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 [19] for infinite-state systems is now an active field of research. These techniques allow considering models like pushdown systems [13], channel systems [1, 14], counter systems [8, 31, 38], 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 [37], LASH [33], TREX [3], and our own FAST [8]. 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 [32].

The problem of convergence. When dealing with infinite-state systems, a naive fixpoint computation procedure for reachability sets, in the style of Procedure 1 (section 3.2), 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 3.2). To

* This work was supported by the ACI Sécurité & Informatique (project Persée) funded by the French Ministry of Research.

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, 12, 14, 26, 38]. Some of them have been implemented [3, 8, 33] and promising case-studies have been reported [1–3, 8, 9]. Acceleration shares some similarities with the widening techniques used in abstract interpretation [22] but also exhibits some clear differences: acceleration aims at exact computation for some given control structures, while widening mostly ignores control structures and usually 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 acceleration 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 outcomes 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 contributions. (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 [31] 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. We show Procedure REACH2 terminates iff it is applied to a flattable (rather than flat) system, which is the first completeness result on symbolic model checking with acceleration. We 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 6) for which tools like LASH and ALV, based on similar technology but restricted heuristics, do not terminate. More generally, the comparisons in section 6 suggest that flat acceleration greatly enhances termination of symbolic reachability set computation, and is fully justified in practice.

Outline. We define the systems under study in section 2, and the symbolic frameworks in section 3. Section 4 introduces the three levels of accelerations and defines flattable systems. Section 5 provides our procedure for flattable systems, and gives several algorithmic and/or heuristic refinements. Section 6 compares several existing tools through the new framework. All omitted proofs can be found in the full version of this paper.

2 Systems and interpretations

Notations. A *(binary) relation* r on some set X is any subset of $X \times X$. We write $x\,r\,x'$ when $(x, x') \in r$ and denote by $r(x)$ the set $\{x' \in X \mid x\,r\,x'\}$. For $Y \subseteq X$, $r(Y)$ is $\bigcup_{x \in Y} r(x)$. Given $r_1, r_2 \subseteq X \times X$, the *compound relation* $r_1 \bullet r_2$ contains all pairs (x, z) s.t. $x\,r_1\,y$ and $y\,r_2\,z$ for some $y \in X$. Note that, in $r_1 \bullet r_2$, relation r_1 is applied first. For $i \in \mathbb{N}$, r^i is defined inductively by $r^0 = Id_X$ and $r^{i+1} = r \bullet r^i$, where Id_X is the *identity* on X. $r^* = \bigcup_{i \in \mathbb{N}} r^i$ is the *reflexive and transitive closure* of r.

Here, a *system* is a finite state control graph extended with a finite number of variables that range over arbitrary domains and are modified by actions when a transition is fired. Specific families of systems have been widely studied (see section 2.1). Formally:

Definition 2.1 (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 formulae 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 2.2 (Interpretation). *Given a (possibly infinite) set of formulae* 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 formulae to relations on* D*.*

Definition 2.3 (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.

In this example the actions may be assignments that can be guarded by Boolean expressions, but we will not specify it more precisely. A possible interpretation for a_1 ,a_2 and a_3 (the actions appearing in S_0) 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.

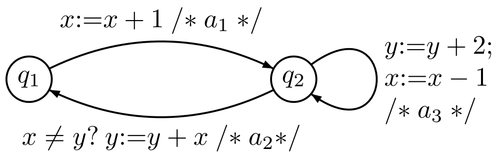


Fig. 1. S_0, a simple uninterpreted system

Behaviour. The *set of configurations* $\mathcal{C}_S$ of S is $Q \times D$, and each transition $t \in T$ is interpreted as a relation $\xrightarrow{t} \subseteq \mathcal{C}_S \times \mathcal{C}_S$ 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 extends to sequences $\pi \in T^*$ of transitions. Let ε denote 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 $[\![\cdot]\!]$ is extended to any language $\mathcal{L} \subseteq \Sigma^*$. In the following we omit the S subscript whenever this causes no ambiguities.

Reachability problems. We are interested in checking *safety properties*, which can be expressed in terms of reachability using standard techniques. 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 $\mathsf{post}(T, X)$ of all configurations reachable in one step from X is denoted by $\mathsf{post}(X)$. The set $\mathsf{post}(T^*, X)$ of all configurations reachable from X is *the reachability set of* X, denoted by $\mathsf{post}^*(X)$.

In practice, we usually ask whether $\mathsf{post}^*(X_0) \subseteq P$, for X_0 a set of initial configurations, and P a set of "safe" configurations. We focus here on the reachability set computation which is the key issue. Since $\mathsf{post}^*(X_0)$ is not recursive in general, the best we can hope for are partially correct procedures, with no guarantees of termination, but that are efficient on interesting subclasses of systems, and in practical case-studies.

Backward computation. One may also rely on *backward reachability* and check if, for a set P of "bad" configurations, $\mathsf{pre}^*(P) \cap X_0$ is empty (with obvious definition for pre). Since, for our level of abstraction, adaptation to backward computation is straightforward, we consider only forward computation. However it is worth remembering that, depending on the case at hand, 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* $\xrightarrow{T^*}$ once and for all (e.g., [21, 25]). Then $\mathsf{post}^*(X_0) = \xrightarrow{T^*} (X_0)$. Our framework extends smoothly in this direction but, since it requires additional notations, we postpone this until the full version of this work.

2.1 Families of systems

Definition 2.4 (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 Definition 2.4:
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 [18, 34]: 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 [26, 38], reversal-counter systems [31], 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 [17]: 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).

3 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 [15, 32, 22].[3]

Definition 3.1 (Symbolic framework). *A* symbolic framework *is a tuple $SF = (\Sigma, D, [\![\cdot]\!]_1, L, [\![\cdot]\!]_2)$ where $I = (\Sigma, D, [\![\cdot]\!]_1)$ is an* interpretation*, L is a set of formulae called* regions*, $[\![\cdot]\!]_2 : L \to 2^D$ is a* region concretization*, and such that there exists 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. $\mathrm{POST} : \Sigma \times L \to L$ is such that $\forall a \in \Sigma, \forall \mathtt{x} \in L$, $[\![\mathrm{POST}(a, \mathtt{x})]\!]_2 = [\![a]\!]_1 ([\![\mathtt{x}]\!]_2)$.

[3] Some weakened versions of the symbolic framework are sometimes considered. 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 [22]). In the following, we do not consider weakened framework.

Notation. Usually given an interpretation $I = (\Sigma, D, [\![\cdot]\!]_1)$ and a set of regions L, $[\![\cdot]\!]_2$ is understood. 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 referring to a system S, if nothing is specified we assume that $S \in \mathcal{F}(I)$.

Well-known symbolic frameworks for some of the families listed in section 2.1 are:
Regular languages: have been used for representing sets of configurations of pushdown systems [13], distributed protocols over rings of arbitrary size [32], and channel systems [36]. Restricted sets of regular languages are sometimes used for better algorithmic efficiency: languages closed by the subword relation [1] or closed by semi-commutations [16].
(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 [18, 26]: are automata recognizing subsets of $\mathbb{Z}^{Var}$ and have been used in the analysis of counter systems.
Real Vector Automata [11]: 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 [24]: are a compact representation for upward-closed subsets of $\mathbb{N}^{Var}$. These sets appear naturally in the backward analysis of broadcast protocols [26] 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 $q_1, \dots, q_{|Q|}$, we work on tuples of regions in $L^{|Q|}$. We extend $[\![\cdot]\!]$ to $L^{|Q|}$ by $[\![(\mathtt{x}_1, \dots, \mathtt{x}_{|Q|})]\!] = \bigcup_{i=1}^{|Q|} \{q_i\} \times [\![\mathtt{x}_i]\!]$. Extensions of $\sqsubseteq$ and $\sqcup$ are component-based. $\textsc{post}$ is extended into $\textsc{post} : T \times L \to L$ by: $\textsc{post}((q_i, a, q_j), (\mathtt{x}_1, \dots, \mathtt{x}_{|Q|}))$ is equal to $(\mathtt{x}'_1, \dots, \mathtt{x}'_{|Q|})$ such that $\mathtt{x}'_p = \perp$ 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 $\textsc{post}(\mathtt{x}) = \bigsqcup_{t \in T} \textsc{post}(t, \mathtt{x})$.

3.1 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 $[\![\mathtt{x}]\!] = X$. Obviously, computing $\mathsf{post}^*(X)$ using regions is feasible only if $\mathsf{post}^*(X)$ is L-definable and the question "is $\mathsf{post}^*([\![\mathtt{x}]\!])$ L-definable?" is undecidable in general.

Furthermore, L-definability of $\mathsf{post}^*(X)$ is not a sufficient condition for its computability. We say below that post^* (or any other function) is *effectively L-definable* if there exists a recursive function $g : L^{|Q|} \to L^{|Q|}$ such that $\forall \mathtt{x} \in L^{|Q|}$, $\mathsf{post}^*([\![\mathtt{x}]\!]) = [\![g(\mathtt{x})]\!]$. (We often abuse terminology and write that "$\mathsf{post}^*([\![\mathtt{x}]\!])$", instead of post^*, "is effectively L-definable"). It can well be the case that $\mathsf{post}^*([\![\mathtt{x}]\!])$ is L-definable but not effectively so (e.g., the family of lossy channel systems and the framework defined by simple regular expressions).

3.2 Standard symbolic model checking procedure

REACH1 (Procedure 1) is the standard symbolic procedure for reachability sets. It is only guaranteed to terminate on L-uniformly bounded 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 (forward version)

Definition 3.2 (**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 $c_2 \in \bigcup_{i \leq n_{\mathtt{x}}} \mathsf{post}^i(\{c_1\})$.*

Theorem 3.3. *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 3.4. Termination for L-uniformly bounded systems does not hold if $\sqsubseteq$ or $\sqcup$ are weak.

In practice systems are rarely L-uniformly bounded and Procedure 1 seldom terminates. A notable exception are the well-structured transition systems with upward-closed sets as regions [28, 27]. They are *L-backward* uniformly bounded so that a backward version of Procedure 1 always terminates.

4 Flat acceleration for flattable systems

4.1 Acceleration techniques

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

Definition 4.1 (**Acceleration**). *A symbolic framework $SF = (I, L)$ supports*

1. loop acceleration *if there exists a recursive function* POST_STAR : $\Sigma \times L \rightarrow L$ *s.t. $\forall a \in \Sigma$, $\forall \mathtt{x} \in L$, $[\![\text{POST\_STAR}(a, \mathtt{x})]\!] = [\![a^*]\!]\,([\![\mathtt{x}]\!])$;*

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

We often write that "S", rather than (I, L), "supports loop acceleration, or flat, ..."

Consider S_0 from Fig. 1 and let $A \subseteq D$. Loop acceleration only concerns action a_3, and comes down to computing $[\![a_3^*]\!]\,(A) = \{(x', y') \in \mathbb{Z}^2 | \exists (x, y) \in A; \exists k \in \mathbb{N}; x' = x - k \wedge y' = y + 2 \cdot k\}$. Flat acceleration requires computability of $[\![(a_1 \cdot a_2)^*]\!]\,(A)$, $[\![(a_1 \cdot a_3 \cdot a_2)^*]\!]\,(A)$, $[\![(a_1 \cdot a_3 \cdot a_3 \cdot a_2)^*]\!]\,(A)$, $[\![(a_3 \cdot a_2 \cdot a_1)^*]\!]\,(A)$ and so on. Global acceleration requires the computation of more complex interleaving of actions, like $[\![(a_1 \cdot a_3^* \cdot a_2)^*]\!]\,(A)$.

Definition 4.1 applies to symbolic frameworks and hence uses sequences of actions. However, in practice, POST_STAR is used with sequences of transitions. Let us illustrate this in the case of flat acceleration: Consider a sequence $\pi = (q_1, a_1, q_2) \cdot (q_3, a_2, q_4) \cdot (q_5, a_3, q_6)$ of transitions. 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 $\text{POST\_STAR}(\pi, (q, \mathtt{x}))$ is $(q, \mathtt{x})$. If the sequence is *valid*, then the sequence is equivalent to $(q_1, a_1 \cdot a_2 \cdot a_3, q_6)$. If the sequence is not a cycle ($q_1 \neq q_6$) it can be fired at most once and $\text{POST\_STAR}(\pi, (q_1, \mathtt{x}))$ is $(q_6, \text{POST}(a_1 \cdot a_2 \cdot a_3, \mathtt{x})) + (q_1, \mathtt{x})$. If the sequence is a cycle (i.e., $q_1 = q_6$) then $\text{POST\_STAR}(\pi, (q, \mathtt{x}))$ is $(q_1, \text{POST\_STAR}(a_1 \cdot a_2 \cdot a_3, \mathtt{x}))$ if $q = q_1$, and $(q, \mathtt{x})$ otherwise. Finally POST_STAR is extended to $L^{|Q|}$ in the obvious way. The extension for global acceleration considers the intersections of the regular language e with the regular languages of transitions from a location q to another location q'.

Example 4.2. **Loop acceleration.** Minsky machines support loop acceleration in frameworks where formulae in L define *upward-closed sets* or *semi-linear sets*. But upward-closed sets (for example) are not expressive enough to support flat acceleration.
Flat acceleration. Counter systems (with finite monoid) equipped with Presburger formulae supports flat acceleration [26, theorem 2]. Other examples are channel systems with cqdd [14, theorem 5.1], non-counting channel systems with slre [27, theorem 5.2] or qdd[12, 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]: their POST_STAR is not recursive.
Global acceleration. Reversal-counter systems [31], 2-dim VASS [34], lossy VASS and other subclasses of VASS with Presburger formulae [35], pushdown systems with regular languages or semi-commutative rewriting systems with APC languages [16], support global acceleration.

Obviously "global $\Rightarrow$ flat $\Rightarrow$ loop". Loop acceleration is often easy to obtain, but rarely sufficient in fixpoint computations. Flat acceleration is more flexible, but often requires good compositional properties of Σ and more complex methods for POST_STAR. Global acceleration is a very strong property that ensures post^* is effectively L-definable. Clearly most interesting families of systems 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.

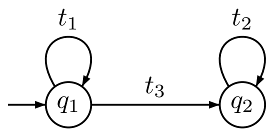
4.2 Restricted linear regular expressions

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 $u_1^* \dots u_n^*$ where, for all i, $u_i \in A^*$. This is closely related to semi-linear regular expressions [27, 30].

Proposition 4.3. *Let S support flat acceleration. Then for any rlre ρ over T and for any $\mathtt{x}_0 \in L^{|Q|}$, $\mathsf{post}(\rho, [\![\mathtt{x}_0]\!])$ is effectively L-definable.*

4.3 Flat systems and flattenings

In general, flat acceleration does not ensure computability of the reachability set. However it does in some cases, for example with "*flat*" systems, that have *no nested loops*. Consider the system on the right: its reachability set can be computed by iterating first t_1, then firing t_3, and finally iterating t_2.



Definition 4.4 (Flat system [20, 27, 30]). *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 $S = (\Sigma, Q, T, D, [\![\cdot]\!])$ is* flat *if $\mathsf{S} = (\Sigma, Q, T)$ is flat.*

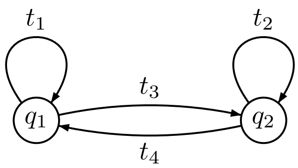
In Fig. 1, S_0 is not flat because its two elementary cycles both visit q_2.

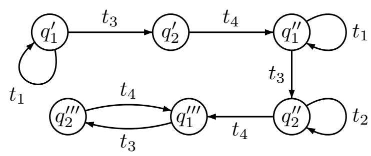
Proposition 4.5. *If S is a flat system supporting flat acceleration, then $\mathsf{post}_S^*([\![\mathtt{x}]\!])$ is effectively L-definable.*

Not all systems of interest are flat, and a possible method for dealing with a non-flat system S is to find an equivalent flat system, called a *flattening* of S.

Definition 4.6 (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 is a form of partial unfolding. The following figure shows a system (left) and one of its flattenings (right).





Assume S' is a flattening of some S. The z folding extends to configurations of S' by $z((q', \mathsf{x})) = (z(q'), \mathsf{x})$. Extension of z to $X \subseteq \mathcal{C}_{S'}$ is defined by:

$$z\Big(\bigcup_{q' \in Q'} \{q'\} \times D_{q'}\Big) \;=\; \bigcup_{q \in Q} \{q\} \times \Big(\bigcup_{q' \in z^{-1}(q)} D_{q'}\Big).$$

This gives an effective extension of z to L-definable subsets of $\mathcal{C}_{S'}$. Given $X' \subseteq \mathcal{C}_{S'}$, Definition 4.6 ensures that $z(\mathsf{post}_{S'}^*(X')) \subseteq \mathsf{post}_S^*(z(X'))$ and that for any language $\mathcal{L} \subseteq T^*$, $z(\mathsf{post}_{S'}(\mathcal{L}, [\![\mathsf{x}']\!])) = \mathsf{post}_S(z(\mathcal{L}), z([\![\mathsf{x}']\!]))$.

Definition 4.7 **(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 a $\mathtt{x}' \in L^{|Q'|}$ such that $z([\![\mathtt{x}']\!]) = [\![\mathtt{x}]\!]$ and $z(\mathsf{post}^*_{S'}([\![\mathtt{x}']\!])) = \mathsf{post}^*_S(z([\![\mathtt{x}']\!]))$.*

Prop. 4.5 extends to flattable systems:

Theorem 4.8. *If S is a L-flattable system supporting flat acceleration, then $\mathsf{post}^*_S([\![\mathtt{x}]\!])$ is effectively L-definable.*

A natural question is whether L-flattable systems are common or rare. It appears that many systems with L-definable reachability sets are flattable. For example 2-dim VASS [34], timed automata [21], k-reversal counter machines, lossy VASS and other subclasses of VASS [35] and all L-uniformly bounded systems (see section 3) are L-flattable. Clearly, 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.

We conclude by noting that L-flattability is undecidable in general, even when restricting to 2-counter systems:

Theorem 4.9. *Assuming the symbolic framework of 2-counter systems and Presburger formulae, the question of whether a 2-counter system S is L-flattable is undecidable.*

5 Computing reachability set using flat acceleration

The previous characterization leads to a complete procedure for flattable systems: (1) enumerate all flattenings S' of S; (2) for each S', compute its reachability set X; (3) check whether $z(X)$ is closed by post in S.

However flattenings are not easy to handle and this motivates the following alternative characterization based on rlre's.

Theorem 5.1. *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 $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}]\!])$.*

Hence 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: see Procedure 2. Observe that REACH2 must choose "*fairly*". Here this means that, in a nonterminating execution of the procedure, each $w \in T^*$ is selected infinitely often. Many simple schemes ensuring such a fair choice are possible.

Theorem 5.2. *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).

Remark 5.3. Termination for L-flattable systems does not hold if the symbolic framework provides only a weak inclusion (or if POST_STAR returns an over-approximation).

procedure $\text{REACH2}(\mathtt{x}_0)$
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**
    Choose fairly $w \in T^*$
    $\mathtt{x} \leftarrow \text{POST\_STAR}(w, \mathtt{x})$
end while
return $\mathtt{x}$

Procedure 2: Computing reachability sets with flat acceleration.

5.1 Faster enumeration of flattenings

A major practical issue with REACH2 is to implement Choose so that we converge quickly to the fixpoint. For this purpose the following heuristic proved very efficient in FAST: one picks a bound $k \in \mathbb{N}$ and restricts Choose to sequences $w \in T^{\leq k}$, i.e., of length at most k. This method, called k-flattable, is eventually stopped by a Watchdog if it does not terminate. Then k is incremented and k-flattable is launched again.

This leads to Procedure REACH3 below. For "fairness" we require that Watchdog fires infinitely often, but only after Choose picked each $w \in T^{\leq k}$ at least once.

procedure $\text{REACH3}(\mathtt{x}_0)$
parameter: S
input: $\mathtt{x}_0 \in L^{|Q|}$
$\mathtt{x} \leftarrow \mathtt{x}_0$; $k \leftarrow 0$
$k \leftarrow k + 1$
start
    while $\text{POST}(\mathtt{x}) \not\sqsubseteq \mathtt{x}$ **do**   /* k-flattable */
        Choose fairly $w \in T^{\leq k}$
        $\mathtt{x} \leftarrow \text{POST\_STAR}(w, \mathtt{x})$
    end while   /* end k-flattable */
with
    when Watchdog stops **goto** 2
return $\mathtt{x}$

Procedure 3: Flat acceleration and circuit length increasing

Theorem 5.4. *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. When implementing REACH3 one faces (at least) two practical problems. First the *size* [4] of the region $\mathtt{x}$ computed so far may be explosive. Then Watchdog

[4] Each set of regions has its own natural measure for size, depending on data structures and implementation.

needs some criterion. Below we describe the implementation choices made in FAST on these two issues, believing that these solutions may adapt to other domains. Let us point out that these choices do not respect exactly the specification for REACH3 since fairness is not ensured, and FAST should be improved in this way.

Choose: In general there is no direct relationship between the size of a region $\mathtt{x}$ and the "size" of its concretization $[\![\mathtt{x}]\!]$. Intermediate regions may be much larger than the final region for $\mathsf{post}^*([\![\mathtt{x}_0]\!])$. To avoid such large regions, Choose selects a next $w \in T^{\leq k}$ such that $|\text{POST\_STAR}(w, \mathtt{x})| < |\mathtt{x}|$. If there is no such w then the size of the current $\mathtt{x}$ is allowed to increase and the next w is picked. In practice, this enumeration works well (while a cyclic enumeration of $T^{\leq k}$ almost always runs out of memory).

Watchdog: FAST's criterion is simply a fixed (but user-modifiable) limit on the number of iterations in k-flattable for any given value of k. This cannot be fair but it works well in practice since, once a k large enough is considered, the fixpoint is usually found within a few iterations.

5.2 Reduction of the number of cycles

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

Definition 5.5 **(k-Reduction).** *Given an interpretation $I = (\Sigma, D, [\![\cdot]\!])$, a k-reduction r maps each system $S = (Q, \Sigma, T, D, [\![\cdot]\!]) \in \mathcal{F}(I)$ to a system $S' = (Q, \Sigma, T', D, [\![\cdot]\!]) \in \mathcal{F}(I)$ 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) $|T'| \leq |T^{\leq k}|$.*

Hence a k-reduction replaces T by a new set T' that can stand for $T^{\leq k}$ but is smaller. In particular, if S is L-flattable, then $r(S)$ is too, and they both have the same reachability set. Obvious (and naive) k-reductions are the removals of identity loops. More useful generic reductions are *conjugation reduction*: only keep one sequence of transitions among each conjugacy class (e.g., keep $t_1 \cdot t_2 \cdot t_3$ but remove $t_2 \cdot t_3 \cdot t_1$ and $t_3 \cdot t_1 \cdot t_2$) and *commuting reduction*: if t_1 and t_2 commute, i.e., if $\xrightarrow{t_1 t_2} = \xrightarrow{t_2 t_1}$, then remove both $t_1 \cdot t_2$ and $t_2 \cdot t_1$ (works since $\xrightarrow{(t_1 \cdot t_2)^*} = \xrightarrow{t_1^* t_2^*}$).

Proposition 5.6. *Conjugation reduction and commuting reduction are k-reductions. Conjugation reduction satisfies $|T'| = \mathcal{O}(\frac{|T^k|}{k})$.*

Beyond these generic reductions, it is worth developing reductions dedicated to a specific interpretation. For linear counter systems with a finite monoid, [26] presents a reduction where $|T'|$ remains polynomial in k (while $|T^{\leq k}|$ is exponential). This appears to be a key reason for FAST's performances.

Here are reduction results for the swimming pool protocol (a VASS with 7 transitions and 6 variables studied in [29]). Computing the reachability set requires considering cycles of length $k = 4$. In the table $V_k \subseteq T^{\leq k}$ is the set of *valid sequences in* $T^{\leq k}$. T' (resp. T'') is from the system after the reduction of [26] (resp. further combined with commuting reduction).

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

6 Conclusion: flat acceleration in practice

6.1 Tools comparison

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

Our framework is useful when comparing ALV, FAST, LASH and TREX, four symbolic model checkers that can perform reachability analysis on counter systems (see section 2.1). We restrict this comparison to the exact forward computation of $\mathsf{post}^*(\llbracket \mathtt{x}_0 \rrbracket)$. ALV [6] handles full counter systems. Regions are Presburger formulae. The heuristic used is similar to REACH1. Both FAST [8] and LASH [33] handle linear counter systems with Presburger formulae: flat acceleration is supported for functions whose monoid is finite, but while FAST really takes advantage of full flat acceleration (Procedure REACH3), the heuristics in LASH are restricted to loop acceleration (Procedure REACH2 where w is chosen in $T^{\leq 1}$ instead of T^*). TREX [3] handles restricted counter systems. Regions are arithmetic formulae (hence $\sqsubseteq$ is not recursive). A partially recursive flat acceleration procedure is available. The heuristic is REACH2 restricted to $T^{\leq k}$ for a user-defined k. See [23] for an in-depth comparison of FAST and TREX. UB, F and kF stands for L-uniformly bounded, L-flattable and L-flattable with length k (UB $\subseteq$ 1F $\subseteq$ kF $\subseteq$ F).

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	?

Procedure comparison on case studies. The following table compares how ALV, FAST and LASH behave in practice. "Yes" means termination within 1200 seconds on a Pentium III 933 MHz with 512 Mb. k is the length of cycles FAST considered in Procedure REACH3. All case studies are infinite-state systems, taken from FAST's web site [8]. Experimental results show strong relationship with the acceleration framework: flat acceleration (FAST) has the better termination results, loop acceleration ($k = 1$) is not always sufficient, while simple iteration (ALV) is not sufficient on these complex examples (results are consistent with [10]). These experiments clearly suggest that **flat acceleration greatly enhances termination and is fully justified in practice, at least for counter systems**.

6.2 Tool design

The flat acceleration framework provides guidelines for designing new techniques and tools. FAST supports completely this framework. Complex case studies have been conducted [8, 9]. The following table shows performances of FAST on a significant pool of counter systems collected on the web sites of tools like ALV, BABYLON [7], BRAIN, LASH and TREX, and ranging from tricky academic puzzles (swimming pool) to complex industrial protocols (TTP). (More examples are given in the full version of this paper.) They all are infinite-state and are thus beyond the scope of traditional model checking techniques and tools. Furthermore, most of these systems also go beyond

System	var	$\vert T\vert$	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

System	var	$\vert T\vert$	sec.	Mb	k
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

VASS or Petri nets, so that methods like covering trees or backward computation do not apply. The results are for forward computation of the reachability set, on an Intel Pentium 933 Mhz with 512 Mb. Comparing them with other complex case studies analyzed with ALV, LASH, and TREX [3, 6, 10, 33] confirms that flat acceleration is a powerful technique for handling infinite-state systems.

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, 2000.
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, 2001.
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. `www.cs.ucsb.edu/~bultan/composite/`.
7. BABYLON. `www.ulb.ac.be/di/ssd/lvbegin/CST/`.
8. 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, 2003
9. S. Bardin, A. Finkel and J. Leroux. FASTer acceleration of counter automata. In *Proc. TACAS'04*, LNCS 2988, pages 576–590, 2004.
10. C. Bartzis and T. Bultan. Widening arithmetic automata. In *Proc. CAV'04*, LNCS 3114, pages 321–333, 2004.
11. 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, 1997.
12. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. SAS'97*, LNCS 1302, pages 172–186, 1997.
13. 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.

14. A Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *TCS*, 221(1–2):211–250, 1999.
15. A. Bouajjani, B. Jonsson, M. Nilsson and T. Touili. Regular Model Checking. *Proc. CAV'00*, LNCS 1855, pages 403–418, 2000.
16. A. Bouajjani, A. Muscholl, and T. Touili. Permutation rewriting and algorithmic verification. In *Proc. LICS'01*, pages 399–408, 2001.
17. D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.
18. 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, 1997.
19. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
20. H. Comon and Y. Jurski. Multiple counters automata, safety analysis, and Presburger arithmetic. In *Proc. CAV'98*, LNCS 1427, pages 268–279, 1998.
21. H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. CONCUR'99*, LNCS 1664, pages 242–257, 1999.
22. P. Cousot. Abstract interpretation. *ACM Comp. Surv.*, 28(2):324–328, 1996.
23. Ch. Darlot, A. Finkel, and L. Van Begin. About Fast and TReX accelerations. In *Proc. AVoCS'04*, ENTCS 128(6), pages 87–103, 2005.
24. 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.
25. J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fund. Informaticae*, 31(1):13–25, 1997.
26. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proc. FSTTCS'02*, LNCS 2556, pages 145–156, 2002.
27. A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems: Application to FIFO automata. *Inf. & Comp.*, 181(1):1–31, 2003.
28. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1–2):63–92, 2001.
29. L. Fribourg and H. Olsén. Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmetic, In *Proc. CONCUR'97*, LNCS 1243, pages 213–227, 1997.
30. L. Fribourg. Petri nets, flat languages and linear arithmetic. In M. Alpuente, editor, *Proc. WFLP'00*, pages 344–365, 2000.
31. O. H. Ibarra, Jianwen Su, Zhe Dang, T. Bultan, and R. A. Kemmerer. Counter machines and verification problems. *TCS*, 289(1):165–189, 2002.
32. 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.
33. LASH. `www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
34. J. Leroux and G. Sutre. On flatness for 2-dimensional vector addition systems with states. In *Proc. CONCUR'04*, LNCS 3170, pages 402–416, 2004.
35. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. ATVA'05*, this volume.
36. J. K. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proc. PSTV '87*, pages 207–219, 1987.
37. T. Rybina and A. Voronkov. Brain: Backward reachability analysis with integers. In *Proc. AMAST'02*, LNCS 2422, pages 489–494, 2002.
38. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. CAV'98*, LNCS 1427, pages 88–97, 1998.

A Proofs missing in the main text

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 $\mathsf{x} \in X^n$.

Theorem A.1. *Given the symbolic framework of 2-counter systems and Presburger formulae, 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 formulae 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 $(q_0, \text{“}x := 0, y := 0, z := 0\text{“}, q_1)$, the transitions $(q_1, \text{“}x := x + 1\text{“}, q_1)$, $(q_1, \text{“}y := y + 1\text{“}, q_1)$, the transitions $(q, \text{“}x := x + 1\text{“}, q)$, $(q, \text{“}x := x - 1\text{“}, q)$, $(q, \text{“}y := y + 1\text{“}, q)$, $(q, \text{“}y := y - 1\text{“}, q)$ $(q, \text{“}z := z + 1\text{“}, q)$, $(q, \text{“}z := z - 1\text{“}, q)$ and for all $q'' \in Q_0 \cup Q_1$ a transition $(q, \text{“}x := x, y := y, z := z\text{“}, q'')$. 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.1 Proof of Theorem 3.3

Theorem 3.3. *Given a symbolic framework $SF = (I, L)$ and a system $S \in \mathcal{F}(I)$*
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.2 Proof of Proposition 4.3

Proposition 4.3. *Let S support flat acceleration. Then for any* rlre *ρ over T and for any $\mathtt{x}_0 \in L^{|Q|}$,* $\mathsf{post}(\rho, [\![\mathtt{x}_0]\!])$ *is effectively L-definable. Furthermore, computing $\mathtt{x} \in L^{|Q|}$ such that $[\![\mathtt{x}]\!] = \mathsf{post}(\rho, [\![\mathtt{x}_0]\!])$ can be done uniformly from ρ and $\mathtt{x}_0$.*

Proof. We reason by induction on ρ. If $\rho = \varepsilon$ then $[\![\mathtt{x}]\!] = \mathsf{post}(\varepsilon, [\![\mathtt{x}]\!])$ and the property is true. Otherwise 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.3 Proof of Proposition 4.5

Proposition 4.5. *If S is a flat system supporting flat acceleration, then $\mathsf{post}^*_S([\![\mathtt{x}]\!])$ is effectively L-definable.*

Proof. In [34] 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.4 Proof of Theorem 4.8

Theorem 4.8. *If S is a L-flattable system supporting flat acceleration, then $\mathsf{post}^*_S([\![\mathtt{x}]\!])$ 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 $[\![\mathtt{x}'']\!] = \mathsf{post}^*_{S'}([\![\mathtt{x}']\!])$ (Prop. 4.5). 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 $\mathtt{y}$, $[\![\mathtt{y}]\!]$ is included in $\mathsf{post}^*([\![\mathtt{x}]\!])$. Hence $[\![\mathtt{y}]\!] = \mathsf{post}^*([\![\mathtt{x}]\!])$. 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.5 Proof of Theorem 4.9

Theorem 4.9. *Assuming the symbolic framework of 2-counter systems and Presburger formulae, the question of whether a 2-counter system S is L-flattable is undecidable.*

Proof. This proof is essentially as for Theorem A.1. 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.6 Proof of Theorem 5.1

Theorem 5.1. *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* $\mathsf{post}^*([\![\mathtt{x}]\!]) = \mathsf{post}(\rho, [\![\mathtt{x}]\!])$.

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 language $\rho_X \subseteq T^*$).

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 4.8). Since S' is flat, using the proof of Prop. 4.5 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 $z(\mathsf{post}_{S'}(\rho', [\![\mathtt{x}']\!])) = \mathsf{post}_S(z(\rho'), z([\![\mathtt{x}']\!])) = \mathsf{post}_S(\rho, z([\![\mathtt{x}']\!]))$.

Hence 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.7 Proof of Theorem 5.2

Theorem 5.2. *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)*.*

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 under-approximations 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 5.1). 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 5.1). 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 under-approximation). 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.8 Proof of Theorem 5.4

Theorem 5.4. *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).

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.9 Proof of Proposition 5.6

Proposition 5.6. *Conjugation reduction and commuting reduction are k-reductions. 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 extends the table from section 6.2. All examples are infinite-state, except those in the *Bounded Petri Nets* category.

The results are taken from forward computation of the reachability set, using an Intel Pentium 933 Mhz with 512 Mb. In the following table, $|T|$ is the number of transitions, $|\mathcal{A}|$ is the size of the binary automaton that represents the reachability set (in FAST, the regions are binary automata). $|w|$ 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 section 5.2). We aborted all computations after 1800 seconds.

In principle there can be several reasons why FAST does not terminate: (1) the reachability set is not Presburger definable, (2) the system (with given $\mathtt{x}_0$) is not L-flattable, (3) the monoid of the system is not finite and our techniques on counter systems do not apply, (4) time and space consumption are too high. Even if it is always difficult to pick one of these explanations when experiment fails, it seems that, on these examples, the main reason for failure is (4). We can rule out (3) in most cases since they are variants of VASS (with zero test or reset) for which finiteness of the monoid is guaranteed [9].

Case study	nb var	$\|T\|$	time (s)	mem (Mb)	$\|\mathcal{A}\|$	$\|w\|$	k	nb 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

Fig. 2: Running FAST (on Pentium 933 Mhz with 512 Mbytes)