Iterating Transducers in the Large ⋆ (extended abstract)

Bernard Boigelot, Axel Legay, and Pierre Wolper

Université de Liège,
Institut Montefiore, B28,
4000 Liège, Belgium
{boigelot,legay,pw}@montefiore.ulg.ac.be,
http://www.montefiore.ulg.ac.be/~{boigelot,legay,pw}/

Abstract. Checking infinite-state systems is frequently done by encoding infinite sets of states as regular languages. Computing such a regular representation of, say, the reachable set of states of a system requires acceleration techniques that can finitely compute the effect of an unbounded number of transitions. Among the acceleration techniques that have been proposed, one finds both specific and generic techniques. Specific techniques exploit the particular type of system being analyzed, e.g. a system manipulating queues or integers, whereas generic techniques only assume that the transition relation is represented by a finite-state transducer, which has to be iterated. In this paper, we investigate the possibility of using generic techniques in cases where only specific techniques have been exploited so far. Finding that existing generic techniques are often not applicable in cases easily handled by specific techniques, we have developed a new approach to iterating transducers. This new approach builds on earlier work, but exploits a number of new conceptual and algorithmic ideas, often induced with the help of experiments, that give it a broad scope, as well as good performance.

1 Introduction

If one surveys much of the recent work devoted to the algorithmic verification of infinite-state systems, it quickly appears that regular languages have emerged as a unifying representation formalism for the sets of states of such systems. Indeed, regular languages described by finite automata are a convenient to manipulate, and already quite expressive formalism that can naturally capture infinite sets. Regular sets have been used in the context of infinite sets of states due to unbounded data (e.g. [BG96,FWW97,WB98,BW02]) as well as in the context of parametric systems (e.g. [KMM+97,PS00]). Of course, whether regular of not, an infinite set of states cannot be computed

⋆ This work was partially funded by a grant of the "Communauté française de Belgique - Direction de la recherche scientifique - Actions de recherche concertées" and by the European IST-FET project ADVANCE (IST-1999-29082).

enumeratively in a finite amount of time. There is thus a need to find techniques for finitely computing the effect of an unbounded number of transitions. Such techniques can be domain specific or generic. Domain specific results were, for instance, obtained for queues in [BG96,BGWW97,BH97], for integers and reals in [Boi99,WB98,WB00,BJW01,BW02,Boi03], for pushdown system in [FWW97,BEM97], and for lossy channels in [AJ96,ABJ98].

Generic techniques appeared in the context of the verification of parametric systems. The idea used there is that a configuration being a word, a transition relation is a relation on words, or equivalently a language of pairs of words. If this language is regular, it can be represented by a finite state automaton, more specifically a finite-state *transducer*, and the problem then becomes the one of iterating such a transducer. Finite-state transducers are quite powerful (the transition relation of a Turing machine can be modelled by a finite-state transducer), the flip side of the coin being that the iteration of such a transducer is neither always computable, nor regular. Nevertheless, there are a number of practically relevant cases in which the iteration of finite-state transducers can be computed and remains finite-state. Identifying such cases and developing (partial) algorithms for iterating finite-state transducers has been the topic, referred to as "regular model checking", of a series of recent papers [BJNT00,JN00,Tou01,DLS01,AJNd02].

The question that initiated the work reported in this paper is, whether the generic techniques for iterating transducers could be fruitfully applied in cases in which domain specific techniques had been exclusively used so far. In particular, our goal was to iterate finite-state transducers representing arithmetic relations (see [BW02] for a survey). Beyond mere curiosity, the motivation was to be able to iterate relations that are not in the form required by the domain specific results, for instance disjunctive relations. Initial results were very disappointing: the transducer for an arithmetic relation as simple as $(x, x + 1)$ could not be iterated by existing generic techniques. However, looking for the roots of this impossibility through a mix of experiments and theoretical work, and taking a pragmatic approach to solving the problems discovered, we were able to develop an approach to iterating transducers that easily handles arithmetic relations, as well as many other cases. Interestingly, it is by using a tool for manipulating automata (LASH [LAS]), looking at examples beyond the reach of manual simulation, and testing various algorithms that the right intuitions, later to be validated by theoretical arguments, were developed. Implementation was thus not an afterthought, but a central part of our research process.

The general approach that has been taken is similar to the one of [Tou01] in the sense that, starting with a transducer T, we compute powers T^i of T and attempt to generalize the sequence of transducers obtained in order to capture its infinite union. This is done by comparing successive powers of T and attempting to characterize the difference between powers of T as a set of states and transitions that are added. If this set of added states, or *increment*, is always the same, it can be inserted into a loop in order to capture all powers of T. However, for arithmetic transducers comparing T^i with T^{i+1} did not yield an

increment that could be repeated, though comparing T^{2^i} with $T^{2^{i+1}}$ did. So, a first idea we used is not to always compare T^i and T^{i+1}, but to extract a sequence of samples from the sequence of powers of the transducer, and work with this sequence of samples. Given the binary encoding used for representing arithmetic relations, sampling at powers of 2 works well in this case, but the sampling approach is general and different sample sequences can be used in other cases. Now, if we only consider sample powers T^{i_k} of the transducers and compute $\bigcup_k T^{i_k}$, this is not necessarily equivalent to computing $\bigcup_i T^i$. Fortunately, this problem is easily solved by considering the reflexive transducer, i.e. $T_0 = T \cup I$, in which case working with an infinite subsequence of samples is sufficient. Finally, for arithmetic transducers, we used the fact that the sequence $T_0^{2^i}$ can efficiently be computed by successive squaring.

To facilitate the comparison of elements of a sequence of transducers, we work with transducers normalized as reduced deterministic automata. Identifying common parts of successive transducers then amounts to finding isomorphic parts which, given that we are dealing with reduced deterministic automata, can be done efficiently. Working with reduced deterministic automata has advantages, but at the cost of frequently applying expensive determinization procedures. Indeed, during our first experiments, the determinization cost quickly became prohibitive, even though the resulting automata were not excessively large. A closer look showed that this was linked to the fact that the subset construction was manipulating large, but apparently redundant, sets of states. This redundancy was pinpointed to the fact that, in the automata to be determinized, there were frequent inclusion relations between the languages accepted from different states. Formally, there is a partial-order relation on the states of the automaton, a state s_1 being greater than a state s_2 (we say s_1 *dominates* s_2), if the language accepted from s_1 includes the language accepted from s_2. Thus, when applying the subset construction, dominated states can always be eliminated from the sets that are generated. Of course, one needs the dominance relation to apply this but, exploiting the specifics of the context in which determinization is applied, we were able to develop a simple procedure that computes a safe approximation of the dominance relation in time quadratic in the size of the automaton to be determinized.

Once the automata in the sequence being considered are constructed and compared, and that an increment corresponding to the difference between successive elements has been identified, the next step is to allow this increment to be repeated an arbitrary number of times by incorporating it into a loop. There are some technical issues about how to do this, but no major difficulty. Once the resulting "extrapolated" transducer has been obtained, one still needs to check that the applied extrapolation is safe (contains all elements of the sequence) and is precise (contains no more). Checking the safety of the extrapolation can easily be done by checking that it remains unchanged when being composed with itself. Checking preciseness is more delicate, but we have develop a procedure that embodies a sufficient criterion for doing so. The idea is to check that any behavior of the transducer with a given number k of copies of the increment, can

be obtained by composing transducers with less than k copies of the increment. This is done by augmenting the transducers to be checked with counters and proving that one can restrict theses counters to a finite range, hence allowing finite-state techniques to be used.

In our experiments, we were able to iterate a variety of arithmetic transducers. We were also successful on disjunctive relations that could not be handled by earlier specific techniques. Furthermore, to test our technique in other contexts, we successfully applied it to examples of parametric systems and to the analysis of a Petri net.

2 Transducers, arithmetic transducers and their iteration

The underlying problem we are considering is reachability analysis for an infinite state system characterized by a transition relation R. Our goal is thus to compute the closure $R^* = \bigcup_{i \geq 0} R^i$ of R. In what follows, it will be convenient to also consider the reflexive closure of R, i.e. $R \cup I$ where I is the identity relation, which will be denoted by R_0; clearly $R^* = R_0^*$.

We will work in the context of regular model checking [BJNT00], in which R is defined over the set of finite words constructed from an alphabet Σ, is regular and is length preserving (i.e. if $(w, w') \in R$, then $|w| = |w'|$). In this case, R can be defined by a finite automaton over the alphabet $\Sigma \times \Sigma$. Such an automaton is called a *transducer* and is defined by a tuple $T = (Q, \Sigma \times \Sigma, q_0, \delta, F)$ where Q is the set of states, $q_0 \in Q$ the initial state, $\delta : Q \times (\Sigma \times \Sigma) \rightarrow 2^Q$ the transition function ($\delta : Q \times (\Sigma \times \Sigma) \rightarrow Q$ if the automaton is deterministic), and $F \subseteq Q$ is the set of accepting states.

As it has been shown in earlier work [KMM+97,PS00,BJNT00,JN00,Tou01] [DLS01,AJNd02] finite-state transducers can represent the transition relation of parametric systems. Using the encoding of integers by words adopted in [WB95,WB98,Boi99,WB00,BJW01,BW02,Boi03], finite-state transducers can represent all Presburger arithmetic definable relations plus some base-dependent relations [Cob69,Sem77,BHMV94].

Example 1. If positive integers are encoded in binary with an arbitrary number of leading 0's allowed, and negative numbers are represented using 2's complement allowing for an arbitrary number of leading 1's, the transducer of Figure 1 defined over the alphabet $\{0,1\} \times \{0,1\}$ represents the relation $(x, x+1) \cup (x, x)$ (see [BW02] for a full description of the encoding).

If relations R_1 and R_2 are respectively represented by transducers $T_1 = (Q_1, \Sigma \times \Sigma, q_{01}, \delta_1, F_1)$ and $T_2 = (Q_2, \Sigma \times \Sigma, q_{02}, \delta_2, F_2)$, the transducer $T_{12} = T_2 \circ T_1$ representing the composition $R_2 \circ R_1$ of R_1 and R_2 is easily computed as $T_{12} = (Q_1 \times Q_2, \Sigma \times \Sigma, (q_{01}, q_{02}), \delta_{12}, F_1 \times F_2)$, where

$$\delta((q_1, q_2), (a, b)) = \{(q_1', q_2') \mid (\exists c \in \Sigma)(q_1' \in \delta_1(q_1, (a, c)) \text{ and } q_2' \in \delta_2(q_2, (c, b)))\}$$

Note that even if T_1 and T_2 are deterministic w.r.t. $\Sigma \times \Sigma$, T_{12} can be nondeterministic.

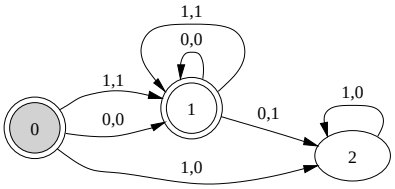

Fig. 1. A transducer for $(x, x+1) \cup (x, x)$.

To compute the closure R^* of a relation represented by a transducer T, we need to compute $\bigcup_{i\geq 0} T^i$, which is a priori an infinite computation and hence we need a *speed up* technique. In order to develop such a technique, we will consider the reflexive closure R_0 of R and use the following result.

Lemma 1. *If R_0 is a reflexive relation and $s = s_1, s_2, \ldots$ is an infinite subsequence of the natural numbers then, $\bigcup_{i\geq 0} R_0^i = \bigcup_{k\geq 0} R_0^{s_k}$.*

The lemma follows directly from the fact that for any $i \geq 0$, there is an $s_k \in s$ such that $s_k > i$ and that, since R_0 is reflexive, $(\forall j \leq i)(R_0^j \subseteq R_0^i)$.

Thus, if we use the transducer T_0 corresponding to the reflexive relation R_0, it is sufficient to compute $\bigcup_{k\geq 0} R_0^{s_k}$ for an infinite sequence $s = s_1, s_2, \ldots$ of "sample points". For instance, in the case of arithmetic transducers working with binary encodings, considering sample points of the form $s_k = 2^k$ turns out to be very useful as illustrated by the following example.

Example 2. Figure 2 shows the transducer for $(x, x+1) \cup (x, x)$ composed with itself 2, 4, 8 and 16 times.

Finally, note that when the sampling sequence consists of powers of 2, the sequence of transducers $T_0^{2^k}$ can be efficiently computed by using the fact that $T_0^{2^{k+1}} = T_0^{2^k} \circ T_0^{2^k}$.

3 Detecting increments

Consider a reflexive transducer T_0 and a sequence $s_1, s_2, \ldots$ of sampling points. Our goal is to determine whether for each $i > 0$, the transducer $T_0^{s_{i+1}}$ differs from $T_0^{s_i}$ by some additional constant finite-state structure. One cannot however hope to check explicitly such a property among an infinite number of sampled transducers. Our strategy consists in comparing a finite number of successive transducers until either a suitable increment can be guessed, or the procedure cannot be carried on further.

For each $i > 0$, let $T_0^{s_i} = (Q^{s_i}, \Sigma \times \Sigma, q_0^{s_i}, \delta^{s_i}, F^{s_i})$. We assume that these transducers are deterministic w.r.t. $\Sigma \times \Sigma$ and minimal. To identify common parts between two successive transducers $T_0^{s_i}$ and $T_0^{s_{i+1}}$ we first look for states of $T_0^{s_i}$ and $T_0^{s_{i+1}}$ from which identical languages are accepted. Precisely, we want

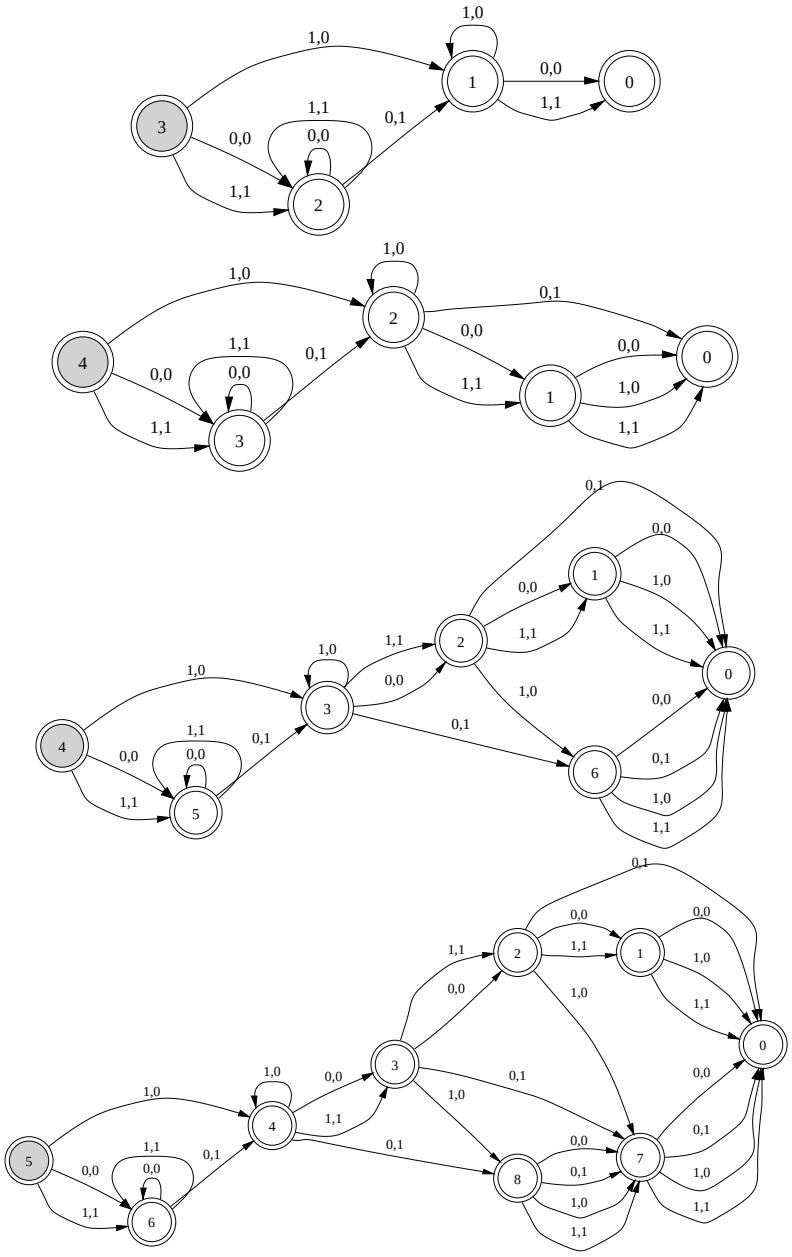

Fig. 2. Transducers for $(x, x+1) \cup (x, x)$ at powers of 2.

to construct a relation $E_f^{s_i} \subseteq Q^{s_i} \times Q^{s_{i+1}}$ such that $(q, q') \in E_f^{s_i}$ iff the language accepted from q in $T_0^{s_i}$ is identical to the language accepted from q' in $T_0^{s_{i+1}}$. Since we are dealing with minimized deterministic transducers, the *forwards equivalence* $E_f^{s_i}$ is one-to-one (though not total) and can easily be computed by partitioning the states of the joint automaton $(Q^{s_i} \cup Q^{s_{i+1}}, \Sigma \times \Sigma, q_0^{s_i}, \delta^{s_i} \cup \delta^{s_{i+1}}, F^{s_i} \cup F^{s_{i+1}})$ according to their accepted language. This operation is easily carried out by Hopcroft's finite-state minimization procedure [Hop71]. Note that because the automata are reduced deterministic, the parts of $T_0^{s_i}$ and $T_0^{s_{i+1}}$ linked by $E_f^{s_i}$ are isomorphic, incoming transitions being ignored.

Next, we search for states of $T_0^{s_i}$ and $T_0^{s_{i+1}}$ that are reachable from the initial state by identical languages. Precisely, we want to construct a relation $E_b^{s_i} \subseteq Q^{s_i} \times Q^{s_{i+1}}$ such that $(q, q') \in E_b^{s_i}$ iff the language accepted in $T_0^{s_i}$ when q' is taken to be the unique accepting state is identical to the language accepted in $T_0^{s_{i+1}}$ when q is taken to be the unique accepting state. Since $T_0^{s_i}$ and $T_0^{s_{i+1}}$ are deterministic and minimal, the *backwards equivalence* $E_b^{s_i}$ can be computed by forward propagation, starting from the pair $(q_0^{s_i}, q_0^{s_{i+1}})$ and exploring the parts of the transition graphs of $T_0^{s_i}$ and $T_0^{s_{i+1}}$ that are isomorphic to each other, if transitions leaving these parts are ignored.

Note that taking into account the reduced deterministic nature of the automata we are considering, the relations $E_f^{s_i}$ and $E_b^{s_i}$ loosely correspond to the forwards and backwards bisimulations used in [DLS01,AJNd02].

We are now able to define our notion of finite-state "increment" between two successive transducers, in terms of the relations $E_f^{s_i}$ and $E_b^{s_i}$.

Definition 1. *The transducer $T_0^{s_{i+1}}$ is* incrementally larger *than $T_0^{s_i}$ if the relations $E_f^{s_i}$ and $E_b^{s_i}$ cover all the states of $T_0^{s_i}$. In other words, for each $q \in Q^{s_i}$, there must exist $q' \in Q^{s_{i+1}}$ such that $(q, q') \in E_f^{s_i} \cup E_b^{s_i}$.*

If $T_0^{s_{i+1}}$ is incrementally larger than $T_0^{s_i}$, the increment consists of the states that are matched neither by $E_f^{s_i}$, nor by $E_b^{s_i}$.

Definition 2. *If $T_0^{s_{i+1}}$ is incrementally larger than $T_0^{s_i}$, then the set Q^{s_i} can be partitioned into $\{Q_b^{s_i}, Q_f^{s_i}\}$, such that*

- *The set $Q_f^{s_i}$ contains the states q covered by $E_f^{s_i}$, i.e., for which there exists q' such that $(q, q') \in E_f^{s_i}$;*
- *The set $Q_b^{s_i}$ contains the remaining states of Q^{s_i} (Definition 1 implies that these states must therefore be covered by $E_b^{s_i}$; the fact that states covered both by $E_b^{s_i}$ and $E_f^{s_i}$ are placed in $Q_f^{s_i}$ is arbitrary, its consequence is that when successive transducers are compared - see below - the part matched to $Q_f^{s_i}$, rather than the part matched to $Q_b^{s_i}$ will grow).*

The set $Q^{s_{i+1}}$ can now be partitioned into $\{Q_H^{s_{i+1}}, Q_{I_0}^{s_{i+1}}, Q_T^{s_{i+1}}\}$, where

- *The head part $Q_H^{s_{i+1}}$ is the image by $E_b^{s_i}$ of the set $Q_b^{s_i}$;*
- *The tail part $Q_T^{s_{i+1}}$ is the image by $E_f^{s_i}$ of the set $Q_f^{s_i}$, dismissing the states that belong to $Q_H^{s_{i+1}}$ (our intention is to have an unmodified head part);*

– *The* increment $Q_{I_0}^{s_{i+1}}$ *contains the states that do not belong to either* $Q_H^{s_{i+1}}$ *or* $Q_T^{s_{i+1}}$.

These definitions are illustrated in Figure 3. Note that given the definition used, the transitions between the head part, increment and tail part must necessarily be in the direction shown in the figure.

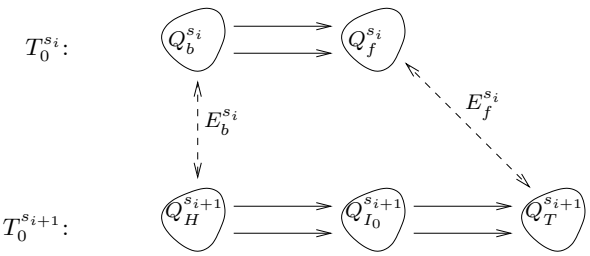

Fig. 3. Partitioning transducer states

Our expectation is that when moving from one transducer to the next in the sequence, the increment will always be the same. We formalize this by defining the incremental growth of a sequence of transducers.

Definition 3. *The sequence of sampled transducers* $T_0^{s_i}$, $T_0^{s_{i+1}}$, …, $T_0^{s_{i+k}}$ grows incrementally *if*

– *for each* $j \in [0, k-1]$, $T_0^{s_{i+j+1}}$ *is incrementally larger than* $T_0^{s_{i+j}}$;
– *for each* $j \in [1, k-1]$, *the increment* $Q_{I_0}^{s_{i+j+1}}$ *is the image by* $E_b^{s_{i+j}}$ *of the increment* $Q_{I_0}^{s_{i+j}}$.

Consider a sequence $T_0^{s_i}$, $T_0^{s_{i+1}}$, …, $T_0^{s_{i+k}}$ that grows incrementally. The tail part $Q_T^{s_{i+j}}$ of $T_0^{s_{i+j}}$, $j \in [2, \ldots, k]$, will then consist of $j-1$ copies of the increment plus a part that we will name the *tail-end part*. Precisely, $Q_T^{s_{i+j}}$ can be partitioned into $\{Q_{I_1}^{s_{i+j}}, Q_{I_2}^{s_{i+k}}, \ldots, Q_{I_{j-1}}^{s_{i+j}}, Q_{T_f}^{s_{i+j}}\}$, where

– for each $\ell \in [1, \ldots, j-1]$, the *tail increment* $Q_{I_\ell}^{s_{i+j}}$ is the image by the relation $E_f^{s_{i+j-1}} \circ E_f^{s_{i+j-2}} \circ \cdots \circ E_f^{s_{i+j-\ell}}$ of the "head" increment $Q_{I_0}^{s_{i+j-\ell}}$, where "$\circ$" denotes the composition of relations;
– the *tail-end set* $Q_{T_f}^{s_{i+j}}$ contains the remaining elements of $Q_T^{s_{i+j}}$.

The situation is illustrated in Figure 4.

Focusing on the last transducer $T_0^{s_{i+k}}$ in a sequence of incrementally growing transducers, its head increment $Q_{I_0}^{s_{i+k}}$ and all the tail increments $Q_{I_\ell}^{s_{i+k}}$, $\ell \in [1, k-1]$ appearing in its tail part $Q_T^{s_{i+k}}$ are images of the increment $Q_{I_0}^{s_{i+1}}$ by a combination of forwards and backwards equivalences. Indeed, by Definition 3, each tail increment is the image of a previous increment by a composition of forwards equivalences, and each head increment is the image of the previous

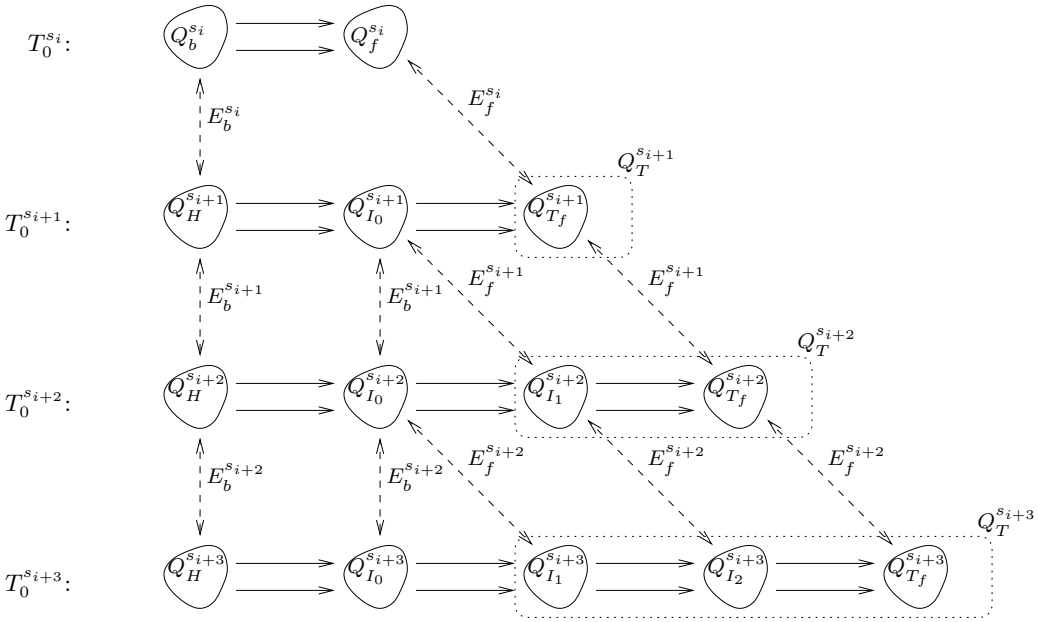

Fig. 4. Incrementally-growing sequence of transducers.

one by a backwards equivalence. Thus, the transition graphs internal to all increments are isomorphic to that of $Q_{I_0}^{s_{i+1}}$, and hence are isomorphic to each other.

Our intention is to extrapolate the transducer $T_0^{s_{i+k}}$ by adding more increments, following a regular pattern. In order to do this, we need to compare the transitions leaving different increments. We use the following definition.

Definition 4. *Let $T_0^{s_{i+k}}$ be the last transducer of an incrementally growing sequence, let $Q_{I_0}^{s_{i+k}}, \ldots, Q_{I_{k-1}}^{s_{i+k}}$ be the isomorphic increments detected within $T_0^{s_{i+k}}$, and let $Q_{T_f}^{s_{i+k}}$ be its "tail end" set. Then, an increment $Q_{I_\alpha}^{s_{i+k}}$ is said to be* communication equivalent *to an increment $Q_{I_\beta}^{s_{i+k}}$ iff, for each pair of corresponding states (q, q'), $q \in Q_{I_\alpha}^{s_{i+k}}$ and $q' \in Q_{I_\beta}^{s_{i+k}}$, and $a \in \Sigma \times \Sigma$, we have that, either*

- *$\delta(q, a) \in Q_{I_\alpha}^{s_{i+k}}$ and $\delta(q', a) \in Q_{I_\beta}^{s_{i+k}}$, hence leading to corresponding states by the existing isomorphism,*
- *$\delta(q, a)$ and $\delta(q', a)$ are both undefined,*
- *$\delta(q, a)$ and $\delta(q', a)$ both lead to the same state of the tail end $Q_{T_f}^{s_{i+k}}$, or*
- *there exists some γ such that $\delta(q, a)$ and $\delta(q', a)$ lead to corresponding states of respectively $Q_{I_{\alpha+\gamma}}^{s_{i+k}}$ and $Q_{I_{\beta+\gamma}}^{s_{i+k}}$.*

In order to extrapolate $T_0^{s_{i+k}}$, we simply insert an extra increment $Q_{I_{e_1}}^{s_{i+k}}$ between the head part of $T_0^{s_{i+k}}$ and its head increment $Q_{I_0}^{s_{i+k}}$ and define the

transitions leaving it in order to make it communication equivalent to $Q_{I_0}^{s_{i+k}}$. Of course, before doing so, it is heuristically sound to check that a sufficiently long prefix of the increments of $T_0^{s_{i+k}}$ are communication equivalent with each other.

4 Extrapolating sequences of transducers and correctness

Consider a transducer T_{e_0} to which extrapolation is going to be applied. The state set of this transducer can be decomposed in a head part Q_H, a series of k increments $Q_{I_0}, \ldots, Q_{I_{k-1}}$ and a tail end part Q_{T_f}. Repeatedly applying the extrapolation step described at the end of the previous section yields a series of extrapolated transducers $T_{e_1}, T_{e_2}, \ldots$. Our goal is to build a single transducer that captures the behaviors of the transducers in this sequence, i.e. a transducer $T_{e_*} = \bigcup_{i \geq 0} T_{e_i}$. The transducer T_{e_*} can simply be built from T_{e_0} by adding transitions to its head increment Q_{I_0} according to the following rule.

For each state $q \in Q_{I_0}$ and $a \in \Sigma \times \Sigma$,

- If $\delta(q, a)$ leads to a state q' in an increment Q_{I_j}, $1 \leq j \leq k-1$, then
- add transitions from q and labelled by a to the state corresponding to q' (by the increment isomorphism) in each of the increments Q_{I_ℓ} with $0 \leq \ell < j$.

The added transitions, which include loops (transitions to $Q_{I_0}^{e_0}$ itself) allow T_{e_*} to simulate the computations of any of the T_{e_i}, $i \geq 0$. Conversely, it is fairly easy to see all computations generated using the added transitions correspond to a computation of some T_{e_i}. Note that the addition of transitions yields a nondeterministic transducer, which needs to be determinized and reduced to be in canonical form.

Having thus constructed an extrapolated transducer T_{e_*}, it remains to check whether this transducer accurately corresponds to what we really intend to compute, i.e. $\bigcup_{i \geq 0} T^i$. This is done by first checking that the extrapolation is *safe*, in the sense that it captures all behaviors of $\bigcup_{i \geq 0} T^i$, and then checking that it is precise, i.e. that it has no more behaviors than $\bigcup_{i \geq 0} T^i$.

Lemma 2. *The transducer T_{e_*} is a safe extrapolation if $L(T_{e_*} \circ T_{e_*}) \subseteq L(T_{e_*})$.*

Indeed, we have that $L(T_0) \subseteq L(T_{e_*})$ and thus by induction that $L(T_0^i) \subseteq L(T_{e_*})$ (recall that T_0 is reflexive).

Determining whether the extrapolation is precise is a more difficult problem for which we provide only a partial solution, in the form of a sufficient criterion. The problem amounts to proving that any word accepted by T_{e_*}, or equivalently by some T_{e_i}, is also accepted by an iteration T_0^j of the transducer T_0. The idea is to check that this can be proved inductively. The property is true by construction for the transducer T_{e_0} from which the extrapolation sequence is built. If we can also prove that, if the property holds for all $j < i$, then it also holds for i, we are done. For this last step, we resort to the following sufficient condition.

Definition 5. *A sequence of extrapolated transducers T_{e_i} is* inductively precise *if, for all i and word $w \in L(T_{e_i})$, there exists $j, j' < i$ such that $w \in L(T_{e_j} \circ T_{e_{j'}})$.*

To check inductive preciseness, we use automata with counters. First, notice that it is possible to add a counter c to T_{e_*}, in such a way that, when a word is accepted, the value of c is the index i of the automaton T_{e_i} of the extrapolation sequence by which the word is in fact accepted. Sketchily, this is done by initializing the counter to 0 and suitably incrementing it when the transitions added to T_{e_0} in order to obtain T_{e_*} are used. The result is a counter automaton $T^c_{e_*}$. Furthermore, we will write $T^{c=i}_{e_*}$ (resp. $T^{c<i}_{e_*}$) to denote the automaton $T^c_{e_*}$ restricted to accepting words only when the value of the counter at the end of the computation is equal to (resp. less than) i.

Using three copies of this automaton, $T^{c_1}_{e_*}$, $T^{c_2}_{e_*}$ $T^{c_2}_{e_*}$, the inductive preciseness criterion can be expressed as

$$\forall w \forall i [w \in L(T^{c_1=i}_{e_*}) \supset w \in L(T^{c_2<i}_{e_*} \circ T^{c_3<i}_{e_*})]$$

or equivalently as

$$\forall w \forall i [w \notin L(T^{c_1=i}_{e_*}) \vee w \in L(T^{c_2<i}_{e_*} \circ T^{c_3<i}_{e_*})].$$

What we need now is an algorithmic way of checking this condition, which is non obvious. The universal quantifier on words can be eliminated by expressing the criterion as a language universality condition, namely

$$\forall i [L(\overline{T}^{c_1=i}_{e_*} \cup (T^{c_2<i}_{e_*} \circ T^{c_3<i}_{e_*})) = (\Sigma \times \Sigma)^*].$$

Besides the fact that we need to check universality of a counter automaton, we are left with two problems: the fact that we have a universal quantifier on counter values and the need to compute the complement of $T^c_{e_*}$, which is nondeterministic. Fortunately both problems can be solved simultaneously as follows. The transitions of $T^c_{e_*}$ are labelled by elements of $\Sigma \times \Sigma$ and by a counter incrementing operation $+i$, where i is in a finite range $0 \leq i \leq d$. Interestingly, if we consider the alphabet to be $\Sigma \times \Sigma \times [0, d]$, it can be seen (details in the full paper) that the automaton is deterministic. Let $Ta^c_{e_*}$ be the counter automaton operating over this augmented alphabet and let $\overline{Ta}^c_{e_*}$ be its complement, which is thus easily obtained by exchanging accepting and non accepting states. Intuitively, working with words in $\Sigma \times \Sigma \times [0, d]$ amounts to working with words in which each symbol is annotated by the expected counter increment. When such a word is accepted by $Ta^c_{e_*}$, the sum of these annotations is exactly equal to the value of the counter. Quantifying on words w and on values i can thus be replaced by quantifying over augmented words $w_a \in (\Sigma \times \Sigma \times [0, d])^*$. What we need to check is thus

$$L(\overline{Ta}^{c_1}_{e_*} \cup (T^{c_2<c_1}_{e_*} \circ T^{c_3<c_1}_{e_*})) = (\Sigma \times \Sigma \times [0, d])^*, \tag{1}$$

which makes sense since any word over $\Sigma \times \Sigma \times [0, d]$ uniquely determines a final value for the counter c_1. The fact that we use T rather than Ta for the second and third copies of the automaton should be interpreted to mean that these copies operate over $\Sigma \times \Sigma$ and thus ignore the counter annotations.

We are thus now left with the problem of checking universality of a three-counter automaton. In fact, since what needs to be checked is two of the differences between these counters, two counters are enough. We are still in the realm of undecidability, but our last step is to show that we can replace these counters by bounded counters and hence reduce the problem to a finite-state one.

Definition 6. *The counter automaton $A^{c_2<c_1}$ is* synchronized *with respect to the counters c_1 and c_2 if there exists $M > 0$ such that each $w \in L(A^{c_2<c_1})$ can be accepted by a path π, each subpath σ of which satisfies $\Delta c_2(\sigma) < \Delta c_1(\sigma) + M$, where $\Delta c_i(\sigma)$ denotes the increment applied to c_i while following σ.*

The following result establishes that a decidable sufficient criterion for preciseness can be obtained by strengthening the condition (1), by imposing synchronization between the counters that need to be compared.

Theorem 3. *It is decidable whether the counter machine defined in (1)*

- *is synchronized with respect to c_1, c_2 and to c_1, c_3, and*
- *accepts the language $(\Sigma \times \Sigma \times [0, d])^*$.*

The proof, based on a reduction to a bounded-counter automaton, is left for the full paper.

5 Using dominance to improve efficiency

Since to ease the comparison of successive transducers we work with reduced deterministic automata, and since transducer composition usually introduces nondeterminism, each transducer composition step implies a potentially costly determinization procedure. Indeed, our experiments showed that this step could be very resource consuming, even though the resulting transducer was not that much larger than the ones being combined. It thus seemed likely that the transducers we were using had some structure that kept them from growing when being composed. If this structure could be exploited, it would be possible to substantially improve the efficiency of the determinization step.

Looking at the states generated during these steps, it appeared that they corresponded to large, but vastly redundant, sets of states of the nondeterministic automaton. This redundancy is due to the fact that there are frequent inclusion relations between the languages accepted from different states of the transducer. We formalize this observation with the following notion of *dominance*, similar to the concept used in the ordered automata of [WB00].

Definition 7. *Given a nondeterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, let A_q be the automaton $A = (Q, \Sigma, \delta, q, F)$, i.e. A where the initial state is q. We say that a state q_1* dominates *a state q_2 (denoted $q_1 \geq q_2$) if $L(A_{q_2}) \subseteq L(A_{q_1})$.*

Clearly, when applying a subset construction, each subset that is generated can be simplified by eliminating dominated states. However, in order to use this, we need to be able to efficiently compute the dominance relation.

A first step is to note that, for deterministic automata, this can be done in quadratic time. The idea of the algorithm is to compute the synchronized product of the automaton with itself, i.e. the product in which the transitions are those simultaneously possible in both components. Now, a state q_1 dominates a state q_2 iff, in the synchronized product, it is impossible from the pair (q_1, q_2) to reach a pair (q_1', q_2') where $q_2' \in F$ and $q_1' \notin F$. So, if we eliminate from the states of the product all pairs from which it is possible to reach such an accepting-nonaccepting pair, the remaining pairs define the dominance relation.

The problem of course is that the automaton to which the determinization and minimization procedure is applied is not deterministic. However, it is obtained from deterministic automata by the composition procedure described in Section 2, and it is possible to approximate the dominance relation of the composed transducer using the dominance relation of the components. Indeed, it is easy to see that if $q_1 \geq q_1'$ in T_1, and $q_2 \geq q_2'$ in T_2, then $(q_1, q_2) \geq (q_1', q_2')$ in $T_2 \circ T_1$; this being only sufficient since we can have dominance in the composed transducer without having dominance in the components. Nevertheless, the dominance relation obtained by combining the component relations is a safe approximation and has proven to be quite satisfactory in practice.

As an example of the performance improvements made possible by the use of dominance, the time required to compute $T^{2^{100}}$ for the transducer given in Example 1 was reduced from 3 hours of CPU time to 3 minutes.

6 Experiments

The results presented in this paper have been tested on a series of case studies. The prototype implementation that has been used relies in part on the LASH package [LAS] for automata manipulation procedures, but implements the specific algorithms needed for transducer implementation. It is a prototype in the sense that the implementation is not at all optimized, that the interfaces are still rudimentary, that the implementation of the preciseness criterion is not fully operational, and that the increment detection procedure that is implemented is not yet the final one.

As a first series of test cases, we used transducers representing arithmetic relations as in Example 1. This example is of course fully handled by our tool, but we could also automatically compute the iteration of the transducers representing $(x, x+k)$ for many values of k. Turning to examples with multiple variables, the closure of the transducers encoding the relations $((x, y, z), (z+1, x+2, y+3))$ and $((w, x, y, z), (w+1, x+2, y+3, z+4))$ were successfully computed. In addition, we could also handle the transducer encoding the transition relation of a Petri net arithmetically represented by $(((x, y), (x+2, y-1)) \cup ((x, y), (x-1, y+2))) \cap \mathbb{N}^2 \times \mathbb{N}^2$. An interesting aspect of this last example is that it is disjunctive and can not be handled by the specific techniques of [Boi03]. In all these examples, the

sampling sequence consists of the powers of 2. In Table 1 we give the number of states of some transducers that were iterated, of their closure, and of the largest power of the transducer that was constructed.

Relation	$\lvert T_0\rvert$	$\lvert T_0^*\rvert$	Max $\lvert T_0^i\rvert$
$(x, x+1)$	3	3	11
$(x, x+7)$	7	9	91
$(x, x+73)$	14	75	933
$(((x,y),(x+2,y-1)) \cup ((x,y),(x-1,y+2))) \cap \mathbb{N}^2 \times \mathbb{N}^2$	19	70	1833
$((x,y),(x+2,y-1)) \cup ((x,y),(x-1,y+2)) \cup ((x,y),(x+1,y+1))) \cap \mathbb{N}^2 \times \mathbb{N}^2$	21	31	635
$((w,x,y,z),(w+1,x+2,y+3,z+4))$	91	251	2680

Table 1. Examples of transducers and their iteration.

We also considered the parametric systems which were used as examples in previous work on transducer iteration. We tried the main examples described in [BJNT00,JN00,Tou01,AJNd02] and our tool was able to handle them. In this case, sampling was not needed in the sense that all powers of the transducer were considered.

7 Conclusions and comparison with other work

As a tool for checking infinite-state systems, iterating regular transducers is an appealing technique. Indeed, it is, at least in principle, independent of the type of system being analyzed and is a natural generalization of the iteration of finite-state relations represented by BDDs, which has been quite successful.

Will the iteration of regular transducers also have a large impact on verification applications? The answer to this question is still unknown, but clearly the possibility of scaling up the technique will be a crucial success factor. This is precisely the direction in which this paper intends to make contributions. Indeed, we believe to have scaled up techniques for iterating transducers both qualitatively and quantitatively. From the qualitative point of view, the idea of sampling the sequence of approximations of the iterated transducer, as well as our increment detection and closing technique have enabled us to handle arithmetic transducers that were beyond the reach of earlier methods. Arithmetic relations were also considered in [JN00,BJNT00], but for a simple unary encoding, which limits the expressiveness of regular transducers. From the quantitative point of view, systematically working with reduced deterministic automata and using efficiency improving techniques such as dominance has enabled us to work with quite complex transducers of significant, if not really large size. At least, our implemented tool can find iterations well beyond what can be done by visually inspecting, and manually computing with, automata.

Our work definitely builds on earlier papers that have introduced the basic concepts used in the iteration of regular transducers. For instance, our technique for comparing successive approximations of the iterated transducer can be linked to the reduction techniques used in [JN00,DLS01,BJNT00,AJNd02]. However, we work from the point of view of comparing successive approximations, rather than reducing an infinite-state transducer. This makes our technique similar to the widening technique found in [BJNT00,Tou01], but in a less restrictive setting. Furthermore, we have a novel technique to check that the "widened" transducer corresponds exactly to the iterated transducer. Also, some of the techniques introduced in this paper could be of independent interest. For instance, using dominance to improve the determinization procedure could have applications in other contexts.

Techniques for iterating transducers are still in their infancy and there is room for much further work. The set of transducers we have handled is still limited and there are many other examples to explore and to learn from in order to improve our technique. Our implementation can still be substantially improved, which can also lead to further applications and results. Finally, there are a number of possible extensions, one being to handle automata with infinite words, which would lead the way to applying the iteration of transducers to dense real-time systems.

Acknowledgement

We thank Marcus Nilsson, Parosh Abdulla, Elad Shahar, and Martin Steffen for answering many email questions on their work.

References

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proceedings 10th International Conference on Computer Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318, Vancouver, Canada, 1998. Springer.

[AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, June 1996.

[AJNd02] P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proceedings 13th International Conference on Concurrency Theory (CONCUR)*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130, Brno, Czech Republic, 2002. Springer.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the 8th International conference of Concurrency Theory (CONCUR 97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150, Warsaw, Poland, july 1997. Springer.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Proceedings of the 8th International Conference on Computer-Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12, New-Brunswick, NJ, USA, July 1996. Springer-Verlag.

[BGWW97] Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The power of QDDs. In *Proc. of Int. Static Analysis Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186, Paris, September 1997. Springer-Verlag.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO channel systems with nonregular sets of configurations. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceeding of 24th International Colloquim on Automata, Languages and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, pages 560–570, Bologna, Italy, 1997. Springer-Verlag.

[BHMV94] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society*, 1(2):191–238, March 1994.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and Tayssir Touili. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[BJW01] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Siena, June 2001. Springer-Verlag.

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. Collection des publications de la Faculté des Sciences Appliquées de l'Université de Liège, Liège, Belgium, 1999.

[Boi03] Bernard Boigelot. On iterating linear transformations over recognizable sets of integers. *Theoretical Computer Science*, 2003. 65 pages, accepted for publication.

[BW02] Bernard Boigelot and Pierre Wolper. Representing arithmetic constraints with finite automata: An overview. In *Proc. International Conference on Logic Programming (ICLP)*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19, Copenhagen, July 2002. Springer-Verlag.

[Cob69] A. Cobham. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory*, 3:186–192, 1969.

[DLS01] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *Proceedings 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297, Paris, France, 2001. Springer.

[FWW97] Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. In Faron Moller, editor, *Infinity'97, Second International Workshop on Verification of Infinite State Systems*, volume 9 of *Electronic Notes in Theoretical Computer Science*, Bologna, July 1997. Elsevier Science Publishers.

[Hop71] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. *Theory of Machines and Computation*, pages 189–196, 1971.

[JN00] B. Jonsson and M. Nilson. Transitive closures of regular relations for verifying infinite-state systems. In S. Graf and M. Schwartzbach, editors, *Proceeding of the 6th International conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1875 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 1997.

[LAS] The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2000.

[Sem77] A. L. Semenov. Presburgerness of predicates regular in two number systems. *Siberian Mathematical Journal*, 18:289–299, 1977.

[Tou01] T. Touili. Regular model checking using widening techniques. In *Proceeding of Workshop on Verification of Parametrized Systems (VEPAS'01)*, volume 50 of *Electronic Notes in Theoretical Computer Science*, 2001.

[WB95] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32, Glasgow, September 1995. Springer-Verlag.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97, Vancouver, July 1998. Springer-Verlag.

[WB00] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, March 2000. Springer-Verlag.