Electronic Notes in Theoretical Computer Science 68 No. 6 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 16 pages

About Fast and TReX accelerations

Christophe Darlot, Alain Finkel [1,2]

LSV & ENS de Cachan
61,av. du président wilson, 94235 CACHAN CEDEX, France

Laurent Van Begin [3]

Université Libre de Bruxelles
Blvd du Triomphe, 1050 Bruxelles, Belgium

Abstract

FAST and TREX tools are designed to analyse systems with infinite state spaces. They both implement algorithms computing the set of reachable states of such systems. Since the state space may be infinite, acceleration techniques are used . In this paper, we study the differences between FAST and TREX acceleration techniques and show that although FAST remains in Presburger logics while accelerating, TREX can produce 1^{st} order arithmetical formulas even when accelerating functions from states belonging to a subset of Presburger.

Key words: FAST, TREX, Acceleration, Counter systems.

1 Introduction

During the last years, progress in model-checking techniques allowed to apply automatic verification not only to finite state systems but also to infinite ones. In this paper, we are interested in the state space exploration of a special class of infinite state systems called *parametric counter systems*, i.e. finite automata extended with unbounded counters. Such models are widely used to describe, for instance, abstractions of cache coherency protocols [Del00], programs with any number of threads [DRV02] and protocols like the TTP/C [BFL04].

More precisely, our issue is to compute for counter systems the exact (infinite) set of states reachable from a set of initial states in a fully automatic way. Let us recall that achieving this goal allows us to automatically verify safety properties. For this we mainly encounter two problems :

[1] Email: darlot@lsv.ens-cachan.fr
[2] Email: finkel@lsv.ens-cachan.fr
[3] Email: lvbegin@ulb.ac.be

- A suitable data structure to represent infinite sets of states must be defined;
- Since the the computation of the reachable state space for counter systems is not guaranteed to terminate in general, several techniques must be defined to force termination in practice.

The so-called *acceleration technique* is one of the most popular techniques that have been investigated to ensure termination of the reachable state space computation. Roughly speaking, the acceleration techniques consist in computing the exact effect of iterating a control loop of a counter system an arbitrary number of times, hence computing a (possibly) infinite set of states in a finite number of steps (see [Boi99]).

Several acceleration techniques have been developed and implemented into tools. In this paper, we set the focus on the acceleration techniques implemented into the tools FAST [Fast] and TREX [Trex] [4]. The underlying technologies used by these tools are different: FAST uses *Presburger arithmetics* (and *Presburger automata*) to represent infinite set of states while TREX uses *Parameterized Difference Bounded Matrices* [AAB00]. As a consequence, the acceleration techniques are completely different. One aim of this paper is to compare them. First, we have to discuss the expressiveness of both models. Particularly, we compare the symbolic representations used in these tools. Second, the acceleration techniques could compute non-manipulable representations of sets, i.e. representations for which no algorithm can compute some important operations or tests such as inclusion or emptiness. Hence, either a semi-algorithm is used in practice (in this case, the algorithm is not guaranteed to terminatebe) or the test is approximated (then we are not guaranteed to compute the exact set of reachable states). Since the computation of the exact reachable state space allows a finer analysis of counter systems, we focus on the characterization of the cases where we are guaranteed to compute manipulable representations.

The paper is organized as follows: Section 2 introduces the notations and concepts used in the rest of the paper. In section 3, we define and compare the input models and symbolic representations used in both tools. In Section 4, we briefly present in a common framework both acceleration techniques, give some new results on TREX acceleration and benchmark both tools on a common set of examples. Last, we give our conclusion in section 5.

2 Preliminaries

Definition 2.1 Let V be a set of variables such that $\mid V \mid = n$.

- The set of *arithmetical terms* over V is defined by the grammar:
  $t ::= 0 \mid 1 \mid x \mid t - t \mid t + t \mid t * t$ where $x \in V$.

[4] Notice that the tool LASH (see [Lash]) also uses accelerations, but its acceleration technique is close to the one used in FAST, so it will not be discussed in this paper.

- A *Presburger term* is a term where no product of variables occurs.
- The set of 1^{st} *order arithmetical formulas* over V is defined by the grammar $\phi ::= t \leq t \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi$ where t is an arithmetical term over V and $x \in V$.
- A Presburger formula is a 1^{st} order arithmetical formula with only Presburger terms.

Examples:
- $(y = x + x)$, is equivalent to $y = 2 * x$ hence this last formula will be considered as a Presburger formula;
- $(y = x * x)$ is not a Presburger formula;
- $(y = x * x) \wedge (x = 1)$ is not a Presburger formula (nevertheless, there exists an equivalent formula $y = 1$ which is in Presburger but there exists no algorithm to decide wether a genereral arithmetical formula belongs to Presburger arithmetics).

Definition 2.2 A set $\mathcal{L} \subseteq \mathbb{Z}^n$ is linear if there exists $B, V_1, \ldots, V_p \in \mathbb{Z}^n, p \in \mathbb{N}$ such that $\mathcal{L} = (B, V_1, \ldots, V_p) = \{B + \sum_{i=1}^{p} \lambda_i V_i, \lambda_i \in \mathbb{Z}\}$.

Notice that finite unions of linear sets and Presburger formulas are known to represent the same sets [GS66].

In the following, we consider $V = C \cup P$ with $C = \{c_1, \ldots, c_q\}$ a set of q counters, $P = \{p_1, \ldots, p_r\}$ a set of r parameters such that C and P are disjoint ($C \cap P = \emptyset$).

Notation: ϕ will always denote a quantifier-free first order arithmetical formula over P and ψ will always denote a formula $\psi = \bigwedge_{i,j \in 1..q} c_i - c_j \prec t_{ij}$ where $\prec \in \{<, \leq\}$, t_{ij} is an arithmetical term over P.

Definition 2.3 For all formula ψ, where $t_{ij} \in \mathbb{Z}$ for all terms t_{ij}, ψ is a *DBM formula*. $\exists p_1, \ldots, p_r \in P.\psi$ is a *Parametric DBM formula* (PDBM formula). $\exists p_1, \ldots, p_q \in P.(\psi \wedge \phi)$ is an *Extended PDBM formula* (EPDBM formula).

The DBM formulas, usually used to describe zones in hybrid systems, can be encoded using special kind of matrices called Difference Bounded Matrices. These representations allow an efficient (polynomial) implementation of operations such as intersection or empty test in the case of DBM formula. They have been extended in [AAB00] to handle parameters (PDBM) and constrained parameters (EPDBM).

Let M be a matrix, we denote $\langle M \rangle$ its multiplicative monoïd ($\langle M \rangle = \{M^k; k \geq 0\}$).

Definition 2.4 An *affine function* $f : \mathbb{N}^n \to \mathbb{N}^n$ is a tuple $f = \langle M, B, G \rangle$ (sometimes denoted $\langle M_f, B_f, G_f \rangle$) where $M \in \mathbb{N}^n \times \mathbb{N}^n$, $B \in \mathbb{N}^n$, G (called *guard*) is in Presburger and for every $v \in \mathbb{N}^n$, $v \in G \Rightarrow f(v) = M \cdot v + B$.

Lets now define a restriction of affine functions called simple affine functions. Informally, these functions can be seen as a set of counter assignments of the form $c_i := c_j + t$ where $c_i, c_j \in C$ and t is a Presburger term over P.

Definition 2.5 An affine function $f = \langle M, B, G \rangle$ is *simple* if G is a PDBM formula and $M = \begin{pmatrix} M_C & M_P \\ 0 & Id \end{pmatrix}$ with $M_P \in \mathbb{N}^q \times \mathbb{N}^r$ and $M_C \in \mathbb{N}^q \times \mathbb{N}^q$ is such that each line contains exactly one "1" (all other elements equals to 0).

3 Models and Symbolic Representations

Models

Definition 3.1 A *parametric counter system* (PCS) is a tuple $S = \langle Q, C, P, \mathcal{F}, R, \mathcal{I} \rangle$ where Q is a finite set of locations, C a set of counters, P a set of parameters, $\mathcal{F}$ a set of affine functions, $R \subseteq Q \times \mathcal{F} \times Q$ is a finite set of symbolic transitions and $\mathcal{I}$ is a tuple $\langle q \in Q, I \subseteq \mathbb{N}^n \rangle$ that describes the initial state with a control state and a set of valuations for the variables in $C \cup P$.

FAST and TREX both analyze PCS but the functions of $\mathcal{F}$ are affine functions for FAST and simple affine functions for TREX. Moreover, I is described by a Presburger formula in FAST and by a PDBM formula in TREX. Figure 1 shows an example of PCS with three counters x, y, z and no parameters (all variables are assigned at least once).

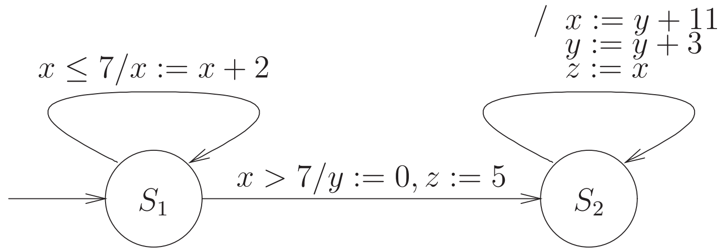

Fig. 1. The Graph of a Parametric Counter System.

The semantics of PCS is defined as follows. A configuration of the PCS $S = \langle Q, C, P, \mathcal{F}, R, \mathcal{I} \rangle$ is a pair $\langle q, v \rangle$ where $q \in Q$ and $v \in \mathbb{N}^m$ is a valuation. The transition relation $\rightarrow$ between the configurations of S is defined as follows: given two configurations $\langle s_1, v_1 \rangle$ and $\langle s_2, v_2 \rangle$ and a function $f \in \mathcal{F}$, we have $\langle \langle q_1, v_1 \rangle, f, \langle q_2, v_2 \rangle \rangle \in \rightarrow$ if and only if there exist $\langle q_1, f, q_2 \rangle \in R$ such that $v_2 = f(v_1)$.

Symbolic Representations

While analyzing parametric counter systems, the possibly infinite set of configurations is represented using different symbolic representations:

FAST algorithm uses Presburger formulas over $V = C \cup P$ (there is no distinction between parameters and counters). FAST uses automaton to encode and efficiently operate on Presburger formulas (see [Boi99] for further details). However, the encoding of a Presburger formula into an automata is non-elementary and the reverse operation is not possible in the general case.

TREX algorithm uses EPDBM formulas over the counters C and the parameters P. The set $N \subseteq P$ of integer variables called *iteration variables* in [AAB00], is used while accelerating in order to set constraints on the number of iterations of the control loops. EPDBM formulas $\varphi = \psi \wedge \phi$ are encoded using both symbolic matrices (for ψ) and logical formulas[5] (for ϕ). The encoding and decoding of PDBM formulas from and to matrices can be done in linear time. Since we here only consider PCS where the initial state is described by a set of assignments $\{c_i := t_i\}$, both terms $c_i - c_j \leq t_{i,j}$ and $c_j - c_i \leq -t_{i,j}$ always appear in the EPDM formulas. For this reason, we can restrict their syntax to the following simpler form: $\bigwedge_{i \in 1..q} c_i = t_i$ where t_i is an arithmetical term over P.

Now, lets compare the expressive power of these symbolic representations. In order to ease the comparison of both tools, we define a new type of formula called *Presburger PDBM* which is basically an EPDBM formula where no product of variables occurs.

Definition 3.2 A *Presburger PDBM formula* is a formula $\exists p_1, \dots, p_n \in P.(\psi \wedge \phi_l)$ where ϕ_l is a quantifier free Presburger formula over the parameters in P.

Proposition 3.3 *PDBM formulas are strictly less expressive than Presburger formulas.*

Proof. This result is obvious because:

(i) Every PDBM formula is syntactically a Presburger formula;

(ii) There exist Presburger formulas that cannot be expressed by a PDBM formula: i.e. the Presburger formula $x \neq y$ cannot be expressed by a PDBM formula since PDBM formula are known to describe convex sets. □

Proposition 3.4 *Presburger PDBM formulas and Presburger formulas are equivalently expressive.*

Proof.

(i) Every Presburger PDBM formula is syntactically a Presburger formula;

(ii) First, we show how to encode a linear set with a Presburger PDBM formula. Let $\mathcal{L} = (B, \{V_1, \dots, V_r\})$ be a linear set with a base $B = (b_1, \dots, b_n)$ and a set of periods $V_i = (v_{i_1}, \dots, v_{i_n})$.

Let us consider the sets of counters $C = \{c_1, \dots, c_n\}$ and the set of parameters $P = \{p_1, \dots, p_{n+r}\}$. We define $\psi \equiv \bigwedge_{i=1}^{n} (c_i = d_i)$ and $\phi =$

[5] TREX uses an external prover (*Omega*) to manipulate these formulas.

$\bigwedge_{i=1}^{n}(d_i = b_i + \sum_{k=1}^{r}(p_{n+k} \cdot v_{k_i}))$. Then the Presburger PDBM $\psi \wedge \phi$ defines (syntactically) the linear set $\mathcal{L}$ (since parameters are existentially quantified in PDBM formulas).

Second, we show how to encode a finite union of linear sets into a Presburger PDBM. The extension is obvious. Given the union of m linear sets $\bigcup_{j=1}^{m} \mathcal{L}^{(j)}$ with $\mathcal{L}^{(j)} = (B^{(j)}, \{V_1^{(j)}, \ldots, V_p^{(j)}\})$, we define ϕ' as follows:

$$\phi' = \bigvee_{j=1}^{q} \bigwedge_{i=1}^{n} (d_i = b_i^{(j)} + \sum_{k=1}^{r} (p_{n+k} \cdot v_{k_i}^{(j)})$$

The Presburger PDBM $\psi \wedge \phi'$ defines the finite union of linear sets. So we proved that any finite union of linear sets can be encoded by a Presburger PDBM. Then, Presburger is included in Presburger PDBM.

□

Proposition 3.5 *EPDBM formulas are strictly more expressive than Presburger formulas.*

Proof. The EPDBM formula $\varphi = \psi \wedge \phi$ with $\psi \equiv (c_1 = p_1 \wedge c_2 = p_2)$ and $\phi \equiv (p_1 = p_2 * p_2)$ represents the set$\{(c_1, c_2) \mid c_1 = c_2 * c_2\}$ which is not a Presburger set. □

From the previous proposition, we can give the inclusion diagram given in figure 2 which sums up the expressiveness of the different symbolic representations used in the tools FAST and TREX.

$$\begin{tikzcd}
(\text{TREX}) & DBM \arrow[r] & PDBM \arrow[r] & Presburger PDBM \arrow[r] \arrow[d, equal] & EPDBM \\
(\text{FAST}) & & & Presburger\ Formula &
\end{tikzcd}$$

Fig. 2. Inclusion Diagram of FAST and TREX Symbolic Representations

4 Study of FAST and TREX Acceleration Techniques

We call acceleration the process which given a function f and a set of states S returns the (exact) set of states reached by applying f an arbitrary number of times from S. More formally,

$$Accelerate(S, f) = f^*(S) = \{s' \mid \exists n \in \mathbb{N}, s \in S \wedge s' = f^n(s)\}.$$

In this section, we briefly describe the respective algorithms of FAST and TREX used to compute the reachable state space of a PCS. In each case, we detail the acceleration technique and we expose the context (main algorithm) in which it is used. More details about those algorithms can be found, respectively, in [FL02] and [AAB00].

In the rest of the paper, we consider that the locations of a PCS are represented by a counter l, i.e. $l = i$ means that the PCS is in the location s_i. So transitions of a PCS are viewed as (simple) affine functions. Let us say that a PCS is flat if its control graph contains no nested loops.

4.1 Fast *Acceleration Algorithm*

The main algorithm of Fast is a classical forward fix-point algorithm extended with acceleration techniques. More precisely, given a PCS $S = \langle Q, C, P, \mathcal{F}, R, \mathcal{I} \rangle$ and a natural number k, the algorithm works as follows. The (infinite) set of valuations that we want to compute is represented by a Presburger formula F. Two steps are performed:

(i) *Initialization:* First, F is defined to be a Presburger formula describing the set of initial configurations of the PCS. Moreover, the set $\mathcal{T}$ of all the possible compositions of functions in R of size lesser or equal to k is computed;

(ii) *Iteration:* Then, while a fix-point is not reached, a function $f \in \mathcal{T}$ is randomly chosen and $F \leftarrow F \cup \mathit{Accelerate}_{\mathit{Fast}}(F, f)$ is computed.

Notice that some heuristics are implemented in Fast to increase on the fly the value of k if necessary to reach a fix-point (see [BFLP03]).

Fast acceleration technique is based on the fact that we can represent the relation corresponding to any number of applications of an affine function f by a Presburger formula *provided* $\langle M_f \rangle$ *is finite*. Notice that deciding whether the monoïd of an affine function is finite is decidable [MS77].

Now we recall how to express f^* as a Presburger formula. We denote $\overline{f}$ the function f where G_f is substituted by *true*. Since $\langle M_f \rangle$ is finite, there exist two integers $a \geq 1$ and $b \geq 1$ such that ${M_f}^a = {M_f}^{a+b}$. The formula $v' = \overline{f}^i(v) \wedge i \geq 0$ with the free variables v', v and i is equivalent to the following Presburger formula (where $\mathbf{0}$ is the vector where all dimensions equal to 0):

$$\left(\bigvee_{r=0}^{a-1} [(v' = \overline{f}^r(v)) \wedge (i = r)] \right) \vee \left(\bigvee_{r=0}^{b-1} \exists n \geq 0 \begin{bmatrix} \wedge (v' = \overline{f}^{a+r}(v) + n.{M_f}^{a+r}.\overline{f}^b(\mathbf{0})) \\ \wedge (i = a + r + n.b) \end{bmatrix} \right)$$

We define the acceleration formula $A(f, F, a, b)$ which is a Presburger formula with a free variable v' and which takes the guards into account:

$$A(f, F, a, b) = \exists v, k.(v \in F \wedge k \geq 0 \wedge v' = \overline{f}^k(v) \wedge (\forall i.0 \leq i < k \Rightarrow \overline{f}^i(v) \in G_f)) \tag{1}$$

Theorem 4.1 **(**Fast **correctness** **[FL02])** *Let $a, b \in \mathbb{N}$, f an affine function such that $\langle M_f \rangle$ is finite and $M^a = M^{a+b}$ then $A(f, F, a, b) = f^*(F)$.*

Moreover, in the case where the function is a translation and the guard a convex polyhedron, [BFL04] gives a new more efficient acceleration algorithm

Algorithm 1: Acceleration of FAST

function $Accelerate_{Fast}$ (F: a Presburger formula, f: an affine function): Presburger formula;

if $\langle M_f \rangle$ *is finite* **then**

  Compute a and b such that ${M_f}^a = {M_f}^{a+b}$;

  return $A(f, F, a, b)$;

(where Formula 1 is simpler). We call this acceleration $Accelerate_{ConvexFast}$. Also notice that fast always termintate on flat PCS.

4.2 TREX *Acceleration Algorithm*

In order to compute the reachable state space of a PCS $S = \langle Q, C, P, \mathcal{F}, R, \mathcal{I} \rangle$, TREX uses an in-depth first construction algorithm of a symbolic reachability tree, i.e. a tree where each node is an EPDBM formula –the root node is an EPDBM formula representing the set of initial states, and edges describe the reachability relation between nodes. This tree construction is extended with an acceleration technique (described further). During the construction of the symbolic reachability tree denoted $SRT_{TReX}(S)$, each node (labelled by a tuple $\langle q \in Q, \varphi \in EPDM \rangle$) is treated in two steps:

(i) *Acceleration computation:* First, TREX tries to perform accelerations on the loops of the control graph. In order to detect the loops, TREX looks back for a symbolic state φ' labelled by the location q and from which the current node is reachable in the symbolic reachability tree. If such a node is detected, the acceleration function is called. Notice that TREX compaoses the functions of the loop to obtain a single function. If a new EPDBM formula is created, the in-depth first exploration resumes from this formula;

(ii) *Direct successor computation:* Second, for each transition $t \in R$, the set of configurations reached by firing t from the current node and denoted $\mathsf{post}_f(\varphi)$ is computed. This set is described by a (set of) EPDBM formula. For each formula a new node is created and the in-depth first exploration resumes recursively from these formulas;

Notice that the in-depth first explotation of the reachability tree allows to discover on-the-fly the loops of the PCS and to compose the functions to be accelerated. However, the size of the detected loops is bounded in TREX tool.

Given two EPDBM formulas $\psi_1 = \bigwedge_{i \in 1..q} c_i = t_i$ and $\psi_2 = \bigwedge_{i \in 1..q} c_i = k_i$, we define the following operations:

$$\psi_2 \dot{-} \psi_1 = \bigwedge_{i \in 1..q} c_i = k_i - t_i, \qquad \psi_1 \dot{+} \psi_2 = \bigwedge_{i \in 1..q} c_i = k_i + t_i$$

$$n(\in N)\dot{\times}\psi_1 = \bigwedge_{i\in 1..q} c_i = n \cdot t_i, \qquad \psi_1 \dot{=} \psi_2 = \bigwedge_{i\in 1..q} k_i = t_i.$$

The function $Accelerate_{TReX}$ (Algorithm 2) works as follows: After one

Algorithm 2: Acceleration of TReX

function $Accelerate_{TReX}$ $((\psi_0 \wedge \phi_0)$:EPDBM formula,f: affine function): EPDBM formula;
if *f is a simple affine function* **then**
  $(\psi_1 \wedge \phi_1) \leftarrow \mathsf{post}_f(\psi_0 \wedge \phi_0)$;
  $(\psi_2 \wedge \phi_2) \leftarrow \mathsf{post}_f^2(\psi_0 \wedge \phi_0)$;
  if $(\psi_1 \wedge \phi_1) \neq false \wedge (\psi_2 \wedge \phi_2) \neq false$ **then**
    $\Delta \leftarrow \psi_1 \dot{-} \psi_0$ /* effect of applying f from φ_0 */ ;
    $\Delta' \leftarrow \psi_2 \dot{-} \psi_1$ /* effect of applying f from φ_1 */ ;
    Let $n \in N$ be a fresh iteration variable;
    if $C1$ *[and $C2$]* **then**
      return $\mathsf{post}_f((\psi_0 \dot{+} (n \dot{\times} \Delta_1)) \wedge \phi_0 \wedge n \geq 0)$;

iteration of a control loop f, we iterate it a second time[6] and compare the effect Δ_1 of the first iteration with the effect Δ_2 of the second iteration. For this, two conditions are verified:

$$\begin{aligned} C1 \equiv & \forall p_1, \ldots, p_{\alpha_p} \in P.\phi_2 \Longrightarrow \Delta \dot{=} \Delta' \\ C2 \equiv & \forall n \geq 0.\mathsf{post}_f^2(\psi_0 \dot{+} (n \dot{\times} \Delta)) \wedge \phi_0 \wedge n \geq 0) = \\ & \quad \mathsf{post}_f(\psi_0 \dot{+} ((n+1) \dot{\times} \Delta)) \wedge \phi_0 \wedge n \geq 0) \end{aligned}$$

Informally, $\Delta_1 = \psi_1 \dot{-} \psi_0$ represents a distance between the symbolic states ψ_1 and ψ_0, which is the *effect* of the loop. $C1$ says that the effect of the first and second iteration of the loop are equal under the assumption ϕ_2 and $C2$ says that each time we apply f, we add Δ to the symbolic state. If $C1 \wedge C2$ is satisfied, we return a symbolic state expressing that if we iterate n times the loop f from φ_0, we obtain the effect given by $n \dot{\times} \Delta_1$. Notice that the computation of $\mathsf{post}_f((\psi_0 \dot{+} (n \dot{\times} \Delta)) \wedge \phi_0 \wedge n \geq 0)$ sets constraints on n (using the guard of the function f). The correctness of this algorithm is given by the following theorem:

Theorem 4.2 (TReX correctness [AAB00]) *$C2$ is equivalent to*

$$\forall n.\mathsf{post}_f^{n+1}(\psi_0 \wedge \phi_0) = \mathsf{post}_f((\psi_0 \dot{+} (n \dot{\times} \Delta)) \wedge \phi_0 \wedge n \geq 0)$$

[6] We give here a simplified version of the algorithm where applying post generates a single EPDBM formula but it can be easily extended in the case of post generating several EPDBM formula (this may happen after the EPDBM normalisation process, not described here).

We can notice that if we check only $C1$, we compute an upper-approximation of the reachability set. In the implementation of the tool TReX only $C1$ is verified but in [AAB00], the authors claim that the acceleration is still exact because the guards describe convex sets and accelerated simple affine functions are translations.

4.3 Analysis of TReX *acceleration technique*

We detail in this section the cases where the TReX acceleration technique can be used and the kind of formula obtained once the acceleration performed.

First, contrary to the case of Fast acceleration technique, the TReX acceleration technique allows to accelerate any simple affine functions without restriction about its monoïd:

Proposition 4.3 *Given a PCS $S = \langle Q, C, P, \mathcal{F}, R, \mathcal{I}\rangle$ and an initial state φ_0, for each infinite path $\varphi_0 \rightarrow^{f^\omega}$ ($f \in \mathcal{F}$) of $SRT_{TReX}(S)$, there exist two integers a and b such that $Accelerate_{TReX}(f^a(\varphi_0), f^b)$ will return $(f^b)^*(f^a(\varphi_0))$.*

Proof. See appendix. □

Notice that this result is not effective since the acceleration test ($C1$), the inclusion test and the empty test of TReX are undecidable. Nevertheless, if the PCS does not contain non-Presburger constraints on parameters in its initial state, i.e. if its initial state is described by a Presburger PDBM formula, the acceleration test $C1$ will be effectively evaluated to *true* at least once until the first acceleration occurs (since its result may introduce non-Presburger terms as explained in next paragraph). Moreover, since the acceleration test $C1$ is undecidable, the acceleration procedure of TReX is not recursive. However, if we suppose we have an oracle for deciding the empty test, the inclusion test and $C1$, TReX terminates on flat systems.

Remark 4.4 *It seems that* TReX *does not terminate in general even if we suppose we have an oracle (for example when used on a Petri net example of [DFS98] where the language of the transitions labels is not regular).*

Second, TReX acceleration result is in the general case a first-order formula, even when accelerating a function with a finite monoïd from a Presburger PDBM (see Proposition 4.5). Nevertheless there exists in that case a Presburger formula equivalent to the resulting first-order formula. Moreover, TReX always produces first-order formulas (even when accelerating from a Presburger formula) when applied to functions with an infinite monoïd for some initial states (see Proposition 4.6).

Proposition 4.5 *There exist a DBM formula φ and a function f such that $\langle M_f\rangle$ is finite and $Accelerate_{TReX}(\varphi, f)$ is not a Presburger PDBM formula.*

Proof. Let us consider the PCS shown in Figure 1 with the initial symbolic state $(s_1, \psi \wedge \phi)$ with $\psi = x = 0 \wedge y = 0 \wedge z = 0$ and $\phi = true$ (ψ is a simple

DBM because $\phi = true$). By applying the TReX procedure on that PCS, we build successively the following EPDBM formulas :

formula	ψ_i	ϕ_i
φ_0	$l = 1 \wedge x = 0 \wedge y = 0 \wedge z = 0$	$true$
φ_1	$l = 1 \wedge x = 2 \wedge y = 0 \wedge z = 0$	$true$
φ_2	$l = 1 \wedge x = 4 \wedge y = 0 \wedge z = 0$	$true$
φ_3 (Acceleration)	$l = 1 \wedge x = (2 + 2 \cdot n_0) \wedge y = 0 \wedge z = 0$	$0 \leq n_0 < 3$
φ_4	$l = 2 \wedge x = (2 + 2 \cdot n_0) \wedge y = 0 \wedge z = 5$	$n_0 = 3$
φ_5	$l = 2 \wedge x = 11 \wedge y = 3 \wedge z = (2 + 2 \cdot n_0)$	$n_0 = 3$
φ_6	$l = 2 \wedge x = 14 \wedge y = 6 \wedge z = 11$	$n_0 = 3$
φ_7 (Acceleration)	$l = 2 \wedge x = (11 + 3 * n_1) \wedge y = (3 + 3 \cdot n_1)$ $\wedge z = (2 + 2 \cdot n_0 - 2 \cdot n_0 \cdot n_1 + 9 \cdot n_1)$	$n_0 = 3 \wedge n_1 \geq 0$

First, TReX accelerates while looping on the location S_1 (steps 0-2) and generates the symbolic state φ_3 (which is a Presburger PDBM). Then, when looping on the location S_2 (steps 4-6), $Accelerate_{TReX}$ computes:

(i) $\Delta = \psi_5 \dot{-} \psi_4 = (\Delta_l = 2 - 2 \wedge \Delta_x = 11 - (2 + 2 \cdot n_0) \wedge \Delta_y = 3 - 0 \wedge \Delta_z = (2 + 2 \cdot n_0) - 5)$ which is equivalent to $\Delta_l = 0 \wedge \Delta_x = 9 - 2 \cdot n_0 \wedge \Delta_y = 3 \wedge \Delta_z = 2 \cdot n_0 - 3$, and

(ii) $\Delta' = \psi_6 \dot{-} \psi_5 = (\Delta'_l = 2-2 \wedge \Delta'_x = 14-11 \wedge \Delta'_y = 6-3 \wedge \Delta'_z = 11-(2+2 \cdot n_0)$ which is equivalent to $\Delta'_l = 0 \wedge \Delta'_x = 3 \wedge \Delta'_y = 3 \wedge \Delta'_z = 9 - 2 \cdot n_0$.

The acceleration test ($C1$, line 9 of Algorithm 2) $\forall n_0 . \phi_5 \Rightarrow \Delta \doteq \Delta'$ succeeds since $\forall n_0 . n_0 = 3 \Rightarrow ((9-2 \cdot n_0 = 3) \wedge (3 = 3) \wedge (2 \cdot n_0 - 3 = 9 - 2 \cdot n_0)$ is true. Then the state returned (line 10 of Algorithm 2) is $\varphi_7 = \mathsf{post}_f(\psi_5 \dot{+} n_1 \dot{\times} \Delta \wedge \phi_5 \wedge n_1 \geq 0)$ which contains the non-Presburger terms "$2 \cdot n_0 \cdot n_1$". Thus the result is not a Presburger formula. □

Proposition 4.6 *For any function f such that $\langle M_f \rangle$ is infinite and $G_f = true$, there exists a Presburger PDBM formula F such that $Accelerate_{TReX}(F, f)$ is not a Presburger PDBM formula.*

Proof. See appendix. □

4.4 A few benchmarks on parametric counter systems

Figure 3 gives a few results on classical examples of counter systems, taken from [Fast]. "-"means that no result is given within a reasonable time (about half an hour). These benchmarks have been performed on a Xeon 2.4Ghz with 1Gb RAM running Linux 2.4. Out of 31 models handled by Fast, only 14 could be translated to TReX entry model (PDBM guards or simple affine functions are too restrictive for some models).

Example	$\mid C \mid$	$\mid P \mid$	t(s) Fast	t(s) TReX
Barber	8	0	1.0	0.1
rtp	9	0	1.1	0.1
Lamport	11	0	1.3	0.2
Peterson	14	0	2.3	0.5
Read-Write	13	0	4.6	0.9
Kanban	16	0	4.8	0.4
Dekker	22	0	9.9	7.5
Producer-Consumer	5	1	0.2	4.6
Ticket2i	6	3	0.5	134.2
Ticket3i	8	4	1.7	-
ICALP 98 Petri net	4	1	6.2	-
Multipoll	18	4	6.7	-
CSM	14	1	21.4	-
FMS	22	3	75.0	-
Swimming Pool	7	2	101.0	-

Fig. 3. Benchmarks on classical systems

We can notice that when applied on systems without parameters, TReX is faster than Fast. A possible reason is that simple operations on DBM (such as empty test or inclusion) can be done in polynomial time. However, even with one parameter, TReX is much slower than Fast; The reason of this inefficiency may be the call of external provers (Omega and Reduce) to handle parameters, the normalization of the EPDBM which is a costly operation (performed by case splitting on parameters values), the kind of specifications used (their control graph mostly consists of one location with many functions looping on it, which may cause TReX not to be as efficient as for specifications with more important control graphs).

5 Conclusion

Figure 4 summaries the results given in the previous sections. The advantage of the TReX acceleration technique is that the functions with a finite or infinite monoïd can be sometimes accelerated. Moreover, the sequences of transitions to accelerate are easily chosen by exploring the reachability tree constructed by the TReX algorithm –in the case of Fast, such heuristics are more difficult to define[7], and a data structure especially designed to represent EPDBM formulas have been defined in [AAB00]. The drawback of this technique is that contrary to the case of Fast acceleration, we are not guaranteed to obtain Presburger formulas when applying the acceleration technique. When manipulating non-Presburger formulas, the satisfiability for example of $C1$ becomes undecidable and TReX is then not guaranteed to compute the exact set of reachable states in general. Moreover, the benchmarks showed that TReX could not handle easily models with many parameters, which is a problem since each acceleration step increase the number of parameters (in the set N of iteration variables). So (E)PDBM do not seem to be a suitable symbolic representations for the analysis of PCS.

	Fast	TReX
Guards	Presburger	PDBM
Functions	Affine functions with finite monoïd	Simple affine functions
Symb. Rep.	Presburger	EPDBM
$\subseteq$ of Symb. Rep.	3-EXP	Undecidable
Acceleration	5-EXP	Undecidable

Fig. 4. Fast and TReX Accelerations

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 419–434. Springer, 2000.

[BFL04] S. Bardin, A. Finkel and J. Leroux FASTer acceleration of counter automata in practice. will appear in Proc. *10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2004)*, 2004.

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast Acceleration of Symbolic Transition systems. In Proc. *15th Int. Conf. Computer Aided Verification (CAV'2003)*, volume 2725 of Lecture Notes in Computer Science, pages 118-121. Springer, 2003.

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1999.

[7] They are actually based on criteria such as the maximum symbolic states sizes allowed which are *a priori* not easy for the user to guess from a given PCS

[Del00] G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In Proc. *12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *LNCS*, pages 53–68. Springer, 2000.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.

[DRV02] G. Delzanno, J-F. Raskin, and L. Van Begin. Towards the Automated Verification of Multithreaded Java Programs. In Proc. *8th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *LNCS*, pages 173–187. Springer, 2002.

[Fast] FAST web. `http://www.lsv.ens-cachan.fr/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. (FSTTCS'2002)*, volume 2556 of Lecture Notes in Computer Science, pages 145-156. Springer, 2002.

[GS66] s. Ginsburg and E. Spanier Semigroups, Presburger formulas and languages. Pacific Journal of Mathematics, volume 16, 1966.

[Lash] LASH web. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[MS77] A. Mandel and I. Simon. On finite semi-groups of matrices. Theoritical Computer Science, 5(2):101–111, October 1977.

[Trex] TRex web. `http://www.liafa.jussieu.fr/~sighirea/trex/`.

A Proofs

Proof. [Proposition 4.3] Let $f(X) = M.X + B$ be a simple affine function (notice that $\langle M_f \rangle$ may be infinite) such that f is a loop of the control graph. Let X_C and X_P be the respective vectors of variables and parameters defined as follows:

$$X_C = \begin{pmatrix} c_1 \\ \dots \\ c_{|C|} \end{pmatrix}, \quad X_P = \begin{pmatrix} p_1 \\ \dots \\ p_{|P|} \end{pmatrix}$$

where $p_1, \dots, p_{|P|} \in P$ and $c_1, \dots, c_{|C|} \in C$. We are only interested in computing the counters values (since parameters values never change), thus let's consider the projection of f on C (called f_C) $f_C(X_C) = M_C.X_C + B'$ with $B' = M_P.X_P + B$ which is a constant term.

Notice that M_C is a matrix such that each line contains at most one 1, others values being 0, thus the monoïd of M_C is obviously finite and then $\exists a, b. M_C^{a+b} = M_C^b$.

Let $\varphi_0 \rightarrow^f \ldots \rightarrow^f \ldots$ be an infinite path of the symbolic transition system built by TReX. When TReX reaches $f^{a+b}(\varphi_0)$, it will try to accelerate f^b from $f^{a+b}(\varphi_0)$ (because it composes the functions that loops on a control state).

By hypotheses, $f^{a+2b}(\varphi_0)$ is defined (because on an infinite path). We can easily show that $f^n_C(x) = M^n_C.x + \sum_{k=1}^n M_C^{k-1}B'$. Let us calculate $(\Delta' - \Delta)$ with $\Delta = f_C^{a+b}(\varphi_0) - f_C^a(\varphi_0)$ and $\Delta' = f_C^{a+2b}(\varphi_0) - f_C^{a+b}(\varphi_0)$:

$$\begin{aligned} f_C^{a+b}(\varphi_0) - f_C^a(\varphi_0) &= (M_C^{a+b}.\varphi_0 + \sum_{k=1}^{a+b} M_C^{k-1}.B') - (M_C^a.\varphi_0 + \sum_{k=1}^{a} M_C^{k-1}.B') \\ &= (M_C^{a+b}.\varphi_0 - M_C^a.\varphi_0) + (\sum_{k=1}^{a+b} M_C^{k-1}.B' - \sum_{k=1}^{a} M_C^{k-1}.B') \\ &= 0 + \sum_{k=a+1}^{a+b} M_C^{k-1}.B' \text{ (because} M_C^{a+b} = M_C^b) \end{aligned}$$

$$\begin{aligned} f_C^{a+2b}(\varphi_0) - f_C^{a+b}(\varphi_0) &= (M_C^{a+2b}.\varphi_0 + \sum_{k=1}^{a+2b} M_C^{k-1}.B') - (M_C^{a+b}.\varphi_0 + \sum_{k=1}^{a+b} M_C^{k-1}.B') \\ &= (M_C^{a+2b}.\varphi_0 - M_C^{a+b}.\varphi_0) + (\sum_{k=1}^{a+2b} M_C^{k-1}.B' - \sum_{k=1}^{a+b} M_C^{k-1}.B') \\ &= 0 + \sum_{k=a+b+1}^{a+2b} M_C^{k-1}.B' \text{ (because} M_C^{a+b} = M_C^b) \end{aligned}$$

$$\begin{aligned} \Delta' - \Delta &= \sum_{k=a+b+1}^{a+2b} M_C^{k-1}.B' - \sum_{k=a+1}^{a+b} M_C^{k-1}.B' \\ &= \sum_{k=a+b+1}^{a+2b} M_C^{k-1}.B' - \sum_{k=a+b+1}^{a+2b} M_C^{k-1}.B' \text{ (because} M_C^{a+b} = M^b) \\ &= 0 \end{aligned}$$

So $\Delta = \Delta'$ is true. Moreover, the formula $\forall P, N.\phi \Longrightarrow \Delta = \Delta'$ is also true (whatever the value of ϕ !), so $C1$ is satisfied and TReX acceleration procedure returns (at worst) a first order formula. □

Proof. [Proposition 4.6] We call $\mathsf{Term}(\psi)$ the set of terms that occur in ψ. Suppose that the theorem does not hold and there exists a function f with an infinite monoïd, three EPDBM formulas $\varphi_1 = \psi_1 \wedge \phi_1$, $\varphi_2 = \psi_2 \wedge \phi_2$ and $\varphi_3 = \psi_3 \wedge \phi_3$ and a symbolic tree constructed by TReX such that we have: (i) a path from φ_1 to φ_2 in the tree corresponding to a control loop where f corresponds to its effect on the variables, and (ii) a path from φ_2 to φ_3 corresponding the the same control loop.

Moreover, we have $\Delta = \psi_2 \dot{-} \psi_1$ with $\mathsf{Term}(\Delta) \subseteq \mathbb{Z}$, $\Delta' = \psi_3 \dot{-} \psi_2$ and $\phi_3 \Rightarrow \Delta \doteq \Delta'$. Let $x_1 \in C$ such that the line corresponding to x_1 in the matrices of $\langle M_f \rangle$ is unbounded ($\neg(\exists n \in \mathbb{N} : \forall M \in \langle M_f \rangle : \sum_{x' \in C \cup P} \mid M_{x_1 x'} \mid \leq n)$). Let $x_1 := x_2 + t_1$ be the simple assignment of x_1 corresponding to the effect of f on x_1. We consider two cases:

(i) If there exist $x_2, \ldots, x_{n+1} \in C$ such that $x_j := x_{j+1} + t_j$ is the simple assignment corresponding to the effect of f on x_j, for j such that $2 \leq j \leq n$ and $x_{n+1} = x_i$ with i such that $1 \leq i \leq n$.

Suppose that $\psi_1 = \bigwedge_{x_1, x_2 \in C} x_1 - x_2 \leq t_{x_1 x_2}$. We have that $t_{0x_j} \equiv t_{0x_{j+1}} + t_j - \Delta_{0x_j}$ for j such that $i \leq j \leq n$. It follows that $t_{0x_i} \equiv t_{0x_i} + t_i + \ldots + t_n - \Delta_{0x_i} - \ldots - \Delta_{0x_n}$, hence $t_i + \ldots + t_n \equiv d$ where $d = \Delta_{0x_i} + \ldots + \Delta_{0x_n}$. Since $\Delta_{0x_j} \in \mathbb{Z}$ ($i \leq j \leq n$), we have that $t_i, \ldots, t_n \in \mathbb{Z}$. Hence the effect of $f^{n+k \cdot j}$ with $0 \leq k \leq n-i$ and $j \geq 1$ on x_1 corresponds to a simple assignment of the form $x_1 := x_{n-k} + \sum_{l=1 \ldots i-1} t_l + j \cdot d_k$ for some $d_k \in \mathbb{Z}$. We conclude that, for a fixed k, the line corresponding to x in the matrices of the functions $f^{n+k \cdot j}$ is the same for all $j \geq 1$, hence the line corresponding to x in the matrices of $\langle M_f \rangle$ in bounded.

(ii) Otherwise, suppose that $x_2 := x_3 + t_2, \ldots, x_{n-1} := x_n + t_{n-1}$ are the simple assignments corresponding to the effect of f on $x_2, \ldots, x_{n-1}$ with $x_1 \neq x_2 \neq \ldots \neq x_{n-1} \neq x_n$ and

(a) either $x_n := x_n + t_n$ is the simple assignment corresponding to the effect of f on x_n. We consider two cases.

$t_n \in \mathbb{Z}$. In that case, we have that $x_n := x_n + (n' - n + 2) \cdot t_n + t_1 + \ldots + t_{n-1}$ is the simple assignment corresponding to the effect of $f^{n'}$ on x_1 for all $n' \geq n$. As a consequence, all the matrices corresponding to the functions $f^{n'}$ with $n' \geq n$ have the same line corresponding to x, hence the line corresponding to x in the matrices of $\langle M_f \rangle$ is bounded.

$t_n \notin \mathbb{Z}$. We have $t_{0x_n} \equiv t_{0x_n} + t_n - \Delta_{0x_n}$, hence $t_n \equiv \Delta_{0x_n}$. However, since $\Delta_{0x_n} \in \mathbb{Z}$, we have $t_n \not\equiv \Delta_{0x_n}$.

(b) or $x_n := t_n$ is the simple assignment corresponding to the effect of f on x_n. In that case, we have that $x_n := t_1 + \ldots + t_n$ is the simple assignment corresponding to the effect of $f^{n'}$ on x_1 for all $n' \geq n$. As a consequence, all the matrices corresponding to the functions $f^{n'}$ with $n' \geq n$ have the same line corresponding to x, hence the line corresponding to x in the matrices of $\langle M_f \rangle$ is bounded.

In all the cases, we conclude to a contradiction. $\square$