Flat acceleration in symbolic model checking

Sébastien Bardin[1], Alain Finkel[1], Jérôme Leroux[2], and Philippe Schnoebelen[1]

[1] LSV: ENS de Cachan & CNRS UMR 8643,
61, av. Pdt. Wilson, 94235 Cachan Cedex, France.
{bardin|finkel|phs}@lsv.ens-cachan.fr
[2] IRISA, Vertecs project, INRIA,
Campus de Beaulieu, 35042 Rennes Cedex, France.
jleroux@irisa.fr

Abstract. Symbolic model-checking provides partially effective verification procedures that can handle systems with an infinite state space. So-called "acceleration techniques" enhance the convergence of fixpoint computations by computing the transitive closure of some transitions. In this paper we develop a new framework for symbolic model-checking with accelerations. We also propose and analyze new symbolic algorithms using accelerations to compute reachability sets.

Key words: verification of infinite-state systems, symbolic model checking, acceleration.

1 Introduction

Context. The development of model checking techniques [18] for infinite-state systems is now an active field of research. These techniques allow considering models like pushdown systems [12], channel systems [1, 13], counter systems [7, 29, 37], and many other versatile families of models. Such models are very expressive and often lead to undecidable verification problems. This did not deter several research teams from developing powerful innovative model checkers for infinite-state systems. For example, tools for checking reachability properties of counter systems are ALV [6], BRAIN [36], LASH [32], MONA [31], TREX [3], and our own FAST [7]. For infinite-state systems, model checking must be "symbolic" since one manipulates (symbolic representations of) potentially infinite sets of configurations. The most popular symbolic representations are based on regular languages: these are quite expressive and automata-theoretical data structures provide efficient algorithms performing set-theoretical operations as well as pre- and post- image computations. With these ingredients, it becomes possible to launch a fixpoint computation for forward or backward reachability sets, as exemplified in [30].

The problem of convergence. When dealing with infinite-state systems, a naive fixpoint computation procedure for reachability sets, in the style of Proc. 1 (page 7), has very little chance to terminate: convergence in a finite number of steps can only occur if the system under study is uniformly bounded (see section 4.2). To make fixpoint computations converge more frequently, so-called "*acceleration techniques*" have been developed. These techniques can compute subsets of the reachability set that are not uniformly bounded. This can be done, for example, by replacing a control loop "`x:=x+1; y:=y-1`" by its transitive closure "`k:=random_int(); x:=x+k; y:=y-k`". Currently, many different acceleration techniques for different families of systems exist [1, 2, 11, 13, 25, 37]. Some of them have been implemented [3, 7, 32] and promising case-studies have been reported [1–3, 7, 8]. Acceleration is quite related to widening techniques [4] in abstract interpretation [21]. While acceleration refers to exact computation, widening trades exactness for termination.

A field in need of foundations. The existing acceleration results usually amount to a (sometimes difficult) theorem stating that the transitive closure of an action, or of a sequence of actions, can be effectively computed. The difficulty of these results usually lies in finding the precise conditions on the action and on the set of initial states that yield effectiveness. How to use such results is not really known: the theorems and algorithms for computing reachability sets with acceleration methods do not exist in general! With some tools, e.g., LASH, the user has to choose which loops to accelerate and how to mix the result with more standard symbolic computation; in other cases, e.g., with TREX, some default strategy is implemented outside of any theoretical framework and without discussions about its efficiency or completeness.

Our contribution. **(1)** We propose the first theoretical framework for symbolic model checking with acceleration. We distinguish three natural levels for accelerations ("*loop*", "*flat*", and "*global*"), depending on which sequences of transitions can be computed: transitive closure of cycles (resp. of length 1) for flat (resp. loop) acceleration; or any regular set of sequences for global acceleration. These levels can account for most acceleration results on specific systems (pushdown systems, channel systems, counter systems, ...). For each level we give a symbolic algorithm with acceleration computing reachability sets and we characterize the conditions necessary for its termination.

Flat acceleration is the most interesting level. As a matter of fact, loop acceleration is not sufficient for many of the example systems we have analyzed with our tool FAST. Furthermore, the majority of existing acceleration results stated at the loop acceleration level may be extended to the level of flat acceleration. At the other end of the spectrum, global acceleration is always sufficient but it occurs very rarely in practice and is essentially restricted to particular subclasses (e.g., pushdown systems, reversal-bounded counter systems [29] or particular subclasses of Petri nets).

(2) We develop new concepts for the algorithmic study of flat acceleration. The notions of *flattenings* and of *flattable systems* provide the required bridge between flat acceleration and the effective computation of the reachability set.

We propose new symbolic procedures and analyze them rigorously. Procedure REACH2 is new. We show it terminates iff it is applied to a flattable (and not only flat) system, which is the first completeness result on symbolic model checking with acceleration. Let us remark that most of the case studies we analyzed in earlier works with FAST are flattable but not flat, underlining the relevance of this concept.

(3) Procedure REACH2 is schematic and it can be specialized in several ways. We propose one such specialization, REACH3, geared towards the efficient search of all flattenings of a nonflat system, without compromising completeness.

It appears that a key issue with REACH3 is the reduction of the number of circuits the procedure has to consider. FAST implements specific algorithms for counter systems that reduce exponentially the number of considered circuits and we show how to generalize these ideas to other families of systems. It is these algorithms that make FAST succeed in verifying several examples (see section 7) for which tools like LASH and ALV, based on similar technology but restricted heuristics, do not terminate. More generally, the comparisons in section 7 suggest that flat acceleration greatly enhances termination of symbolic reachability set computation, and is fully justified in practice.

This "theory of accelerations" is a new theoretical framework, not a compilation or survey of known acceleration results. It gives a common theoretical background justifying existing results and tools. It suggests research agendas for different families of systems. Our results are not technically difficult but we think they can (and must!) be used to compare and to rationally improve existing tools like TREX, LASH, and others, or to design new tools based on new acceleration results.

Outline. We define the systems under study in section 3, and the symbolic frameworks in section 4. Section 5 introduces the three levels of accelerations and defines flattable systems. Section 6 provides our procedure for flattable systems, and gives several algorithmic and/or heuristic refinements. Section 7 compares several existing tools through the new framework. As a rule, all important proofs are given in the appendix.

2 Notations

A *binary relation* r (shortly *relation*) on some set X is any subset of $X \times X$. We shortly write $x\,r\,x'$ whenever $(x, x') \in r$. We denote by $r(x)$ the set $r(x) = \{x' \in X \mid xrx'\}$. We extend this notation to a subset $Y \subseteq X$ by $r(Y) = \{x' \in X \mid \exists y \in Y, yrx'\}$. Given two binary relations r_1, r_2 on X, the *composed binary relation* $r_1 \bullet r_2$ on X is defined by $x\,(r_1 \bullet r_2)\,x'$ iff $x\,r_1\,y$ and $y\,r_2\,x'$ for some $y \in X$. $r_1 \bullet r_2$ corresponds to applying first r_1 then r_2. Id_X is the *identity relation on* X. r^i is defined by $r^0 = Id_X, r^{i+1} = r \bullet r^i$. r^* is the *reflexive and transitive closure of* r.

3 Systems and interpretations

A *system* is a finite state control graph extended with a finite number of variables ranging over arbitrary domains and modified by actions when a transition is fired. Specific families of systems have been widely studied (see subsection 3.1).

Definition 3.1 (Uninterpreted system). *An* uninterpreted system S *is a tuple* $\mathsf{S} = (Q, \Sigma, T)$*, where* Q *is a finite set of* locations*,* Σ *is a (possibly infinite) set of formulas called* actions*,* $T \subseteq Q \times \Sigma \times Q$ *is a finite set of* transitions.

Given a *uninterpreted system* $\mathsf{S} = (Q, \Sigma, T)$, the *source*, *target* and *action* mappings $\alpha : T \to Q$, $\beta : T \to Q$ and $l : T \to \Sigma$ are defined as follows: for any transition $t = (q, \sigma, q') \in T$, $\alpha(t) = q, \beta(t) = q', l(t) = \sigma$.

Definition 3.2 (Interpretation). *Given a (possibly infinite) set of formulas Σ and a set D, an* interpretation *I of Σ, shortly an* interpretation*, is a tuple $I = (\Sigma, D, [\![\cdot]\!])$ such that $[\![\cdot]\!] : \Sigma \to 2^{D \times D}$ maps formulas to relations on D.*

Definition 3.3 (System). *An* interpreted system *S (shortly a system) is a pair (S, I) of an uninterpreted system $\mathsf{S} = (Q, \Sigma, T)$ and an interpretation $I = (\Sigma, D, [\![\cdot]\!])$ of Σ, shortly written $S = (Q, \Sigma, T, D, [\![\cdot]\!])$.*

Fig. 1 displays S_0, a simple uninterpreted system, in graphical notation. The underlying set of actions Σ is left unspecified in this example, but the reader should recognize that it contains assignments that can be guarded by Boolean expressions. a_1, a_2, a_3 identify the three actions effectively used by S_0. A possible interpretation for this set of actions assumes that the domain D is $\mathbb{Z}^{\{x,y\}}$, or equivalently $\mathbb{Z}^2$, i.e., we decide that x and y range over integers. We then interpret the actions in the obvious way. For example $[\![a_2]\!] = \{((x, y), (x', y')) \mid x \neq y \wedge y' = y + x \wedge x' = x\}$. This turns S_0 into an interpreted system S_0.

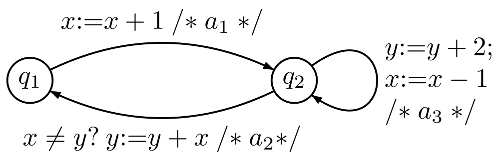


Fig. 1. S_0, a simple uninterpreted system

Behaviour. The *set of configurations* $\mathcal{C}_S$ of S is $Q \times D$, and the semantics of each transition $t \in T$ is given by a relation $\xrightarrow{t} \subseteq \mathcal{C}_S \times \mathcal{C}_S$. $\xrightarrow{t}$ is defined by: $(q, \mathsf{x}) \xrightarrow{t} (q', \mathsf{x}')$ if $q = \alpha(t), q' = \beta(t)$ and $(\mathsf{x}, \mathsf{x}') \in [\![l(t)]\!]$. This definition can be extended to the set T^* of all sequences of transitions. Let us denote ε the *empty word*. Then $\xrightarrow{\varepsilon} = Id_{\mathcal{C}_S}$ and $\xrightarrow{t \cdot \pi} = \xrightarrow{t} \bullet \xrightarrow{\pi}$. We also define $\xrightarrow{\mathcal{L}}$ for *any language* $\mathcal{L} \subseteq T^*$ by $\xrightarrow{\mathcal{L}} = \bigcup_{\pi \in \mathcal{L}} \xrightarrow{\pi}$. Similarly $[\![\cdot]\!]$ can be extended to any language $\mathcal{L} \subseteq \Sigma^*$.

Reachability problems. We are interested in checking safety properties of systems. Safety properties are both intuitive as they can be described in terms of sets of (safe) configurations $P \subseteq \mathcal{C}_S$, and meaningful since they allow to express deadlock freedom, mutual exclusion, overflows and so on. For any $X \subseteq \mathcal{C}_S$ and any $\mathcal{L} \subseteq T^*$, we define $\mathsf{post}_S(\mathcal{L}, X) = \{x' \in \mathcal{C}_S \mid \exists x \in X;\ (x, x') \in \xrightarrow{\mathcal{L}}\}$. The set $\mathsf{post}_S(T, X)$ of all configurations reachable in one step from X is denoted by $\mathsf{post}_S(X)$.The set $\mathsf{post}_S(T^*, X)$ of all configurations reachable from X is *the reachability set of* X, denoted by $\mathsf{post}^*_S(X)$.

Given an initial set of configurations X_0, checking a safety property P can be done by (1) computing $\mathsf{post}^*_S(X_0)$, and (2) checking that $\mathsf{post}^*_S(X_0) \subseteq P$. We focus in this paper on the reachability set computation which is the key issue. Since $\mathsf{post}^*_S(X_0)$ is not recursive in general (Minsky machines, channel systems [16], and so on), the best we can hope for are partially correct procedures, with no guarantee of termination, but efficient on large subclasses and practical case-studies.

Backward computation. Another way is to proceed backward, computing the co-reachability set $\mathsf{pre}^*_S(P)$ and checking that $X_0 \subseteq \mathsf{pre}^*_S(P)$. Since for our level of abstraction, adaptation to backward computation is straightforward, we consider only forward computation. However it is worth noticing that depending on particular cases, one of the approaches may be more adapted than the other. Along the paper specific results for backward computation are pointed out.

Transition relation computation. A third approach is to compute *the reachability relation* $\xrightarrow{T^*}$, and then $\mathsf{post}^*(X_0) = \xrightarrow{T^*} (X_0)$. Our framework can be extended in this direction. For it requires additional notations, it is not treated here.

Notation. Whenever S is implicitly known, it is omitted in notations.

3.1 Family of systems

Definition 3.4 (Family of systems). *Given an interpretation $I = (\Sigma, D, [\![\cdot]\!])$,* the family of systems built on I *(shortly* the family of systems*) denoted by $\mathcal{F}(I)$ is the class of all systems $S = (Q, \Sigma, T, D, [\![\cdot]\!])$ using I to interpret actions.*

Well known models can be obtained by instantiating Def 3.4. Let us denote by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ the sets of positive integers, integers and reals. Then

Minsky machines: are obtained by defining $D = \mathbb{N}^{Var}$ where $Var = \{x_1, x_2, \ldots\}$ is a set of variables, and Σ as the set of increments "$x_i := x_i + 1$", guarded decrements "$x_i > 0?\ x_i := x_i - 1$" and 0-tests "$x_i = 0?$" with the obvious interpretation.
Counter systems [17,33]: are obtained by considering the same domain, or a variant $D = \mathbb{Z}^{Var}$, and all actions definable in Presburger arithmetic. Many restrictions exist, e.g., linear systems where actions are linear transformations with guards expressed in Presburger [25,37], reversal-counter systems [29], many extensions of VASS (or Petri nets) and so on.
Pushdown systems: the domain is $D = \Gamma^*$, the set of all words on some stack alphabet Γ. Actions add or remove letters on or from the top of the stack.
Channel systems [16]: consider the domain is $D = (\Gamma^*)^C$ where C is a set of fifo channels, and Γ is some alphabet of messages. Actions add messages at one end of the channels and consume them at the other end.
Timed automata [5]: consider the domain $D = \mathbb{R}_+^{Var}$. Here some actions are guarded by simple linear (in)equalities and they can only reset clocks. Other actions, left implicit in the standard presentation, account for time elapsing.
Hybrid systems [4]: extend timed automata in that the real-valued variables do not increase uniformly when time elapses. Rather they each increase according to their own rate (as given by the current location).

4 A symbolic framework for symbolic model checking

In practice model checking procedures use symbolic representations (called here *regions*) to manipulate sets of configurations. The definition below follows directly from ideas expressed for example in [14,21,30].

Definition 4.1 (Symbolic framework). *A* symbolic framework *is a tuple $(\Sigma, D, [\![\cdot]\!]_1, L, [\![\cdot]\!]_2)$ where $I = (\Sigma, D, [\![\cdot]\!]_1)$ is an interpretation, L is a set of formulas called* regions*, $[\![\cdot]\!]_2 : L \rightarrow 2^D$ is a* region concretization*, such that there existing a decidable relation $\sqsubseteq$ and recursive functions $\sqcup$,* POST *satisfying*

1. *there exists an element* $\bot \in L$ *such that* $[\![\bot]\!]_2 = \emptyset$.
2. $\sqsubseteq \subseteq L \times L$ *is such that for all* $\mathtt{x}_1, \mathtt{x}_2 \in L$, $\mathtt{x}_1 \sqsubseteq \mathtt{x}_2$ *iff* $[\![\mathtt{x}_1]\!]_2 \subseteq [\![\mathtt{x}_2]\!]_2$.
3. $\sqcup : L \times L \to L$ *is such that* $\forall \mathtt{x}_1, \mathtt{x}_2 \in L$, $[\![\mathtt{x}_1 \sqcup \mathtt{x}_2]\!]_2 = [\![\mathtt{x}_1]\!]_2 \cup [\![\mathtt{x}_2]\!]_2$.
4. $\textsc{post} : \Sigma \times L \to L$ *is such that* $\forall a \in \Sigma, \forall \mathtt{x} \in L, [\![\textsc{post}(a, \mathtt{x})]\!]_2 = [\![a]\!]_1 ([\![\mathtt{x}]\!]_2)$.

Notation. Let us denote by $[\![\cdot]\!] : \Sigma \cup L \to 2^{D \times D} \cup 2^D$ the unique function extending $[\![\cdot]\!]_1$ and $[\![\cdot]\!]_2$. Usually given an interpretation $I = (\Sigma, D, [\![\cdot]\!]_1)$ and a set of regions L, $[\![\cdot]\!]_2$ is well-known. Thus in the following, we write $[\![\cdot]\!]$ for both $[\![\cdot]\!]_1$ and $[\![\cdot]\!]_2$, and we denote symbolic frameworks as $SF = (I, L)$. In the rest of the paper, we fix an arbitrary symbolic framework $SF = (I, L)$. When refering to a system S, if nothing is specified we assume that $S \in \mathcal{F}(I)$.

In some approaches, the symbolic framework may be weakened. A *weak inclusion* ensures only that $\mathtt{x}_1 \sqsubseteq \mathtt{x}_2$ implies $[\![\mathtt{x}_1]\!] \subseteq [\![\mathtt{x}_2]\!]$ while a *weak union* satisfies $[\![\mathtt{x}_1]\!] \cup [\![\mathtt{x}_2]\!] \subseteq [\![\mathtt{x}_1 \sqcup \mathtt{x}_2]\!]$ (typical widening in abstract interpretation [4, 21]). In the following, we do not consider weakened framework.

Well-known symbolic frameworks for some of the families listed in 3.1 are:

Regular languages: have been used for representing sets of configurations of pushdown systems [12], distributed protocols over rings of arbitrary size [30], and channel systems [35]. Restricted sets of regular languages are sometimes used for better algorithmic efficiency: languages closed by the subword relation [1] or closed by semi-commutations [15].

(finite union of) Convex polyhedra [4]: are conjunctions of linear inequalities defining subsets of $\mathbb{R}_+^{Var}$, relevant in the analysis of hybrid systems.

Number Decision Diagrams [17, 25]: are automata recognizing subsets of $\mathbb{Z}^{Var}$ and have been used in the analysis of counter systems.

Real Vector Automata [10]: are Büchi automata recognizing subsets of $\mathbb{R}_+^{Var}$ and have been used in the analysis of linear hybrid systems.

Difference Bounds Matrices [5]: are a canonical representations for convex subsets of $\mathbb{R}_+^{Var}$ defined by simple diagonal and orthogonal constraints that appear in timed automata.

Covering Sharing Trees [23]: are a compact representation for upward-closed subsets of $\mathbb{N}^{Var}$. These sets appears naturally in the backward analysis of broadcast protocols [25] and several monotonic extensions of Petri nets.

Given a system S with a set of locations Q, and $X \subseteq \mathcal{C}_S$, $\mathsf{post}^*(X)$ is of the form $\bigcup_{q \in Q} \{q\} \times D_q$ where the D_q are subsets of D. Assuming an implicit ordering on locations $q_1, \ldots, q_{|Q|}$, we work on tuples of regions in $L^{|Q|}$. We extend $[\![\cdot]\!]$ to $L^{|Q|}$ by $[\![(\mathtt{x}_1, \ldots, \mathtt{x}_{|Q|})]\!] = \bigcup_{i \leq |Q|} \{q_i\} \times [\![\mathtt{x}_i]\!]$. Extensions of $\sqsubseteq$ and $\sqcup$ are component-based. $\textsc{post}$ is extended into $\textsc{post} : T \times L \to L$ by: $\textsc{post}((q_i, a, q_j), (\mathtt{x}_1, \ldots, \mathtt{x}_{|Q|}))$ is equal to $(\mathtt{x}'_1, \ldots, \mathtt{x}'_{|Q|})$ such that $\mathtt{x}'_p = \bot$ if $p \neq j$, $\textsc{post}(a, \mathtt{x}_i)$ otherwise. $\textsc{post}$ is then extended to sequence of transition in the obvious way. We define $\textsc{post} : L^{|Q|} \to L^{|Q|}$ by $\textsc{post}(\mathtt{x}) = \bigsqcup_{t \in T} \textsc{post}(t, \mathtt{x})$.

4.1 Limits of the symbolic approach

A subset of configurations $X \subseteq \mathcal{C}_S$ is *L-definable* if there exists $\mathtt{x} \in L^{|Q|}$ such that $[\![\mathtt{x}]\!] = X$. Computing $\mathsf{post}^*(X)$ using regions is feasible only if $\mathsf{post}^*(X)$ is L-definable. The question "is $\mathsf{post}^*([\![\mathtt{x}]\!])$ L-definable?" is undecidable.

Theorem 4.2. *Given the symbolic framework of 2-counter systems and Presburger formulas, a 2-counter system S, and $\mathtt{x}_0 \in L^{|Q|}$, then whether $\mathsf{post}^*([\![\mathtt{x}_0]\!])$ is L-definable or not is undecidable.*

It must be clear that L-definability of $\mathsf{post}^*(X)$ is not a sufficient condition for feasibility of its computation.

Definition 4.3 (Effective L-definability). *A* function $f : 2^{\mathcal{C}_S} \times 2^{\mathcal{C}_S} \rightarrow 2^{\mathcal{C}_S}$ *is* effectively L-definable *if there exists a recursive function $g_f : L^{|Q|} \rightarrow L^{|Q|}$ such that* $\forall \mathtt{x} \in L^{|Q|}$, $f([\![\mathtt{x}]\!]) = [\![g_f(\mathtt{x})]\!]$.

It can be the case that for all $\mathtt{x} \in L^{|Q|}$, $\mathsf{post}^*([\![\mathtt{x}]\!])$ is L-definable while post^* is not effectively L-definable. For example, consider the family of lossy channels systems and the framework defined by simple regular expressions. With no ambiguity, we write now $f([\![\mathtt{x}]\!])$ is effectively L-definable instead of f is effectively L-definable.

4.2 Standard symbolic model-checking procedure

The iterative procedure 1 is derived from the algorithm for finite systems.

procedure REACH1($\mathtt{x}_0$)
parameter: S
input: $\mathtt{x}_0 \in L^{|Q|}$
$\mathtt{x} \leftarrow \mathtt{x}_0$
while $\text{POST}(\mathtt{x}) \not\sqsubseteq \mathtt{x}$ **do**
  $\mathtt{x} \leftarrow \text{POST}(\mathtt{x}) \sqcup \mathtt{x}$
end while
return $\mathtt{x}$

Procedure 1: Standard symbolic model checking algorithm

Definition 4.4 (L-uniformly bounded). *A system S is L-* uniformly bounded *if for all $\mathtt{x} \in L^{|Q|}$, there exists $n_{\mathtt{x}} \in \mathbb{N}$ such that, for all $c_1 \in Q \times [\![\mathtt{x}]\!]$ and $c_2 \in Q \times D$, if $c_2 \in \mathsf{post}^*(\{c_1\})$ then $c_2 \in \bigcup_{i \le n_{\mathtt{x}}} \mathsf{post}^i(\{c_1\})$.*

Theorem 4.5. *Given a symbolic framework $SF = (I, L)$ and a system $S \in \mathcal{F}(I)$*
1. *When* REACH1 *terminates,* $[\![\text{REACH1}(\mathtt{x}_0)]\!] = \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (partial correctness).
2. REACH1 *terminates on any input iff S is L-uniformly bounded* (termination).

Remark 4.6. If $\sqsubseteq$ or $\sqcup$ are weak, the above termination result does not hold anymore.

Well-structured transition systems [26] with upward-closed sets are L-backward uniformly bounded. This applies for Petri nets and many of their monotonic extensions, or lossy channels systems. However in practice, systems are rarely L- (backward) uniformly bounded and Proc. 1 seldom terminates.

5 Flat acceleration for flattable systems

5.1 Acceleration techniques

In order to improve the convergence of the previous procedure, *acceleration techniques* consist in computing the transitive closure of some transitions.

Definition 5.1 (Acceleration). *A symbolic framework SF supports*

1. *loop acceleration if there exists a recursive function* $\text{POST\_STAR} : \Sigma \times L \to L$ *such that* $\forall a \in \Sigma$, $\forall \mathtt{x} \in L$, $[\![\text{POST\_STAR}(a, \mathtt{x})]\!] = [\![a^*]\!]\,([\![\mathtt{x}]\!])$*;*
2. *flat acceleration if there exists a recursive function* $\text{POST\_STAR} : \Sigma^* \times L \to L$ *such that* $\forall \pi \in \Sigma^*$, $\forall \mathtt{x} \in L$, $[\![\text{POST\_STAR}(\pi, \mathtt{x})]\!] = [\![\pi^*]\!]\,([\![\mathtt{x}]\!])$*;*
3. *global acceleration if there exists a recursive function* $\text{POST\_STAR} : \mathit{RegExp}(\Sigma) \times L \to L$ *such that for any regular expression* a *over* Σ*, for any* $\mathtt{x} \in L$, $[\![\text{POST\_STAR}(\mathsf{e}, \mathtt{x})]\!] = [\![\mathsf{e}]\!]\,([\![\mathtt{x}]\!])$.

Let $A \subseteq D$. In Fig. 1, loop acceleration concerns only action a_3, and comes down to computing $A' = [\![a_3^*]\!]\,(A) = \{(x', y') \in \mathbb{Z}^2 | \exists (x, y) \in A; \exists k \in \mathbb{N}; x' = x - k \wedge y' = y + 2 \cdot k\}$. Flat acceleration requires that $[\![(a_1 \cdot a_2)^*]\!]\,(A)$, $[\![(a_1 \cdot a_3 \cdot a_2)^*]\!]\,(A)$, $[\![(a_1 \cdot a_3 \cdot a_3 \cdot a_2)^*]\!]\,(A)$, $[\![(a_3 \cdot a_2 \cdot a_1)^*]\!]\,(A)$ and so on are computable. Global acceleration requires the computation of more complex interleaving of actions, like the nested loops $[\![(a_1 \cdot a_3^* \cdot a_2)^*]\!]\,(A)$ *(configurations* (q_1, A') *reachable from* (q_1, A)*).*

In all cases, POST_STAR can be extended to handle transitions. Let us explain the extension for flat acceleration. Consider a sequence of transitions $(q_1, a_1, q_2) \cdot (q_3, a_2, q_4) \cdot (q_5, a_3, q_6)$. Then there are two cases. If the sequence is invalid (i.e. $q_2 \neq q_3$ or $q_4 \neq q_5$) then the associated relation is empty, and the acceleration returns the identity relation. If the sequence is valid, then the transition is equivalent to $(q_1, a_1 \cdot a_2 \cdot a_3, q_6)$. If the sequence is not a cycle ($q_1 \neq q_6$), the iteration is equivalent to firing the transition only once. We compute it using the POST operation (and adding the identity relation). Finally if the sequence is a cycle $\pi = (q_1, a_1 \cdot a_2 \cdot a_3, q_1)$, the acceleration is: $\text{POST\_STAR}(\pi, (q, \mathtt{x})) = (q, \mathtt{x})$ if $q \neq q_1$, $(q, \text{POST\_STAR}(a_1 \cdot a_2 \cdot a_3, \mathtt{x}))$ otherwise. POST_STAR is finally easily extended to $L^{|Q|}$. The extension for global acceleration considers the intersections of the regular langage e with the regular langages of transitions from a location q to another location q'.

Loop acceleration. All the symbolic frameworks defined from Minsky machines and equipped with sets of formulas like *formulas defining upward-closed sets*, *Presburger formulas defining semi-linear sets* support loop acceleration. Upward-closed sets for example are not expressive enough to support flat acceleration.
Flat acceleration. Counter systems (with finite monoid) equipped with Presburger formulas supports flat acceleration [25, theorem 2]. Other examples are channel systems with cqdd [13, theorem 5.1], non-counting channel systems with slre [26, theorem 5.2] or qdd[11, theorem 6], lossy channel systems with sre [1, corollary 6.5]. Restricted counter systems used by TREX equipped with arithmetics almost supports flat acceleration [2, lemma 5.1] except that the POST_STAR is not recursive.
Global acceleration. Reversal-counter systems [29], 2-dim VASS [33], lossy VASS and other subclasses of VASS with Presburger formulas [34], pushdown systems with

regular languages or semi-commutative rewriting systems with APC language [15], support global acceleration.

Obviously global $\Rightarrow$ flat $\Rightarrow$ loop. Loop acceleration is easy to get, but rarely sufficient to lead to fixpoint computation. Flat acceleration is more flexible, but often requires good compositional properties of Σ and rather complex methods for POST_STAR. Finally global acceleration is a very strong property, ensuring the effective computation of $\mathsf{post}^*(\llbracket \mathtt{x} \rrbracket)$ for any $\mathtt{x} \in L^{|Q|}$. Clearly most of the systems we want to cope with do not support global acceleration since they are Turing powerful. Then for our purpose, flat acceleration is likely to be the best compromise. The rest of the paper will focus on flat acceleration.

Notation. In the rest of the paper we write S *supports loop (resp. loop, flat, global) acceleration* instead of (I, L) *supports loop (resp. loop, flat, global) acceleration*.

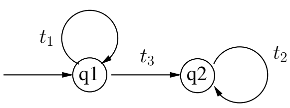
5.2 Restricted linear regular expressions

Flat acceleration allows to compute the effect of more general expressions than iterations of sequences of actions. Given an alphabet A, a *restricted linear regular expression* (rlre) over A is a regular expression ρ of the form $u_1^* \ldots u_n^*$, where $u_i \in A^*$. This is closely related to semi-linear regular expressions [26, 28].

Proposition 5.2. *Given a system S supporting flat acceleration, then for any* rlre *ρ over T and for any $\mathtt{x}_0 \in L^{|Q|}$,* $\mathsf{post}(\rho, \llbracket \mathtt{x}_0 \rrbracket)$ *is effectively L-definable.*

Actually, there exists a recursive function with input $(\rho, \mathtt{x}_0)$ producing $\mathtt{x} \in L^{|Q|}$ such that $\llbracket \mathtt{x} \rrbracket = \mathsf{post}(\rho, \llbracket \mathtt{x}_0 \rrbracket)$.

5.3 Flat systems



In general, flat acceleration does not ensure the reachability set computability. However on some particular systems, flat acceleration is sufficient. For example when the system has *no nested loop*. Considering the system on the right, reachability set computation is achieved by iterating first t_1, then firing t_3 and finally iterating t_2. We call such systems flat. The system of figure 1 is not flat, because of the two *elementary cycles* on q_2, labelled by a_3 and $a_2 \cdot a_1$. An elementary cycle is a valid sequence of transitions which does not visit any location twice, except that the first location and the last one can be the same.

Definition 5.3 (Flat system [19, 26, 28]). *An uninterpreted system* $\mathsf{S} = (\Sigma, Q, T)$ *is* flat *if for any location q, there exists at most one elementary cycle containing q. A system $S = (\Sigma, Q, T, D, \llbracket\cdot\rrbracket)$ is* flat *if* $\mathsf{S} = (\Sigma, Q, T)$ *is flat.*

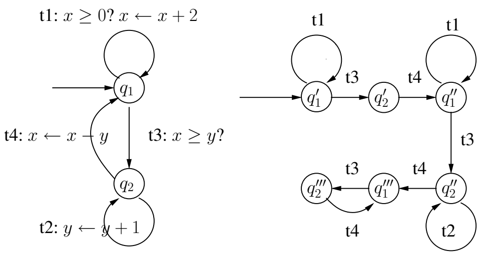
Proposition 5.4. *Given a flat system S supporting flat acceleration, then* $\mathsf{post}^*(\llbracket \mathtt{x} \rrbracket)$ *is effectively L-definable.*

5.4 Flattening of non-flat systems

Not all systems of interest are flat. For an arbitrary system, we introduce *flattening*, which consists in finding a flat system S', equivalent to S w.r.t. reachability, and compute on S' instead of S.

Definition 5.5 (Flattening). *A system $S' = (Q', \Sigma, T', D, [\![\cdot]\!])$ is a flattening of a system $S = (Q, \Sigma, T, D, [\![\cdot]\!])$ if (1) S' is flat and (2) there exists a mapping $z : Q' \to Q$, called* folding*, such that $\forall (q'_1, w, q'_2) \in T'$, $(z(q'_1), w, z(q'_2)) \in T$.*

Flattening generalizes unfolding, allowing non nested loops. The following figure shows a system (left) and one of its flattenings (right). Assume S is a system and S' one of its flattening as defined above. We extend the folding z to configurations of S' by $z((q', x)) = (z(q'), x)$. Extension of z to $X \subseteq \mathcal{C}_{S'}$ is defined by:



$$z(\textstyle\bigcup_{q' \in Q'} \{q'\} \times D_{q'}) = \bigcup_{q \in Q} \{q\} \times (\bigcup_{(q' \in Q', z(q') = q)} D_{q'}).$$

This gives an effective extension of z to L-definable subsets of $\mathcal{C}_{S'}$. Given $X' \subseteq \mathcal{C}_{S'}$, definition 5.5 ensures that $z(\mathsf{post}^*_{S'}(X')) \subseteq \mathsf{post}^*_S(z(X'))$ and that for any language $\mathfrak{L} \subseteq T^*$, $z(\mathsf{post}_{S'}(\mathfrak{L}, [\![\mathtt{x}']\!])) = \mathsf{post}_S(z(\mathfrak{L}), z([\![\mathtt{x}']\!]))$.

Definition 5.6 (L- flattable). *A system $S = (Q, \Sigma, T, D, [\![\cdot]\!])$ is* L-flattable *iff for any $\mathtt{x} \in L^{|Q|}$, there exists a flattening $S' = (Q', \Sigma, T', D, [\![\cdot]\!])$ of S and $\mathtt{x}' \in L^{|Q'|}$ such that $z([\![\mathtt{x}']\!]) = [\![\mathtt{x}]\!]$ and $z(\mathsf{post}^*_{S'}([\![\mathtt{x}']\!])) = \mathsf{post}^*_S(z([\![\mathtt{x}']\!]))$.*

Theorem 5.7. *Let S be a L-flattable system supporting flat acceleration. Then* $\mathsf{post}^*([\![\mathtt{x}]\!])$ *is effectively L-definable.*

5.5 About flattable systems

A natural question is whether L-flattable systems are well-spread or not. A first negative result is that we cannot decide if a system is L-flattable, even if restricting to 2-counter systems.

Theorem 5.8. *Given the symbolic framework of 2-counter systems and Presburger formulas, then whether a 2-counter system S is L-flattable or not is undecidable.*

However many systems with L-definable reachability set appear to be flattable. For example 2-dim VASS [33], timed automata [20], k-reversal counter machines, lossy VASS and other subclasses of VASS [34] and all L-uniformly bounded systems (see section 4) are L-flattable. It must be clear that there is no equivalence in general: lossy channel systems have L-definable reachability sets but are not flattable. Interesting open questions are whether well-known subclasses with L-definable reachability sets (like Presburger definable VASS) are L-flattable or not.

6 Computing reachability set using flat acceleration

6.1 A first procedure

The previous characterization gives a complete procedure for flattable systems: (1) enumerate all flattenings; (2) compute their reachability set; (3) check the fixpoint. But flattenings are not easily manipulable. Here is a new characterization of flattable based on rlre. Indeed proposition 5.4 shows that for flat systems, post^* is effectively L-definable using POST_STAR over some rlre. Theorem 5.7 states that for a flattable system S, post^* is also effectively L-definable, using a flattening of S. The next theorem unifies these results.

Theorem 6.1. *A system $S = (Q, \Sigma, T, D, [\![\cdot]\!])$ is L- flattable iff for all $\mathtt{x} \in L^{|Q|}$, there exists a rlre ρ over T such that* $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}]\!])$.

Reachability set computation for flattable systems reduces to exploring the set of rlre over T, which can be achieved by increasing a sequence of rlre (Proc. 2). *Fairness:* we assume that if Choose is called infinitely often, each $w \in T^*$ is selected infinitely often. This can be ensured for example by enumerating all $w \in T^*$ such that $|w| \leq 1$, then all $w \in T^*$ such that $|w| \leq 2$, and so on.

```
procedure REACH2(x0)
parameter: S
input: x0 ∈ L^|Q|
x ← x0
while POST(x) ⋢ x do
    Choose fairly w ∈ T*
    x ← POST_STAR(w, x)
end while
return x
```

Procedure 2: Flat acceleration.

Theorem 6.2. *Given a symbolic framework $SF = (I, L)$ and a system $S \in \mathcal{F}(I)$*

1. *When* REACH2 *terminates,* $[\![\text{REACH2}(\mathtt{x}_0)]\!] = \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (partial correctness).
2. REACH2 *terminates on any input iff S is L- flattable* (termination).

This termination result does not hold if the symbolic framework provides only a weak inclusion, or if POST_STAR is an overapproximation.

6.2 Faster enumeration of flattenings

A major issue is to implement Choose such that the fixpoint is reached quickly. Instead of considering all sequences in T^*, a bound k is chosen, and the previous procedure restricted to non empty sequences of length $\leq k$ (denoted $T^{\leq k}$) is launched ($\mathsf{k\text{-}flattable}$). The search is eventually stopped, k is incremented and $\mathsf{k\text{-}flattable}$ is launched again. We assume that $\mathsf{Watchdog}$ is fired infinitely often, but only after that each $w \in T^{\leq k}$ has been selected at least once by Choose *(fairness)*.

```
procedure REACH3(x_0)
parameter: S
input: x_0 ∈ L^{|Q|}
x ← x_0 ; k ← 0
k ← k + 1
start
    while POST(x) ⋢ x do        /* k-flattable */
        Choose fairly w ∈ T^{≤k}
        x ← POST_STAR(w, x)
    end while                   /* end k-flattable */
with
    when Watchdog stops goto 2
return x
```

Procedure 3: Flat acceleration and circuit length increasing

Theorem 6.3. *Given a symbolic framework $SF = (I, L)$ and a system $S \in \mathcal{F}(I)$*

1. *When* REACH3 *terminates,* $[\![\text{REACH3}(\mathtt{x}_0)]\!] = \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (partial correctness).
2. REACH3 *terminates for any input iff S is L- flattable* (termination).

Technical issues. There still remain two practical problems. First the *size*[3] of $\mathtt{x} \in L^{|Q|}$ computed so far may be intractable. Second Watchdog needs a stop criterion. We describe the implementation in FAST of these two procedures, and believe that the solutions proposed can be adapted to over domains. We want to point out that the implementation we describe does not follow exactly the specification of REACH3 since fairness is not ensured anymore. FAST should be improved in this way.

Choose. In general there is no monotonic relationship between the size of a region and the size of its concretization (w.r.t. $\subseteq$). Intermediate regions may have a size much larger than the one of the fixpoint. Such intermediate regions must be avoided. Choose selects the next $w \in T^{\leq k}$, such that $|\text{POST\_STAR}(w, \mathtt{x})| < |\mathtt{x}|$. If there is no such w, then the next one is selected. In practice, a cyclic enumeration almost always run out of memory, while this enumeration works well.

Watchdog. Let us denote by depth the number of iterations in the macro k-flattable (reset when exiting the macro). Our stop criterion for Watchdog is a maximal limit on depth. In practice, when a sufficient k is reached, the fixpoint is computed within few iterations.

6.3 Reduction of the number of cycles

A remaining issue in REACH3 is the exponential cardinal in k of $T^{\leq k}$. We introduce the notion of reduction to compact the number of relevant transitions.

Definition 6.4 **(k-Reduction).** *Given an interpretation $I = (\Sigma, D, [\![\cdot]\!])$, a k-reduction $r : \mathcal{F}(I) \to \mathcal{F}(I)$ maps to each system $S = (Q, \Sigma, T, D, [\![\cdot]\!]) \in \mathcal{F}(I)$ a system*

[3] Each set of regions has its own measure for size. For exemple, a relevant size for Presburger formulas may be the number of nodes of the associated binary automaton.

$S' = (Q, \Sigma, T', D, [\![\cdot]\!]) \in \mathcal{F}(I)$ *such that: (1)* $\forall t' \in T', \xrightarrow{t'} \subseteq \xrightarrow{T^*}$, *(2)* $\forall w \in T^{\leq k}, \exists \rho \in$ *rlre*(T'). $\xrightarrow{w^*} \subseteq \xrightarrow{\rho}$, *(3)* $|T'| \leq |T^{\leq k}|$.

Conditions 1 and 2 ensure that if S is L- flattable with length k, then S' is L-flattable and has the same reachability set. Removing identity loops from $T^{\leq k}$ is a naive reduction, as well as $Id_{\mathcal{F}(I)}$. The following reductions are much more useful. *Conjugation reduction*: remove sequences of transitions equivalent w.r.t. conjugation to another sequence (e.g. $t_1 \cdot t_2 \cdot t_3$ and $t_2 \cdot t_3 \cdot t_1$). *Commuting reduction*: if t_1 and t_2 commute, i.e. $\xrightarrow{t_1} \bullet \xrightarrow{t_2} = \xrightarrow{t_2} \bullet \xrightarrow{t_1}$, then remove both $t_1 \cdot t_2$ and $t_2 \cdot t_1$.

Proposition 6.5. *The conjugation reduction and the commuting reduction are k-reductions. The conjugation reduction satisfies $|T'| = \mathcal{O}(\frac{|T^k|}{k})$.*

In addition to these generic reductions, it is worthwhile to develop reductions dedicated to a specific interpretation. [25] presents a reduction for linear counter systems with a finite monoid, such that $|T'|$ remains *polynomial in k*, while $|T^{\leq k}|$ is exponential in k. This appears to be a keypoint in FAST performances. Here are the reduction results for the swimming pool protocol, an infinite VASS with 7 transitions (6 var.), studied in [27]. Cycles of length 4 are required to compute the reachability set. $V_k \subseteq T^{\leq k}$ is the set of valid sequences of length $\leq k$. T' (resp. T'') is the reduced system with reduction of [25] (resp. combined with commuting transitions).

k	$\lvert V_k\rvert$	$\lvert T'\rvert$	$\lvert T''\rvert$
1	7	7	7
2	36	21	16
3	156	56	28
4	**578**	**126**	**47**
5	1890	252	86

7 Conclusion: flat acceleration in practice

7.1 Tools comparison

	ALV	FAST	LASH	TREX
system	full	linear		restricted
symbolic rep.	Presburger formula			arith. undec. $\sqsubseteq$
acceleration	no	flat	loop	$\approx$ flat
termination	UB	F	1F	kF (oracle $\sqsubseteq$)

We use our framework to compare symbolic model-checkers ALV, FAST, LASH and TREX, designed to check safety properties on counter systems (see definitions in section 3.1). We restrict this comparison to the exact forward computation of $\mathsf{post}^*([\![\mathrm{x}_0]\!])$. **ALV[6]** works on full counter systems. Regions are Presburger formulas. The heuristic used is similar to REACH1. **FAST[7]** and **LASH[32]** work both on linear counter systems equipped with Presburger formulas. Flat acceleration is supported for functions whose monoid is finite, but while FAST really takes advantage of full flat acceleration (Proc. REACH3), heuristics in LASH are restricted to loop acceleration (Proc. REACH2 where w is chosen in $T^{\leq 1}$ instead of T^*). **TREX[3]** manipulates restricted counter systems. Since regions are arithmetics formulas, inclusion is undecidable. A partially recursive flat acceleration procedure is available. The heuristic is REACH2 restricted to $T^{\leq k}$ for a user-defined k. [22] compares FAST and TREX in depth. UB, F and kF stands for L- uniformally bounded, L- flattable and L-flattable with length k (UB $\subseteq$ 1F $\subseteq$ kF $\subseteq$ F).

Procedure comparison on case studies. The following table compares how ALV, FAST and LASH behave in practice. Comparison is made between termination and non termination (after 1200 seconds), on a Pentium III 933 MHz with 512 Mbytes. k is the length of cycles used by FAST. Case studies are taken from [24]. They are all infinite systems. Experimental results are strongly related to the acceleration framework: the tool closer to the framework (FAST) is the one with better termination results, while simple iteration (ALV) is not sufficient on these complex examples (results are consistent with [9]). *Experiments clearly suggest that flat acceleration enhances greatly termination and is fully justified in practice, at least for counter systems.*

System	ALV	LASH	FAST	k
TTP	no	**yes**	**yes**	1
prod/cons (2)	no	**yes**	**yes**	1
prod/cons (N)	no	no	**yes**	2
lift control, N	no	no	**yes**	2
train	no	no	**yes**	2
consistency	no	no	**yes**	3
CSM, N	no	no	**yes**	2
swimming pool	no	no	**yes**	4
PNCSA	no	no	no	?
IncDec	no	no	no	?
BigJAVA	no	no	no	?

7.2 Tool design

Flat acceleration framework provides guidelines to design from scratch new techniques and tools. FAST supports completely this framework. Complex case studies have been conducted [7, 8]. The following table shows performances of FAST on a significant pool of infinite counter systems, collected among web sites of other model-checkers ALV, BABYLON, BRAIN, LASH and TREX. They range from tricky academic puzzles (swimming pool) to complex industrial protocols (TTP). Since they have infinite state-space, they are beyond the scope of traditional model-checking techniques and tools. Moreover, most of these systems are also beyond VASS/Petri nets, then nice methods like covering tree or backward computation do not work anymore. Of course one can try to build a finite abstraction of the system and check it with finite state tools. However we are interested here in *exact automatic computation*. The results are for forward computation of the reachability set, on an Intel Pentium 933 Mhz with 512 Mbytes. Other complex case-studies have been performed with ALV, LASH and TREX [3, 6, 9, 32]. *This confirms that flat acceleration is useful to handle infinite systems.*

System	var	$\|T\|$	sec.	MB.	k
CSM	13	13	45.57	6.31	2
FMS	22	20	157.48	8.02	2
Multipoll	17	20	22.96	5.13	1
Kanban	16	16	10.43	6.54	1
swimming pool	9	6	111	29.06	4
last i.-first s.	17	10	1.89	2.74	1
PC Java(2)	18	14	13.27	3.81	1
PC Java(N)	18	14	723.27	12.46	2
Central server	13	8	20.82	6.83	2
Consistency	12	8	275	7.35	3
M.E.S.I.	4	4	0.42	2.44	1
M.O.E.S.I.	4	5	0.56	2.49	1
Synapse	3	3	0.30	2.23	1
Illinois	4	6	0.97	2.64	1
Berkeley	4	3	0.49	2.75	1
Firefly	4	8	0.86	2.59	1
Dragon	5	8	1.42	2.72	1
Futurebus+	9	10	2.19	3.38	1
lift - N	4	5	4.56	2.90	3
barber m4	8	12	1.92	2.68	1
ticket 2i	6	6	0.88	2.54	1
ticket 3i	8	9	3.77	3.08	1
TTP	10	17	1186.24	73.24	1

References

[1] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *FMSD*, 25(1):39–65, 2004.

[2] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. CAV'00*, LNCS 1855, pages 419–434.

[3] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A tool for reachability analysis of complex systems. In *Proc. CAV'01*, LNCS 2102, pages 368–372.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, Pei-Hsin Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *TCS*, 138(1):3–34, 1995.

[5] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

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

[7] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *Proc. CAV'03*, LNCS 2725, pages 118–121.

[8] S. Bardin, A. Finkel and J. Leroux. FASTer acceleration of counter automata. In *Proc. TACAS'04*, LNCS 2988, pages 576–590.

[9] C. Bartzis and T. Bultan. Widening arithmetic automata. In *Proc. CAV'04*, LNCS 3114, pages 321–333.

[10] B. Boigelot, L. Bronne, and S. Rassart. Improved reachability analysis method for strongly linear hybrid systems. In *Proc. CAV'97*, LNCS 1254, pages 167–178

[11] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. SAS'97*, LNCS 1302, pages 172–186.

[12] A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems, and P. Wolper. An efficient automata approach to some problems on context-free grammars. *IPL*, 74(5–6):221–227, 2000.

[13] A Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *TCS*, 221(1–2):211–250, 1999.

[14] A. Bouajjani, B. Jonsson, M. Nilsson and T. Touili. Regular Model Checking. *Proc. CAV'00*, LNCS 1855, pages 403–418.

[15] A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. In *Proc. LICS'01*, pages 399–408.

[16] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.

[17] T. Bultan, R. Gerber, and W. Pugh. Symbolic model-checking of infinite state systems using Presburger arithmetic. In *Proc. CAV'97*, LNCS 1254, pages 400–411.

[18] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[19] H. Comon and Y. Jurski. Multiple counters automata, safety analysis, and Presburger arithmetic. In *Proc. CAV'98*, LNCS 1427, pages 268–279.

[20] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. CONCUR'99*, LNCS 1664, pages 242–257.

[21] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL'77*, pages 238–252.

[22] C. Darlot, A. Finkel, and L. Van Begin. About Fast and TReX accelerations. In *Proc. AVoCS'04*, ENTCS (to appear).

[23] G. Delzanno, J.-F. Raskin, and L. Van Begin. Covering sharing trees: a compact data structure for parameterized verification. *JSTTT*, 5(2–3):268–297, 2004.

[24] `http://www.lsv.ens-cachan.fr/fast/`.

[25] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proc. FSTTCS'02*, LNCS 2556, pages 145–156.

[26] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems: Application to FIFO automata. *I&C*, 181(1):1–31, 2003.

[27] L. Fribourg and H. Olsén Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmetic, In *Proc. CONCUR'97*, pages 213–227.

[28] L. Fribourg. Petri nets, flat languages and linear arithmetic. In M. Alpuente, editor, *Proc. WFLP'00*, pages 344–365.

[29] O. H. Ibarra, Jianwen Su, Zhe Dang, T. Bultan, and R. A. Kemmerer. Counter machines and verification problems. *TCS*, 289(1):165–189, 2002.

[30] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256(1–2):93–112, 2001.
[31] N. Klarlund, A. Mller, and M. I. Schwartzbach. MONA implementation secrets. *JFCS*, 13(4):571–586, 2002.
[32] `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
[33] J. Leroux and G. Sutre. On flatness for 2-dimensional vector addition systems with states. In *Proc. CONCUR'04*, LNCS 3170, pages 402–416.
[34] J. Leroux and G. Sutre. Flat counter automata almost everywhere! INRIA technical report (in preparation). Submitted.
[35] J. K. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proc. PSTV '87*, pages 207–219.
[36] T. Rybina and A. Voronkov. Brain: Backward reachability analysis with integers. In *Proc. AMAST'02*, LNCS 2422, pages 489–494.
[37] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. CAV'98*, LNCS 1427, pages 88–97.

A Proofs

Notations Given a set X, for any index $i \in [1 \dots n]$, we denote by $\mathsf{x}[i]$ the i^{th} *component* of a n-tuple $\mathsf{x} \in X^n$.

A.1 Proof of theorem 4.2

Theorem *Given the symbolic framework of 2-counter systems and Presburger formulas, a 2-counter system S, and $\mathtt{x}_0 \in L^{|Q|}$, then whether $\mathsf{post}^*(\llbracket \mathtt{x}_0 \rrbracket)$ is L-definable or not is undecidable.*

Proof. We reduce the reachability problem, undecidable for 4-counter systems. It is not a restriction since 2-counter systems can encode any fixed number of counters. We consider a weaker variant, location reachability, still undecidable. The location reachability problem is the following. We consider a 4-counter system S_0 equipped with 4 variables x, y, y_0 and z ranging over $\mathbb{N}$ and a finite set Q of locations, an initial configuration (q_0, c_0) where $q_0 \in Q$ and $c_0 \in \mathbb{N}^4$, and a location $q \in Q$. Then we want to decide if there is a run of the counter system on input (q_0, c_0) such that q is reached.

Suppose that for any (S', q', c') we can decide if $\mathsf{post}^*_{S'}((q', c'))$ is definable by a Presburger formula. Let us remind that Presburger formulas cannot expressed multiplication among variables (typically $z = x \times y$). We proceed as follows. We transform S_0 into S_1 by: adding a finite number of new locations Q_1 and new transitions over Q_1, starting at $q_1 \in Q_1$, coding a multiplication of counter x by counter y, and the result is assigned to z in location $q_z \in Q_1$ (counter y_0 is used to remember the value of y during the operation). Then we had some more transitions. A transition $(q_0, “x := 0, y := 0, z := 0“, q_1)$, the transitions $(q_1, “x := x + 1“, q_1)$, $(q_1, “y := y + 1“, q_1)$, the transitions $(q, “x := x + 1“, q)$, $(q, “x := x - 1“, q)$, $(q, “y := y + 1“, q)$, $(q, “y := y - 1“, q)$ $(q, “z := z + 1“, q)$, $(q, “z := z - 1“, q)$ and for all $q'' \in Q_0 \cup Q_1$ a transition $(q, “x := x, y := y, z := z“, q'')$. Then it is easy to verify that $\mathsf{post}^*_{S_1}((q_0, c_0))$ is L-definable (and equals to $(Q_0 \cup Q_1) \times \mathbb{N}^3$) iff q is reachable (otherwise the reachability set projected on q_z is $\{(x, y, z) | z = x \times y\}$). ☐

A.2 Proof of theorem 4.5

Theorem.

1. *When* REACH1 *terminates,* $\llbracket \text{REACH1}(\mathtt{x}_0) \rrbracket = \mathsf{post}^*(\llbracket \mathtt{x}_0 \rrbracket)$ (partial correctness).
2. REACH1 *terminates on any input iff S is L- uniformly bounded* (termination).

Proof. Partial correctness: when the procedure terminates, $\text{REACH1}(\mathtt{x}_0)$ is a fixpoint of POST, then $\llbracket \text{REACH1}(\mathtt{x}_0) \rrbracket$ is a fixpoint of post. Moreover at each iteration of the procedure, $\llbracket \mathtt{x} \rrbracket \subseteq \mathsf{post}^*(\llbracket \mathtt{x}_0 \rrbracket)$. This ensures that $\llbracket \text{REACH1}(\mathtt{x}_0) \rrbracket$ is equal to the least fixpoint of post, i.e. $\mathsf{post}^*(\llbracket \mathtt{x0} \rrbracket)$.

Termination: We suppose S is L- uniformly bounded. Given $\mathtt{x}_0 \in L^{|Q|}$, there exists $n_{\mathtt{x}_0}$ such that $\mathsf{post}^*(\llbracket \mathtt{x}_0 \rrbracket) = \bigcup_{i \leq n_{\mathtt{x}_0}} \mathsf{post}^i(\llbracket \mathtt{x}_0 \rrbracket)$. It is straightforward that after $n_{\mathtt{x}_0}$ iterations, REACH1 terminates. We suppose now that REACH1 terminates on any input. Then for any $\mathtt{x}_0 \in L^{|Q|}$, the fixpoint is reached after $n_{\mathtt{x}_0}$ iterations. $n_{\mathtt{x}_0}$ is the constant in the definition of L- uniformly bounded. ☐

A.3 Proof of proposition 5.2

Proposition. *Given a system S supporting flat acceleration, then for any* rlre ρ *over* T *and for any* $\mathtt{x} \in L^{|Q|}$, $\mathsf{post}(\rho, [\![\mathtt{x}]\!])$ *is L-definable.*

Proof. We reason by induction on ρ. If $\rho = \varepsilon$ then $[\![\mathtt{x}]\!] = \mathsf{post}(\varepsilon, [\![\mathtt{x}]\!])$ and the property is true. Otherwiser if $\rho = u^* \cdot \rho_1$ where $u \in T^*$, then we apply the induction hypothesis to $\mathsf{post}(\rho_1, [\![\text{POST\_STAR}(u, x)]\!])$. □

A.4 Proof of proposition 5.4

Proposition. *Given a flat system S supporting flat acceleration, then* $\mathsf{post}^*([\![\mathtt{x}]\!])$ *is effectively L-definable.*

Proof. In [33] it is proved that for a flat system S, there exists a semi-linear regular expression (slre) ρ' over T such that for all $\mathtt{x} \in L^{|Q|}$, $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho', [\![\mathtt{x}]\!])$. Moreover it is proved that ρ' is effectively computable. Recall that a slre over T is a regular linear expression of the form $\Sigma_i u_{i,1} w_{i,1}^* \dots u_{i,n} w_{i,n}^*$, where $u_{i,j}, w_{i,j} \in T^*$. Let us define the rlre ρ over T by $\rho = \Pi_i u_{i,1}^* w_{i,1}^* \dots u_{i,n}^* w_{i,n}^*$. It is easy to verify that $\mathsf{post}(\rho, [\![\mathtt{x}]\!]) = \mathsf{post}(\rho', [\![\mathtt{x}]\!]) = \mathsf{post}^*([\![\mathtt{x}]\!])$. Using proposition 4.2, we are done. □

A.5 Proof of theorem 5.7

Theorem. *Let S be a L- flattable system supporting flat acceleration. Then* $\mathsf{post}^*([\![\mathtt{x}]\!])$ *is effectively L-definable.*

Proof. First notice that since regions are closed by finite union ($\sqcup$ operator), z is easily extended into $z : L^{|Q'|} \to L^{|Q|}$. This construction is effective. Consider a system S and $\mathtt{x} \in L^{|Q|}$, we enumerate all $(S', z, \mathtt{x}')$ such that S' is a flattening of S with folding z, and $\mathtt{x}' \in L^{|Q'|}$ such that $[\![z(\mathtt{x}')]\!] = [\![\mathtt{x}]\!]$ (using $\sqsubseteq$). For each $(S', z, \mathtt{x}')$, since S' is flat we can compute $\mathtt{x}''$ such that $[\![\mathtt{x}'']\!] = \mathsf{post}^*_{S'}([\![\mathtt{x}']\!])$ (proposition 5.4). Then we compute $\mathtt{y} = z(\mathtt{x}'') \in L^{|Q|}$ and check whether $\text{POST}_S(\mathtt{y}) \sqsubseteq \mathtt{y}$ or not. When it is the case then $[\![\mathtt{y}]\!]$ is an invariant of post_S. By definition of flattenings and construction of $\mathtt{y}$, $[\![\mathtt{y}]\!]$ ne peut qu'être inférieur à $\mathsf{post}^*([\![\mathtt{x}]\!])$. Donc $[\![\mathtt{y}]\!] = \mathsf{post}^*([\![\mathtt{x}]\!])$. Since S is L-flattable, such a $(S', z, \mathtt{x}')$ exists and will eventually be found (even if there are finitely many $(S', z, \mathtt{x}')$, they can be enumerated). □

A.6 Proof of theorem 5.8

Theorem *Given the symbolic framework of 2-counter systems and Presburger formulas, then whether a 2-counter system S is L- flattable or not is undecidable.*

Proof. This is essentially the same proof than theorem 4.2. The location reachability problem is reduced in the same way. Notice that q is reachable iff S_1 is L- flattable (If q is reachable, compute $\mathbb{N}^3$ on q then use each new transition once to propagate $\mathbb{N}^3$ on every location $q'' \in Q \cup Q_1$; otherwise $\mathsf{post}^*_{S_1}(c)$ is not L-definable for any c, then S_1 cannot be L- flattable). □

A.7 Proof of theorem 6.1

Theorem. *A system $S = (Q, \Sigma, T, D, [\![\cdot]\!])$ is L- flattable iff for all* $\mathtt{x} \in L^{|Q|}$*, there exists a* rlre *ρ over T such that* $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}]\!])$.

Proof. Given $\mathtt{x} \in L^{|Q|}$, if there exists a rlre $\rho_{\mathtt{x}}$ such that $\mathsf{post}^*_S([\![\mathtt{x}]\!]) = \mathsf{post}_S(\rho_{\mathtt{x}}, [\![\mathtt{x}]\!])$, we deduce naturally a flattening S'_X of S (intuitively the uninterpreted system of S'_X is the automata recognizing the langage $\rho_X \subseteq T^*$).

Let us prove the converse. Let us assume that S is L-flattable. By definition there exists a flat system S', a flattening z and $\mathtt{x}'$ such that $z([\![\mathtt{x}']\!]) = [\![\mathtt{x}]\!]$ and $z(\mathsf{post}^*_{S'}\,[\![\mathtt{x}']\!]) = \mathsf{post}^*_S([\![\mathtt{x}]\!])$. Moreover we can build effectively $(S', z, \mathtt{x}')$ by enumeration (see proof of theorem 5.7). Since S' is flat, using the proof of proposition 5.4 we deduce that there exists ρ' a rlre over T' verifying $\mathsf{post}^*_{S'}([\![\mathtt{x}']\!]) = \mathsf{post}_{S'}(\rho', [\![\mathtt{x}']\!])$. We denote $\rho = z(\rho')$. By definition of flattening, ρ is a rlre over T (each transition of a flattening corresponds to a transition in the original system, the property extends to sequences and languages). By reasoning on sequences of transitions and then languages, we can prove that for any $\mathcal{L} \subseteq T^*$, $z(\mathsf{post}_{S'}(\mathcal{L}, [\![\mathtt{x}']\!])) = \mathsf{post}_S(z(\mathcal{L}), z([\![\mathtt{x}']\!]))$. We then deduce that $z(\mathsf{post}_{S'}(\rho', [\![\mathtt{x}']\!])) = \mathsf{post}_S(z(\rho'), z([\![\mathtt{x}']\!])) = \mathsf{post}_S(\rho, z([\![\mathtt{x}']\!]))$.

It comes that there exists $\mathtt{x}'$ such that $z([\![\mathtt{x}']\!]) = [\![\mathtt{x}]\!]$ and $\mathsf{post}^*_S([\![\mathtt{x}]\!]) = z(\mathsf{post}^*_{S'}([\![\mathtt{x}']\!])) = z(\mathsf{post}_{S'}(\rho', [\![\mathtt{x}']\!])) = \mathsf{post}_S(\rho, [\![\mathtt{x}]\!])$. □

A.8 Proof of theorem 6.2

Theorem.

1. *When* REACH2 *terminates,* $[\![\text{REACH2}(\mathtt{x}_0)]\!] = \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (partial correctness).
2. REACH2 *terminates on any input iff S is L- flattable* (termination).

Proof. **Partial correctness**: straightforward from the definition of POST_STAR and $\sqsubseteq$. **Termination**: first remark that if REACH2 terminates it returns the fixpoint, since computations in our procedure are always underapproximations of the reachability set. The finite sequence of selected $w \in T^*$ during the successful computation provides a rlre ρ over T^* such that $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}]\!])$. Thus if REACH2 terminates for all input, then S is L- flattable (theorem 6.1). Assume now that S is L- flattable, and consider $\mathtt{x}_0 \in L^{|Q|}$. There exists a rlre ρ over T^* such that $\mathsf{post}^*([\![\mathtt{x}_0]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}_0]\!])$ (theorem 6.1). Let us denote $\rho = u_1^* \dots u_n^*$. Since Choose is fair, the sequence ρ' of w selected by choose will eventually be of the form $\rho' = w_1^* \dots w_m^*$ where there exists $i_1, \dots, i_n$ such that $w_{i_1} = u_1, \dots, w_{i_n} = u_n$. It will eventually be the case because all $w \in T^*$ are repeated infinitely often thanks to fairness condition. Moreover the identity relation being contained in each step of acceleration, each step of computation contains entirely the previous step. Then we get that $\mathsf{post}(\rho, [\![\mathtt{x}_0]\!]) \subseteq \mathsf{post}(\rho', [\![\mathtt{x}_0]\!]) \subseteq \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (remember we can only compute underapproximation). Since $\mathsf{post}^*([\![\mathtt{x}_0]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}_0]\!])$, we get that $\mathsf{post}^*([\![\mathtt{x}_0]\!]) = \mathsf{post}(\rho', [\![\mathtt{x}_0]\!])$ and the computation will stop at that stage (returning the fixpoint). □

A.9 Proof of theorem 6.3

Theorem.
1. *When* REACH3 *terminates,* $[\![\text{REACH3}(\mathtt{x}_0)]\!] = \mathsf{post}^*([\![\mathtt{x}_0]\!])$ (partial correctness).
2. REACH3 *terminates for any input iff S is L- flattable* (termination).

Proof. **Partial correctness**: straightforward from the definition of POST_STAR and $\sqsubseteq$.
Termination: fairness of Choose on $T^{\leq k}$ and Watchdog, and re-using computations of each previous task k-flattable ensure fairness of Choose on T^*. Then we use the same arguments than for theorem 6.2. □

A.10 Proof of proposition 6.5

Proposition. *The conjugation reduction and the commuting reduction are reductions. The conjugation reduction satisfies* $|T'| = \mathcal{O}(\frac{|T^k|}{k})$.

Proof. **Conjugation reduction.** Given three transitions t_1, t_2 and t_3, we do not need to consider $t_2 \cdot t_3 \cdot t_1$ and $t_3 \cdot t_2 \cdot t_1$ since $\xrightarrow{(t_2 \cdot t_3 \cdot t_1)^*}$ and $\xrightarrow{(t_3 \cdot t_2 \cdot t_1)^*}$ can be computed easily from $\xrightarrow{(t_1 \cdot t_2 \cdot t_3)^*}$. For example $\xrightarrow{(t_2 \cdot t_3 \cdot t_1)^*} = Id \cup \xrightarrow{t_2} \bullet \xrightarrow{t_3} \bullet \xrightarrow{(t_1 \cdot t_2 \cdot t_3)^*} \bullet \xrightarrow{t_1}$.
Commuting reduction. If t_1 and t_2 satisfies $\xrightarrow{t_1 \cdot t_2} = \xrightarrow{t_2 \cdot t_1}$ then $\xrightarrow{(t_1 \cdot t_2)^*}$ is equal to $\xrightarrow{t_1^*} \bullet \xrightarrow{t_2^*}$, therefore we can remove safely both $t_1 \cdot t_2$ and $t_2 \cdot t_1$. □

B Practical use of flat acceleration: FAST

The following table shows performances of FAST on a significant pool of counter systems, mainly collected among web sites of other model-checkers, like ALV, BABYLON[4], BRAIN, LASH and TREX. They range from tricky academic puzzles (swimming pool) to complex industrial protocols (TTP). Since most of them have infinite state-space (except whose in category *Bounded Petri Nets*), they are beyond the scope of traditional model-checking techniques and tools. Moreover, most of these systems are also beyond VASS/Petri nets, then nice methods for WSTS like covering tree or backward computation do not work anymore.

The results are taken from forward computation of the reachability set, using an Intel Pentium 933 Mhz with 512 Mbytes. In the following table, $|T|$ is the number of transitions, $|\mathcal{A}|$ is the size of the computed binary automaton (regions of FAST). $|w|$ is the length of the rlre computed so far, k is the maximal length of cycle ($T^{\leq k}$), the number of cycles is given after reductions (commuting transitions and specific reduction of subsection 6.3). "-" indicates an unknown result (termination does not terminate under 1800 seconds).

There can have several reasons for FAST non termination: (1) the reachability set is not Presburger definable, (2) the system is not L- flattable, (3) the monoid of the system is not finite and our techniques on counter systems do not applied, (4) the three previous conditions does not hold but time and space consumption are too high. Even if

[4] `http://www.ulb.ac.be/di/ssd/lvbegin/CST/`

it is difficult to understand why some experiments fail it seems that on these examples, the main factor of failure is (4). At least, since most of the systems are variants of VASS (with zero test or reset), the monoid is finite.

Case study	variables	$\|T\|$	time (s)	mem. (MB)	$\|A\|$	$\|w\|$	k	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
Unbounded 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
Unbounded Counter Systems								
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
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	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
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
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