FAST: Fast Acceleration of Symbolic Transition systems (tool presentation)

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 large or even 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 which is expected to terminate in most practical cases. We follow the second approach. The techniques used in FAST are based on *acceleration* [FL02,ABS01,WB00]. 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 by such a system. In [FL02], it is proved that for linear systems whose associated square matrices generate a finite multiplicative monoid – namely *finite linear systems*, 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.

2 Related tools

We present four main tools able to cope with integer variables infinite systems, two of them using acceleration or similar techniques.

LASH [LAS] provides a Number Decision Diagram library (NDD) that represents infinite sets of positive and negative integer vectors for exact backward and forward reachability analysis with accelerations. Guards are more restrictive than with FAST as LASH only allows convex guards that are not stable by union nor quantifiers. Moreover, users must provide cycles to be accelerated. Actions are given by general linear transformations. Note that LASH also provides a Real Vector Automata library (RVA) to represent real vectors for reachability analysis without acceleration, and a Queue-content Decision Diagram library (QDD) suited for sets of unbounded FIFO queues contents. NDD, RVA and QDD cannot be used simultaneously.

TReX [ABS01] provides a Parametric Difference-Bound Matrix library (PDBM) that represents real and integer vectors for backward and forward reachability analysis with accelerations; the analysis is exact only for integer vectors. The use of PDBM to represent both integers and reals requires actions and guards to be restrictive (TReX only uses the classical guards and actions of the timed automata theory). Note that TReX also provides a Simple Regular Expression library (SRE) to represent lossy `FIFO` channels contents.

BRAIN [BRA] uses a Bases and Periods Semi-linear representation library that represents non-negative integer vectors; guards contain no quantification and actions are general linear transformations. BRAIN implements a backward reachability algorithm without acceleration.

BABYLON [ADG+02] uses a Covering Sharing Trees library (CST) to represent upward closed sets of non-negative vectors for exact backward reachability analysis without acceleration. Actions are defined by general linear transformations whereas guards are restricted to be upward closed; if non upward closed guards are used, the backward reachability analysis does not remain exact.

BRAIN and BABYLON (as well as FAST) are restricted to non-negative integers. All examples of integer variables systems we have found use only non-negative integers. Most of these tools (LASH, TReX, BRAIN) are available on the net.

3 Architecture

Figure 1 shows the inputs and outputs of FAST.

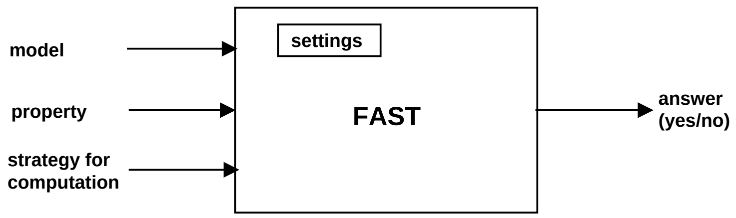

Fig. 1. FAST inputs and outputs

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.

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). This automata description for non-negative integer arithmetic is based on works like [BC96,WB00]. 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, but there was a drop in performances. 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 studied in [FO97]. The TTP protocol is a complex group membership protocol, using elaborate guards. We use the abstraction proposed in [BM02]. The table in figure 2 presents the number of variables and transitions of the models, the computation time, the memory consumption, the number of states of the automaton representing the reachability set, the number of accelerations required to compute this automaton, the length of the cycles which were accelerated and the number of cycles considered by FAST. 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.

5 Conclusion and Future Work

In this paper, we presented FAST, a tool for infinite systems analysis using acceleration techniques. It deals with systems described with Presburger arithmetics. The user can provide both a property to check and a computation policy to enhance the performances according to the model specificities. We also compared FAST with related tools, and showed results obtained on case studies.

In the future, we intend to improve FAST in different ways. Enhancing performances can be achieved by improving the heuristic to find cycles to accelerate or using techniques such as invariants. Extending FAST to allow nondeterministic assignment or to support integers (or even reals) will increase the expressiveness.

Another concern will be to connect FAST to other tools such as DESIGN/CPN or SPIN. FAST version 0.3 will be available on Jérôme Leroux home page [FAS] in February 2003.

References

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: a Tool for Reachability analysis of Complex Systems. In *Proc. 13th Int. Conf. Computer Aided Verification (CAV'2001), Paris, France, July 2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.

[ADG+02] P. Ammirati, G. Delzanno, P. Ganty, G. Geeraerts, J.-F. Raskin, and L. Van Begin. Babylon: An integrated toolkit for the specification and verification of parameterized systems. In Giorgio Delzanno, Sandro Etalle, and Maurizio Gabbrielli, editors, *Specification, Analysis and Validation for Emerging Technologies in Computational Logic*, volume 94 of *Datalogiske Skrifter*, Roskilde, Denmark, July 2002.

[BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *Proc. Coll. on Trees in Algebra and Programming (CAAP'96)*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 1996.

[BM02] A. Bouajjani and A. Merceron. Parametric verification of a group membership algorithm. In *Proc. 7th Int. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'2002), Oldenburg, Germany, Sep. 2002*, volume 2469 of *Lecture Notes in Computer Science*, pages 311–330. Springer, 2002.

[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.

[FO97] L. Fribourg and H. Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 1997.

[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/`.

[WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. 6th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.

A Heuristic

The algorithm used in FAST is presented below. First we introduced data types and parameters used in the algorithm. We considerer two specific types: *Set* and *functions*.

- *Sets* are (symbolic) sets of configurations. We denote by A_S the automaton representing the *Set* S and $|A_S|$ the number of states of this automaton.
- A *function* is a linear function (eventually obtained by composition of other linear functions). The length of a *function* is the number of composition required to build it. *Functions* are sets of functions.
- We denote by $C_k(L)$ the set of *functions* with $length = k$ built on the set of functions L.
- We denote by c^*S, with c a function and S a set, the set obtained from S by acceleration of c, i.e. $c^*S = \bigcup c^i(S)$.
- $N_{St} \in \mathbb{N} \cup \{+\infty\}$ is used in the stop criterion to limit the size of the automaton representing the *Set* computed by the algorithm.
- $N_{Ac} \in \mathbb{N} \cup \{+\infty\}$ is used in the stop criterion to limit the number of accelerations used.

Semi-algorithm 1 FAST

input:
 L : *set of linear functions*,
 S_{init} : *set of initial states*,
 $Stop$: *boolean expression depending on two parameters* N_{St} *and* N_{Ac},
 used as a stop criterion

output:
 returns the reachability set S of (S_{init}, L)

 var :
 Functions $C \leftarrow L$
 int $k \leftarrow 1$
 Set S

 repeat
  $C \leftarrow C \cup C_k(L)$
  $S \leftarrow S_{init}$
  while $(\exists c_1 \in C.\ S \subset c_1^* S) \wedge \neg Stop(N_{St}, N_{Ac})$ **do**
   $S \leftarrow c_1^* S$
   while $\exists c_2 \in C.\ |A_{c_2^* S}| < |A_S|$ **do**
    $S \leftarrow c_2^* S$
   end while
  end while
  $k \leftarrow k + 1$
 until $\neg Stop(N_{St}, N_{Ac})$
 return S

B Case Study

Table 1: Results using an Intel Pentium 933 Mhz with 512 Mbytes

Case study	variables	transitions	time (s)	memory (MB)	n. of states	n. of accelerations	cycle length	n. of cycles
Bounded Petri Nets								
Producer/Consumer	5	3	0.41	2.37	7	3	1	3
Lamport ME	11	9	2.70	2.88	5	11	1	9
Dekker ME	22	22	21.72	5.48	5	36	1	22
RTP	9	12	2.24	2.76	5	8	1	12
Peterson ME	14	12	4.97	3.78	5	12	1	12
Reader/Writer	13	9	9.68	23.14	9	23	1	9
Petri Nets								
CSM	13	13	45.57	6.31	6	32	2	35
FMS	22	20	157.48	8.02	21	23	2	46
Multipoll	17	20	22.96	5.13	35	13	1	20
Kanban	16	16	10.43	6.54	4	2	1	16
Mesh2x2	32	32	$\geq$ 1800	-	-	-	-	-
Mesh3x2	52	54	$\geq$ 1800	-	-	-	-	-
Manufacturing system	7	6	$\geq$ 1800	-	-	-	-	-
Manufacturing system (check deadlock freedom)	13	6	$\geq$ 1800	-	-	-	-	-
PNCSA	31	38	$\geq$ 1800	-	-	-	-	-
extended ReaderWriter	24	22	$\geq$ 1800	-	-	-	-	-
SWIMMING POOL	9	6	111	29.06	30	9	4	47
Petri Nets with Transfer Arcs								
Last-in First-served	17	10	1.89	2.74	9	12	1	10
Esparza-Finkel-Mayr	6	5	0.79	2.55	5	2	1	5
Broadcast Protocols								
Inc/Dec	32	28	$\geq$ 1800	-	-	-	-	-
Producer/Consumer with Java threads - 2	18	14	13.27	3.81	13	53	1	14
Producer/Consumer with Java threads - N	18	14	723.27	12.46	58	86	2	75
2-Producer/2-Consumer with Java threads *2 types of producers and 2 types of consumers*	44	38	$\geq$ 1800	-	-	-	-	-
Central Server system	13	8	20.82	6.83	5	11	2	25
Consistency Protocol	12	8	275	7.35	7	9	3	98
M.E.S.I. Cache Coherence Protocol	4	4	0.42	2.44	6	3	1	4
M.O.E.S.I. Cache Coherence Protocol	4	5	0.56	2.49	7	3	1	5
Synapse Cache Coherence Protocol	3	3	0.30	2.23	6	2	1	3

Case study	variables	transitions	time (s)	memory (MB)	n. of states	n. of accelerations	cycle length	n. of cycles
Broadcast Protocols								
Illinois Cache Coherence Protocol	4	6	0.97	2.64	6	4	1	6
Berkeley Cache Coherence Protocol	4	3	0.49	2.75	7	2	1	3
Firefly Cache Coherence Protocol	4	8	0.86	2.59	7	3	1	8
Dragon Cache Coherence Protocol	5	8	1.42	2.72	6	5	1	8
Futurebus+ Cache Coherence Protocol	9	10	2.19	3.38	12	8	1	10
Others								
lift controller - N	4	5	4.56	2.90	14	4	3	20
bakery	8	20	$\geq$ 1800	-	-	-	-	-
barber m4	8	12	1.92	2.68	5	8	1	12
ticket 2i	6	6	0.88	2.54	22	5	1	6
ticket 3i	8	9	3.77	3.08	77	10	1	9
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

C Tools for symbolic reachability set computation (numeric variables)

	model			acceleration					
	variable type	guards	actions	acceleration	automated cycle search	forward	backward	exact computation	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

Fig. 3. A comparison of different tools for reachability set computation