Tool Presentation: FAST Extended Release

Sébastien Bardin[1], Jérôme Leroux[2], and Gérald Point[2]

[1] LSV: ENS de Cachan & CNRS,
61, av. Pdt. Wilson, 94235 Cachan Cedex, France.
bardin@lsv.ens-cachan.fr
[2] LABRI: Uni. Bordeaux 1 & CNRS,
351, cours de la Libération
F-33405 Talence cedex, France
{leroux,point}@labri.fr

Abstract. FAST is a tool designed for the analysis of counter systems, i.e. automata extended with unbounded integer variables. Despite the reachability set is not recursive in general, FAST implements several innovative techniques such as acceleration and circuit selection to solve this problem in practice. In its latest version, the tool is built upon an open architecture: the Presburger library is manipulated through a clear and convenient interface, thus any Presburger arithmetics package can be plugged to the tool. We provide four implementations of the interface using LASH, MONA, OMEGA and a new *shared automata* package with computation cache. Finally new features are available, like different acceleration algorithms.

Keywords: counter systems verification, acceleration, generic Presburger interface, automata with cache computation.

1 Introduction

The automatic verification of reactive systems is a major field of research. A popular way of modeling such systems is by means of (synchronized) automata extended with variables. The automata represent the control structure of the system, while variables encode data. FAST is a tool for the analysis of systems manipulating unbounded integer variables. We check *safety properties* by computing the reachability set of the systems. Even if this reachability set is not necessarily recursive, we use innovative techniques (*acceleration, flattening, reduction*) to increase convergence. FAST relies heavily on Presburger arithmetics for both system/properties specification and symbolic representation of infinite sets of states. FAST theoretical background is described in [7, 2, 1, 3].

In our opinion, the following facts make FAST a valuable tool for counter system analysis. (1) Since counter systems and Presburger constraints are very expressive, FAST can be applied to a large spectrum of applications and the tool is not tied to a particular specific case-study. (2) Despite the inherent theoretical limitations, the analysis succeeds in most practical cases. (3) FAST design is fully based on a clear theoretical framework. Abilities and limits of the tool are identified: the tool is complete relatively to the class of flattable systems [3]. Since many decidable subclasses of counter systems are flattable [8], FAST provides a unified and efficient verification algorithm for many well-studied classes of counter systems. (4) Finally the user can guide the tool via a script language, which is useful since termination cannot be guaranteed.

Experimentations. FAST has been tested over a pool of 40 infinite-state systems, and the computation succeeded in around 80% of the tests [2, 1]. In [5] FAST is used to prove properties of a class of communication protocols manipulating *counters and queues*. A comparison of FAST and other tools in [3] shows that FAST provides a very efficient engine for (forward) reachability set computation of counter systems.

Extended Release. This new version provides the following improvements: (1) an open architecture based on an analysis engine and a convenient interface (API) for Presburger arithmetics. We provide adaptations of the standard packages LASH [9], MONA [10] and OMEGA [11] to the API; (2) a new Presburger package implementing the API via shared automata [6] equipped with a *computation cache*; (3) various add-ons both in the analysis engine and in the interface.

2 Open architecture

The architecture of the tool has been redesigned, and the tool is now divided in two parts: on the one side, a counter system analysis engine built upon a generic Presburger API; on the other side various implementations of this API. These different libraries can be re-used easily in various applications, independently of FAST and counter system analysis, corresponding to a recurrent demand.

*The generic Presburger programming interface (*GENEPI*).* The API requires only basic operations on Presburger formulas such as conjunction, disjunction, negation, (inverse) projection and satisfiability testing. The API is easy to use, and it is also quite easy to adapt existing Presburger packages to the API.

Implementations of the API. We provide three implementations of the API based upon standard packages LASH, MONA and OMEGA. The MONA implementation corresponds to the former version of FAST.

Potential applications. People concerned with Presburger packages can take advantage of our open architecture and API in at least two ways. *(1) Presburger developers.* People interested in developing a Presburger package can easily linked it to FASTER and use the tool and the 40 case-studies as *intensive benchmarking* for their package. *(2) Presburger users.* People interested in developing any application requiring Presburger arithmetics can use our generic Presburger API, and then select through the set of implementations which one fits most their application.

3 The shared automata package

We have also developed from scratch an implementation of the API using shared automata introduced by Couvreur in [6]. These automata share their strongly connected components in a bdd-like manner. It allow to implement important features for intensive computation, such as cache computation and constant-time equality testing. Our library is functional, but the computation cache is not yet well optimized. However it has already permitted to speed up computation time by a factor 3. The shared automata package is called PresTaf.

4 New features in analysis

The tool has been extended with new capacities, both in the analysis engine and in the interface. (1) One can specify the acceleration algorithm, choosing between standard acceleration and convex acceleration [1]. The last one considers restricted functions but is more efficient. Different search heuristics are also available. (2) One can specify some circuits to be used during the analysis. (3) Finally we developed a tool to transform a Petri net in pnml format into a Fast model. The language pnml [4] describes various extensions of Petri nets and it is under standardization.

5 Comparison of Presburger libraries

We present in figure 1 the performances (time spent in seconds) of Faster depending on our different implementations of Presburger arithmetics. Columns V and T denote respectively the number of variables and transitions in the system. All these systems have infinite reachability sets, except Dekker.

system	V	T	Mona*	Lash	PresTaf**	Omega
Central Server system	13	8	5.94	91.1	7.20	43.3
Consistency Protocol	12	8	77.4	2400	140	50.3
Producer/Consumer Java	18	14	446	2520	57.6	$\geq$ 3600
CSM - N	13	13	13.1	241	12.5	616
Dekker ME	22	22	11.4	287	12.8	$\geq$ 3600
Last-in First-served	17	10	0.65	8.12	1.13	13.9
Multipoll	17	20	7.25	283	8.55	295
SWIMMING POOL	9	6	44.1	993	48.6	$\geq$ 3600

* This implementation corresponds to the former version of Fast.
** A computation cache is available, but not yet optimized.

Fig. 1. Comparison of different Presburger implementations

Even though the computation cache implemented in PresTaf is not fully optimized, figure 1 shows that PresTaf and Mona have significantly the same execution time. LASH seems outperformed by the two previous libraries. Recall that (1) LASH provides Presburger implementation for negative and non-negative integers, thanks to more complex algorithms, and (2) LASH does not implement any computation cache. OMEGA is also outperformed. The tool appears to compute unduly complicated Presburger formulas (even with the simplification method provided by the package), while LASH, MONA and PRESTAF benefit from canonical representations of formulas.

In the previous table, the memory used is not given because, due to cache computation, this value is not representative. Without computation cache, since the internal representations of LASH, MONA and PRESTAF are slightly the same, the three implementations require slightly the same amount of memory.

Availability. FASTER, GENEPI and PRESTAF are available at `http://altarica.labri.fr/`. The tool, the API and the libraries are freely available under the GPL license. The analysis engine is written in C++ and the different implementations of the API are written in C. FASTER has been tested on an Intel PC running Linux and **gcc 4.0.2**.

Acknowledgments. We are grateful to Jean-Michel Couvreur for providing us advices on the implementation of shared automata, and to Ales Smrcka for adapting OMEGA source code to recent compilers.

References

1. S. Bardin, A. Finkel, and J. Leroux. Faster acceleration of counter automata. In *TACAS'2004*. LNCS 2988. Springer, 2004.
2. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *CAV'2003*. LNCS 2725. Springer, 2003.
3. S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA'2005*. LNCS 3707. Springer, 2005.
4. J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, technology and tools. In *ICATPN'2003*. LNCS 2679. Springer, 2003.
5. J. Billington, G. E. Gallasch, and L. Petrucci. FAST verification of the class of stop-and-wait protocols modelled by coloured Petri nets. *Nordic Journal of Computing*. To appear.
6. J.-M. Couvreur. A bdd-like implementation of an automata package. In *CIAA'2004*. LNCS 3317. Springer, 2004.
7. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FST&TCS'2002*. LNCS 2556. Springer, 2002.
8. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *ATVA'2005*. LNCS 3707. Springer, 2005.
9. LASH homepage. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`
10. MONA homepage. `http://www.brics.dk/mona/index.html`
11. OMEGA homepage. `http://www.cs.umd.edu/projects/omega/`