/~sighirea`) provides packages for symbolic representation of configurations of lossy FIFO channels and parametric timed automata and clock automata. For parametric timed automata and clock automata, the symbolic representation used is the *Parametric Difference Bound Matrix (PDBM)* [AAB00], i.e. a tuple (M, ϕ), where M is a DBM with expressions on parameters (like bounds on the values of the clocks and counters), and ϕ is a constraint on the parameters. The constraints on parameters are represented using an internal library, called FOAF (*First Order Arithmetical Formulas*). Linear and non-linear constraints between real and integer parameters are represented by the syntactic trees. Some heuristics are used to simplify the constraints and to share common sub-formulas. FOAF calls the decision procedures of *Omega* and *Reduce* tools to check the validity of a set of constraints.

For the considered models, the termination of the state-space exploration is not guaranteed in general. However, efficient *extrapolation techniques* are used to help termination. These techniques are based on computing (guessing automatically an over approximation of) the effect of the iteration of control loops detected dynamically during the search.

The work done this year on TReX concerns the following points:

- the improvement of the modular architecture of TReX,
- the integration of NDD library of LASH for the representation of constraints on integer parameters [DS01], and
- the improvement of the extrapolation techniques.

2.2 Modularization

The aim of this work is to transform TReX into an open software, such that its user may choose between several symbolic representations, decision procedures, extrapolation techniques, or exploration algorithms. We have rewritten a part of TReX in order to obtain a module for representing constraints on parameters. This allows us to integrate the NDD representation provided by a library of LASH (see the following paragraph). We also plan to add the RVA representation developed in LASH.

2.3 Integration of the NDD Library

The use of FOAF library for the representation of constraints presents the advantages of being very general to represent linear and non-linear constraints between integer and real parameters and it gives an explicit representation that is easy to read by an user. However, it puts the following problems: (1) representation by syntactic trees has no canonical form, so the simplification of constraints is difficult, (2) it may be expensive in memory consumption for particular classes of constraints, (3) it uses external decision procedures, which call is time consuming. To evaluate the impact of these disadvantages on the performances of TReX, we integrated the NDD library of LASH in order to represent linear constraints between integers.

The NDD library provides a representation based on finite automata accepting the binary coding of sets of integers. This representation has a canonical form, so the simplifications of a set of constraints is natural. Also, the NDD library includes internal decision procedures. However, we cannot say that the NDD library is better than the FOAF library because: it is less general w.r.t. the kind of constraints represented, the finite-automata encoding may consume a lot of memory when big integer constants are used in constraints (FOAF representation is not sensible to this problem), the representation is not explicit, and its translation into a readable formula is difficult.

The experimental results confirm the analysis above. On the example of the lift controller, the execution time is improved by the use of NDDs with 30% to 40% when the number of floors is a small constant or a parameter. The FOAF representation and the decision procedures of *Omega* give better results when the number of floors is a big constant.

In conclusion, the work done in this direction has two benefits. It allows us to improve the performances of TReX for analysis of counter automata. It also suggests several improvements for the NDD library w.r.t. its use in the particular context of TReX. For example, during the analysis with TReX, the same constraint may be generated and stored several times. In order to avoid to store duplicate copies, it would be useful to put each automaton into a hash table. This implies to have an equality operator on NDDs which is more efficient to test than the equality of languages. The structural equality of automata may be a solution to the implementation of this operator.

2.4 Improvement of the Extrapolation Techniques

As explained above, the version 1.3 of TReX tries to apply the extrapolation procedure on each simple control loop detected during the analysis. despite its advantage of being fully automatic, this strategy is very expensive when the number of circuits that can be extrapolated is small. Indeed, the extrapolation test may require a lot of time and memory.

In order to solve this problem, the new version of TReX allows the user either to freeze the application of the extrapolation or to specify a set of control states where the loops having the origin may be extrapolated. The last option is less automatic and uses the experience of the user on the model analyzed. In this way, the user may follows a methodology which, in a first step, uses TReX with extrapolation tried systematically. During the analysis, TReX displays the number of extrapolations tried and effectively done. Based on this information, if the exploration does not terminate, the second step is to use TReX by providing the list of states where the extrapolation should be tried.

The experiences done with this new version of TReX are encouraging and validate the methodology above. For the root contention protocol of IEEE 1394 ("FireWire") standard, we obtain a reduction of the exploration time up to 50% and of the memory consumption up to 20%. This is due to the absence of cycles that can be extrapolated. For the BRP protocol and the lift controller, the improvement obtained is not very important (1% for time and 5% for memory) because most of the cycles can be extrapolated.

3 The LASH Tool

The *Liège Automata-based Symbolic Handler (LASH)* is a collection of tools for representing finite and infinite sets with the help of finite-state representation systems, and for exploring infinite state spaces [LAS].

The work carried out during the second year of the project has been integrated in the release 0.9 of LASH, which contains the following improvements over previous versions :

- A stable and efficient library for creating and manipulating *Real Vector Automata (RVA)*, which are finite-state representations suited for sets of real and integer values [BBR97]. Thanks to a previous result [BJW01], RVA are implemented efficiently by means of weak deterministic infinite-word automata. The RVA library provides the first effective implementation of a decision procedure for the first-order additive theory of real and integer numbers [BW02]. RVA can also be used as canonical representations of sets defined in this theory.
- An extended library of operations over *Number Decision Diagrams (NDDs)*, which allow to represent and manipulate sets of unbounded integer vectors.
- A new compiling front-end for the state-space exploration tool, based on the IF 2.0 language. This compiler has been designed so as to accept full-fledged IF 2.0 models, abstracting out the language features that cannot be handled by LASH.
- Fined-tuned algorithms for the essential state-space exploration operations (in particular, for the computation and application of meta-transitions).
- An improved documentation.

The following features are currently under development and will be part of the next release of LASH :

- A state-space exploration module for timed systems, relying on the expressive power of RVA.
- An extension of our prototype visualization tool to systems using both real and integer variables.
- A general arithmetic solver relying on the decision procedures implemented in LASH.
- A more efficient way of representing sets with an orthogonal structure (i.e., sets that can be expressed as the Cartesian product of smaller sets) by finite-state structures. The implementation of this recent theoretical result would make it possible to store and manipulate much more efficiently NDDs and RVA, thus improving the tool capacities.
- A general interface for accessing easily the various features offered by LASH.

4 The Moped Tool

Edinburgh has continued working on the verification of systems with (possibly recursive) procedures. The formal model employed in this context is that of pushdown systems, which can be seen as automata extended with a stack for keeping track of active procedure calls. This model can handle both control flow and the variables used by such programs with the restriction that variables come from finite domains.

Previously Edinburgh had developed a model-checker for pushdown systems and the temporal logic LTL. The checker (called Moped) has now been extended with an interface to Slam [BR00], a verification suite built by Microsoft Research. In co-operation with MSR, Slam and Moped have been combined and applied to the verification of device drivers with up to 10.000 lines of code. The verification is done in a process of iterative refinement: A C program under consideration is abstracted to a program with only Boolean predicates, which can then be model-checked. Since the abstraction step can lead to spurious counterexamples, this process may have to be repeated several times with increasingly accurate abstractions. In this context, Moped can assume the role of the model-checker.

In the experiments, Moped performed significantly faster than the model-checker originally built into the suite. A detailed account of the experiments and of the optimizations and design choices which were necessary to achieve these results is given in [Sch02a]. Moped (including its source code) has been made available to the public [Sch02b].

5 Foundations and New Algorithmic Methods

5.1 Analysis of Pushdown Systems

In [KPV02b], the authors address the problem of checking pushdown specifications expressed as nondeterministic pushdown parity tree automata (PD-NPT). They show that the model-checking problem for regular systems and PD-NPT specifications can be solved in time exponential in the system and the specification. The model-checking algorithm proposed in the paper involves a new solution to the nonemptiness problem of nondeterministic pushdown tree automata, where the best known upper bound is improved from a triple-exponential to a single exponential. The model-checking problem for context-free systems and PD-NPT specifications is also considered and shown to be undecidable.

5.2 Analysis of Timed Petri Nets

In [AN02], the undecidability of (action based) linear-time temporal logic (LTL) is established for timed Petri nets. This is to be contrasted with decidability of both the problem of checking safety properties for timed Petri nets, and the problem of checking LTL formulas for (untimed) Petri nets. In the paper, the undecidability result is shown through a reduction from a similar problem for lossy counter machines.

5.3 Analysis of Infinite-State Sequential Systems

In [KPV02a], an automata-theoretic framework is developed for reasoning about linear properties of infinite-state sequential systems. The framework is based on the observation that states of such systems, which carry a finite but unbounded amount of information, can be viewed as nodes in an infinite tree, and transitions between states can be simulated by finite-state automata. Checking that the system satisfies a temporal property can then be done by an alternating two-way automaton that navigates through the tree. For branching properties, the framework is known and the two-way alternating automaton is a tree automaton. Applying the framework for linear properties results in algorithms that are not optimal. Indeed, the fact that a tree automaton can split to copies and simultaneously read all the paths of the tree has a computational price and is irrelevant for linear properties. The paper introduces path automata on trees. The input to a path automaton is a tree, but the automaton cannot split to copies and it can read only a single path of the tree. In particular, two-way nondeterministic path automata enable exactly the type of navigation that is required in order to check linear properties of infinite-state sequential systems. As has been the case with finite-state systems, the automata-theoretic framework is quite versatile, which is demonstrated by solving several versions of the model-checking problem for LTL specifications and prefix-recognizable systems. The proposed algorithm is exponential in both the size of (the description of) the system and the size of the LTL specification, and a matching lower bound is given. This is the first optimal algorithm for solving the LTL model-checking problem for prefix recognizable systems. The framework introduced in the paper also handles systems with regular labeling, and in fact it is shown that LTL model checking with respect to pushdown systems with regular labeling is intereducible with LTL model checking with respect to prefix-recognizable systems with simple labeling.

5.4 Analysis of Concurrent programs with procedures

In [BET03], the authors present a generic approach to the static analysis of concurrent programs with procedures. They model programs as communicating pushdown systems. It is known that typical dataflow problems for this model are undecidable, because the emptyness problem for the intersection of context-free languages, which is undecidable, can be reduced to them. In this paper, an algebraic framework for defining upper approximations of context-free languages, is proposed. The approximations belong to different classes of commutative languages, i.e. languages that contain a word if and only if they contain all its permutations. The authors show how to compute such approximations by combining automata theoretic techniques with algorithms for solving systems of polynomial inequations in commutative Kleene algebras.

5.5 Representation of Downward-Closed Languages

A recent work by the Uppsala team [AMN02] addresses the problem of analyzing monotonic systems, which cover Petri nets, lossy channel systems, relational automata, broadcast and cache-coherence protocols, timed Petri nets, ... For such systems, the exact set of reachable states often has a more complicated structure that its downward closure, since it contains much more information about system behavior. In [AMN02], the Uppsala team proposes a novel formalism for representing downward-closed languages, called the *downward closed language generators (dlgs)*. It is shown that the standard set operations such that intersection, union and entailment checking, needed for performing symbolic reachability analysis, are computable for dlgs, which brings them close to their practical use in a tool.

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of the 12th CAV*, volume 1855 of *LNCS*, pages 419–434. Springer Verlag, July 2000.

[AMN02] P. A. Abdulla, P. Mahata, and A. Nylén. Downward closed language generators. Technical report, Uppsala University, Sweden, 2002.

[AN02] P. A. Abdulla and A. Nylén. Undecidability of ltl for timed petri nets. In *Proceedings of 4th International Workshop on Verification of Infinite-State Systems*, 2002.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proceedings of the 9th International Conference on Computer-Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 167–177, Haifa, Israel, June 1997. Springer-Verlag.

[BCAS01] A. Bouajjani, A. Collomb-Annichini, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proceedings of CAV'01*, LNCS. Springer Verlag, 2001.

[BET03] Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. A Generic Approach to the Static Analysis of Concurrent Programs with Procedures. In *30th Symp. on Principles of Programming Languages (POPL'03)*, New Orleans, USA, January 2003. ACM.

[BJW01] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proceedings of IJCAR '01*, number 2083 in LNAI, pages 611–625, 2001.

[BR00] T. Ball and S. Rajamani. The SLAM project, 2000. http://research.microsoft.com/slam.

[BW02] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In *Proc. International Conference on Logic Programming (ICLP)*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19, Copenhagen, July 2002. Springer-Verlag.

[DS01] S. Delaune and M. Sighireanu. Integration of ndd in trex. Technical report, LIAFA, September 2001.

[KPV02a] O. Kupferman, N. Piterman, and M. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proceedings of 14th International Conference on Computer-Aided Verification*, 2002.

[KPV02b] O. Kupferman, N. Piterman, and M. Vardi. Pushdown specifications. In *Proceedings of 9th International Conference on Logic for Programming Artificial Inteligence and Reasoning*, 2002.

[LAS] The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[Sch02a] S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, TU-München, 2002. Submitted.

[Sch02b] S. Schwoon. Moped – A Model-Checker for Pushdown Systems, 2002. http://www7.in.tum.de/∼schwoon/moped.