FAST: Fast Acceleration of Symbolic Transition systems

Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci

LSV, CNRS UMR 8643
ENS de Cachan
61 avenue du président Wilson
F-94235 CACHAN Cedex
FRANCE
{bardin,finkel,leroux,petrucci}@lsv.ens-cachan.fr

Abstract. FAST is a tool for the analysis of infinite systems. This paper describes the underlying theory, the architecture choices that have been made in the tool design. The user must provide a model to analyse, the property to check and a computation policy. Several such policies are proposed as a standard in the package, others can be added by the user. FAST capabilities are compared with those of other tools. A range of case studies from the literature has been investigated.

1 Introduction

Model-checking is a wide-spread technique in critical systems verification. Several efficient model-checkers, such as SMV [SMV], SPIN [SPI] or DESIGN/CPN [CPN], are available. However, these tools are restricted to finite systems whereas many real systems are infinite, because of parameters or unbounded data structures.

FAST is a tool designed to allow automatic verification of systems modeled by automata augmented with (unbounded) integer variables (*extended counter automata*). The main issue addressed by FAST is the computation of the *exact* (*infinite*) set of configurations reachable from a given set of initial configurations. Let us recall that verification of safety properties can be reduced to reachability of a given configuration from a set of initial configurations.

A lot of properties are in general undecidable, but there are two ways to deal with undecidability. The first one is to consider decidable subclasses, thus reducing the expressiveness of the model, while the second one is to accept only a semi-algorithm, which does not terminate in the general case but is expected to terminate in most practical cases. We follow the second approach. The techniques used in FAST are based on *acceleration* [FL02]. It comes down to computing the (exact) effect of iterating a control loop of *an arbitrary length* (*cycle*). These cycles are automatically chosen. Both forward and backward reachability are allowed. FAST works on *linear systems*, i.e. *finite sets of linear functions* whose definition domains are defined by a *Presburger formula over non-negative integers*. Most systems with integer variables can be described in such a way. In

[FL02], it is proved that for linear systems whose associated square matrices generate a finite multiplicative monoid – namely *finite linear systems*, the acceleration of a loop terminates. It turns out that most integer variables systems appear to be finite linear systems. Even though termination is not guaranteed, in practice, FAST deals with a large amount of examples of our extended counter automata model (see section 4). We believe that Presburger arithmetic is sufficient to model these problems and that most systems are effectively computable.

FAST is freely available on Jérôme Leroux home page [FAS].

2 Related tools

Table 1 summarises a comparison of the main tools able to cope with integer variables infinite systems. No performance comparison is provided, as these tools do not require the same user implication, they take different input models, more or less limited, and hence perform different computations.

	variable type	guards	actions	acceleration	auto. cycle search	forward	backward	exact comput.	engine
FAST	$\mathbb{N}$	Presburger	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	yes	yes	yes	yes	yes	LNDD, MONA
LASH	$\mathbb{Z}$	convex sets	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	yes	no	yes	yes	yes	NDD
	$\mathbb{R}$	convex sets	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	no	-	yes	yes	yes	RVA
TReX	$\mathbb{Z}$	$\bigwedge \begin{cases} x_i \leq x_j + c \\ x_i \leq c \\ x_i \geq c \end{cases}$	$\bigwedge \begin{cases} x_i = x_j + c \\ x_i = c \end{cases}$	yes	yes	yes	yes	yes	PDBM
	$\mathbb{R}$		$\bigwedge \begin{cases} x_i = x_j \\ x_i = 0 \end{cases}$	yes	yes	yes	yes	no	PDBM
BRAIN	$\mathbb{N}$	$\bigwedge x_i \leq x_j + c$	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	no	-	no	yes	yes	period basis
BABYLON	$\mathbb{N}$	$\bigwedge x_i \leq x_j + c$	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	no	-	no	yes	no	CST
HYTECH	$\mathbb{R}$	convex sets	$\overrightarrow{x} = A.\overrightarrow{x} + \overrightarrow{b}$	no	-	yes	yes	no	convex polyhedra
ALV	$\mathbb{Z}$	Presburger	Presburger	no	-	yes	yes	yes	OMEGA, BDD

Table 1. A comparison of different tools for reachability set computation

3 Architecture

Figure 1 shows the inputs and outputs of FAST. FAST requires a model of the system to analyze, a reachability property to check and a strategy to direct the computation. If it terminates, the tool answers whether the property is satisfied or not. *Settings* can also be optionally set by the user, such as the ordering of variables and stop criteria.

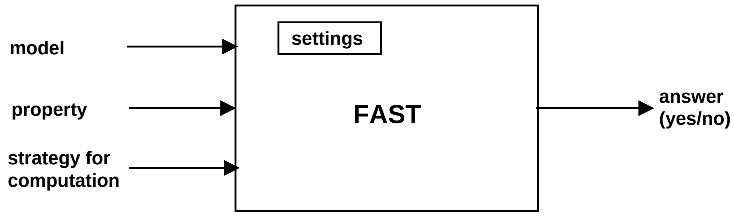

Fig. 1. FAST inputs and outputs

Strategies allow the user to direct "by-hand" the computation. Strategies make it possible to describe standard model-checking features such as forward or backward reachability as well as more advanced constructs like a sequence of incremental submodel analysis. This has been successfully used to verify the TTP protocol (see section 4). Concretely, the user describes strategies through a high-level language allowing to manipulate Presburger definable sets of integers, linear functions, booleans and providing primitives for pre^* and $post^*$ operations.

Presburger definable sets of integers are internally represented by Labeled Number Decision Diagrams (LNDDs). LNDDs allow to represent any Presburger formula and provides basic operations on sets (intersection, negation, inclusion or emptiness test) as well as more advanced constructs like the acceleration of a cycle described in [FL02]. Our implementation uses packages from MONA [MON], providing automata operations. An extended version of FAST for integer arithmetic has also been developed, built on LASH libraries. But there was a drop in performances, integer variables being implicitly encoded as two non-negative integer variables. Since all the case studies considered only deal with non-negative integers, we decided to first limit FAST to non-negative integers.

4 Results

FAST has been applied to a large number of examples (about 40), ranging from Petri nets to abstract multi-threaded JAVA programs, mainly taken from [Del]. About 80% of these case studies could effectively be verified. It proves that choices made during FAST design, like having only a semi-algorithm or restricting FAST to non-negative integers, are sound for practical infinite systems verification. Moreover, most of these examples require only a basic predefined strategy (a forward search), thus only little input from the user.

Figure 2 presents the performances obtained by FAST on ten of the most representative examples. Dekker ME is a bounded Petri net, other examples are infinite state systems because of parameters (lift controller) or unbounded integer variables (FMS). Despite its number of variables and transitions, the Swimming Pool protocol is a highly non-trivial protocol. The TTP protocol is a complex group membership protocol, using elaborate guards. The tool computes efficiently these examples. A forward search has been used for all examples. For the particular case of TTP, a more elaborate strategy was also tested, leading to considerable increase in computation time.

Case Study	variables	transitions	time (s)	mem.(MB)	n. states	n. acc	c. length	n. cycles
Dekker ME	22	22	21.72	5.48	5	36	1	22
CSM	13	13	45.57	6.31	6	32	2	35
FMS	22	20	157.48	8.02	21	23	2	46
Swimming Pool	9	6	111	29.06	30	9	4	47
Producer/Consumer with Java threads - N	18	14	723.27	12.46	58	86	2	75
Lift Controller - N	4	5	4.56	2.90	14	4	3	20
TTP	10	17	1186.24	73.24	1140	31	1	17
TTP (ad hoc strategy)	10	17	246.67	72.87	1140	16	1	17

Fig. 2. Results using an Intel Pentium 933 MHz with 512 Mbytes

Considering the case studies that could not be verified (9 out of 40), we propose three reasons for FAST not to terminate. First of all, the input model can be such that FAST cannot terminate, either some loops have infinite associated monoids or the reachability set is not *flatable*, i.e. not computable using a finite set of accelerations [FL02]. Second, the computation may lead to large automata and saturate the memory. Finally, there may be too many cycles to consider, and then the heuristic used by FAST to find cycles to be accelerated reaches its limits.

We currently develop an interface, for the user to set parameters, e.g. the cycle length, which will allow us to cope with these drawbacks.

References

[ALV] ALV *homepage.* `http://www.cs.ucsb.edu/~bultan/composite/`.

[BRA] *The* BRAIN *homepage.* `http://www.cs.man.ac.uk/~voronkov/BRAIN/`.

[CPN] DESIGN/CPN *online.* `http://www.daimi.au.dk/designCPN`.

[Del] G. Delzanno. *Home Page – Giorgio Delzanno.* `http://www.disi.unige.it/person/DelzannoG/`.

[FAS] FAST *homepage.* `http://www.lsv.ens-cachan.fr/~leroux/fast/`.

[FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proc. 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'2002), Kanpur, India, Dec. 2002*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2002.

[HYT] HYTECH *homepage.* `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.

[LAS] *The* LASH *Toolset.* `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[MON] *the* MONA *project.* `http://www.brics.dk/mona/`.

[SMV] SMV *homepage.* `http://www-cad.eecs.berkeley.edu/~kenmcmil/`.

[SPI] SPIN *homepage.* `http://spinroot.com/spin/`.

[TRE] TReX *homepage.* `http://www.liafa.jussieu.fr/~sighirea/trex/`.