On (Omega-)Regular Model Checking

Axel Legay
Universié de Liège
Institut Montefiore, B28
Liège, Belgium

Pierre Wolper
Universié de Liège
Institut Montefiore, B28
Liège, Belgium

Checking infinite-state systems is frequently done by encoding infinite sets of states as regular languages. Computing such a regular representation of, say, the set of reachable 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 performances.

Categories and Subject Descriptors: D.2.4 [**Formal Methods**]: Model checking—*Software/Program Verification*; F.1.1 [**Automata**]:

General Terms: Verification, Theory, Algorithms, Implementation

Additional Key Words and Phrases: (Omega-)Regular Model Checking, Transducers, Extrapolation, Infinite-State System.

Authors' e-mail : {legay,pw}@montefiore.ulg.ac.be
Authors' website : http://www.montefiore.ulg.ac.be/∼{legay,pw}/
Axel Legay is now with INRIA Rennes, France.
The present article is an improved version of [Boigelot et al. 2003], [Boigelot et al. 2004], [Legay 2007] (a part of), and [Legay 2008].

1. INTRODUCTION

At the heart of all the techniques that have been proposed for exploring infinite state spaces, is a symbolic representation that can finitely represent infinite sets of states. In early work on the subject, this representation was domain specific, for example linear constraints for sets of real vectors. For several years now, the idea that a generic finite-automaton based representation could be used in many settings has gained ground, starting with systems manipulating queues and integers [Wolper and Boigelot 1995; Boigelot et al. 1997; Wolper and Boigelot 1998; 2000], then moving to parametric systems [Kesten et al. 1997], and, finally, reaching systems using real variables [Boigelot et al. 1998; Boigelot et al. 2001; 2005; Boigelot and Wolper 2002].

For exploring an infinite state space, one does not only need a finite representation of infinite sets, but also techniques for finitely computing the effect of an unbounded number of transitions. Such techniques can be domain specific or generic. Domain specific techniques exploit the specific properties and representations of the domain being considered and were, for instance, obtained for queues in [Boigelot and Godefroid 1996; Bouajjani and Habermehl 1997], for integers and reals in [Boigelot 1999; Boigelot and Wolper 2002; Boigelot et al. 2003; Boigelot and Herbreteau 2006; Finkel and Leroux 2002; Bardin et al. 2004; Bardin et al. 2005], for pushdown system in [Finkel et al. 1997; Bouajjani et al. 1997], and for lossy channels in [Abdulla and Jonsson 1996]. Generic techniques consider finite-automata representations and provide algorithms that operate directly on this representation, mostly disregarding the domain for which it is used.

Generic techniques appeared first in the context of the verification of systems whose states can be encoded by *finite words*, such as parametric systems. The idea used there is that a configuration being a finite word, a transition relation is a relation on finite words, or equivalently a language of pairs of finite 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 modeled 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 [Kesten et al. 1997; Bouajjani et al. 2000; Boigelot et al. 2003; 2004; Jonsson and Nilsson 2000; Bouajjani et al. 2004; Touili 2001; Dams et al. 2002; Abdulla et al. 2003].

The question that initiated the work presented 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, one of our goals was to iterate finite-state transducers representing arithmetic relations (see [Boigelot and Wolper 2002] 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 [LASH]), 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.

The general approach that has been taken is similar to the one of [Touili 2001] 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., $T_0 = T \cup T_{Id}$ where T_{Id} is the identity transducer, in which case working with an infinite subsequence of samples is sufficient.

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). An easy to check sufficient condition for the extrapolation to be safe is that it remains unchanged when being composed with itself. Checking preciseness is more delicate, but we have developed 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.

Taking advantage of the fact that our extrapolation technique works on automata, not just on transducers, we consider computing reachable states both by computing the closure of the transducer representing the transition relation, and by repeatedly applying the transducer to a set of initial states. The first approach yields a more general object and is essential if one wishes to extend the method to the verification of temporal properties ([Bouajjani et al. 2000; Pnueli and Shahar 2000; Abdulla et al. 2004; Bouajjani et al. 2004]), but the second is often less demanding from a computational point of view and can handle cases that are out of reach for the first. Preciseness is not always possible to check when working with state sets rather than transducers, but this just amounts to saying that what is computed is possibly an overapproximation of the set of reachable states, a situation which is known to be pragmatically unproblematic.

Going further, the problem of using Regular Model Checking technique for systems whose states are represented by infinite (omega) words has been addressed. This makes the representation of sets of reals possible as described in [Boigelot et al. 2001; Boigelot et al. 2003]. To avoid the hard to implement algorithms needed for some operations on infinite-word automata, only omega-regular sets that can be defined by weak deterministic Büchi automata [Muller et al. 1986] are considered. This is of course restrictive, but as is shown in [Boigelot et al. 2001; 2005], it is sufficient to handle sets of reals defined in the first-order theory of linear constraints. Moreover using such a representation leads to algorithms that are very similar to the ones used in the finite word case, and allows us to work with reduced deterministic automata as a normal form. Due to these advantages and properties, one can show that the technique developed for the finite word case can directly be adapted to weak deterministic Büchi automata up to algorithmic modifications.

Our technique has been implemented in a tool called T(0)RMC (Tool for (Omega-)Regular Model Checking), which has been tested on several classes of infinite-state systems. It is worth mentioning that the ability of T(O)RMC to extrapolate a sequence of automata has other applications than solving the (ω-)Regular Reachability Problems. As an example, the tool has been used in a semi-algorithm to compute the convex hull of a set of integer vectors [Cantin et al. 2007; 2008]. T(O)RMC was also used to compute a symbolic representation of the simulation relation between the states of several classes of infinite-state systems with the aim of verifying temporal properties [Bouajjani et al. 2004].

Structure of the paper. The paper is structured as follows. In Section 2, we recall the elementary definitions on automata theory that will be used throughout the rest of the paper. Section 3 presents the (ω-)Regular Model Checking framework as well as the problems we want to solve. Sections 4, 5, 6, and 7 describe our main results. Implementation and experiments are discussed in Section 8. Finally, Sections 9 concludes the paper with a comparison with other works on the same topic.

2. BACKGROUND ON AUTOMATA THEORY

In this section, we introduce several notations, concepts, and definitions that will be used throughout the rest of this paper. The set of natural numbers is denoted by $\mathbb{N}$, and $\mathbb{N}_0$ is used for $\mathbb{N} \setminus \{0\}$.

2.1 Relations

Consider a set S, a set $S_1 \subseteq S$, and two binary[1] relations $R_1, R_2 \subseteq S \times S$. The identity relation on S, which is denoted R_{id}^S (or R_{id} when S is clear from the context), is the set $\{(s,s)|s \in S\}$. The *image* of S_1 by R_1, denoted $R_1(S_1)$, is the set $\{s' \mid (\exists s \in S_1)((s,s') \in R_1)\}$. The *composition* of R_1 with R_2, denoted $R_2 \circ R_1$, is the set $\{(s,s') \mid (\exists s'')((s,s'') \in R_1 \wedge (s'',s') \in R_2)\}$. The *i*th *power* of R_1 ($i \in \mathbb{N}_0$), denoted R_1^i, is the relation obtained by composing R_1 with itself i times. The *zero-power* of R_1, denoted R_1^0, corresponds to the identity relation. The *transitive closure* of R_1, denoted R_1^+, is given by $\bigcup_{i=1}^{i=+\infty} R_1^i$, its *reflexive transitive closure*, denoted R^*, is given by $R_1^+ \cup R_{id}^S$. The *domain* of R_1, denoted $Dom(R_1)$, is given by $\{s \in S \mid (\exists s' \in S)((s,s') \in R_1)\}$.

2.2 Words and Languages

An *alphabet* is a (nonempty) finite set of distinct symbols. A *finite word* of length n over an alphabet Σ is a mapping $w : \{0,\ldots,n-1\} \rightarrow \Sigma$. An *infinite word* , also called $\omega-$word, over Σ is a mapping $w : \mathbb{N} \rightarrow \Sigma$. We denote by the term *word* either a finite word or an infinite word, depending on the context. The *length* of the finite word w is denoted by $|w|$. A finite word w of length n is often represented by $w = w(0)\cdots w(n-1)$. An infinite word w is often represented by $w(0)w(1)\cdots$. The sets of finite and infinite words over Σ are denoted by Σ^* and by Σ^ω, respectively. We define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A *finite-word (respectively infinite-word) language* over Σ is a (possibly infinite) set of finite (respectively, infinite) words over Σ. Consider L_1 and L_2, two finite-word (respectively, infinite-word) languages. The *union* of L_1 and L_2, denoted $L_1 \cup L_2$, is the language that contains all the words that belong either to L_1 or to L_2. The *intersection* of L_1 and L_2, denoted $L_1 \cap L_2$, is the language that contains all the words that belong to both L_1 and L_2. The *complement* of L_1, denoted $\overline{L_1}$ is the language that contains all the words over Σ that do not belong to L_1.

We alos introduce *synchronous product* and *projection*, which are two operations needed to define relations between languages.

Definition 2.1. Consider L_1 and L_2 two languages over Σ.

—If L_1 and L_2 are finite-word languages, the synchronous product $L_1 \bar{\times} L_2$ of L_1 and L_2 is defined as follows

$$L_1 \bar{\times} L_2 = \{(w(0),w(0)')\ldots(w(n),w(n)') \mid \\ w = w(0)w(1)\ldots w(n) \in L_1 \wedge w' = w(0)'w(1)'\ldots w(n)' \in L_2\}.$$

—If L_1 and L_2 are ω-languages, the synchronous product $L_1 \bar{\times} L_2$ of L_1 and L_2 is defined as follows

$$L_1 \bar{\times} L_2 = \{(w(0),w(0)')(w(1),w(1)')\cdots \mid \\ w = w(0)w(1)\ldots \in L_1 \wedge w' = w(0)'w(1)'\cdots \in L_2\}.$$

The language $L_1 \bar{\times} L_2$ is defined over the alphabet Σ^2.

[1]The term "binary" will be dropped in the rest of the paper.

Definition 2.1 directly generalizes to synchronous products of more than two languages. Given two finite (respectively, infinite) words w_1, w_2 (with $|w_1| = |w_2|$ if the words are finite) and two languages L_1 and L_2 with $L_1 = \{w_1\}$ and $L_2 = \{w_2\}$, we use $w_1 \bar{\times} w_2$ to denote the *unique* word in $L_1 \bar{\times} L_2$.

Definition 2.2. Suppose L a language over the alphabet Σ^n and a natural $1 \leq i \leq n$. The projection of L on all its components except component i, denoted $\Pi_{\neq i}(L)$, is the language L' such that

$$\Pi_{\neq i}(L) = \{w_1 \bar{\times} \ldots \bar{\times} w_{i-1} \bar{\times} w_{i+1} \bar{\times} \ldots \bar{\times} w_n \mid (\exists w_i)(w_1 \bar{\times} \ldots \bar{\times} w_{i-1} \bar{\times} w_i \bar{\times} w_{i+1} \bar{\times} \ldots \bar{\times} w_n \in L)\}.$$

2.3 Automata

Definition 2.3. An automaton over Σ is a tuple $A = (Q, \Sigma, Q_0, \triangle, F)$, where

—Q is a finite set of *states*,

—Σ is a *finite* alphabet,

—$Q_0 \subseteq Q$ is the set of *initial states*,

—$\triangle \subseteq Q \times \Sigma \times Q$ is a finite *transition relation*, and

—$F \subseteq Q$ is the set of accepting states (the states in $Q \setminus F$ are the *nonaccepting* states).

Let $A = (Q, \Sigma, Q_0, \triangle, F)$ be an automaton. If $(q_1, a, q_2) \in \triangle$, then we say that there is a *transition* from q_1 (the *origin*) to q_2 (the *destination*) labeled by a. We sometimes abuse the notations, and write $q_2 \in \triangle(q_1, a)$ instead of $(q_1, a, q_2) \in \triangle$. Two transitions $(q_1, a, q_2), (q_3, b, q_4) \in \triangle$ are *consecutive* if $q_2 = q_3$. Given two states $q, q' \in Q$ and a finite word $w \in \Sigma^*$, we write $(q, w, q') \in \triangle^*$ if there exist states $q_0, \ldots, q_{k-1}$ and $w(0), \ldots, w(k-2) \in \Sigma$ such that $q_0 = q$, $q_{k-1} = q'$, $w = w(0)w(1) \cdots w(k-2)$, and $(q_i, w(i), q_{i+1}) \in \triangle$ for all $0 \leq i < k-1$. Given two states $q, q' \in Q$, we say that the state q' is *reachable* from q in A if $(q, a, q') \in \triangle^*$. The automaton A is *complete* if for each state $q \in Q$ and symbol $a \in \Sigma$, there exists at least one state $q' \in Q$ such that $(q, a, q') \in \triangle$. An automaton can easily be completed by adding an extra nonaccepting state.

A *finite run* of A on a finite word $w : \{0, \ldots, n-1\} \rightarrow \Sigma$ is a labeling $\rho : \{0, \ldots, n\} \rightarrow Q$ such that $\rho(0) \in Q_0$, and $(\forall 0 \leq i \leq n-1)((\rho(i), w(i), \rho(i+1)) \in \triangle)$. A finite run ρ is *accepting* for w if $\rho(n) \in F$. An *infinite run* of A on an infinite word $w : \mathbb{N} \rightarrow \Sigma$ is a labeling $\rho : \mathbb{N} \rightarrow Q$ such that $\rho(0) \in Q_0$, and $(\forall 0 \leq i)((\rho(i), w(i), \rho(i+1)) \in \triangle)$. An infinite run ρ is *accepting* for w if $\mathit{inf}(\rho) \cap F \neq \emptyset$, where $\mathit{inf}(\rho)$ is the set of states that are visited infinitely often by ρ.

We distinguish between *finite-word automata* that are automata accepting finite words, and *Büchi automata* that are automata accepting infinite words. A finite-word automaton accepts a finite word w if there exists an accepting finite run for w in this automaton. A Büchi automaton accepts an infinite word w if there exists an accepting infinite run for w in this automaton. The set of words accepted by A is the *language accepted by* A, and is denoted $L(A)$. Any language that can be represented by a finite-word (respectively, Büchi) automaton is said to be *regular* (respectively, ω-*regular*).

The automaton A may behave nondeterministicaly on an input word, since it may have many initial states and the transition relation may specify many possible transitions for each state and symbol. If $|Q_0| = 1$ and for all state $q_1 \in Q$ and symbol $a \in \Sigma$ there is at most one state $q_2 \in Q$ such that $(q_1, a, q_2) \in \triangle$, then A is *deterministic*. In order to emphasize this property, a deterministic automaton is denoted as a tuple $(Q, \Sigma, q_0, \delta, F)$, where q_0 is the unique initial state and $\delta : Q \times \Sigma \to Q$ is a partial function deduced from the transition relation by setting $\delta(q_1, a) = q_2$ if $(q_1, a, q_2) \in \triangle$. Operations on languages directly translate to operations on automata, and so do the notations.

One can decide weither the language accepted by a finite-word or a Büchi automaton is empty or not. It is also known that finite-word automata are closed under determinization, complementation, union, projection, and intersection [Hopcroft 1971]. Moreover, finite-word automata admit a minimal form, which is unique up to isomorphism [Hopcroft 1971].

Though the union, intersection, synchronous product, and projection of Büchi automata can be computed efficiently, the complementation operation requires intricate algorithms that not only are worst-case exponential, but are also hard to implement and optimize (see [Vardi 2007] for a survey). The core problem is that there are Büchi automata that do not admit a deterministic/minimal form. To working with infinite-word automata that do own the same properties as finite-word automata, we will restrict ourselves to *weak* automata [Muller et al. 1986] defined hereafter.

Definition 2.4. For a Büchi automaton $A = (\Sigma, Q, q_0, \delta, F)$ to be weak, there has to be partition of its state set Q into disjoint subsets $Q_1, \ldots, Q_m$ such that for each of the Q_i, either $Q_i \subseteq F$, or $Q_i \cap F = \emptyset$, and there is a partial order $\leq$ on the sets $Q_1, \ldots, Q_m$ such that for every $q \in Q_i$ and $q' \in Q_j$ for which, for some $a \in \Sigma$, $q' \in \delta(q, a)$ ($q' = \delta(q, a)$ in the deterministic case), $Q_j \leq Q_i$.

A weak automaton is thus a Büchi automaton such that each of the strongly connected components of its graph contains either only accepting or only non-accepting states.

Not all ω-regular languages can be accepted by deterministic weak Büchi automata, nor even by nondeterministic weak automata. However, there are algorithmic advantages to working with weak automata : deterministic weak automata can be complemented simply by inverting their accepting and non-accepting states; and there exists a simple determinization procedure for weak automata [Safra 1992], which produces Büchi automata that are deterministic, but generally not weak. Nevertheless, if the represented language can be accepted by a deterministic weak automaton, the result of the determinization procedure will be *inherently weak* according to the definition below [Boigelot et al. 2001] and thus easily transformed into a weak automaton.

Definition 2.5. A Büchi automaton is *inherently weak* if none of the reachable strongly connected components of its transition graph contain both accepting (visiting at least one accepting state) and non-accepting (not visiting any accepting state) cycles.

This gives us a pragmatic way of staying within the realm of deterministic weak Büchi automata. We start with sets represented by such automata. This is preserved by union, intersection, synchronous product, and complementation operations. If a projection is needed, the result is determinized by the known simple procedure. Then, either the result is inherently weak and we can proceed, or it is not and we are forced to use the classical algorithms for Büchi automata. The latter cases might never occur, for instance if we are working with automata representing sets of reals definable in the first-order theory of linear constraints [Boigelot et al. 2001].

A final advantage of weak deterministic Büchi automata is that they admit a minimal form, which is unique up to isomorphism [Löding 2001].

2.4 Relations on Automata States

We will also use the following definitions.

Definition 2.6. Given two automata $A_1 = (Q_1, \Sigma, Q_{01}, \triangle_1, F_1)$ and $A_2 = (Q_2, \Sigma, Q_{02}, \triangle_2, F_2)$, we define

—the forward equivalence relation $E_f, \subseteq Q_1 \times Q_2$, which is an equivalence relation on states of A_1 and A_2 with $(q_1, q_2) \in E_f$ iff $L_{q_1}^{F_1}(A_1) = L_{q_2}^{F_2}(A_2)$;

—the backward equivalence relation $E_b \subseteq Q_1 \times Q_2$, which is an equivalence relation on states of A with $(q_1, q_2) \in E_b$ iff $L_{Q_{01}}^{q_1}(A_1) = L_{Q_{02}}^{q_2}(A_2)$ and $q_1 \in Q_{01}$ iff $q_2 \in Q_{02}$.

Definition 2.7. We say that two automata $A_1 = (Q_1, \Sigma, Q_{01}, \triangle_1, F_1)$ and $A_2 = (Q_2, \Sigma, Q_{02}, \triangle_2, F_2)$ are isomorphic iff there exists an isomorphism relation $R \subseteq Q_1 \times Q_2$ such that

—R is a bijection[2],

—for each $a \in (\Sigma \cup \{\epsilon\})$ and $q_1, q_2 \in Q_1$, $(q_1, a, q_2) \in \triangle_1 \Leftrightarrow (R(q_1), a, R(q_2)) \in \triangle_2$,

—for each $(q, q') \in R$, $q \in Q_{01} \Leftrightarrow q' \in Q_{02}$,

—for each $(q, q') \in R$, $q \in F_1 \Leftrightarrow q' \in F_2$.

Isomorphism property. By Definition 2.7 is a bijection, we have that if A_1 and A_2 are isomorphic and A_1 and A_3 are isomorphic, then A_2 and A_3 are isomorphic. Moreover, R is transitive.

The following theorem will be used in the rest of the paper.

Theorem 2.8. *Let $A_1 = (Q_1, \Sigma, q_{01}, \delta_1, F_1)$ and $A_2 = (Q_2, \Sigma, q_{02}, \delta_2, F_2)$ be two minimal automata. Let E_f^1 and E_b^1 be bijective forward and backward equivalence relations between states of A_1 and A_2. We have the following results.*

[2] We restrict to bijections mainly because this definition will be used with minimal automata.

(1) E_f^1 *is an isomorphism relation between* A_1 *and* A_2*;*

(2) E_f^1 *is an isomorphism relation between* A_1 *and* A_2*.*

PROOF. This is a direct consequence of Definition 2.7 and the fact that E_f^1 and E_b^1 are bijective relation on minimal automata. □

2.5 Transducers

In this paper, we will consider relations that are defined over sets of words. We use the following definitions taken from [Nilsson 2001]. For a finite-word (respectively, infinite-word) language L over Σ^n, we denote by $\lfloor L \rfloor$ the finite-word (respectively, infinite-word) relation over Σ^n consisting of the set of tuples $(w_1, w_2, \ldots, w_n)$ such that $w_1 \bar{\times} w_2 \bar{\times} \ldots \bar{\times} w_n$ is in L. The arity of such a relation is n. Note that for $n = 1$, we have that $L = \lfloor L \rfloor$. The relation R_{id} is the *identity relation*, i.e., $R_{id} = \{(w_1, w_2, \ldots, w_n) | w_1 = w_2 = \ldots = w_n\}$. A relation R defined over Σ^n is *(ω-)regular* if there exists a (ω-)regular language L over Σ^n such that $\lfloor L \rfloor = R$.

We now introduce transducers that are automata for representing (ω-)regular relations over Σ^2.

Definition 2.9. A transducer over Σ^2 is an automaton T over Σ^2 given by $(Q, \Sigma^2, Q_0, \triangle, F)$, where

—Q is the *finite* set of *states*,

—Σ^2 is the *finite* alphabet,

—$Q_0 \subseteq Q$ is the set of *initial states*,

—$\triangle : Q \times \Sigma^2 \times Q$ is the *transition relation*, and

—$F \subseteq Q$ is the set of accepting states (the states that are not in F are the *nonaccepting* states).

Given an alphabet Σ, the transducer representing the identity relation over Σ^2 is denoted T_{id}^{Σ} (or T_{id} when Σ is clear from the context). All the concepts and operations defined for finite automata can be used with transducers. The only reason to particularize this class of automata is that some operations, such as composition, are specific to relations. In the sequel, we use the term “transducer” instead of “automaton” when using the automaton as a representation of a relation rather than as a representation of a language. We sometimes abuse the notations and write $(w_1, w_2) \in T$ instead of $(w_1, w_2) \in \lfloor L(T) \rfloor$. Given a pair $(w_1, w_2) \in T$, w_1 is the *input word*, and w_2 is the *output word*. The transducers we consider here are often called *structure-preserving*. Indeed, when following a transition, a symbol of the input word is replaced by exactly one symbol of the output word.

EXAMPLE 2.10. *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 represents the relation* $(x, x+1) \cup (x, x)$ *(see [Boigelot and Wolper 2002] for a full description of the encoding.*

Given two transducers T_1 and T_2 over the alphabet Σ that represents two relations R_1 and R_2, respectively. The *composition* of T_1 by T_2, denoted $T_2 \circ T_1$ is the

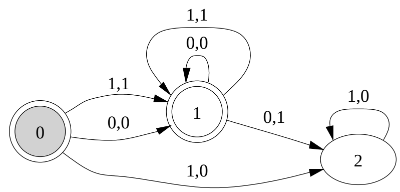


Fig. 1. A transducer T for $(x, x+1) \cup (x, x)$. The initial state of the automaton is colored in gray, and the final state is surrounded by a double circle (this convention will be followed throughout the rest of the paper).

transducer that represents the relation $R_2 \circ R_1$. We denote by T_1^i ($i \in \mathbb{N}_0$) the transducer that represents the relation R_1^i. The *transitive closure* of T is $T^+ = \bigcup_{i=1}^{\infty} T^i$; its *reflexive transitive closure* is $T^* = T^+ \cup T_{id}$. The transducer T is *reflexive* if and only if $L(T_{id}) \subseteq L(T)$. Given an automaton A over Σ that represents a set S, we denote by $T(A)$ the automaton representing the *image* of A by T, i.e., an automaton for the set $R(S)$.

Let T_1 and T_2 be two finite-word (respectively, Büchi) transducers defined over Σ^2 and let A be a finite-word automaton (respectively, Büchi) automaton defined over Σ. We observe that $T_2 \circ T_1 = \pi_{\neq 2}[(T_1 \bar{\times} T_{id}^{\Sigma}) \cap (T_{id}^{\Sigma} \bar{\times} T_2)]$ and $T(A) = \pi_{\neq 1}[(A^{\Sigma} \bar{\times} \Sigma) \cap T]$, where A^{Σ} is an automaton accepting Σ^* (respectively, Σ^{ω}). As a consequence, the composition of two finite-word ((weak) Büchi) transducers is a finite-word transducer. However, the composition of two deterministic weak Büchi transducer is a weak Büchi transducer whose deterministic version may not be weak. A same observation can be made about the composition of a transducer with an automaton.

3. (ω)-REGULAR MODEL CHECKING : FRAMEWORK AND PROBLEMATIC

In Section 3.1, we introduce the ω)-regular model checking framework. In Section 3.2, we present the problem that will be in the paper.

3.1 The framework

In this paper, we suppose that states of a system are encoded by words over a fixed alphabet. If the states are encoded by finite words, then sets of states can be represented by finite-word automata and relations between states by finite-word transducers. This setting is referred to as *Regular Model Checking* [Kesten et al. 1997; Wolper and Boigelot 1998]. If the states are encoded by infinite words, then sets of states can be represented by deterministic weak Büchi automata and relations between states by deterministic weak Büchi transducers. This setting is referred to as *ω-Regular Model Checking* [Boigelot et al. 2004]. Formally, a finite automata-based representation of a system can be defined as follows.

Definition 3.1. A *(ω-)regular system* for a system $\mathcal{T} = (S, S_0, R)$ is a triple $M = (\Sigma, A, T)$, where

—Σ is a finite alphabet over which the states are encoded as finite (respectively, infinite) words;
—A is a deterministic finite-word (respectively, deterministic weak Büchi) automaton over Σ that represents S_0;
—T is a deterministic finite-word (respectively, deterministic weak Büchi) transducer over Σ^2 that represents R. In the rest of the paper, T is assumed to be reflexive.

In the finite-word case, an execution of the system is an infinite sequence of same-length finite words over Σ. The Regular Model Checking framework was first used to represent parametric systems [Abdulla et al. 2002; Bouajjani and Touili 2002; Kesten et al. 1997; Abdulla et al. 1999; Bouajjani et al. 2000; Kesten et al. 2002]. The framework can also be used to represent various other models, which includes linear integer systems [Wolper and Boigelot 1995; 2000], FIFO-queues systems [Boigelot and Godefroid 1996], *XML* specifications [Bouajjani et al. 2006; Touili and d'Orso 2006], and heap analysis [Bouajjani et al. 2005; Bouajjani et al. 2006].

As an illustration we give details on how to represent parametric systems. Let P be a process represented by a finite-state system. A parametric system for P is an infinite family $S = \{S_n\}_{n=0}^{\infty}$ of networks where for a fixed n, S_n is an *instance* of S, *i.e.,* a network composed of n copies of P that work together in parallel. In the Regular Model Checking framework, the finite set of states of each process is represented as an alphabet Σ. Each state of an instance of the system can then be encoded as a finite word $w = w(0)\ldots w(n-1)$ over Σ, where $w(i-1)$ encodes the current state of the ith copy of P. Sets of states of several instances can thus be represented by finite-word automata. Observe that the states of an instance S_n are all encoded with words of the same length. Consequently, relations between states in S_n can be represented by binary finite-word relations, and eventually by transducers.

Example 3.2. *Consider a simple example of parametric network of identical processes implementing a token ring algorithm. Each of these processes can be either in idle or in critical mode, depending on whether or not it owns the unique token. Two neighboring processes can communicate with each other as follows: a process owning the token can give it to its right-hand neighbor. We consider the alphabet $\Sigma = \{N, T\}$. Each process can be in one of the two following states : T (has the token) or N (does not have the token). Given a word $w \in \Sigma^*$ with $|w| = n$ (meaning that n processes are involved in the execution), we assume that the process whose states are encoded in position $w(0)$ is the right-hand neighbor of the one whose states are encoded in position $w(n-1)$. The transition relation can be encoded as the union of two regular relations that are the following:*

(1) $(N,N)^*(T,N)(N,T)(N,N)^*$ *to describe the move of the token from $w(i)$ to $w(i+1)$ (with $0 \le i \le n-2$), and*
(2) $(N,T)(N,N)^*(T,N)$ *to describe the move of the token from $w(n-1)$ to $w(0)$.*

The set of all possible initial states where the first process has the token is given by TN^*.

In the infinite-word case, an execution of the system is an infinite sequence of infinite words over Σ. The ω-Regular Model Checking framework has been used for handling systems with both integer and real variables [Boigelot and Wolper 2002; Boigelot et al. 2005], such as linear hybrid systems with a constant derivative (see examples in [Alur et al. 1995]).

3.2 The problematic and an overview of the solution

It is known that verifying properties of systems in the (ω-)Regular Model Checking framework generally reduces to solving the *(ω-)Regular Reachability Problems* [Pnueli and Shahar 2000; Bouajjani et al. 2000; Boigelot et al. 2004; Abdulla et al. 2004; Legay 2007; Bouajjani et al. 2004] that are defined hereafter.

Definition 3.3. Let A be a deterministic finite-word (respectively, deterministic weak Büchi) automaton, and T be a reflexive deterministic finite-word (respectively, deterministic weak Büchi) transducer. The (ω-)Regular Reachability Problems for A and T are the following:

(1) *Computing $T^*(A)$:* the goal is to compute a finite-word (respectively, weak Büchi) automaton representing $T^*(A)$. If A represents a set of states S and T a relation R, then $T^*(A)$ represents the set of states that can be reached from S by applying R an arbitrary number of times.

(2) *Computing T^*:* the goal is to compute a finite-word (resp. weak Büchi) transducer representing the reflexive transitive closure of T. If T represents a subset of a power of a reachability relation R, then T^* represents its closure.

Remark 3.4. Observe that while $T^i(A)$ (respectively, T^i) will be deterministic weak for any i, $T^*(A)$ (respectively, T^*) may not be deterministic.

Existing approaches. The (ω-)Regular Reachability Problems are undecidable [Apt and Kozen 1986], but partial solutions exist. Studying those solutions is the subject of the rest of this paper. Among the techniques to solve the (ω-)Regular Reachability Problems, one distinguishes between *domain specific and generic* techniques. Domain specific techniques exploit the specific properties and representations of the domain being considered and were for instance obtained for systems with FIFO-queues in [Boigelot and Godefroid 1996; Bouajjani and Habermehl 1997], for systems with integers and reals in [Boigelot 1999; Boigelot and Wolper 2002; Boigelot et al. 2003], for pushdown systems in [Finkel et al. 1997; Bouajjani et al. 1997], for lossy queues in [Abdulla and Jonsson 1996], and well-quasi orders in [Geeraerts et al. 2005; Abdulla et al. 2000]. Generic techniques [Kesten et al. 1997; Bouajjani et al. 2000; Jonsson and Nilsson 2000; Bouajjani et al. 2004; Boigelot et al. 2003; 2004; Touili 2001; Dams et al. 2002; Abdulla et al. 2003; Vardhan et al. 2004; 2005] consider automata-based representations and provide algorithms that operate directly on these representations, mostly disregarding the domain for which it is used.

Our contribution. In this paper, we propose a generic technique for solving the (ω-)Reachability Problems.

Our generic approach. Given a possibly infinite sequence $A^1, A^2, \ldots$ of automata, the limit of this sequence is an automaton A^* such that $L(A^*) = \bigcup L(A^i)$. Consider a finite-word transducer T and a finite-word automaton A. We first observe that the computations of both T^* and $T^*(A)$ can be reduced to the computation of the limit of a possibly infinite sequence of automata. Indeed, computing T^* amounts to compute the limit of T_{id}, T^1, T^2, T^3, $\ldots$, and computing $T^*(A)$ amounts to compute the limit of A, $T^1(A)$, $T^2(A)$, $T^3(A)$, $\ldots$ (observe that since T and A are deterministic, each element in each sequence will be deterministic). We propose a generic technique which can compute the limit of a sequence of automata by extrapolating one of its finite *sampling sequence*, *i.e.* selected automata from a finite prefix of the sequence. The extrapolation step proceeds by comparing successive automata in the sampling sequence, trying to identify the difference between these in the form of an *increment*, and *extrapolating* the repetition of this increment by adding loops to the last automaton of the sequence. After the extrapolation has been built, one has to check whether it corresponds to the limit of the sequence. If this is the case, the computation terminates, otherwise, another sampling sequence has to be chosen. This is a semi-algorithm since there is no guarantee that (1) we can find a sampling sequence that can be extrapolated, and (2) the result of the extrapolation will be the desired closure.

Challenges. We need to provide : (1) a characterization of the sampling sequence, (2) (a) a precise definition of the concept of increment (b) an algorithm to detect increments between automata, (3) extrapolation algorithms, and (4) sufficient criteria to check whether the extrapolation is safe (enough is obtained) and precise (nothing is added) .

Structure of the presentation. The presentation of our solution is organized as follows. Section 4 discusses the sampling sequence. Section 5 propose a definition for increments and present an algorithm to detect successive increments between automata. Section 6 presents several extrapolation algorithms. Finally, Section 7 introduces sufficiant criteria to determine the correctness of the extrapolation. An implementation of those results as well as some experiments are discussed in Sections 8.

Remark 3.5.

(1) Our algorithms depend on properties that are shared by finite-word automata and deterministic weak Büchi automata. As a consequence, the solution we will propose for the finite-word case also apply to deterministic weak Büchi automata. There are only minor technical differences in the extrapolation algorithm[3]. We thus propose a global presentation that does not distinguish between finite-word and deterministic weak Büchi automata, except when this is needed.

(2) Our theoretical contribution is twofolds. (1) we present a methodology to ex-

[3]The extrapolation algorithm for finite-word automata can also be used for deterministic weak Büchi automata. However, this will often lead to an over-approximation of the closure (see Section 6.3), Example 6.15.

trapolate a possibly infinite sequence of automata by considering one of its finite prefixes, and (2) we present an example of sufficient correctness criteria. In [Bouajjani et al. 2004; Cantin et al. 2008], we have shown that our solution to (1) is general and has other applications than solving the ($\omega-$)Regular Reachability problems (see Section B.8 for some more details).

4. CHOOSING THE SAMPLING SEQUENCE

Choosing the sampling sequence is a tricky issue and there is no guarantee that this can be done in a way that ensures that the extrapolation step can be applied. However, there are heuristics that are very effective for obtaining a sampling sequence that can be extrapolated. The following lemma shows that the sampling sequence can be selected quite arbitrarily, assuming that T is *reflexive*.

LEMMA 4.1. *Let T be a reflexive transducer and A be an automaton. If $s = s_0, s_1, s_2, \ldots$ is an infinite increasing subsequence of the natural numbers, then $L(T^*) = \bigcup_{k\geq 0} L(T^{s_k})$ and, similarly, $L(T^*(A)) = \bigcup_{k\geq 0} L(T^{s_k}(A))$.*

PROOF. 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 T is reflexive, $(\forall j \leq i)(L(T^j) \subseteq L(T^i))$ (respectively, $L(T^j(A)) \subseteq L(T^i(A))$). $\square$

As an example, for the cases of FIFO-queue, pushdown, and parametric systems, we observed that considering sample points of the form $s_k = ak$, where $a \in \mathbb{N}$ is a constant, turns out to be very useful. For the case of arithmetic, we observed that the useful sampling points are often of the form $s_k = a^k$. Sampling sequences with sampling points of the form $s_k = ak$ are called *linear*, while sampling sequences with sampling points of the form $s_k = a^k$ are called *exponential*.

EXAMPLE 4.2. *Figure 2 shows the minimal transducer of Example 2.10 composed with itself 2, 4, 8 and 16 times. The difference between the graphs for T^4 and T^8 takes the form of an increment represented by the set of states $\{2,6\}$ in T^8. This increment is repeated between T^8 and T^{16}. Consequently, T^{16} differs from T^4 by the addition of two increments represented by the sets $\{3,8\}$ and $\{2,7\}$.*

5. DETECTING INCREMENTS

We consider a finite sequence $A^1, A^2, A^3, \ldots, A^n$ of finite automata that are either all minimal finite-word automata or all minimal weak Büchi automata. Our goal is to determine whether, for sufficiently large i, the automaton A^{i+1} differs from A^i by some additional constant finite-state structure. Our strategy, consists in comparing a finite number of successive automata until a suitable increment can be detected. In Section 5.1, we will present a methodology to detect an increment between two automata. In Section 5.2, we present a methodology to check whether the increment detected between two automaa in a sequence stays the same when moving from one automaton to the next one in the sequence.

5.1 Detection between two automata

For each $i > 0$, let $A^i = (Q^i, \Sigma, q_0^i, \delta^i, F^i)$. To identify common parts between two successive automata A^i and A^{i+1} we first look for states of A^i and A^{i+1} from

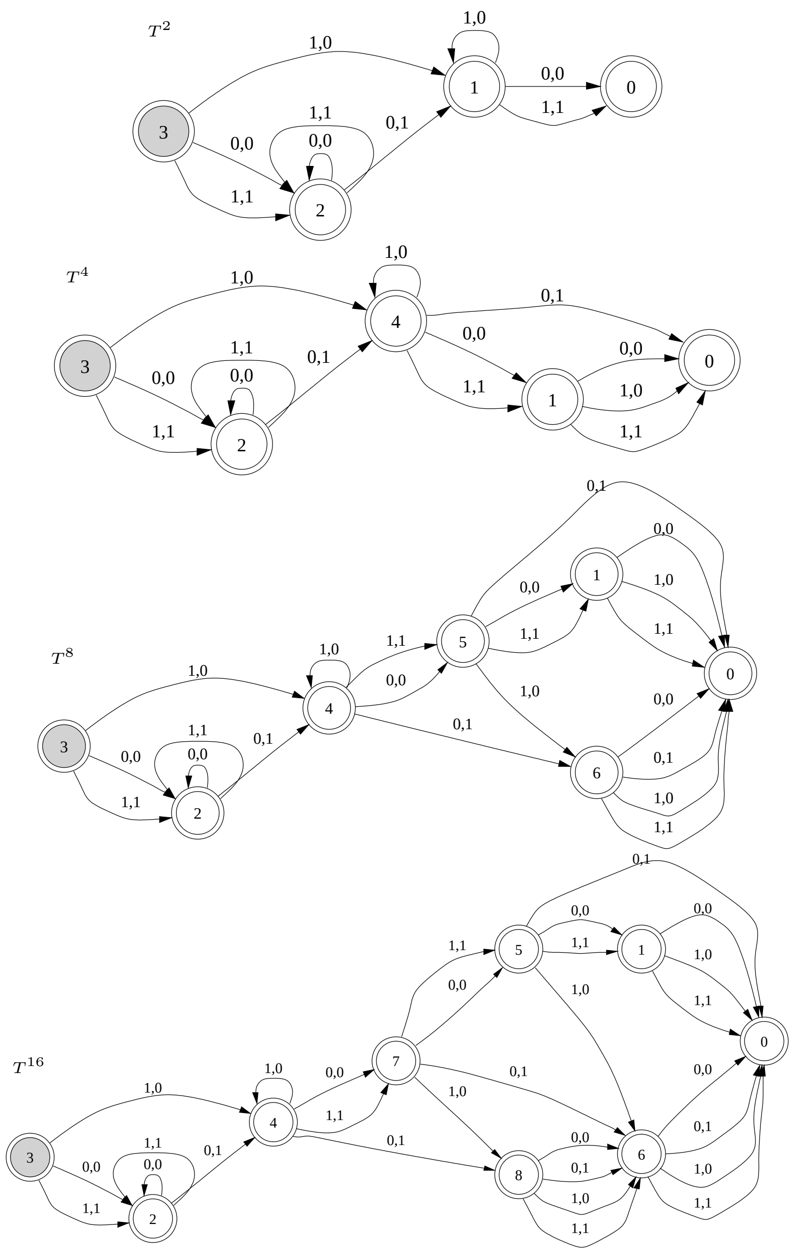


Fig. 2. Transducer of Example 4.2 at powers of two.

which identical languages are accepted. Precisely, we compute a forward equivalence relation $E_f^i \subseteq Q^i \times Q^{i+1}$ between A^i and A^{i+1}. Since we are dealing with deterministic minimal automata, the forwards equivalence E_f^i is one-to-one (though not total) and can easily be computed by partitioning the states of the joint automaton $(Q^i \cup Q^{i+1}, \Sigma, q_0^i, \delta^i \cup \delta^{i+1}, F^i \cup F^{i+1})$ according to their accepted language. For finite-word automata, this operation is easily carried out by Hopcroft's finite-state minimization procedure [Hopcroft 1971]. For weak Büchi automata, one uses the variant introduced in [Löding 2001].

Remark 5.1. Note that because the automata are minimal, the parts of A^i and A^{i+1} linked by E_i^f are isomorphic (see Definition 2.7), incoming transitions being ignored.

Next, we search for states of A^i and A^{i+1} that are reachable from the initial state by identical languages. Precisely, we compute a backward equivalence relation $E_b^i \subseteq Q^i \times Q^{i+1}$ between A^i and A^{i+1}. Since A^i and A^{i+1} are deterministic and minimal, the *backwards equivalence* E_b^i can be computed by forward propagation, starting from the pair (q_0^i, q_0^{i+1}) and exploring the parts of the transition graphs of A^i and A^{i+1} that are isomorphic to each other, if transitions leaving these parts are ignored.

Remark 5.2. Note that because the automata are minimal, the parts of A^i and A^{i+1} linked by E_b^i are isomorphic, outgoing transitions being ignored.

We now define a notion of finite-state *increment* between two successive automata, in terms of the relations E_f^i and E_b^i.

Definition 5.3. Let $A^i = (Q^i, \Sigma, q_0^i, \delta^i, F^i)$ and $A^{i+1} = (Q^{i+1}, \Sigma, q_0^{i+1}, \delta^{i+1}, F^{i+1})$ be two minimal automata. Let E_b^i and E_f^i be respectively, the backward and forward equivalences computed between A^i and A^{i+1}. The automaton A^{i+1} is *incrementally larger* than A^i if the relations E_f^i and E_b^i cover all the states of A^i. In other words, for each $q \in Q^i$, there must exist $q' \in Q^{i+1}$ such that $(q, q') \in E_b^i \cup E_f^i$.

If A^{i+1} is incrementally larger than A^i, the *increment* consists of the states that are matched neither by E_f^i, nor by E_b^i. We will use the following notations :

(1) the set Q^i can be partitioned into $\{Q_b^i, Q_f^i\}$, such that
  —The set Q_f^i contains the states q covered by E_f^i, i.e., for which there exists q' such that $(q, q') \in E_f^i$;
  —The set Q_b^i contains the remaining states.
(2) The set Q^{i+1} can be partitioned into $\{Q_H^{i+1}, Q_{I_0}^{i+1}, Q_T^{i+1}\}$, where
  —The *head part* Q_H^{i+1} is the image by E_b^i of the set Q_b^i;
  —The *tail part* Q_T^{i+1} is the image by E_f^i of the set Q_f^i, dismissing the states that belong to Q_H^{i+1};
  —The *increment* $Q_{I_0}^{i+1}$ contains the states that do not belong to either Q_H^{i+1} or Q_T^{i+1}.

We now give two examples that illustrate the above concepts.

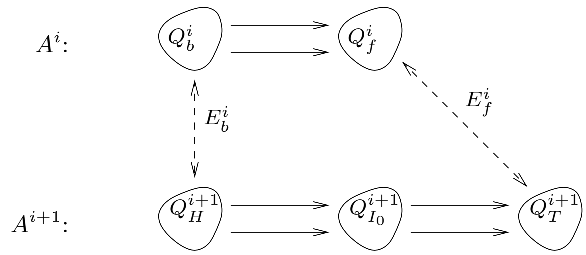


Fig. 3. Partitioning automata states, A^{i+1} is *incrementally larger* than A^i.

Example 5.4. *Definition 5.3 and the notations above are illustrated in Figure 3. The figure only depicts the relations that are introduced by Definition 5.3. In the figure, the double black arrows between Q_H^{i+1} and $Q_{I_0}^{i+1}$ are for the existence of transitions from states of Q_H^{i+1} to states of $Q_{I_0}^{i+1}$. Observe that there could not be a transition from a state in Q_T^{i+1} to Q_H^{i+1} because the two automata are in minimal form and there exists an forward equivalence relation between Q_f^i and Q_T^{i+1}.*

Example 5.5. *Let T^4 and T^8 be two transducers given in Figure 2. Transducer T^8 is incrementally larger than transducer T^4. The forward equivalence between T^4 and T^8 is given by $E_f^4 = \{(0,0),(1,1)\}$. The part of the backward equivalence that contains states that are not covered by the forward equivalence is given by the set $E_b^4 = \{(3,3),(2,2),(4,4)\}$. The set of states of T^4 can be partitionned in two sets : (1) $Q_f^4 = \{0,1\}$, that contains the states covered by E_f^4 and (2) $Q_b^4 = \{2,3,4\}$ that contains the states covered by E_b^4. The head part Q_H^8 of T^8 is the set $\{2,3,4\}$; its tail part is the set $Q_T^8 = \{0,1\}$. Finally, the increment in T^8 is the set of states that are not covered by E_f^8 or E_b^8, i.e., $Q_{I_0}^8 = \{5,6\}$.*

5.2 Detection between multiple automata

Our expectation is that, when moving from one automaton to the next in the sequence, the increment will always be the same. We formalize this property with the following definition.

Definition 5.6. Let $S_I = A^i, A^{i+1}, \ldots, A^{i+k}$ and for each $0 \le j \le k$, let $A^{i+j} = (Q^{i+j}, \Sigma, q_0^{i+j}, \delta^{i+j}, F^{i+j})$ be an automaton. For each $0 \le j < k$, let E_b^{i+j} and E_f^{i+j} be respectively, the backward and the forward equivalences computed between A^{i+j} and A^{i+j+1}. The sequence S_I is an *incrementally growing* sequence if

(1) For each $0 \le j \le k$, A^{i+j} is minimal;
(2) For each $0 \le j \le k-1$, A^{i+j+1} is incrementally larger than A^{i+j};
(3) For each $1 \le j \le k-1$, the *head* increment $Q_{I_0}^{i+j+1}$, which is detected between A^{i+j} and A^{i+j+1}, is the image by E_b^{i+j} of the increment $Q_{I_0}^{i+j}$.

Consider a subsequence $S_I = A^i, A^{i+1}, \ldots, A^{i+k}$ of $A^1, \ldots, A^n$ that grows incrementally. According to Definition 5.6, two successive automata in the sequence satisfy the definition of incrementally larger (item 2 in Definition 5.6), which means that their exists a backward relation between the successive head parts. Moreover, their exists also a backward relation between the successive head increments (item 3,

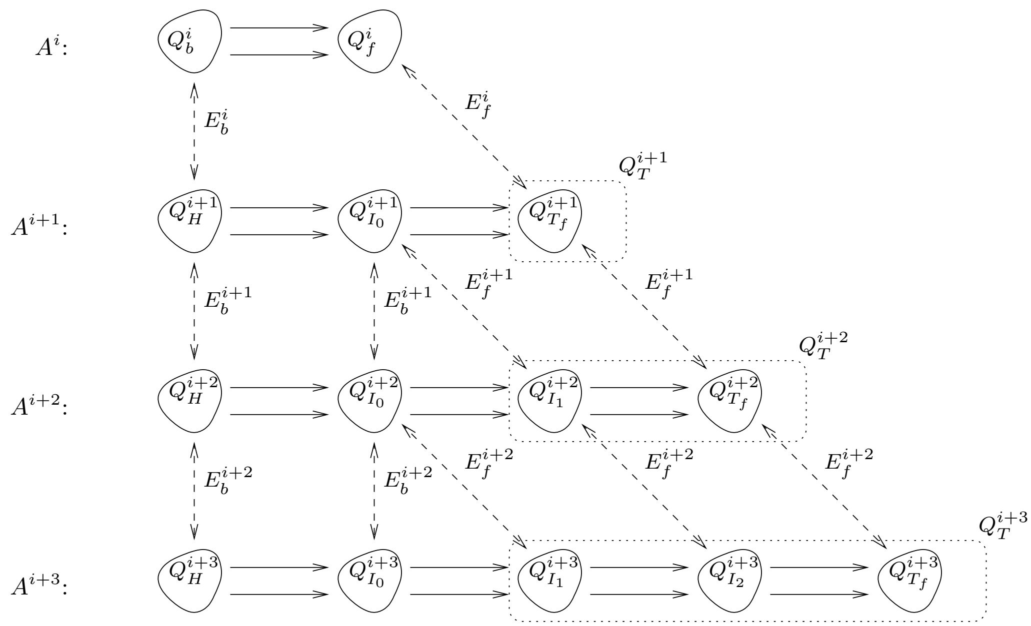


Fig. 4. Automata in an incrementally growing sequence.

Definition 5.6). This implies that for $2 \leq j \leq n$, the tail part Q_T^{i+j} of A^{i+j} consist of $j-1$ tail increments plus a part that we will name the *tail-end set*. Precisely, according to Definition 5.6, Q_T^{i+j} can be partitioned into $\{Q_{I_1}^{i+j}, Q_{I_2}^{i+j}, \ldots, Q_{I_{j-1}}^{i+j}, Q_{T_f}^{i+j}\}$, where

—For each $1 \leq \ell \leq j-1$, the *tail increment* $Q_{I_\ell}^{i+j}$ is the image by the relation $E_f^{i+j-1} \circ E_f^{i+j-2} \circ \cdots \circ E_f^{i+j-\ell}$ of the *head increment* $Q_{I_0}^{i+j-\ell}$;

—The *tail-end set* $Q_{T_f}^{i+j}$ contains the remaining elements of Q_T^{i+j}.

EXAMPLE 5.7. *Definition 5.6 and the discussion above are illustrated in Figure 4, which represents an incrementally growing sequence of automata. Two successive automata in this sequence satisfy Definition 5.3. The relations between the automata are given by Definitions 5.6 and 5.3 (which is embedded in Definition 5.6, item (2)). Forward equivalences between successive automata are due to Definition 5.3. Backward equivalences between the successive head parts are due to Definition 5.6, while backward equivalences between the head increments are due to Definition 5.6 (item (3)). Observe that the picture does not depicts all the possible backward and forward relations between the automata in the sequence. Only the relations that are introduced in Definitions 5.6 and 5.3 are depicted. Those relations will be exploited by the extrapolation procedure. The double black arrow between* $Q_{I_0}^{i+3}$ *to* Q_T^{i+3} *indicate that there may be transitions from states of* $Q_{I_0}^{i+3}$ *to states of* $Q_{I_0}^{i+3} = Q_{I_1}^{i+3} \cup Q_{I_2}^{i+3} \cup Q_{T_f}^{i+3}$.

EXAMPLE 5.8. *Consider Transducers T^4, T^8, and T^{16} given in Figure 2. As observed in Example 5.5, T^8 is incrementally larger than T^4. The head part of T^8 is the set $Q^8_H = \{2,3,4\}$; its tail part is the set $Q^8_T = \{0,1\}$; the increment discovered between T^4 and T^8 is $Q^8_{I_0} = \{5,6\}$. It is easy to see that T^{16} is incrementally larger than T^8. The head part of T^{16} is the set $Q^{16}_H = \{2,3,4\}$; its tail part is the set $Q^{16}_T = \{0,1,5,6\}$. Finally, the increment discoverd between T^8 and T^{16} is the set $Q^{16}_{I_0} = \{7,8\}$. The tail part of T^{16} is divided into two sets : (1) a tail increment $Q^{16}_{I_1} = \{5,6\}$, that is a copy of the increment detected between T^4 and T^8, and (2) a tail-end set $Q^{16}_{T_f} = \{0,1\}$. Let $E^8_f = \{(0,0),(1,1),(5,5),(6,6)\}$ be the forward equivalence between T^8 and T^{16}. We can see that increment $Q^{16}_{I_1}$ is the image of increment $Q^8_{I_0}$ by the relation E^8_f. Moreover, $Q^{16}_{I_0}$ is the image of $Q^8_{I_0}$ by E^8_b. By Theorem 2.8 and the isomorphism property, we have that $Q^{16}_{I_0}$ and $Q^{16}_{I_1}$ are isomorphic.*

Given an automaton A^{i+j} in the sequence S_I, we define its *growing decomposition* w.r.t. S_I, denoted $GROW_{(S_I)}(A^{i+j})$, to be the ordered list $\{Q^{i+j}_H, \{Q^{i+j}_{I_0}, \ldots, Q^{i+j}_{I_{j-1}}\}, Q^{i+j}_T\}$. It is easy to see that, since we are working with minimal automata, the head increment $Q^{i+j}_{I_0}$ of A^{i+j} and all its tail increments $Q^{i+j}_{I_\ell}$, $\ell \in [1, j-1]$ appearing in its tail part Q^{i+j}_T are images of the head increment $Q^{i+1}_{I_0}$ detected between A^i and A^{i+1} by a combination of forward and backward equivalences (See Figure 4 and the end of Example 5.8). This observation extends to all the automata in S_I. Consequently the transition graphs internal[4] to all increments of all the automata in the sequence are isomorphic to that of $Q^{i+1}_{I_0}$, and hence are (by Theorem 2.8 and the isomorphism property) isomorphic to each othe. In the rest of the thesis, this isomorphism relation between two increments is called the *increment isomorphism relation*. Observe also that, since we are working with minimal automata, for each $j \in [1, k-1]$ we have the following:

—The head part Q^{i+j+1}_H is the image by E^{i+j}_b of the head part Q^{i+j}_H. By Theorem 2.8, the internal transition graphs of the head parts of all the automata in the sequence S_I are isomorphic to each other. This isomorphism relation is called the *head isomorphism relation*;

—The tail-end set $Q^{i+j+1}_{T_f}$ is the image by E^{i+j}_f of the tail-end set $Q^{i+j}_{T_f}$. By Theorem 2.8, the internal transition graphs of the tail-end sets of all the automata in the sequence S_I are isomorphic to each other. This isomorphism relation is called the *tail-end set isomorphism relation*.

Our intention is to extrapolate the last automaton of an incrementally growing sequence of automata by adding more increments, following a regular pattern. In order to do this, we need to compare and characterize the transitions leaving different increments.

Definition 5.9. Let $A^{i+k} = (Q^{i+k}, \Sigma, q^{i+k}_0, \delta^{i+k}, F^{i+k})$ be the last automaton of an incrementally growing sequence of automata $S_I = A^i, A^{i+1}, \ldots, A^{i+k}$. Assume that $GROW_{(S_I)}(A^{i+k}) = \{Q^{i+k}_H, \{Q^{i+k}_{I_0}, \ldots, Q^{i+k}_{I_{k-1}}\}, Q^{i+k}_{T_f}\}$. Then, an increment

[4]The transition graph only contains transitions between states of the increment.

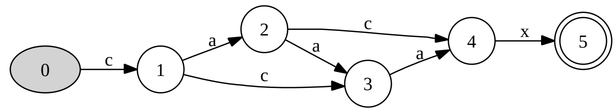


Fig. 5. An automaton for Definition 5.9.

$Q^{i+k}_{I_\alpha}$ $(0 \leq \alpha \ \leq \ k-1)$ is said to be *communication equivalent* to an increment $Q^{i+k}_{I_\beta}$ $(0 \leq \beta \ \leq \ k-1)$ if and only if, for each pair of corresponding states (by the increment isomorphism) (q, q'), $q \in Q^{i+k}_{I_\alpha}$ and $q' \in Q^{i+k}_{I_\beta}$, and $a \in \Sigma$, we have that, either

—$\delta^{i+k}(q, a) \in Q^{i+k}_{I_\alpha}$ and $\delta^{i+k}(q', a) \in Q^{i+k}_{I_\beta}$, hence leading to corresponding states by the existing increment isomorphism between $Q^{i+k}_{I_\alpha}$ and $Q^{i+k}_{I_\beta}$, or

—$\delta^{i+k}(q, a)$ and $\delta^{i+k}(q', a)$ are both undefined, or

—$\delta^{i+k}(q, a)$ and $\delta^{i+k}(q', a)$ both leading to the same state of the tail end $Q^{i+k}_{T_f}$, or

—there exists some $\gamma > 0$ such that $\delta^{i+k}(q, a)$ and $\delta^{i+k}(q', a)$ lead to corresponding states by the increment isomorphism between $Q^{i+k}_{I_{\alpha+\gamma}}$ and $Q^{i+k}_{I_{\beta+\gamma}}$ $(0 \leq \alpha+\gamma, \beta+\gamma \leq k-1)$.

The definition easily generalizes to increments of different automata.

Example 5.10. *Consider the automaton of Figure 5, whose set of states is given by* $\{0, 1, 2, 3, 4, 5\}$. *Assume that* Q *contains three increments that are* $Q_{I_0} = \{1\}$, $Q_{I_1} = \{2\}$, *and* $Q_{I_2} = \{3\}$. *The increments* Q_{I_0} *and* Q_{I_1} *are communication stable. The property does not hold for* Q_{I_0} *and* Q_{I_2} *since a transition labeled with* c *is not defined from states* 3.

For the same reasons, we also need to compare the transitions leaving the head part of different automata in the sequence.

Definition 5.11. Let $A^{i+k-1} = (Q^{i+k-1}, \Sigma, q_0^{i+k-1}, \delta^{i+k-1}, F^{i+k-1})$ and $A^{i+k} = (Q^{i+k}, \Sigma, q_0^{i+k}, \delta^{i+k}, F^{i+k})$ be the two last automata of an incrementally growing sequence of automata $S_I = A^i, A^{i+1}, \ldots, A^{i+k}$. Assume that $GROW_{(S_I)}(A^{i+k-1}) = \{Q_H^{i+k-1}, \{Q_{I_0}^{i+k-1}, \ldots, Q_{I_{k-2}}^{i+k-1}\}, Q_{T_f}^{i+k-1}\}$ and $GROW_{(S_I)}(A^{i+k}) = \{Q_H^{i+k}, \{Q_{I_0}^{i+k}, \ldots, Q_{I_{k-1}}^{i+k}\}, Q_{T_f}^{i+k}\}$. We say that A^{i+k-1} and A^{i+k} are *communication stable* if and only if for each pair of corresponding states (by the increment isomorphism) (q, q'), $q \in Q_H^{i+k-1}$ and $q' \in Q_H^{i+k}$, and $a \in \Sigma$, we have that, either

—$\delta^{i+k-1}(q, a) \in Q_H^{i+k-1}$ and $\delta^{i+k}(q', a) \in Q_H^{i+k}$, hence leading to corresponding states by the existing head isomorphism between Q_H^{i+k-1} and Q_H^{i+k}, or

—$\delta^{i+k-1}(q, a)$ and $\delta^{i+k}(q', a)$ are both undefined, or

—$\delta^{i+k-1}(q, a) = q_f^{i+k-1} \in Q_{T_f}^{i+k-1}$ and $\delta^{i+k}(q', a) = q_f^{i+k} \in Q_{T_f}^{i+k}$, hence leading to corresponding states by the existing tail-end set isomorphism between $Q_{T_f}^{i+k-1}$ and $Q_{T_f}^{i+k}$, or

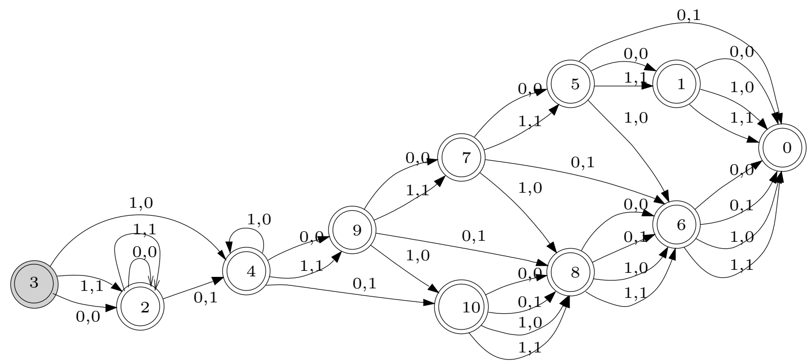


Fig. 6. A transducer that is incrementally larger than T^{16} given in Figure 2.

—$\delta^{i+k-1}(q,a) \in Q_{I_x}^{i+k-1}$ and $\delta^{i+k}(q',a) \in Q_{I_x}^{i+k}$, hence leading to corresponding states by the existing increment isomorphism between $Q_{I_x}^{i+k-1}$ and $Q_{I_x}^{i+k}$ $(0 \leq x \leq k-1)$.

6. EXTRAPOLATION ALGORITHMS

To extrapolate a possibly infinite sequence of minimal finite-word (respectively, minimal weak Büchi) automata $A^1, A^2, \ldots$ we try to extract and extrapolate one of its finite incrementally growing sampling sequences $S_I = A^{s_0}, \ldots, A^{s_k}$. The "candidate" extrapolation for $A^1, A^2, \ldots$ is then given by the *extrapolation* of the sequence S_I. Let $A^{e_0} = A^{s_k}$ be the last automaton of S_I. In order to extrapolate S_I, we simply insert an extra increment between the head part of A^{e_0} and its head increment $Q_{I_0}^{e_0}$, and define its outgoing transitions in order to make this extra increment communication equivalent to $Q_{I_0}^{e_0}$.

Example 6.1. *Let T^4, T^8, T^{16} be the incrementally growing sequence of transducers given in Figure 2. The transducer T^{e_0} given in figure 6 is obtained by adding to T^{16} an increment $Q_{I_0}^{e_0} = \{9, 10\}$ that is communication equivalent to $Q_{I_0}^{16}$. We observe that T^{e_0} is incrementally larger and T^{16}.*

By repeatedly applying the extrapolation step we obtain an extrapolated *infinite* sequence of automata $A^{e_0}, A^{e_1}, \ldots$ which is assumed to be the infinite extension of the sampling sequence S_I. Formally, the *extrapolated sequence* of *origin* A^{e_0} is the infinite sequence of minimal automata $A^{e_0}, A^{e_1}, \ldots$ such that

—For each $i \geq 0$, $A^{s_0}, A^{s_1}, \ldots, A^{s_{k-1}}, A^{e_0}, A^{e_1}, \ldots, A^{e_i}$ grows incrementally;

—For each $i > 0$, A^{e_i} is communication stable with A^{e_0};

—For each $i > 0$, the head increment detected between $A^{e_{i-1}}$ and A^{e_i} is communication equivalent to $Q_{I_0}^{e_0}$.

The limit A^{e_*} of the extrapolated sequence of origin A^{e_0} is thus an extrapolation of the limit of $A^1, A^2, \ldots$.

In this section, we present procedures to build a finite representation for A^{e_*}. We also show that it is possible to add a counter c to A^{e_*} in such a way that when a word is accepted, the value of c is the smallest index i of the automaton A^{e_i} of the extrapolation sequence by which the word is in fact accepted. This counter-based construction will be used in Section 7 to build a criterion to check whether the extrapolation is precise (i.e., is not an over-approximation). For technical reasons, the cases of finite-word and weak Büchi automata are considered separately in Sections 6.2 and 6.3. Before going into the details of those constructions, we will first introduce a theory for counter automata (Section 6.1).

Remark 6.2. The reader who is interested in an over-approximation of A^* which may not be precise can skip Section 6.1 and Propositions 6.13 and 6.19 given in Sections 6.2 and 6.3, respectively.

6.1 Counter-automata

We start with the definition of a counter automaton.

Definition 6.3. A counter-word automaton (counter automaton for short) over an alphabet Σ is a tuple $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$, where

—$n \in \mathbb{N}$ is the counter *dimension* of A,
—$\mathbf{c} = (c_1, \ldots, c_n)$ is a *vector of counters whose values range over the natural numbers.* A *counter valuation* $\mathbf{v} \in \mathbb{N}^n$ for $\mathbf{c}$ is a vector of natural numbers, where the *ith* component of $\mathbf{v}$ assigns a value to c_i,
—Q is a set of *states* (unless stated otherwise, Q is assumed to be finite),
—Σ is a *finite* alphabet,
—$Q_0 \subseteq Q$ is a set of *initial states*,
—$\triangle \subseteq Q \times (\Sigma \times \mathbb{N}^n) \times Q$ is a finite *transition relation*, and
—$F \subseteq Q$ is a set of accepting states.

Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a counter automaton. If $(q_1, (a, \mathbf{v}), q_2) \in \triangle$, then we say that there is a *transition* from q_1 (the *origin*) to q_2 (the *destination*) labeled by a, and associated to the *counter valuation* $\mathbf{v}$. The initial value of each counter is 0, and each time a transition is followed, the current values of the counters are incremented with the counter valuation associated to the transition. Given a counter automaton $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$, the *maximal increment value* of A_c is the smallest $d \in \mathbb{N}$ such that $\triangle \subseteq Q \times (\Sigma \times [0, d]^n) \times Q$. The maximal increment value can always be computed by enumerating the elements of the finite transition relation. As finite automata, counter automata are graphically represented with edge-labeled directed graphs. We emphasize the counter increment vector associated to each transition by preceding it with the symbol "+".

Our aim is to associate counter valuations to the words accepted by a counter automaton. For doing so, we first define a notion of accepted language that does not take the counters into account. We propose the following definition.

Definition 6.4. Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a counter automaton. The *counterless* automaton corresponding to A_c is the finite automaton $A = (Q, \Sigma, Q_0, \triangle', F)$, where

$$\triangle' = \{(q, a, q') \in Q' \times \Sigma \times Q' \mid (\exists \mathbf{v} \in \mathbb{N}^n)((q, (a, \mathbf{v}), q') \in \triangle)\}.$$

Definition 6.5. The *language* accepted by a counter automaton A_c, denoted $L(A_c)$, is the language accepted by its corresponding counterless automaton. If $w \in L(A_c)$, then we say that w is *accepted* by A_c.

We now describe how and when a counter automaton can assign counter values to the words it accepts. Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a counter automaton. Assume first that A_c describes a set of finite words. A *run* of A_c on a finite word $w : \{0, \ldots, m-1\} \rightarrow \Sigma$ is a labeling $\rho : \{0, \ldots, m\} \rightarrow (Q \times \mathbb{N}^n)$ such that

(1) $\rho(0) \in (Q_0 \times \mathbf{0})$, and
(2) $(\forall 0 \leq i \leq m-1)$, $\rho(i+1) = (q_{i+1}, \mathbf{v_{i+1}})$ if and only if $\rho(i) = (q_i, \mathbf{v_i})$ and there exists $(q_i, (w(i), \mathbf{v}), q_{i+1}) \in \triangle$ with $\mathbf{v_{i+1}} = \mathbf{v_i} + \mathbf{v}$.

Let $\rho(m-1) = (q_f \times \{\mathbf{v}\})$. If $q_f \in F$, then we say that ρ is an *accepting run* and that w is *accepted* by A_c with the counter valuation $\mathbf{v}$. Otherwise ρ is *rejecting* for w. The automaton A_c being a finite-word automaton, we can always associate at least one counter valuation to each word $w \in L(A_c)$. Observe that if the counterless automaton of A_c behaves non deterministically on w, then this word may be associated to several counter valuations. There can be accepting and nonaccepting runs that assign the same counter valuation to w.

We now switch to the case of infinite words. A run of A_c on an infinite word $w : \mathbb{N} \rightarrow \Sigma$ is a labeling $\rho : \mathbb{N} \rightarrow (Q \times \mathbb{N}^n)$ such that

(1) $\rho(0) \in (Q_0 \times \mathbf{0})$, and
(2) $(\forall 0 \leq i)$, $\rho(i+1) = (q_{i+1}, \mathbf{v_{i+1}})$ if and only if $\rho(i) = (q_i, \mathbf{v_i})$ and there exists $(q_i, (w(i), \mathbf{v}), q_{i+1}) \in \triangle$ with $\mathbf{v_{i+1}} = \mathbf{v_i} + \mathbf{v}$.

Contrary to the finite-word case, it is generally not possible to associate a counter valuation to ρ. Indeed, there could be the case that the counters are incremented an unbounded number of times. There are however sub-classes of infinite-word counter automata for which it is always possible to assign a counter valuation to each of its runs. This is illustrated with the following definition.

Definition 6.6. Let $A_c = (n, Q, \Sigma, Q_0, \triangle, F)$ be a weak Büchi counter automaton. We say that A_c is *run-bounded* if for each of its accepting strongly connected components $S \subseteq F$ and states $q_1, q_2 \in S$, any transition that goes from q_1 to q_2 is associated with the counter valuation $\mathbf{0}$.

The structure of a run-bounded weak Büchi counter automaton ensures that for each of its runs, after having followed a finite number of transitions, the values of the counters are no longer incremented. Hence, one can reason on a finite prefix of the run to deduce its counter valuation. Let $A_c = (n, Q, \Sigma, Q_0, \triangle, F)$ be a run-bounded weak Büchi counter automaton and ρ be one of its runs. We say that ρ is an *accepting* run and that w is *accepted* by A_c with the counter valuation $\mathbf{v}$ if and only if $\mathit{inf}(\rho) \cap (F \times \{\mathbf{v}\}) \neq \emptyset$, where $\mathit{inf}(\rho)$ is the set of configurations that appear infinitely often in ρ. Otherwise ρ is *rejecting* for w.

In the rest of this paper, we will only consider finite-word and run-bounded weak Büchi counter automaton. We can now define a notion of counter language, which takes the counters into account.

Definition 6.7. The *counter language* of a counter automaton A_c, denoted $\mathcal{L}(A_c)$, is the set of pairs $(w, \mathbf{v})$ such that w can be accepted by A_c with counter valuation $\mathbf{v}$.

We will also use the definition of *counter-zero automaton*.

Definition 6.8. Let $A = (Q, \Sigma, Q_0, \triangle, F)$ be a finite-word (respectively, Büchi automaton), the *counter-zero automaton* corresponding to A is the one-dimensional counter automaton $A_c = (1, \mathbf{c_1}, Q, \Sigma, Q_0, \triangle', F)$, where

—$\triangle' = \{(q, (a, \mathbf{0}), q') \in Q \times (\Sigma \times \mathbf{0}) \times Q \mid (q, a, q') \in \triangle\}$.

Remark 6.9. The class of counter-word automata is particular with respect to existing classes of counter automata[5] such as reversal bounded counter automata [Ibarra 1978], constraint automata [**?**], Parikh automata [**?**], or weighted automata [**?**]. Indeed, counter-word automata use the counter part of the automaton to assign counter valuations to a word when this word is accepted by the automaton, rather than to restrict the language accepted by the automaton. Introducing constraints on the counters before the word is accepted[6] generally leads to more powerful models[7] for which most problems are undecidable. The expressiveness of those models is not needed for the practical applications we considered in the paper.

6.2 Extrapolation for Finite-word Automata

Assume A^{e_0} to be a finite-word automaton. We propose to build a finite representation of A^{e_*} by adding to A^{e_0} new transitions that simulate the existence of additional increments.

Consider the automaton A^{e_0} with $GROW_{(S_I)}(A^{e_0}) = \{Q_H^{e_0}, \{Q_{I_0}^{e_0}, \ldots, Q_{I_{k-1}}^{e_0}\}, Q_{T_f}^{e_0}\}$. Suppose the existence of a transition labeled by a from a state x of $Q_{I_0}^{e_0}$ to a state x' of $Q_{I_3}^{e_0}$. Since, the increment $Q_{I_0}^{e_1}$ added between A^{e_0} and A^{e_1} is communication equivalent to $Q_{I_0}^{e_0}$, there must exist a transition t labeled by a from the state isomorphic to x in $Q_{I_0}^{e_1}$ to the state isomorphic to x' in $Q_{I_2}^{e_1}$. Our construction simulates t in A^{e_0} by adding a transition t' labeled by a from x to the state isomorphic to x' in $Q_{I_2}^{e_0}$. This construction can be repeated for the addition of a second increment. The simulation of "more than two increments" is done by adding transitions between states of $Q_{I_0}^{e_0}$. Due to the communication equivalence property, a similar principle has to be applied for outgoing transitions from $Q_H^{e_0}$. The situation is illustrated in Figure 7 where a part of A^{e_0} has been represented. The dashed transitions in the figure are the transitions added during the extrapolation process.

Formally, a finite representation of A^{e_*} can be built from A^{e_0} with the construction underlined in the following proposition.

PROPOSITION 6.10. *Let A^{e_0} defined over Σ be a minimal finite-word automaton which is the last automaton of an incrementally growing sequence of automata S_I. Assume that $GROW_{(S_I)}(A^{e_0}) = \{Q_H^{e_0}, \{Q_{I_0}^{e_0}, \ldots, Q_{I_{k-1}}^{e_0}\}, Q_{T_f}^{e_0}\}$. One can compute a finite-word automaton A^{e_*} that represents the limit of the extrapolated sequence of origin A^{e_0}.*

[5] As an example, we cannot test the values of the counters.
[6] As an example, one could associate constraints on each transition.
[7] As an example, models that can recognize nonregular languages [**?**].

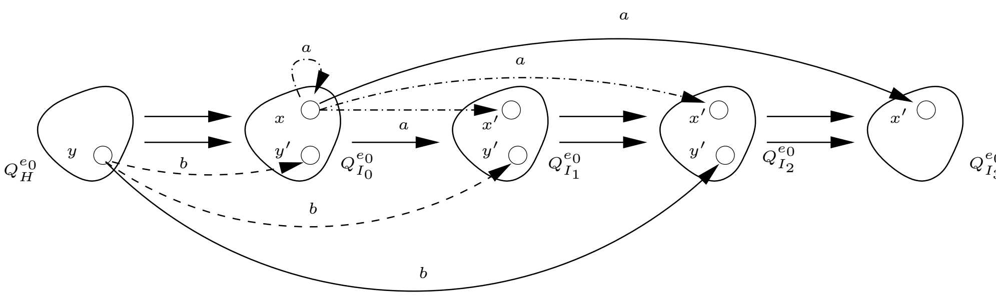


Fig. 7. Illustration of the extrapolation procedure for finite-word automata.

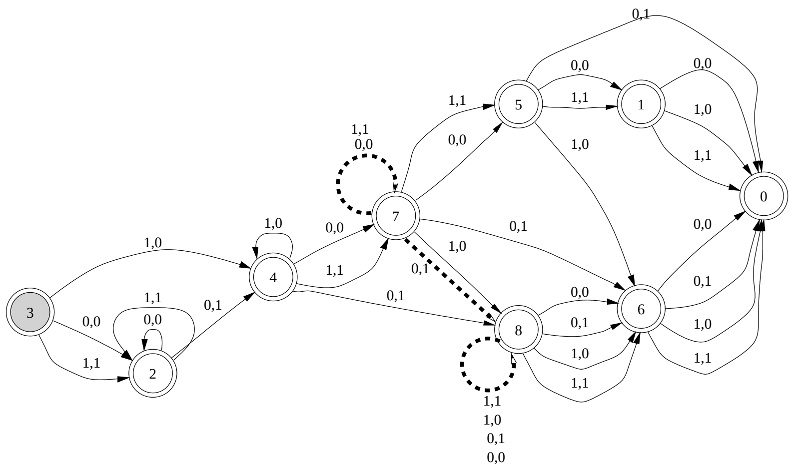


Fig. 8. Transducer for Example 6.11.

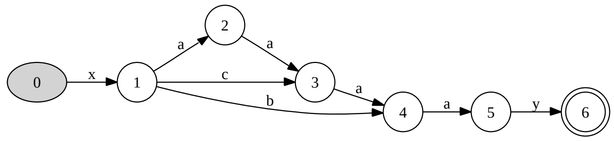
Proof. Let δ be the transition relation of A^{e_0}. The automaton A^{e_*} can be built from A^{e_0} by augmenting δ using the following rule:

> For each state $q \in Q_H^{e_0} \cup Q_{I_0}^{e_0}$ and $a \in \Sigma$, if $\delta(q, a)$ leads to a state q' in an increment Q_{I_j}, $1 \leq j \leq k-1$, then for each $0 \leq \ell < j$, add a transition (q, a, q''), where q'' is the state corresponding to q' (by the increment isomorphism) in $Q_{I_\ell}^{e_0}$.

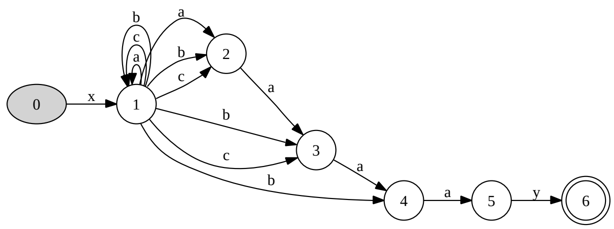
To see that any word acepted by an A^{e_i} $(i \geq 0)$ can also be accepted by A^{e_*}, notice that any run of an A^{e_i} can be simulated by A^{e_*}, using the added transitions. Conversely, any run of A^{e_*} is finite and thus only uses the added cyclic transitions a finite number of times. It can thus be simulted by some A^{e_i}. The principle is illustrated in Example 6.11. □

We illustrate the construction with two examples.

Example 6.11. *Figure presents an application of the extrapolation procedure to the transducer T^{16} given in Figure 2. The dashed transitions are those added by the extrapolation process. As an example, the word* $(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)$,



(a) A^{e_0}



(b) A^{e_*}

Fig. 9. Automata for Example 6.12.

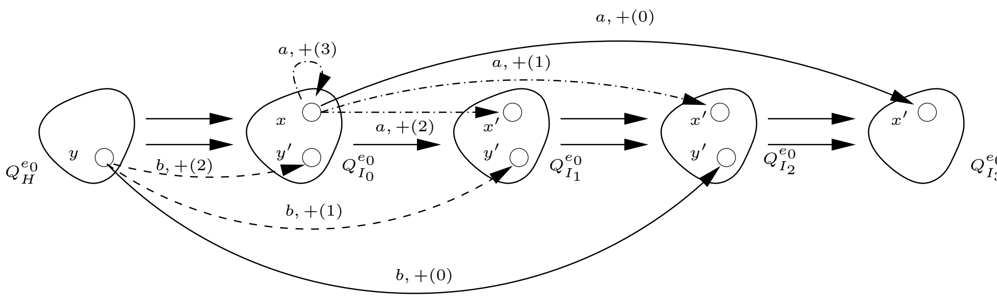


Fig. 10. Illustration of the extrapolation algorithm for finite-word automata with the addition of counter values.

which can be accepted only if we follow a dashed transition, is also accepted by the transducer T^{e_0} *given in Figure 6.*

EXAMPLE 6.12. *Consider the minimal finite-word automaton* A^{e_0} *given in Figure 9(a), with* $Q_H^{e_0} = \{0\}$, $Q_{I_0}^{e_0} = \{1\}$, $Q_{I_1}^{e_0} = \{2\}$, $Q_{I_2}^{e_0} = \{3\}$, $Q_{I_3}^{e_0} = \{4\}$, *and* $Q_{T_f}^{e_0} = \{5, 6\}$. *Applying the construction of Proposition 6.10 to* A^{e_0} *gives the automaton* A^{e_*} *in Figure 9(b).*

We now show that it is possible to add a counter c to A^{e_*} in such a way that when a word is accepted, the value of c is the smallest index i of the automaton A^{e_i} of the extrapolation sequence by which the word is in fact accepted. Our construction labels each transition added to A^{e_0} with a value that represents the number of increments simulated by this transition. In Figure 10 we sketch the construction for the automaton given in Figure 7.

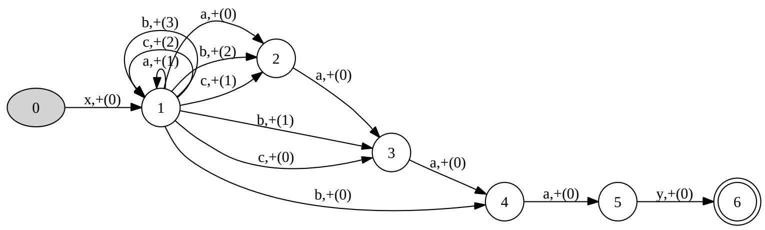


Fig. 11. Automaton for Example 6.14.

PROPOSITION 6.13. *Let $A^{e_0} = (Q, \Sigma, Q_0, \delta, F)$ be a minimal finite-word automaton which is the last automaton of a finite incrementally growing sequence of automata S_I. Assume that $GROW_{(S_I)}(A^{e_0}) = \{Q_H^{e_0}, \{Q_{I_0}^{e_0}, \ldots, Q_{I_{k-1}}^{e_0}\}, Q_{T_f}^{e_0}\}$ and let $A^{e_0}, A^{e_1}, \ldots$ be the extrapolated sequence of origin A^{e_0}. One can compute a finite-word counter automaton $A_c^{e_*}$ such that (1) $L(A_c^{e_*}) = \bigcup_{i \geq 0} L(A^{e_i})$, (2) for each $(w, i) \in \mathcal{L}(A_c^{e_*})$, $w \in L(A^{e_i})$, and (3) for each $i \geq 0$, $w \in L(A^{e_i})$, $0 \leq j \leq i$ exists such that $(w, j) \in \mathcal{L}(A_c^{e_*})$.*

PROOF. Let δ be the transition relation of A^{e_0}. The one-dimensional counter automaton $A_c^{e_*}$ is given by $(1, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$, with $\triangle$ defined as follows:

—Start with $\triangle = \{\emptyset\}$;

—For each $(q, a, q') \in \delta$, add $(q, (a, \mathbf{0}), q')$ to $\triangle$;

—For each state $q \in Q_H^{e_0} \cup Q_{I_0}^{e_0}$ and $a \in \Sigma$,

If $\delta(q, a)$ leads to a state q' in an increment Q_{I_j}, $1 \leq j \leq k-1$, then for each $0 \leq \ell < j$, add to $\triangle$ a transition $(q, (a, j - l), q'')$, where q'' is the state corresponding to q' (by the increment isomorphism) in $Q_{I_\ell}^{e_0}$.

□

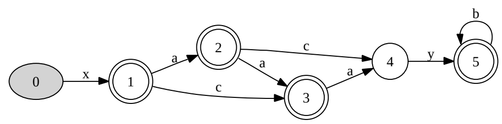
Let $A_c^{e_0}$ be the counter-zero automaton corresponding to A^{e_0}. We directly see that for each $i > 0$, $w \in L(A^{e_i}) \setminus L(A^{e_0})$, $1 \leq j \leq i$ exists such that $(w, j) \in \mathcal{L}(A_c^{e_*}) \setminus \mathcal{L}(A_c^{e_0})$. Indeed, since $w \notin L(A^{e_0})$, any accepted run on w must pass by states of one of the added increments and j cannot be equal to 0.

EXAMPLE 6.14. *Figure 11 presents the result of applying the construction of Proposition 6.13 to Automaton A^{e_0} of Example 6.12.*

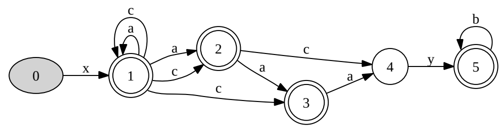
6.3 Extrapolation for Weak Büchi Automata

Assume now A^{e_0} to be a deterministic weak Büchi automaton. In such a case, a finite representation of the extrapolated sequence of origin A^{e_0} cannot be computed with the construction of Proposition 6.10.

EXAMPLE 6.15. *Consider the minimal weak Büchi automaton A^{e_0} given in Figure 12(a), with $Q_H^{e_0} = \{0\}$, $Q_{I_0}^{e_0} = \{1\}$, $Q_{I_1}^{e_0} = \{2\}$, $Q_{I_2}^{e_0} = \{3\}$, and $Q_{T_f}^{e_0} = \{4, 5\}$. Applying the construction of Proposition 6.10 to A^{e_0} gives the automaton A^{e_*} in Figure 12(b). This automaton accepts the word xa^ω which cannot be accepted by one of the automata A^{e_i} in the extrapolated sequence of origin A^{e_0}.*



(a) A^{e_0}



(b) A^{e_*}

Fig. 12. A weak Büchi automaton and its extrapolation with the construction of Proposition 6.10
.

The example above shows that applying the construction of Proposition 6.10 to A^{e_0} may introduce new cycles from states of $Q^{e_0}_{I_0}$ to themselves. Since the accepting runs of the A^{e_i} can only go through a *finite number* of increments, it is essential to make these cycles nonaccepting. The problem can easily be solved, as stated with the following proposition.

PROPOSITION 6.16. *Let A^{e_0} defined over Σ be a minimal weak Büchi automaton which is the last element of an incrementally growing sequence of automata S_I. Assume that $GROW_{(S_I)}(A^{e_0}) = \{Q^{e_0}_H, \{Q^{e_0}_{I_0}, \ldots, Q^{e_0}_{I_{k-1}}\}, Q^{e_0}_{T_f}\}$. One can compute a weak Büchi automaton A^{e_*} that represents the limit of the extrapolated sequence of origin A^{e_0}.*

PROOF. Let δ be the transition relation of A^{e_0}. The automaton A^{e_*} that represents the limit of the extrapolated sequence whose origin is A^{e_0} can be built from A^{e_0} by augmenting its set of states and transitions with the following rules:

(1) Build an isomorphic copy $A_{I_0 copy}$ of the automaton formed by the states in $Q^{e_0}_{I_0}$, the transitions between them, and the outgoing transitions from these states to states in $Q^{e_0}_{I_1}$, $Q^{e_0}_{I_2}$, $\ldots$, $Q^{e_0}_{I_{k-1}}$, and $Q^{e_0}_{T_f}$;

(2) Make all the states of $A_{I_0 copy}$ nonaccepting;

(3) For each state $q \in Q^{e_0}_{I_0} \cup Q^{e_0}_H$ and $a \in \Sigma$, if $\delta(q, a)$ leads to a state q' in an increment $Q^{e_0}_{I_j}$, $1 \leq j \leq k-1$, then
   - (a) For each $1 \leq \ell < j$, add a transition (q, a, q''), where q'' is the state corresponding to q' (by the increment isomorphism) in $Q^{e_0}_{I_\ell}$. Also, add a transition (q, a, q''), where q'' is the state corresponding to q' in $A_{I_0 copy}$;
   - (b) If $q \in Q_{I_0}$, then let q_{copy} be the state corresponding to q in $A_{I_0 copy}$. For each $1 \leq \ell < j$, add a transition (q_{copy}, a, q''), where q'' is the state corresponding to q' (by the increment isomorphism) in $Q^{e_0}_{I_\ell}$. Also, add a transition (q_{copy}, a, q''), where q'' is the state corresponding to q' in $A_{I_0 copy}$.

□

The construction in the proposition above follows from the one given in Proposition 6.10. The only slight difference is in the duplication of the head increment, which

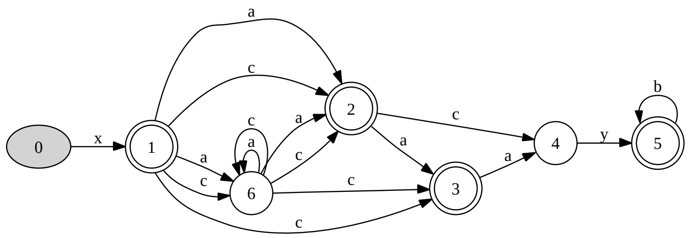
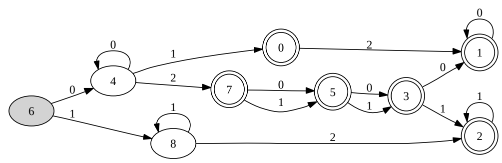
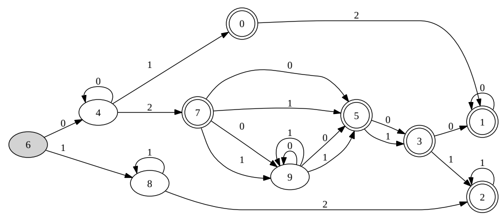


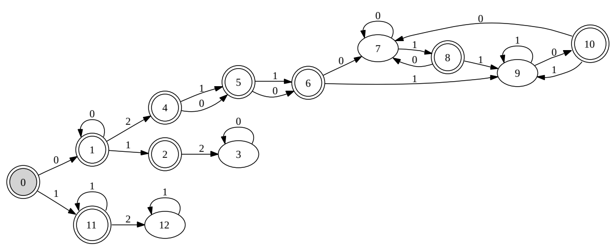
Fig. 13. A weak Büchi automaton for Example 6.17 .



(a) A^{e_0}



(b) $A_1^{e_*}$



(c) $A_2^{e_*}$

Fig. 14. Büchi automata for the proof of Proposition 6.18 .

is needed to make sure that new cycles added to A^{e_0} are nonaccepting.

Example 6.17. *The automaton in Figure 13 is the result of applying the construction of Proposition 6.16 to Automaton A^{e_0} of Example 6.15.*

Proposition 6.18. *Let A^{e_*} be the result of applying the construction of Proposition 6.16 to A^{e_0}, the last automaton of a finite incrementally growing sequence of deterministic weak Büchi automata. The automaton A^{e_*} may not be weak deter-*

ministic.

PROOF. Consider the minimal weak Büchi automaton A^{e_0} given in Figure 14(a), with $Q_H^{e_0} = \{6,4\}$, $Q_{I_0}^{e_0} = \{7\}$, $Q_{I_1}^{e_0} = \{5\}$, and $Q_{T_f}^{e_0} = \{0,1,2,3,8\}$. Applying the construction of Proposition 6.16 to A^{e_0} gives the nondeterministic weak Büchi automaton A^{e_*} in Figure 14(b). In this automaton, the state labeled by 9 is the duplication of $Q_{I_0}^{e_0}$. The result of determinizing $A_1^{e_*}$ is the deterministic co-Büchi automaton $A_2^{e_*}$ that is given in Figure 14(c). It is easy to see that this automaton is not inherently weak and, consequently, cannot be turned to a weak Büchi automaton. □

Following what has been done for the case of finite-word automata, we now propose to add a counter c to A^{e_*} in such a way that when a word is accepted, the value of c is the smallest index i of the automaton A^{e_i} of the extrapolated sequence by which the word is in fact accepted.

PROPOSITION 6.19. *Let $A^{e_0} = (Q, \Sigma, Q_0, \delta, F)$ be a minimal weak Büchi automaton which is the last element of an incrementally growing sequence of automata S_I. Assume that $GROW_{(S_I)}(A^{e_0}) = \{Q_H^{e_0}, \{Q_{I_0}^{e_0}, \dots, Q_{I_{k-1}}^{e_0}\}, Q_{T_f}^{e_0}\}$ and let $A^{e_0}, A^{e_1}, \dots$ be the extrapolated sequence of origin A^{e_0}. One can compute a run-bounded weak Büchi counter automaton $A_c^{e_*}$ such that (1) $L(A_c^{e_*}) = \bigcup_{i \geq 0} A^{e_i}$, (2) for each $(w,i) \in \mathcal{L}(A_c^{e_*})$, $w \in L(A^{e_i})$, and (3) for each $w \in L(A^{e_i})$, $j \leq i$ exists such that $(w,j) \in \mathcal{L}(A_c^{e_*})$.*

PROOF. Let δ be the transition relation of A^{e_0}. The one-dimensional counter automaton $A_c^{e_*}$ is given by $(1, \mathbf{c}, Q', \Sigma, Q_0, \triangle, F)$, with Q and $\triangle$ defined as follows:

(1) Start with $\triangle = \{\emptyset\}$;

(2) For each $(q,a,q') \in \delta$, add $(q,(a,\mathbf{0}),q')$ to $\triangle$;

(3) Build an isomorphic copy $A_{I_0\,copy}$ of the automaton formed by the states in $Q_{I_0}^{e_0}$, the transitions between them, and the outgoing transitions from these states to states in $Q_{I_1}^{e_0}$, $Q_{I_2}^{e_0}$, $\dots$, $Q_{I_{k-1}}^{e_0}$, and $Q_{T_f}^{e_0}$. All the transitions are associated with the counter increment 0;

(4) Make all the states of $A_{I_0\,copy}$ nonaccepting;

(5) For each state $q \in Q_{I_0}^{e_0} \cup Q_H^{e_0}$ and $a \in \Sigma$, if $\delta(q,a)$ leads to a state q' in an increment $Q_{I_j}^{e_0}$, $1 \leq j \leq k-1$, then

   (a) For each $1 \leq \ell < j$, add to $\triangle$ a transition $(q,(a,j-l),q'')$, where q'' is the state corresponding to q' (by the increment isomorphism) in $Q_{I_\ell}^{e_0}$. Also, add a transition $(q,(a,j),q'')$, where q'' is the state corresponding to q' in $A_{I_0\,copy}$;

   (b) If $q \in Q_{I_0}$, then let q_{copy} be the state corresponding to q in $A_{I_0\,copy}$. For each $1 \leq \ell < j$, add to $\triangle$ a transition $(q_{copy},(a,j-l),q'')$, where q'' is the state corresponding to q' (by the increment isomorphism) in $Q_{I_\ell}^{e_0}$. Also, add a transition $(q_{copy},(a,j),q'')$, where q'' is the state corresponding to q' in $A_{I_0\,copy}$.

□

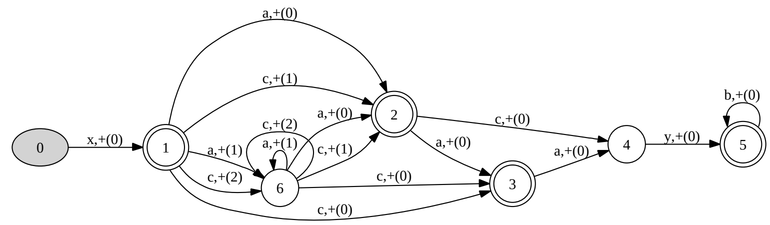


Fig. 15. Automaton for Example 6.20.

Let $A_c^{e_0}$ be the counter-zero automaton corresponding to A^{e_0}. From the observations above, we directly see that for each $i \in \mathbb{N}_0$ $w \in L(A^{e_i}) \setminus L(A^{e_0})$, $1 \leq j \leq i$ exists such that $(w, j) \in \mathcal{L}(A_c^{e_*}) \setminus \mathcal{L}(A_c^{e_0})$.

Example 6.20. *Figure 15 presents the result of applying the construction of Proposition 6.19 to Automaton A^{e_0} of Example 6.15.*

7. SAFETY AND PRECISENESS

After having constructed a finite automaton A^{e_*} representing the extrapolation of a sequence A^1, $A^2, \ldots$ of automata, it remains to check whether it accurately corresponds to what we really intend to compute, i.e., $\bigcup_{i>0} A^i$. This is done by first checking that the extrapolation is *safe*, in the sense that it captures all behaviors of $\bigcup_{i>0} A^i$, and then checking that it is *precise*, i.e., that it has no more behaviors than $\bigcup_{i>0} A^i$. We check both properties using sufficient conditions. We develop separately these conditions for the two (ω-)Regular Reachability Problems in Sections 7.2 and 7.3, respectively. For both cases, our solution for the preciseness problem builds on counter automata. Before going into the details of those constructions, we will first introduce new concepts in our theory for counter automata (Section 7.1).

Remark 7.1. The reader who is interested in a safe, but not necessarily precise solution can skip Section 7.1 and the description of the preciseness criteria given in Sections 7.2 and 7.3.

Remark 7.2. While the safety check we propose is rather standard, there are many other possible preciseness criteria depending of the problem under consideration.

7.1 On operations between and on counter-automata

We first extend the operations of intersection and composition defined for finite automata to counter automata. We propose the following definitions.

Definition 7.3. Let $A_{c_1} = (n_1, \mathbf{c_1}, Q_1, \Sigma, Q_{01}, \triangle_1, F_1)$ and $A_{c_2} = (n_2, \mathbf{c_2}, Q_2, \Sigma, Q_{02}, \triangle_2, F_2)$ be two finite-word (respectively, run-bounded weak Büchi) counter automata. The *counter-intersection* between A_{c_1} and A_{c_2}, denoted $A_{c_1} \cap_c A_{c_2}$, is the finite-word (respectively, run-bounded weak Büchi) counter automaton $A_c = (n_1 + n_2, \mathbf{c_1} \times \mathbf{c_2}, Q, \Sigma, Q_0, \triangle, F)$ with $L(A_c) = L(A_{c_1}) \cap L(A_{c_2})$ and $\mathcal{L}(A_c) = \{(w, \mathbf{v}) \in \Sigma^\infty \times \mathbb{N}^{n_1+n_2} \mid (\exists (w, \mathbf{v_1}) \in \mathcal{L}(A_{c_1}))(\exists (w, \mathbf{v_2}) \in \mathcal{L}(A_{c_2}))(\mathbf{v} = \mathbf{v_1} \times \mathbf{v_2})\}$.

Definition 7.4. Let $T_{c_1} = (n_1, \mathbf{c_1}, Q_1, \Sigma^2, Q_{01}, \triangle_1, F_1)$ and $T_{c_2} = (n_2, \mathbf{c_2}, Q_2, \Sigma^2, Q_{02}, \triangle_2, F_2)$ be two finite-word (respectively, run-bounded weak Büchi) counter transducers. The *counter-composition* of T_{c_1} by T_{c_2}, denoted $T_{c_2} \circ_c T_{c_1}$, is the finite-word (respectively, run-bounded weak Büchi) counter transducer $T_c = (n_1 + n_2, \mathbf{c_1} \times \mathbf{c_2}, Q, \Sigma^2, Q_0, \triangle, F)$, with $L(T_c) = L(T_2 \circ T_1)$ and $\mathcal{L}(T_c) = \{(w, \mathbf{v}) \in \Sigma^\infty \times \mathbb{N}^{n_1+n_2} \mid (\exists (w_1, \mathbf{v_1}) \in \mathcal{L}(T_{c_1}))(\exists (w_2, \mathbf{v_2}) \in \mathcal{L}(T_{c_2}))(\mathbf{v} = \mathbf{v_1} \times \mathbf{v_2} \wedge w = w_2 \circ w_1)\}$.

Definition 7.5. Let $T_1 = (Q_1, \Sigma^2, Q_{01}, \triangle_1, F_1)$ be a finite-word (respectively, run-bounded weak Büchi) transducer, and $A_{c_2} = (n_2, \mathbf{c_2}, Q_2, \Sigma, Q_{02}, \triangle_2, F_2)$ be a finite-word (respectively, run-bounded weak Büchi) counter automaton. The *counter-image* of A_{c_2} by T_1, denoted $T_1(A_{c_2})$, is the finite-word (respectively, Büchi) counter automaton $A_c = (n_2, \mathbf{c_2}, Q, \Sigma, Q_0, \triangle, F)$, where $L(A_c) = L(T_1(A_{c_2}))$ and $\mathcal{L}(A_c) = \{(w, \mathbf{v_2}) \in \Sigma^\infty \times \mathbb{N}^{n_2} \mid (\exists w_1 \in L(T_1))(\exists (w_2, \mathbf{v_2}) \in \mathcal{L}(A_{c_2}))(w = w_2 \circ w_1)\}$.

Let A_c be a n-dimensional counter automaton over the alphabet Σ, and d its maximal increment value. The *extended automaton* of A_c, denoted $(A_c)^e$, is the finite automaton (without counters) obtained from A_c by augmenting the label of each of its transitions with its corresponding counter valuation. We have the following definition.

Definition 7.6. Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a counter automaton whose maximal increment value is d. The *extended* automaton corresponding to A_c is the finite automaton $A = (Q, \Sigma', Q_0, \triangle', F)$, where

—$\Sigma' = \Sigma \times [0, d]^n$, and
—$\triangle' = \{(q, a', q') \in Q' \times \Sigma' \times Q' \mid (\exists \mathbf{v} \in \mathbb{N}^n)((q, (a, \mathbf{v}), q') \in \triangle \wedge a' = a \times \mathbf{v})\}$.

A n-dimensional counter automaton over an alphabet Σ and whose maximal increment value is d can be viewed as a finite automaton over an alphabet $\Sigma \times [0, d]^n$ and, alternatively, a finite automaton over an alphabet $\Sigma \times [0, d]^n$ can be viewed as a n-dimensional counter automaton over an alphabet Σ and whose maximal increment value is d. The alphabet $\Sigma \times [0, d]^n$ is referred to as the *extended alphabet* of A_c.

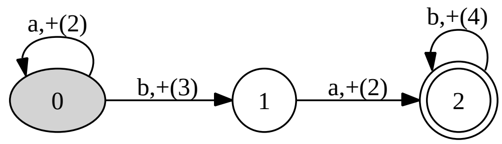
If A_c is a finite-word counter automaton, then we say that it is *universal* if and only if $L((A_c)^e) = (\Sigma \times [0, d]^n)^*$. If A_c is a run-bounded weak Büchi counter automaton, then it is *universal* if and only if $L((A_c)^e) = (\Sigma \times [0, d]^n)^*(\Sigma \times 0)^\omega$.

Definition 7.7. Consider two counter automata A_{c_1} and A_{c_2} of same dimensions. The *extended intersection (respectively, union)* between A_{c_1} and A_{c_2}, denoted $A_{c_1} \cap_e A_{c_2}$ (respectively, $A_{c_1} \cup_e A_{c_2}$), is a counter automaton A_c such that $(A_c)^e = (A_{c_1})^e \cap (A_{c_2})^e$ (respectively, $(A_c)^e = (A_{c_1})^e \cup (A_{c_2})^e$).

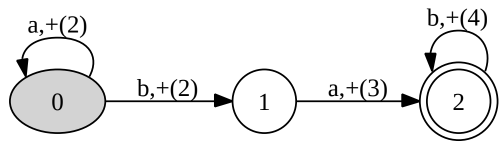
The extended intersection (respectively, union) of two counter automata can easily be computed by applying a classical intersection (respectively, union) algorithm to their extended version. We also have the following proposition.

Proposition 7.8. *The extended intersection/union of two run-bounded weak Büchi counter automata is a run-bounded weak Büchi counter automaton.*

The problem of testing the equivalence between counter languages is known to be undecidable for many classes of counter automata [Ibarra 1978], but decidability t]



(a) A_1



(b) A_2

Fig. 16. Two finite-word counter automata.

results exist for some very particular classes [Roos 1988]. The algorithms involved in those decidability results are known to be of high complexity and difficult to implement. Rather than trying to extend those results to counter-word automata, we preferred to propose a sufficient criterion that can easily be implemented with simple automata-based manipulations. Our criterion is formalized with the following proposition.

Proposition 7.9. *Let A_{c_1} and A_{c_2} be two finite-word (respectively, Büchi) counter automata of same dimension. If $L(A^e_{c_1}) = L(A^e_{c_2})$, then $\mathcal{L}(A_{c_1}) = \mathcal{L}(A_{c_2})$.*

Example 7.10. *Consider the two finite-word counter automata A_{c_1} and A_{c_2} given in Figure 16. The automaton $A^e_{c_1}$ does not accept the same language as $A^e_{c_2}$. However $\mathcal{L}(A_{c_1}) = \mathcal{L}(A_{c_2})$.*

The projection operation for finite automata extends to a *counter projection* for counter automata. We have the following definition.

Definition 7.11. Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a counter automaton. For $1 \leq i \leq n$, the projection of A_c w.r.t. counter c_i, denoted $\Pi_{\neq c_i}(A_c)$ is the counter automaton $A'_c = (n-1, \mathbf{c}', Q, \Sigma, Q_0, \triangle', F)$, where $\mathbf{c}' = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$, $L(A_c) = L(A'_c)$, and $\mathcal{L}(A'_c) = \{(w, \mathbf{c_1} \times \mathbf{c_2}) \in \Sigma^\infty \times \mathbb{N}^{n-1} \mid (\exists\, \mathbf{c_3} \in \mathbb{N})$ $((w, \mathbf{c_1} \times \mathbf{c_3} \times \mathbf{c_2}) \in \mathcal{L}(A_c))\}$.

In the rest of the paper, we use the shortcut $\Pi_{(\neq\{c_1, c_2, \ldots, c_n\})}(A)$ for $\Pi_{(\neq c_1)}(\Pi_{(\neq c_2)} \ldots (\Pi_{(\neq c_n)}(A)) \ldots)$.

We now present a methodology that given a counter automaton A, computes another counter automaton A' whose accepting words are those of A that satisfy counter constraints. We start with the following definition.

Definition 7.12. Let A_c be a finite-word (respectively, run-bounded weak Büchi) n-dimensional counter automaton and $1 \leq i, j \leq n$ be an integer. We define $(A_c)^{c_i > c_j}$ to be the counter automaton obtained from A_c by removing all the accepting runs that do not assign a greater value to c_i than to c_j. The automaton $(A_c)^{c_i>c_j}$ may have an infinite set of states since its language may not be regular.

In the rest of the paper, we use the notation $(A_c)^{(c_1>\{c_2,\ldots,c_n\})}$ to denote $(\ldots((A_c)^{c_1>c_2})^{c_1>c_3}\ldots)^{c_1>c_n}$.

Let A_c be a finite-word (respectively, run-bounded weak Büchi) n-dimensional counter automaton over Σ and whose maximal increment value is d. A way to compute $(A_c)^{c_1>c_2}$ could be to build a universal finite-word (respectively, run-bounded weak Büchi automaton) A^U defined over the same extended alphabet as A_c and then take the extended intersection between $(A^U)^{c_i>c_j}$ and A^c. For any word $w \in \Sigma^*$ (respectively, $w \in \Sigma^\omega$), the automaton $(A^U)^{c_i>c_j}$ contains all the accepting runs on w that satisfy the condition $c_i > c_j$. Hence, taking the extended intersection between $(A^U)^{c_i>c_j}$ and A_c will remove from A_c all the accepting runs that do not satisfy $c_i > c_j$.

Since there is no bound on the difference between the values of c_i and c_j before a word is accepted, the automaton $(A^U)^{c_i>c_j}$ will have an infinite number of states. Indeed, there should be one state for each possible value of $c_i - c_j$. To avoid to handle infinite-state automata, we impose a synchronization between the counters that need to be compared. As a consequence, we may not exactly compute $(A_c)^{c_i>c_j}$, but an automaton whose language and counter language are subsets of those of $(A_c)^{c_i>c_j}$. As we shall see in the two next sections, imposing this synchronization is sufficient for the applications we will consider. We have the following definition.

Definition 7.13. Let $A_c = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ be a finite-word (respectively, run-bounded weak Büchi) counter automaton and a *synchronization bound* $M \in \mathbb{N}$. Let $\Delta c_l(\sigma)$ denotes the difference between the value associated to the counter c_l in the last and in the first state of the subrun σ of a run ρ on w. The automaton A_c is *M-synchronized* with respect to the counters c_i and c_j if $L(A_c) = L(A_c)^{c_i>c_j}$, and for each $w \in L(A_c)$ and each accepting run ρ on w, we have $\|\Delta c_j(\sigma) - \Delta c_i(\sigma)\| \leq M$.

Definition 7.14. The finite-word (respectively, run-bounded weak) counter automaton $A^{MU} = (n, \mathbf{c}, Q, \Sigma, Q_0, \triangle, F)$ is $M-$Universal-synchronized w.r.t. counters c_i and c_j if and only if it is $M-$synchronized w.r.t. c_i and c_j, and $L(A^{MU}) = \Sigma^*$ (respectively, $L(A^{MU}) = \Sigma^\omega$).

Rather than computing $(A_c)^{c_i>c_j}$, we propose to compute a M-synchronized automaton whose language and counter language are subsets of those of $(A_c)^{c_i>c_j}$. For this, we intersect A_c with a *M-Universal-synchronized automaton*. Observe that we can have a possibly infinite number of automata which are $M-$Universal-synchronized w.r.t. c_i and c_j. When taking the extended intersection between a counter automaton A_c and a M-Universal-synchronized automaton A^{MU} defined over the same extended alphabet, we obtain an automaton which is M-synchronized and whose language and counter language are subsets of those of A_c. The requirement $L(A^{MU}) = \Sigma^*$ (respectively, $L(A^{MU}) = \Sigma^\omega$) in Definition 7.14 is to make sure that accepting runs are removed from A_c only if they do not satisfy the constraints over c_i and c_j.

7.2 Transitive Closure of a Transducer

Consider a reflexive deterministic finite-word (respectively, deterministic weak Büchi) transducer T and let T^{e_0} be the last element of an incrementally growing sampling sequence S_I of powers of T. Assume that T^{e_0} is the origin of an extrapolated sequence $T^{e_0}, T^{e_1}, \ldots$. The limit of this sequence is the transducer T^{e_*} with $L(T^{e_*}) = \bigcup_{i=0}^{\infty} L(T^{e_i})$ that has been computed by applying the construction of Proposition 6.10 (respectively, Proposition 6.16) to T^{e_0}. We provide sufficient criteria to test whether $L(T^*) = L(T^{e_*})$.

We first determine whether T^{e_*} is a safe extrapolation of T, i.e., whether $L(T^*) \subseteq L(T^{e_*})$. For this, we propose the following result.

Proposition 7.15. *Let T_1 and T_2 be two reflexive transducers defined over the same alphabet. If $L(T_2 \circ T_2) \subseteq L(T_2)$ and $L(T_1) \subseteq L(T_2)$, then $L(T_1^*) \subseteq L(T_2)$.*

Proof. We show by induction that for each $i > 0$, $L(T_1^i) \subseteq L(T_2)$. The base cases, i.e., $L(T_1^0) \subseteq L(T_2)$ and $L(T_1) \subseteq L(T_2)$, hold by hypothesis. Suppose now that $i > 1$ and that the result holds for any $k < i$. It is easy to see that $L(T_1^i) \subseteq L(T_2)$. Indeed, $L(T_1^i) = L(T_1^{i-1} \circ T_1) \subseteq L(T_2 \circ T_1) \subseteq L(T_2 \circ T_2) \subseteq L(T_2)$. The first inclusion holds by induction, the second because $L(T_1) \subseteq L(T_2)$, and the third is by hypothesis. □

By construction, $L(T) \subseteq L(T^{e_*})$ and, moreover, T is reflexive. Consequently, Proposition 7.15 states that if $L(T^{e_*} \circ T^{e_*}) \subseteq L(T^{e_*})$, then T^{e_*} is a safe extrapolation of T^*. This criterion is only sufficient since their could exist two words $w, w' \in L(T^{e_*})$ such that $w, w' \notin L(T^*)$ and $w \circ w' \notin L(T^{e_*})$. In practice, checking the condition expressed by Proposition 7.15 requires to complement T^{e_*}. Indeed, this condition is equivalent to checking whether the language accepted by the automaton which is the intersection of the automaton for $T^{e_*} \circ T^{e_*}$ and the one for the complement of T^{e_*} is empty or not. When working with weak automata, T^{e_*} is by construction weak but generally not deterministic (see Proposition 6.18). Our approach consists in determinizing T^{e_*}, and then checking whether the resulting transducer is inherently weak. In the positive case, this transducer can be turned into a weak deterministic one and easily be complemented by inverting the sets of accepting and nonaccepting states. Otherwise a Büchi complementation algorithm has to be applied.

We now turn to determine whether T^{e_*} is a precise extrapolation of T, i.e., whether $L(T^{e_*}) \subseteq L(T^*)$. For this, we again provide a partial solution in the form of a sufficient criterion. The "preciseness" 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^j of the transducer T. 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 T^{e_j} with $j < i$, then it also holds for T^{e_i}, we are done. For this, we propose the following theorem.

Theorem 7.16. *Let T and T^{e_*} be two transducers and T^{e_0} be a power of T. Assume an infinite sequence of transducers $T^{e_0}, T^{e_1}, \ldots$, and let $L(T^{e_*}) = \bigcup_{i=0}^{\infty} L(T^{e_i})$.*

If

$$\forall w, \forall i > 0 \; [w \in L(T^{e_i}) \setminus L(T^{e_0}) \Rightarrow \exists 0 \leq j, j' < i, w \in L(T^{e_j} \circ T^{e'_j})], \qquad (1)$$

then $L(T^{e_*}) \subseteq L(T^*)$.

Proof. The proof is by induction: we show that for each $i \geq 0$, $L(T^{e_i}) \subseteq L(T^*)$. The base case, i.e., $L(T^{e_0}) \subseteq L(T^*)$, holds by hypothesis. Suppose now that $i > 0$ and that the result holds for any $j < i$. We show that $L(T^{e_i}) \subseteq L(T^*)$. Consider a word $w \in L(T^{e_i})$. If $w \in L(T^{e_0})$, then the result holds. If $w \notin L(T^{e_0})$ then, by Condition (1) there exist $j, j' < i$, $w' \in L(T^{e_j})$, $w'' \in L(T^{e_{j'}})$ such that $w = w' \circ w''$. Since, by inductive hypothesis $w', w'' \in L(T^*)$, $n_1, n_2 \in \mathbb{N}$ exist such that $w' \in L(T^{n_1})$ and $w'' \in L(T^{n_2})$. We thus have $w \in L(T^{n_1+n_2})$. □

Theorem 7.16 reduces the problem of checking the preciseness of T^{e_*} to the one of testing whether Condition (1) is satisfied or not. We now go one step further and reduce this test to automata-based manipulations.

Lemma 7.17. *Let T^{e_0} be the last element of an incrementally growing sampling sequence S_I of transducers, and $T_c^{e_0}$ be the counter-zero automaton corresponding to T^{e_0}. Assume that T^{e_0} is the origin of an extrapolated sequence $T^{e_0}, T^{e_1}, \ldots$ and let $T_{c_1}^{e_*}, T_{c_2}^{e_*}, T_{c_3}^{e_*}$ be three copies of the counter transducer $T_c^{e_*}$ which is obtained by applying the construction of Proposition 6.13 (respectively, Proposition 6.19) to T^{e_0}. If*

$$\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T_{c_1}^{e_*} \cap_c (T_{c_2}^{e_*} \circ_c T_{c_3}^{e_*}))^{c_1 > \{c_2,c_3\}})]) = \mathcal{L}(T_c^{e_*}) \setminus \mathcal{L}(T_c^{e_0}), \qquad (2)$$

then

$$\forall w, \forall i > 0 \; [w \in L(T^{e_i}) \setminus L(T^{e_0}) \Rightarrow \exists 0 \leq j, j' < i, w \in L(T^{e_j} \circ T^{e'_j})].$$

Proof. Observe that the counter language of $\pi_{(\neq\{c_2,c_3\})}[(T_{c_1}^{e_*} \cap_c (T_{c_2}^{e_*} \circ_c T_{c_3}^{e_*}))^{c_1 > \{c_2,c_3\}}$ is the counter language of $T_c^{e_*}$ from where one has removed all the pairs (w, i) for which there is not $(w', j < i), (w'', j' < i) \in \mathcal{L}(T_c^{e_*})$ with $w = w' \circ w''$. For each i and each word w, if $w \in L(T^{e_i}) \setminus L(T^{e_0})$ then, by Proposition 6.13 (respectively, Proposition 6.19), there exists $k > 0 \in \mathbb{N}$ such that $(w, k \leq i) \in \mathcal{L}(T_c^{e_*}) \setminus \mathcal{L}(T_c^{e_0})$. Since Condition (2) holds, there exist $j, j' \in \mathbb{N}$ with $j, j' < k \leq i$ and two words w', w'' such that $(w', j) \in \mathcal{L}(T_c^{e_*})$ and $(w'', j') \in \mathcal{L}(T_c^{e_*})$, with $w = w' \circ w''$. By Proposition 6.13 (respectively, Proposition 6.19), $w' \in L(T^{e_j})$ and $w'' \in L(T^{e_{j'}})$ and $w \in L(T^{e_j} \circ T^{e_{j'}})$. □

We can now state our main result.

Theorem 7.18. *Let T be a transducer, T^{e_0} the last element of an incrementally growing sampling sequence S_I of powers of T, and $T_c^{e_0}$ the counter-zero automaton corresponding to T^{e_0}. Assume that T^{e_0} is the origin of an extrapolated sequence $T^{e_0}, T^{e_1}, \ldots$ and let T^{e_*} be the transducer that has been obtained by applying the construction of proposition 6.10 (respectively, Proposition 6.16) to T^{e_0}. Let $T_{c_1}^{e_*}, T_{c_2}^{e_*}, T_{c_3}^{e_*}$ be three copies of the counter transducer $T_c^{e_*}$ which is obtained by applying the construction of Proposition 6.13 (respectively, Proposition 6.19) to T^{e_0}. If $\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T_{c_1}^{e_*} \cap_c (T_{c_2}^{e_*} \circ_c T_{c_3}^{e_*}))^{c_1 > \{c_2,c_3\}})]) = \mathcal{L}(T_{c_1}^{e_*}) \setminus \mathcal{L}(T_c^{e_0})$, then $L(T^{e_*}) \subseteq L(T^*)$.*

Proof. By Proposition 6.10 (respectively, Proposition 6.16), we have $L(T^{e_*}) = \bigcup_{i=0}^{\infty} L(T^{e_i})$.

According to Lemma 7.17, since

$$\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}})]) = \mathcal{L}(T^{e_*}_c) \setminus \mathcal{L}(T^{e_0}_c),$$

we have

$$\forall w, \forall i > 0\ [w \in L(T^{e_i}) \setminus L(T^{e_0}) \Rightarrow \exists 0 \leq j, j' < i, w \in L(T^{e_j} \circ T^{e'_j})].$$

It follows from Theorem 7.16 that $L(T^{e_*}) \subseteq L(T^*)$. □

Condition (2) can be checked as follows :

(1) Observe that, since $\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}})])$ is disjoint from, checking $\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}})]) = \mathcal{L}(T^{e_*}_c) \setminus \mathcal{L}(T^{e_0}_c)$ is equivalent to check $\mathcal{L}(\pi_{(\neq\{c_2,c_3\})}[(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}})] \cup_e T^{e_0}_c) = \mathcal{L}(T^{e_*}_c)$, which avoid to compute $\mathcal{L}(T^{e_*}_c) \setminus \mathcal{L}(T^{e_0}_c)$. Computing $\mathcal{L}(T^{e_*}_c) \setminus \mathcal{L}(T^{e_0}_c)$ is a hard problem, which requires the ability to distinguish between accepting and nonaccepting runs that assign the same counter valuation to a given word.

(2) There are algorithms to compute $\cap_c$, $\circ_c$, and $\pi_{(\neq\{c_2,c_3\})}$. Those algorithms directly follow from the definitions given in Section 7.1. Observe that if T is weak, then the counter automaton for $T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3})$ is run-bounded weak.

(3) We do not compute the one-counter automaton for $(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}}$, but a M-synchronized counter automaton whose language and counter languages may be subsets of those of $(T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3}))^{c_1 > \{c_2,c_3\}}$. We follow the methodology described in Section 7.1, and compute the extended-intersection between the automaton $T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3})$ and two finite-word (respectively, run-bounded weak Büchi) M-Universal-synchronized counter automata, one which is synchronized w.r.t. counters c_1 and c_2, and the other one w.r.t. counters c_1 and c_3. Assume that Σ^2 is the alphabet of T and d is the maximal increment value of $T^{e_*}_c$. The extended alphabet of T_c is $\Sigma^2 \times [0,d]$, and the one of $T^{e_*}_{c_1} \cap_c (T^{e_*}_{c_2} \circ_c T^{e_*}_{c_3})$ is thus $\Sigma^2 \times [0,d]^3$ (see constructions for $\circ_c$ and $\cap_c$).

(4) We reduce the problem of checking the equivalence between the counter languages of the two members of the equality to the one of checking the equivalence between the languages of their extended automata.

Observe that, if $L(T^*) = L(T^{e_*})$, then the transducers T^{e_i} ($i \geq 0$) should constitute new elements in an extension of the sampling sequence S_i, i.e., if $S_I = T^{s_0}, T^{s_1}, \ldots, T^{s_k}$ with $T^{s_k} = T^{e_0}$, then the extension is $T^{s_0}, T^{s_1}, \ldots, T^{s_k}, T^{s_{k+1}}, T^{s_{k+2}}, \ldots$, with $T^{s_{k+i}} = T^{e_i}$ for each $i \geq 0$. Condition (1) is thus particularly designed to hold for sampling sequences where each transducer can be obtained by a single composition of transducers that appear before in the sequence. Indeed, the condition can be read as follows: *each transducer T^{e_i} in the extended sampling sequence is the composition of two transducers T^{e_j} and $T^{e_{j'}}$ that appear before in this sequence.* If more than one composition is needed, then the condition may not be satisfied even if $L(T^{e_*}) = L(T^*)$. Condition (1) can be adapted to work with other sampling sequences. This is illustrated with the following example.

Example 7.19. *If each transducer in the sampling sequence is obtained by composing n transducers that appear before in the sequence, then one can test whether the following condition holds*

$$\forall w, \forall i > 0 \ [w \in L(T^{e_i}) \backslash L(T^{e_0}) \Rightarrow \exists 0 \leq j_1, \ldots, j_n < i, w \in L(T^{e_{j_1}} \circ \ldots \circ T^{e_{j_n}})], \quad (3)$$

rather than to test whether Condition (1) holds.

Theorem 7.18 easily extends to other sampling sequences.

7.3 Limit of a Sequence of Reachable Sets

This section lifts the results obtained in the previous section to the case where one computes the limit of a sequence of reachable states. We consider a reflexive finite-word (respectively, deterministic weak Büchi) transducer T and a deterministic finite-word (respectively, deterministic weak Büchi) automaton A. Let A^{e_0} be the last automaton of an incrementally growing sampling sequence S_I of A, $T^1(A)$, $T^2(A)$, $T^3(A)$, and assume that A^{e_0} is the origin of an extrapolated sequence $A^{e_0}, A^{e_1}, \ldots$. The limit of this sequence is the automaton A^{e_*} with $L(A^{e_*}) = \bigcup_{i=0}^{\infty} L(A^{e_i})$ that has been computed by applying the construction of Proposition 6.10 (respectively, Proposition 6.16) to A^{e_0}. We provide sufficient criteria to test whether $L(T^*(A)) = L(A^{e_*})$.

We first determine whether A^{e_*} is a safe extrapolation of $T^*(A)$, i.e., whether $L(T^*(A)) \subseteq L(A^{e_*})$. For this, we propose the following result.

Proposition 7.20. *Let A_1 and A_2 be two automata defined over the same alphabet Σ and with $L(A_1) \subseteq L(A_2)$. Let T be a reflexive transducer over Σ^2. If $L(T(A_2)) \subseteq L(A_2)$ then $L(T^*(A_1)) \subseteq L(A_2)$.*

Proof. By hypothesis, we have $L(A_1) \subseteq L(A_2)$. We show by induction that for each $i > 0$, $L(T^i(A_1)) \subseteq L(A_2)$. The base cases, i.e., $L(A_1) \subseteq L(A_2)$ and $L(T(A_1)) \subseteq L(A_2)$, hold by hypothesis. Suppose now that $i > 1$ and that the result holds for any $j < i$. It is easy to see that $L(T^i(A_1)) \subseteq L(A_2)$. Indeed, $L(T^i(A_1)) = L(T(T^{i-1}(A_1))) \subseteq L(T(A_2)) \subseteq L(A_2)$. The first inclusion holds by induction and the second because $L(T(A_2)) \subseteq L(A_2)$. □

Proposition 7.20 states that checking whether A^{e_*} is a safe extrapolation of $\bigcup_{i=0}^{\infty} T^i(A)$ can be done by checking whether $L(T(A^{e_*})) \subseteq L(A^{e_*})$. It is worth mentioning that this criterion is only sufficient. Indeed, their could exist a word $w \in L(A^{e_*})$ such that $w \notin L(T^*(A))$ and $w \notin L(T(A^{e_*}))$.

We now turn to determine whether A^{e_*} is a precise extrapolation of $T^*(A)$, i.e., whether $L(A^{e_*}) \subseteq L(T^*(A))$. As in Section 7.2, we use an inductive argument, which is formalized with the following theorem.

Theorem 7.21. *Let T be a transducer and A, A^{e_*} be two automata. Let $A^{e_0} = T^k(A)$, and consider an infinite sequence of automata $A^{e_0}, A^{e_1}, \ldots$, with $L(A^{e_*}) = \bigcup_{i=0}^{\infty} L(A^i)$. If*

$$\forall w, \forall i > 0 \ [w \in L(A^{e_i}) \setminus L(A^{e_0}) \Rightarrow \exists 0 \leq j < i, w \in L(T(A^{e_j}))], \quad (4)$$

then $L(A^{e_*})) \subseteq L(T^*(A)$.

Proof. The proof is by induction: we show that for each $i \geq 0$, $L(A^{e_i}) \subseteq L(T^*(A))$. The base case, i.e., $L(A^{e_0}) \subseteq L(T^*(A))$, holds by hypothesis. Suppose now that $i > 0$ and that the result holds for any $j < i$. We show that $L(A^{e_i}) \subseteq L(T^*)$. Consider a word $w \in L(A^{e_i})$. If $w \in L(A^{e_0})$, then the result holds. Assume now that $w \notin L(A^{e_0})$. By Condition (4), there exists $j < i$ such that $w \in L(T(A^{e_j}))$. Since, T is reflexive and by inductive hypothesis, there exists n such that $L(A^{e_j}) \subseteq L(T^n(A))$. We thus have $w \in L(T^{n+1}(A))$. $\square$

We now go one step further and reduce the verification of Condition (4) to simple automata-based manipulations.

Lemma 7.22. *Let T be a reflexive transducer and A be an automaton. Let A^{e_0} be the last automaton of an incrementally growing sampling sequence S_I of A, $T^1(A)$, $T^2(A)$, $T^3(A)$, and assume that A^{e_0} is the origin of an extrapolated sequence $A^{e_0}, A^{e_1}, \ldots$ and let $A^{e_*}_{c_1}, A^{e_*}_{c_2}$ be two copies of the counter automaton $A^{e_*}_c$ that is obtained by applying the construction of Proposition 6.13 (respectively, Proposition 6.19) to $(A^{e_0}, GROW_{(S_I)}(A^{e_0}))$. Let $A^{e_0}_c$ be the counter-zero automaton corresponding to A^{e_0}. If*

$$\mathcal{L}(\pi_{(\neq c_2)}[(A^{e_*}_{c_1} \cap_c T(A^{e_*}_{c_2}))^{c_1 > c_2})]) = \mathcal{L}(A^{e_*}_c) \setminus \mathcal{L}(A^{e_0}_c), \qquad (5)$$

then

$$\forall w, \forall i > 0\ [w \in L(A^{e_i}) \setminus L(A^{e_0}) \Rightarrow \exists 0 \leq j < i, w \in L(T(A^{e_j}))].$$

Proof. Observe that the counter language of $\pi_{(\neq c_2)}[(A^{e_*}_{c_1} \cap_c T(A^{e_*}_{c_2}))^{c_1 > c_2}$ is the counter language of $A^{e_*}_c$ from where one has removed all the pairs (w, i) for which there is no pair $(w', j < i) \in \mathcal{L}(A^{e_*}_c)$ with $w \in L(T(A_{w'}))$ (where $A_{w'}$ is an automaton whose language is $\{w'\}$) have been removed. For each i and each word w, if $w \in L(A^{e_i}) \setminus L(A^{e_0})$ then, by Proposition 6.13 (respectively, Proposition 6.19), there exists $k > 0 \in \mathbb{N}$ such that $(w, k \leq i) \in \mathcal{L}(A^{e_*}_c)$. Since Condition (5) holds, there exists $j \in \mathbb{N}$ with $j < k \leq i \in \mathbb{N}$ and a word w' such that $(w', j) \in \mathcal{L}(T^{e_*}_c)$ with $w = L(T(A_{w'}))$. By Proposition 6.13 (respectively, Proposition 6.19), $w' \in L(A^{e_j})$ and $w \in L(T(A^{e_j}))$. $\square$

Finally, we obtain our main result.

Theorem 7.23. *Let T be a reflexive transducer and A be an automaton. Let A^{e_0} be the last automaton of an incrementally growing sampling sequence S_I of A, $T^1(A)$, $T^2(A)$, $T^3(A)$, and assume that A^{e_0} is the origin of an extrapolated sequence $A^{e_0}, A^{e_1}, \ldots$ Let A^{e_*} be the automaton that has been obtained by applying the construction of Proposition 6.10 (respectively, Proposition 6.16) to A^{e_0}, and let $A^{e_*}_{c_1}, A^{e_*}_{c_2}$ be two copies of the counter automaton $A^{e_*}_c$ that is obtained by applying the construction of Proposition 6.13 (respectively, Proposition 6.19) to A^{e_0}. Let $A^{e_0}_c$ be the counter-zero automaton corresponding to A^{e_0}. If*

$$\mathcal{L}(\pi_{(\neq c_2)}[(A^{e_*}_{c_1} \cap_c T(A^{e_*}_{c_2}))^{c_1 > c_2})]) = \mathcal{L}(A^{e_*}_c) \setminus \mathcal{L}(A^{e_0}_c),$$

then $L(A^{e_*}) \subseteq L(T^*(A))$.

PROOF. By Proposition 6.10 (respectively, Proposition 6.16), we have $L(A^{e_*}) = \bigcup_{i=0}^{\infty} L(A^{e_i})$.

According to Lemma 7.22, since

$$\mathcal{L}(\pi_{(\neq c_2)}[(A_{c_1}^{e_*} \cap_c T(A_{c_2}^{e_*}))^{c_1 > c_2})]) = \mathcal{L}(A_c^{e_*}),$$

we have

$$\forall w, \forall i > 0\ [w \in L(A^{e_i}) \setminus L(A^{e_0}) \Rightarrow \exists 0 \leq j < i, w \in L(T(A^{e_j}))].$$

It follows from Theorem 7.22 that $L(A^{e_*}) \subseteq L(T^*(A))$. □

Theorem 7.23 states a sufficient criterion to check whether A^{e_*} is a precise extrapolation of $T^*(A)$. This criterion amounts to test whether Condition (4) holds. For this last item, we can proceed like for Condition (2).

Observe that, if $L(T^*(A)) = L(A^{e_*})$, then the automata A^{e_i} ($i \geq 0$) may constitute new elements in an extension of the sampling sequence S_I, i.e., if $S_I = A^{s_0}, A^{s_1}, \ldots, A^{s_k}$ with $A^{s_k} = A^{e_0}$, then the extension is $A^{s_0}, A^{s_1}, \ldots, A^{s_k}, A^{s_{k+1}}, A_{s_{k+2}}, \ldots$, with $A^{s_{k+i}} = A^{e_i}$ for each $i \geq 0$. Condition (4) is thus particularly designed to hold for sampling sequences where each element can be obtained from the previous one by a single application of the transducer T. Indeed, the condition can be read as follows: *each automaton A^{e_i} in the extended sampling sequence can be obtained by applying T to an element that appears before in the sequence.* If more applications of T are needed, then we may have to adapt the condition. This is illustrated with the following example.

EXAMPLE 7.24. *If each element in the sampling sequence is obtained by applying the transducer T $k > 1$ times to the previous element in the sequence, then one can test whether the following condition holds*

$$\forall w, \forall i > 0\ [w \in L(A^{e_i}) \setminus L(A^{e_0}) \Rightarrow \exists 0 \leq j < i, w \in L(T^k(A^{e_j}))]. \tag{6}$$

rather than to check Condition (4).

This observation states for sampling sequences where the number of applications of T needed to build each element from the previous one is constant. In [Legay 2007], we proposed another approach that consists in associating to each state of the system an integer variable that counts the number of applications of the reachability relation needed to reach this state from the initial set of states. Using this "counter variable", we can propose a preciseness criterion whose induction is based on the number of applications of the reachability relation rather than on the position in the sampling sequence. Contrary to the techniques presented in this section, the counters are no longer introduced during the extrapolation process, but are present in all the steps of the computation. This is a "key point" to ensure the preciseness when considering a nonlinear sampling sequence, but this clearly influence the extrapolation process and the increments detection. As observed in [Legay 2007], this approach is of particular interest when dealing with systems that manipulate integer/real variables. However, the solution in [Legay 2007] is not a panacea. Indeed, as an example, it is known that the transitive closure of the relation $\{(x, 2x)\}$ in basis 2 is regular, but the transitive closure of the relation $\{((x, y), (2x, y + 1))\}$ is not regular.

8. SUMMARY OF THE EXPERIMENTS

The results presents in this paper have been implemented in a tool called T(O)RMC, which build on the LASH toolset [LASH]. The T(O)RMC toolset has been evaluated over more than 100 case studies. This section only briefly recaps the classes of problems for which T(O)RMC has been used so far. More details on the experiments can be found in the electronic appendix.

We first used T(O)RMC to compute an automata-based representation of the set of reachable states of several infinite-states systems, including parametric systems, FIFO-queue systems, and systems manipulating integer variables. Others experiments concerned the computation of the transitive closure of several arithmetic relations. It is worth mentioning that the disjunctive nature of some relations sometimes prevents the direct use of specific domain-based techniques [Finkel and Leroux 2002; Boigelot and Herbreteau 2006]. We also applied T(O)RMC to the challenging problem of analyzing linear hybrid systems. One of the case studies consisted of computing a precise representation of the set of reachable states of several versions of the *leaking gas burner*. To the best of our knowledge, only the technique in [Boigelot and Herbreteau 2006] was able to handle the cases we considered. Among the other experiments, we should also mention the computation of the set of reachable states of an augmented version of the IEEE Root Contention Protocol [Legay 2007], which has been point out to be a hard problem [Simons and Stoelinga 2001]. The ability of T(O)RMC to compute the limit of an infinite sequence of automata has other applications. As an example, the tool has been used in a semi-algorithm to compute the convex hull of a set of integer vectors [Cantin et al. 2008]. T(O)RMC was also used to compute a symbolic simulation over the state-space of an infinite-state system, with the aim of verifying temporal properties [Bouajjani et al. 2004].

9. A BRIEF COMPARISON WITH OTHER GENERIC TECHNIQUES

The Regular Model Checking framework has first been proposed in [Kesten et al. 1997] as a uniform paradigm for algorithmic verification of parametric systems. The contributions in [Kesten et al. 1997] are an automata-based representation of parametric systems and an algorithm to compute the transitive closure of the finite-word transducer representing the reachability relation of such systems. One major difference with our work is thus that the construction in [Kesten et al. 1997] can only be applied to a very specific class of finite-word transducers.

In [Bouajjani et al. 2000; Abdulla et al. 2003], Nilsson et al. also proposed to use forward and backward equivalences to compute the transitive closure of a transducer. Starting from a transducer T, they derive a transducer, called the *history transducer* whose states are *columns* (words) of states of T. The history transducer is an infinite representation of the transitive closure of T. Since the set of states of this transducer is infinite, it is inappropriate for computational purposes. To overcome this difficulty, the authors propose to compute a finite-state transducer, which is an abstraction of the history transducer. The abstraction transducer is generated on-the-fly by a procedure which starts from the original transducer T, and then incrementally adds new transitions and merges equivalent states. To compute the abstract transducer, the authors approximate the equivalence relation on states of the history transducer with the help of forward and backward relations computed on the successive powers of the transducer. The construction exploits the rewritting properties of transducer composition and cannot be applied to the more general problem of extrapolating a sequence of automata. The results of [Bouajjani et al. 2000; Abdulla et al. 2003] have been implemented in a tool called the RMC toolset (states for Tool for Regular Model Checking) [RMC], and tested on several parametric and queue systems for which good results have been obtained [Nilsson 2005]. Unfortunately, it seems that the relations used to merge the states of the successive unions have been designed to handle parametric and queue systems only. To the best of our knowledge, the RMC toolset cannot be used with other classes of systems such as linear integer systems. In our work, foward and backward relations are used to discover regularities that appear and that are repeated between successive automata in a sequence. Our approach, which can handle integer systems, is more general as it can extrapolate a sequence of automata (not only a sequence of transducers), which is of particular interest when considering other applications than model checking, e.g., convex hull computation.

In [Dams et al. 2002], Dams, Lakhnech, and Steffen proposed a non-implemented simulation-based technique to compute T^+. This technique is similar to those proposed in [Bouajjani et al. 2000; Abdulla et al. 2003], but the definition of the equivalence relation differs. The framework in [Abdulla et al. 2003] is more general than the one in [Dams et al. 2002] (see the introduction in [Abdulla et al. 2003] for an argument).

In [Touili 2001; 2003], Touili proposed another extrapolation-based technique to solve the Regular Reachability Problems. The results presented in this paper share some notions with those in [Touili 2001; 2003]. Indeed, the core idea in the work of Touili is to compute an extrapolation of a finite-word transducer by comparing a finite prefix of its successive powers, trying to detect increments between them. One major drawback of Touili's work, which is not implemented, is that no efficient method is provided to detect the increments. Also, there is no way to check whether the same increment is continuously repeated. This is crucial as several increments may be needed to compute a safe extrapolation.

In [Vardhan et al. 2004; Vardhan 2006], Vardhan et al. apply machine learning techniques from [Angluin 1987; Rivest and Shapire 1993] to learn a finite-word automaton that represents the set of reachable states of a regular system. The results in [Vardhan et al. 2004; Vardhan 2006] have been implemented in a tool called LEVER [Vardhan and Viswanathan 2006], which has been applied to FIFO-queue and linear integer systems. The work does not consider the infinite-word case and is thus less general than our approach. We also mention that in [Habermehl and Vojnar 2004], Habermehl et al. also proposed to use a learning-based approach to compute the set of reachable states of several parametric systems.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tocl/20YY-V-N/p1-URLaxel`.

Thanks

We thank Bernard Boigelot for a fruitful collaboration on preliminary versions of this work. We are grateful to Frederic Herbreteau who helped us with HAT. We also thank Marcus Nilsson, Julien d'Orso, Parosh Abdulla, Sbastien Jodogne, Elad Shahar, Martin Steffen, Tayssir Touili, and Mahesh Viswanathan for answering many questions regarding their works and case studies.

REFERENCES

Abdulla, P. A., Bouajjani, A., Jonsson, B., and Nilsson, M. 1999. Handling global conditions in parameterized system verification. In *Proc. 11th CAV*. LNCS, vol. 1633. Springer, 134–145.

Abdulla, P. A., Cerans, K., Jonsson, B., and Tsay, Y. 2000. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation 160,* 1-2, 109–127.

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

Abdulla, P. A., Jonsson, B., Mahata, P., and d'Orso, J. 2002. Regular tree model checking. In *Proc. 14th CAV*. LNCS, vol. 2404. Springer, 555–568.

Abdulla, P. A., Jonsson, B., Nilsson, M., and d'Orso, J. 2003. Algorithmic improvements in regular model checking. In *Proc. 15th CAV*. LNCS, vol. 2725. Springer, 236–248.

Abdulla, P. A., Jonsson, B., Nilsson, M., d'Orso, J., and Saksena, M. 2004. Regular model checking for ltl(mso). In *Proc. 16th CAV*. LNCS, vol. 3114. Springer, 348–360.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138,* 1, 3–34.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation 75,* 2, 87–106.

Apt, K. R. and Kozen, D. 1986. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letter 22,* 6, 307–309.

Bardin, S., Finkel, A., and Leroux, J. 2004. Faster acceleration of counter automata in practice. In *Proc. 10th TACAS*. LNCS, vol. 2988. Springer, 576–590.

Bardin, S., Finkel, A., Leroux, J., and Schnoebelen, P. 2005. Flat acceleration in symbolic model checking. In *Proc. 3th ATVA*. LNCS, vol. 3707. Springer, 474–488.

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

Boigelot, B. and Godefroid, P. 1996. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. 8th CAV*. LNCS, vol. 1102. Springer, 1–12.

Boigelot, B., Godefroid, P., Willems, B., and Wolper, P. 1997. The power of qdds. In *Proc. 4th SAS*. LNCS, vol. 1302. Springer, 172–186.

Boigelot, B. and Herbreteau, F. 2006. The power of hybrid acceleration. In *Proc. 18th CAV*. LNCS, vol. 4144. Springer, 438–451.

Boigelot, B., Herbreteau, F., and Jodogne, S. 2003. Hybrid acceleration using real vector automata. In *Proc. 15th CAV*. LNCS, vol. 2725. Springer, 193–205.

Boigelot, B., Jodogne, S., and Wolper, P. 2001. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. IJCAR*. LNCS, vol. 2083. Springer, Siena, Italy, 611–625.

Boigelot, B., Jodogne, S., and Wolper, P. 2005. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic 6,* 3, 614–633.

Boigelot, B., Legay, A., and Wolper, P. 2003. Iterating transducers in the large. In *Proc. 15th CAV*. LNCS. Springer, 223–235.

Boigelot, B., Legay, A., and Wolper, P. 2004. Omega-regular model checking. In *Proc. 10th TACAS*. LNCS, vol. 2988. Springer, 561–575.

Boigelot, B., Rassart, S., and Wolper, P. 1998. On the expressiveness of real and integer arithmetic automata. In *Proc. 25th ICALP*. LNCS, vol. 1443. Springer, 152–163.

Boigelot, B. and Wolper, P. 2002. Representing arithmetic constraints with finite automata: An overview. In *Proc. 18th ICLP*. LNCS, vol. 2401. Springer, 1–19.

Bouajjani, A., Esparza, J., and Maler, O. 1997. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th CONCUR*. LNCS, vol. 1243. Springer, 135–150.

Bouajjani, A. and Habermehl, P. 1997. Symbolic reachability analysis of fifo channel systems with nonregular sets of configurations. In *Proc. 24th ICALP*. LNCS, vol. 1256. Springer, 560–570.

Bouajjani, A., Habermehl, P., Moro, P., and Vojnar, T. 2005. Verifying programs with dynamic 1-selector-linked structures in regular model checking. In *Proc. 11th TACAS*. LNCS, vol. 3440. Springer, 13–29.

Bouajjani, A., Habermehl, P., Rogalewicz, A., and Vojnar, T. 2006. Abstract regular tree model checking of complex dynamic data structures. In *Proc. 13th SAS*. LNCS, vol. 4134. Springer, 52–70.

Bouajjani, A., Habermehl, P., and Vojnar, T. 2004. Abstract regular model checking. In *Proc. 16th CAV*. LNCS, vol. 3114. Springer, 372–386.

Bouajjani, A., Jonsson, B., Nilsson, M., and Touili, T. 2000. Regular model checking. In *Proc. 12th CAV*. LNCS, vol. 1855. Springer, 403–418.

Bouajjani, A., Legay, A., and Wolper, P. 2004. Handling liveness properties in (omega-)regular model checking. In *Proc. 6th INFINITY*. ENTCS, vol. 138(3). Elsevier Science Publishers.

Bouajjani, A. and Touili, T. 2002. Extrapolating tree transformations. In *Proc. 14th CAV*. LNCS, vol. 2404. Springer, 539–554.

Cantin, F., Legay, A., and Wolper, P. 2007. Computing convex hulls by automata iteration. In *Proc. 1th AUTOMATHA*.

Cantin, F., Legay, A., and Wolper, P. 2008. Computing convex hull by automata iteration. In *Proc. 13th CIAA*. LNCS, vol. 5148. Springer, 112–121.

Dams, D., Lakhnech, Y., and Steffen, M. 2002. Iterating transducers. *Journal of Logic and Algebraic Programming (JLAP) 52-53*, 109–127.

Finkel, A. and Leroux, J. 2002. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proc. 22rd FSTTCS*. LNCS, vol. 2556. Springer, 145–156.

Finkel, A., Willems, B., and Wolper, P. 1997. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd INFINITY*. ENTCS, vol. 9. Elsevier Science Publishers.

Fribourg, L. and Olsén, H. 1997. Reachability sets of parameterized rings as regular languages. In *Proc. 2nd INFINITY*. ENTCS, vol. 9. Elsevier Science Publishers.

Geeraerts, G., Raskin, J.-F., and Begin, L. V. 2005. Expand, enlarge and check... made efficient. In *Proc. 17th CAV*. LNCS, vol. 3576. Springer, 394–407.

Habermehl, P. and Vojnar, T. 2004. Regular model checking using inference of regular languages. In *Proc. 6th INFINITY*. ENTCS, vol. 138(3). Elsevier Science Publishers.

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

Ibarra, O. H. 1978. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM 25,* 1, 116–133.

Jonsson, B. and Nilsson, M. 2000. Transitive closures of regular relations for verifying infinite-state systems. In *Proc. 6th TACAS*. LNCS, vol. 1785. Springer, 220–234.

KESTEN, Y., MALER, O., MARCUS, M., PNUELI, A., AND SHAHAR, E. 1997. Symbolic model checking with rich assertional languages. In *Proc. 9th CAV*. LNCS, vol. 1254. Springer, 424–435.

KESTEN, Y., PNUELI, A., SHAHAR, E., AND ZUCK, L. D. 2002. Network invariants in action. In *Proc. 13th CONCUR*. LNCS, vol. 2421. Springer, 101–115.

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

LEGAY, A. 2007. *Generic Techniques for the Verification of Infinite-State Systems*. Collection des publications de la Faculté des Sciences Appliquées de l'Université de Liège, Liège, Belgium.

LEGAY, A. 2008. T(o)rmc: A tool for (omega-)regular model checking. In *Proc. 20th CAV*. LNCS, vol. 5123. Springer, 548–551.

LÖDING, C. 2001. Efficient minimization of deterministic weak $\omega-$automata. *Information Processing Letters 79,* 3, 105–109.

MULLER, D. E., SAOUDI, A., AND SCHUPP, P. E. 1986. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*. Springer, Rennes, 275–283.

NILSSON, M. 2001. Regular model checking. M.S. thesis, Uppsala University.

NILSSON, M. 2005. Regular model checking. Ph.D. thesis, Uppsala University.

PNUELI, A. AND SHAHAR, E. 2000. Liveness and acceleration in parameterized verification. In *Proc. 12th CAV*. LNCS, vol. 1855. Springer, 328–343.

RIVEST, R. L. AND SHAPIRE, R. E. 1993. Inference of finite automata using homing sequences. *Information and Computation*.

RMC. The regular model checking tool (RMC). Available at `http://www.it.uu.se/research/docs/fm/apv/rmc`.

ROOS, R. S. 1988. Deciding equivalence of deterministic one-counter automata in polynomial time with applications to learning. Ph.D. thesis, Smith College.

SAFRA, S. 1992. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proceedings of the 24th ACM Symposium on Theory of Computing*. Victoria.

SIMONS, D. P. L. AND STOELINGA, M. 2001. Mechanical verification of the ieee 1394a root contention protocol using uppaal2k. *International Journal on Software Tools for Technology Transfer (STTT) 3,* 4, 469–485.

TOUILI, T. 2001. Regular model checking using widening techniques. *ENTCS 50,* 4.

TOUILI, T. 2003. Analyse symbolique de systèmes infinis basée sur les automates: Application à la vérification de systèmes paramétrés. Ph.D. thesis, Université de Paris 7.

TOUILI, T. AND D'ORSO, J. 2006. Regular hedge model checking. In *Proc. 4th Int. IFIP Conf. on Theoretical Computer Science (TCS06)*.

VARDHAN, A. 2006. Learning to verify systems. Ph.D. thesis, Univeristy of Illinois.

VARDHAN, A., SEN, K., VISWANATHAN, M., AND AGHA, G. 2004. Actively learning to verify safety for fifo automata. In *Proc. 24th FSTTCS*. LNCS, vol. 3328. Springer, 494–505.

VARDHAN, A., SEN, K., VISWANATHAN, M., AND AGHA, G. 2005. Using language inference to verify omega-regular properties. In *Proc. 11th TACAS*. LNCS, vol. 3440. Springer, 45–60.

VARDHAN, A. AND VISWANATHAN, M. 2006. Lever: A tool for learning based verification. In *Proc. 18th CAV*. LNCS, vol. 4144. Springer, 471–474.

VARDI, M. Y. 2007. From church and prior to psl. Available at `http://www.cs.rice.edu/ vardi/papers/index.html`.

WOLPER, P. AND BOIGELOT, B. 1995. An automata-theoretic approach to presburger arithmetic constraints. In *Proc. 2nd SAS*. LNCS, vol. 983. Springer, 21–32.

WOLPER, P. AND BOIGELOT, B. 1998. Verifying systems with infinite but regular state spaces. In *Proc. 10th CAV*. LNCS, vol. 1427. Springer, 88–97.

WOLPER, P. AND BOIGELOT, B. 2000. On the construction of automata from linear arithmetic constraints. In *Proc. 6th TACAS*. LNCS, vol. 1785. Springer, 1–19.

Received Month Year; revised Month Year; accepted Month Year

This document is the online-only appendix to:

On (Omega-)Regular Model Checking

Axel Legay
Universié de Liège
Institut Montefiore, B28
Liège, Belgium

Pierre Wolper
Universié de Liège
Institut Montefiore, B28
Liège, Belgium

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY, Pages 1–45.

A. IMPLEMENTATION

This section briefly discusses an implementation of our results. The experiments are presented in Section B.

A.1 Heuristics

Implementing the technique presented in this paper requires potentially costly composition and determinization procedures. In [Boigelot et al. 2003; 2004; Legay 2007], we proposed two heuristics that, in some situations, reduced to computation time from days to seconds. Experimental results, which are presented in Chapter 7 of [Legay 2007], show that those heuristics are particularly useful when working with arithmetic systems.

A.2 The T(O)RMC Toolset

The results presented in this paper have been implemented in the *T(O)RMC* (states for *Tool for (ω-)Regular Model Checking*) toolset [Legay 2008], which relies on the *LASH Toolset* [LASH] for automata manipulations.

The LASH toolset is a tool for representing infinite sets and exploring infinite state spaces. It is based on finite-state representations, which rely on finite automata for representing and manipulating infinite sets of values over various data domains. The tool is composed of several C functions grouped into packages. The LASH toolset implements several specific algorithms for solving the (ω-)regular reachability problems of several classes of infinite-state systems, which include FIFO-queue systems [Boigelot and Godefroid 1996; Boigelot et al. 1997], systems with integer variables [?], and linear hybrid systems [Boigelot et al. 2003; Boigelot and Herbreteau 2006].

T(O)RMC extends the LASH toolset with the generic algorithm presented in this paper. Contrary to the specific algorithms of LASH, the algorithm of T(O)RMC is applicable to any system that can be represented in the (ω-)Regular Model Checking framework. This makes it possible to handle classes of infinite-state systems that are beyond the scope of specific algorithms, e.g., parametric systems. T(O)RMC is divided into three packages, which are briefly described hereafter.

(1) *The transducer package* that provides data structures and algorithms to manipulate transducers (composition, image computation, ...). The package also provides several heuristics to improve the efficiency of the operations.
(2) *The extrapolation package* for detecting increments in a sequence of automata, and extrapolating a finite sampling sequence. The tool allows the user to precise (1) which sampling strategy has to be used, and (2) how to build the successive elements in the infinite sequence.
(3) *The correctness package* that provides data structures and algorithms to check the correctness of the extrapolation for several classes of problems. The package also contains all the data structures and algorithms to manipulate counter-word automata.

T(O)RMC can be used to compute an extrapolation of a possibly infinite sequence of automata $S = A^1, A^2, \dots$. For this, the user has to provide the following two functions:

—A function named *SAMPLING* that takes as arguments two integers i and j. Each time T(O)RMC calls the function, it sets i and j to the indexes of two automata A^i and A^j, such that A^j is incrementally larger than A^i. The function returns an automaton A^k which is assumed, by the user, to be the next automaton in a sampling sequence whose two last elements are A^i and A^j.

—A function named *CHECK* that takes as argument an automaton A^{e_*}. If the function returns yes, then T(O)RMC assumes that A^{e_*} is the extrapolation expected by the user. This is this function that implements the checks for safety and preciseness.

To extrapolate the infinite sequence of automata S, T(O)RMC behaves as follows:

(1) T(O)RMC computes finite prefixes of S until it finds two automata A^i and A^j such that A^j is incrementally larger than A^i.

(2) T(O)RMC then tries to compute an incrementally growing sampling sequence S_I, assuming that the two first elements of this sequence are A^i and A^j. The automata are added one by one to the sampling sequence, using the function *SAMPLING*. Each time a new automaton is added, the tool checks whether S_I is still incrementally growing. If no, then T(O)RMC goes back to point (1) and consider a prefix of a longer size. If yes, then T(O)RMC extrapolates S_I and produces an automaton A^{e_*}. This extrapolation is followed by a call to the function *CHECK* on A^{e_*}. If the function returns yes, then the computation

System	$\lvert A_{S_0}\rvert$	$\lvert T\rvert$	$\lvert T^i(A_{S_0})\rvert$	$\lvert T^*(A_{S_0})$	T1(s)	T2(s)
Token Ring	2	7	5	2	0,003	0,007
Bakery	1	6	12	3	0,012	0,03
Szymanski	1	14	250	7	1,311	2,970
Burns	1	4	10	2	0,017	0,078
Dijkstra	2	9	74	2	0,566	0,847

Table I. Results for parametric systems.

terminates, and A^{e_*} is the automaton returned by the tool. If the function returns no, then the tool tries to increase S_I by adding one more automaton.

B. AN OVERVIEW OF THE EXPERIMENTS

The T(O)RMC toolset has been applied to a banch of case studies (see [Legay 2007] for the full description). This section briefly recaps classes of problems for which T(O)RMC has been used so far and give the details of some of the experiments. Other experiments can be found in Chapters 7 and 13 of [Legay 2007]. Our experiments have been conducted with an INTEL Pentium 1,6 Ghz and 512 Mega bytes of memory

B.1 Parametric Systems

Parametric systems, which were informally introduced in Example 3.2 of Section 3.1, are parametric version of mutual exclusion protocols such as bakery algorithm, Dijkstra's algorithm, Szymanski's algorithm and Burn's algorithm (see [**?**] for a description of those algorithms). It is know that states of parametric systems can be represented by finite-word automata (see [Touili 2001] and [Legay 2007] for examples of encodings). We computed a finite-word automaton for the set of reachable states of several parametric Systems, including all those that are considered in [Touili 2001; Dams et al. 2002; Abdulla et al. 2003] (see [Legay 2007] for a description of the encoding in regular model checking). We always obtained a save and precise extrapolation in seconds. In all those experiments, the sampling sequence was linear. Moreover all the experiments were carried out with less that two mega bytes of memory. The later is not surprising since the transducers for the transition relation are relatively small. In Table I, we give the number of states of the minimal and deterministic form of the automaton A_{S_0} representing the initial set of states, of the transducer T representing the reachability relation, of the largest power $T^i(A_{S_0})$ that was constructed, and of the set of reachable states $T^*(A_{S_0})$. We denote by T1(s) the time needed to compute a safe extrapolation of the set of reachable states, and by T2(s) the time needed to compute a safe extrapolation and to assess its preciseness. Let d be the maximal counter increment introduced in the extrapolation process. In order to apply the preciseness criterion (see item 4 after Proposition 2), we have worked with $(2\times d)$-Universal-Synchronized automata whose counters can be incremented from 0 to d.

B.2 Communication protocols

We computed the set of reachable states of several communication protocols (with a FIFO politic) whose unbounded communication channel can be represented by a

System	$\lvert A_{S_0}\rvert$	$\lvert T\rvert$	$\max \lvert T^i(A_{S_0})\rvert$	$\lvert T^*(A_{S_0})$	T1(s)	T2(s)
Alternating bit	4	41	224	21	1,6	29,5
Producer consummer	5	68	223	15	1,5	2,6

Table II. Results for queue systems.

finite-word automaton [Boigelot and Godefroid 1996]. See Table II for some results. The observations we made for communication protocols are identical to those we made for parametric systems. This suggest that those two classes of systems exhibit similar behaviors.

B.3 Relations on Integer and Real Variables

We used T(O)RMC to compute a safe extrapolation of the reflexive transitive closure of transducers representing relations over $\mathbb{Z}^n \times \mathbb{Z}^n$. In all our experiments, relations were encoded in base 2 by finite-word transducers, following the scheme presented in Exemple 2.10. We have been able to compute the transitive closure 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 $\{((x,y),(y+1,x-2))\} \cap \mathbb{N}^2 \times \mathbb{N}^2$ 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 [**?**]. In all these examples, the sampling sequence consists of the powers of 2. In Table III we give the number of states of some transducers (column T) that were iterated, of their closure (column T^*), of the largest power of the transducer that was constructed (column T^i), as well as the time (column T(s)) and the memory consumed (column Mb). Most of the computation time was consumed when computing the successive automata in order to build the incrementally growing sampling sequence[8], but not by the extrapolation technique which lasted in a few seconds.

We tried to establish the preciseness of the extrapolations using the technique presented in Section 7.2, but we were unable to manage the intermediate structures required by the algorithms[9]. We suspect that is related to the need of combining two different encodings for integer variable values: the binary encoding of [Wolper and Boigelot 2000] for the variables of the relation and the encoding of the counters involved in the preciseness check.

We also conducted experiments on relations between real variables. Those relations can be represented by deterministic weak Büchi automata. The observations

[8]For systems defined over $\mathbb{N}$ and $\mathbb{R}$, we observed that while the minimal form of the composition of two transducers is generally proportional to the size of the original transducers, the nondeterministic form is often of quadratic size.

[9]We have been able to establish preciseness by using another preciseness criterion, see Section B.9.

Relation	$\|T\|$	$\|T^i\|$	$\|T^*\|$	T(s)	Mb
$\{(x, x+1)\}$	3	9	3	0,02	0,02
$\{(x, x+73)\}$	14	637	75	5,05	3,5
$\{(x, x+177)\}$	16	1492	179	11,344	8,1
$\{((x,y),(y+1,x-2))\} \cap \mathbb{N}^2 \times \mathbb{N}^2$	45	1104	166	3,496s	1,7
$[\{((x,y),(x+2,y-1))\} \cup$ $\{((x,y),(x-1,y+2))\}] \cap \mathbb{N}^2 \times \mathbb{N}^2$	20	855	51	67,399	47,7
$[\{((x,y),(x+2,y-1))\} \cup \{((x,y),(x-1,y+2))\}$ $\cup \{((x,y),(x+1,y+1))\}] \cap \mathbb{N}^2 \times \mathbb{N}^2$	22	493	34	150,791	25,5
$\{((w,x,y,z),(w+1,x+2,y+3,z+4))\}$	91	1500	251	11,7	16,5

Table III. Examples of transducers and of their iteration.

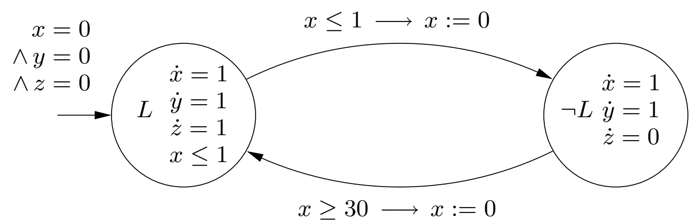


Fig. 17. Hybrid automaton modeling the leaking gas burner.

we made are similar to those on relations between integer variables.

B.4 System with Integer variables

T(O)RMC was also used to compute the set of reachable states of several systems whose set of states are valuation of integer variables (See Section 7.4.1 of [Legay 2007] for various results). The observations we made for those systems are similar to those of the previous section. Since we end up working with a non-linear sampling sequence, we were not able to use the preciseness criterion introduced in this paper. However, we compared our safe extrapolation with the correct answer produced by FAST [**?**], and established that they were equivalent. FASTS generally runs five to ten times faster than T(0)RMC, but it can only handle integer systems, while our approach is general.

B.5 Linear Hybrid systems

We also applied T(O)RMC to the more challenging problem of analyzing linear hybrid Systems. Sets of states and transitions relations of such systems can be represented by deterministic weak Büchi automata that are often called RVAs (see [Boigelot et al. 2001]). We report here the experiments conducted on two case studies that are the *leaking gas burner* described in [Alur et al. 1995] and the *root contention protocol* described in [Simons and Stoelinga 2001].

B.5.1 *The Leaking Gas Burner.* This system consists in a gas burner that leaks during periods of less than one time unit (t.u.), these periods being separated by at least 30 t.u. A linear hybrid automaton modeling the leaking gas burner is given in Figure 17. The states L and $\neg L$ correspond to the leaking and non-leaking situations, respectively, x measures the leaking periods and the interval between them, y is absolute time, and z keeps track of the accumulated leaking duration. We use the following relations:

—$R_1 = \{((x,y,z),(x',y',z')) \mid (\exists k \geq 0)((x,x' \leq 1) \wedge (x' = x+k) \wedge (y' = y+k) \wedge (z' = z+k))\}$ is the time-step transition relation at location L,

—$R_2 = \{((x,y,z),(0,y,z)) \mid x \leq 1\}$ is the discrete-step transition relation whose origin and destination are L and $\neg L$, respectively.

—$R_3 = \{((x,y,z),(x',y',z)) \mid (\exists k \geq 0)((x' = x+k) \wedge (y' = y+k))\}$ is the time-step transition relation at location $\neg L$,

—$R_4 = \{((x,y,z),(0,y,z)) \mid x \geq 30\}$ is the discrete-step transition relation whose origin and destination are $\neg L$ and L, respectively,

—$R'_4 = \{((x,y,z),(0,y,z)) \mid x \geq 3\}$,

—$R = (R_4 \circ R_3 \circ R_2 \circ R_1) \cup R_1$,

—$R' = (R'_4 \circ R_3 \circ R_2 \circ R_1) \cup R_1$.

As a first experiment, we computed an automaton representing a safe extrapolation of the reachable variables values at location L. For this we built a RVA A_{S_0} representing the initial set of states $S_0 \equiv (x = 0) \wedge (y = 0) \wedge (z = 0)$ as well as a minimal weak Büchi transducer T for the relation R. Those two sets were encoded in base 2. We used T(O)RMC to compute a safe extrapolation of $T^*(A_{S_0})$. The sampling sequence used by the tool consists of the powers of 2. The minimal transducer T has 2407 states, the minimal automaton A_{S_0} has 39 states, and the minimal representation of the safe extrapolation 676 states. The minimal form of the largest automaton considered by the increment detection procedure had 1625 states. The computation took 30 minutes and 42 seconds, and the maximal memory used was $22,7$ MB (this value was reached when building R'). We were not able to assess the preciseness of our result using the preciseness criterion presented in this paper. However, we compared our safe extrapolation with the set produced by HAT, and established that they were equivalent. The computation time for this example is quite huge, which is justified by the need to apply 2^7 times a "big" (in term of number of states) transducer to the automaton representing S_0 in order to find an increment that can be extrapolated. The tool HAT, which uses a specific algorithm [Boigelot and Herbreteau 2006], performs about ten times faster than T(O)RMC.

The set of reachable variable values at location L can be characterized by the following set:

$$\begin{gathered} Reach = \{(x,y,z) \in \mathbb{R}^3 \mid (x = z) \wedge (y = z) \wedge (0 \leq x \leq 1)\} \cup \\ \{(x,y,z) \in \mathbb{R}^3 \mid (\exists k \geq 1)((0 \leq x \leq 1) \wedge (-k \leq x - z \leq 0) \wedge (z - y \leq -30k))\}. \end{gathered}$$

It is easy to see that the value -30 has an impact on the size of the automaton representing $Reach$. Indeed, for each value of k, the values of z and y which are solutions of the equation $z - y \leq -30k$ have encodings that differ by a large number of bits. This information has to be stored somewhere in the automaton. Replacing -30 by -3, we obtain a smaller automaton. Taking this observation into account, we computed an automaton representing a safe extrapolation of the reachable variables values at location L, but we used relation R' instead of relation R, which amounts to replace -30 by -3 in the the discrete-step relation (R_4) of the linear hybrid automaton in Figure 17.

The minimal form of the transducer representing R' has 1267 states and the minimal representation of the safe extrapolation has 222 states. The minimal form of the largest automaton considered by the increment detection procedure had 317 states. The computation took $75,759$ seconds, and the maximal memory used was $23,1$ MB (this value was reached when building the transducer for the relation R').

B.5.2 *The Root Contention Protocol.* In this section, we analyze a version of the root contention protocol (RCP). This protocol is part of the IEEE 1394 standard, which defines a communication protocol suited for a high-speed serial bus connecting computer peripherals. The RCP is a leader election protocol, designed for arbitrating conflicts between bus nodes during the identification and organization of the elements connected to the bus [Simons and Stoelinga 2001].

The behavior of RCP relies on timers. When contention occurs between two nodes, each of them chooses a random bit of information, which determines the time it will wait before trying again to become the leader. Several parameters characterize the time constants inherent to the protocol: the physical transmit time *delay*, the minimum and maximum waiting delays *minfast*, *maxfast* for the choice "0", and the minimum and maximum waiting delays *minslow*, *maxslow* for the choice "1". The protocol specifications restrict these parameters to satisfy the following constraints [Simons and Stoelinga 2001]:

$$\begin{gathered}
\mathit{delay} > \mathit{0}, \\
\mathit{minfast} > \mathit{0}, \\
\mathit{maxfast} \geq \mathit{minfast}, \\
\mathit{minslow} > \mathit{maxfast}, \\
\mathit{maxslow} > \mathit{minslow}, \\
2 \times \mathit{delay} \leq \mathit{minfast}, \\
2 \times \mathit{delay} \leq \mathit{minslow} - \mathit{maxfast}.
\end{gathered}$$

It is worth mentioning that this automaton was reduced by using a symmetry relation on pairs of similar states, and by merging together states with the same evolution laws and invariants linked by guardless and transformationless edges (see [?] for details).

In Figure 18, we give the linear hybrid automaton[10] for the RCP that has been proposed in [Simons and Stoelinga 2001]. The initial location is labeled by *start-start*, the initial value of variable x (which count the time in each location) is 0, and the values of the parameters are assumed to satisfy the constraints above. In [?], it is shown that for each location, one can compute a RVA representing the reachable variable values, this without using acceleration techniques.

An extended model of the RCP has been proposed in [**?**], where an extra variable t is added for keeping track of global time. This variable is initialized at 0, evolving

[10]We use the symbolic representation proposed in [Alur et al. 1995].

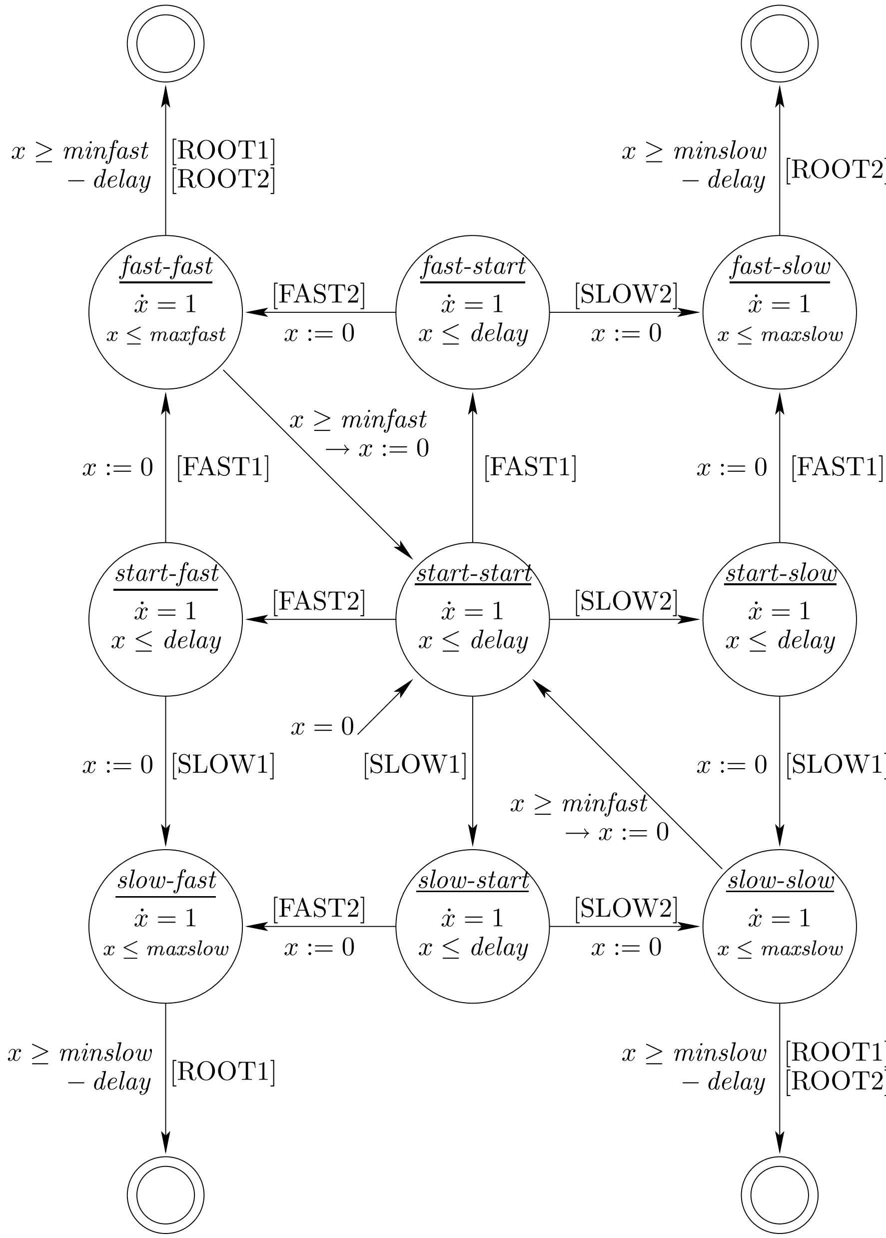


Fig. 18. Hybrid automaton modeling the root contention protocol.

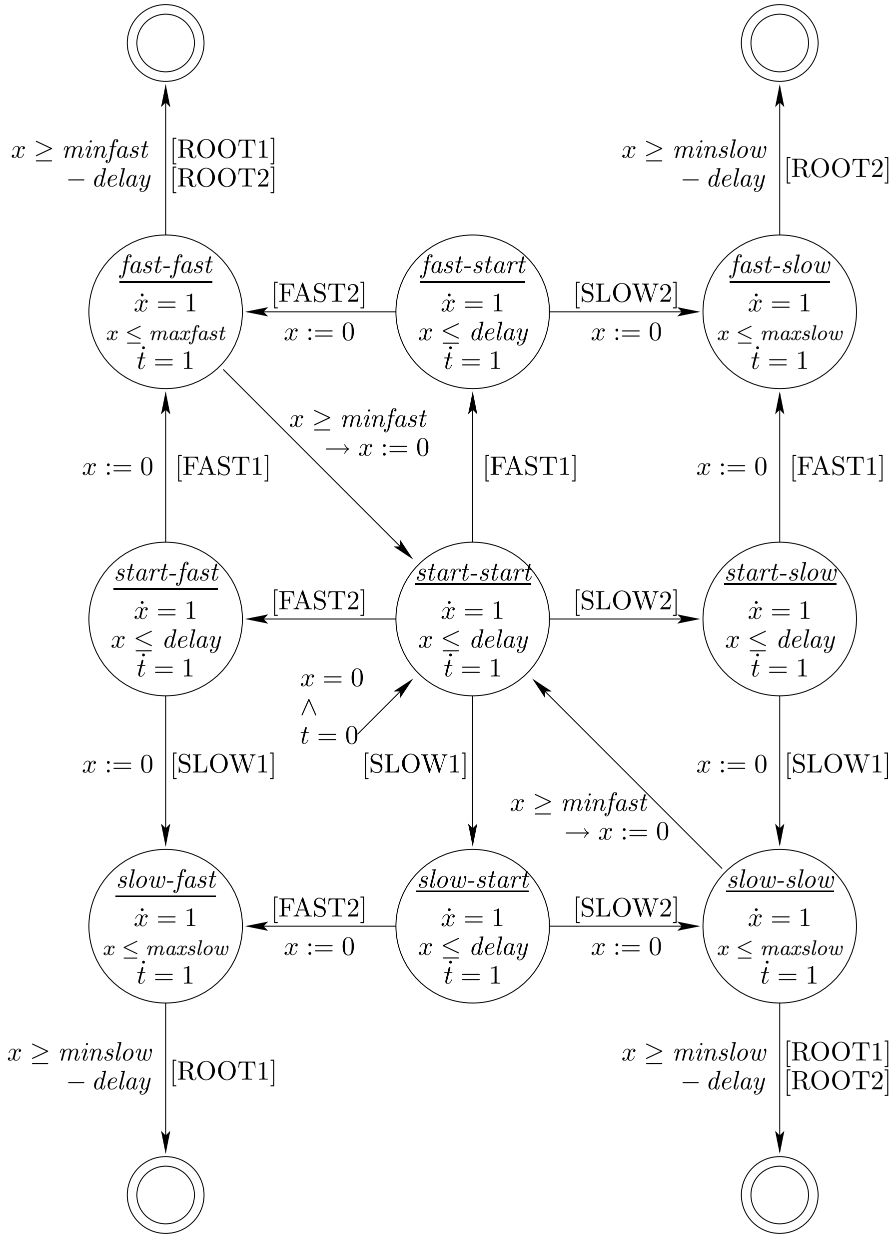


Fig. 19. Hybrid automaton modeling an extended version of the root contention protocol, with *delay = 1*, *minslow = 6*, *maxslow = 7*, *minfast = 3*, and *maxfast = 3*.

at rate 1 in all locations and, unlike variable x, is not being modified by the discrete-step transitions. The linear hybrid automaton A_2 corresponding to this extended model is presented in Figure 19.

In [**?**], it is shown that the set of reachable variable values in any location of A_2 cannot be represented by a RVA. The solution proposed in [**?**] is to manually set the value of the parameters. Using the values *delay* = *1*, *minslow* = *6*, *maxslow* = *7*, *minfast* = *2*, *maxfast* = *3*, we cannot compute the reachable variables values in a finite amount of time. However, those values can be represented by a RVA.

We used T(O)RMC to compute a RVA representing the reachable variable values of x and t (in this order) at location *start-start*. We use the following relations:

—$R_1 = \{((x,t)(x',t')) \mid (\exists k{\geq}0)((x,x'{\leq}\,delay)(x'=x+k)\wedge(t'=t+k))\}$,

—$R_2 = \{((x,t),(0,t))\}$,

—$R_3 = \{((x,t)(x',t')) \mid (\exists k{\geq}0)((x,x'{\leq}\,maxfast)(x'=x+k)\wedge(t'=t+k))\}$,

—$R_4 = \{((x,t)(x',t')) \mid (\exists k{\geq}0)((x,x'{\leq}\,maxslow)(x'=x+k)\wedge(t'=t+k))\}$,

—$R_5 = \{((x,t),(0,t)) \mid x \geq minfast\}$,

—$R = (R_5 \circ R_3 \circ R_2 \circ R_1) \cup (R_5 \circ R_4 \circ R_2 \circ R_1) \cup R_1$.

The relation R above is the union of the relation representing the cycle "*start-start, fast-start,fast-fast, start-start*" with the relation representing the cycle "*start-start, start-slow, slow-slow, start-start*" and the one representing the cycle "*start-start,start-start*".

We computed an automaton representing a safe extrapolation of the reachable variables values at location L. For this we build a RVA A_{S_0} representing the initial state $S_0 = (x = 0 \wedge t = 0)$ as well as a minimal weak Büchi transducer for the relation R. We used an encoding in base 2. The minimal and deterministic form of the transducer representing R has 355 states, the automaton A_{S_0} has 15 states, and the representation of the safe extrapolation 41 states. The minimal form of the largest automaton considered by the increment detection procedure had 114 states. The computation took $9,4$ seconds, and the maximal memory used is $22,7$ MB (this value was reached when building the transducer for the relation R). We were not able to asses the preciseness of our result using the preciseness criterion presented in this paper. However, we compared our safe extrapolation with the set produced by HAT, and established that they were equivalent.

B.6 A note on the incrementally growing sampling sequence

One question that has not been addressed is "in the experiments, what is the size of the incrementally growing sampling sequence for producing a safe extrapolation?"'. From the theory point of view, one can already extrapolate an incrementally growing sampling sequence with 3 automata. However, based on our experiments, it seems that safe extrapolations only appear after having detected two increments that are communication equivalent and stable in one of the automata of the sequence. For the experiments considered in this paper (and in [Legay 2007]), those properties generally appear for incrementally growing sampling sequences with between 3 to 5 automata.

B.7 A note on performances

The main goal of T(O)RMC is not performance improvement, but to allow experimentation with automata sequence extrapolation in a variety of context that goes beyond (ω-)regular model checking problems. As such T(O)RMC is slower than tools that are specific to solving such model checking problems for the arithmetic domain (e.g. FAST [?], LIRA [?], LASH), but is perfectly competitive when handling other regular model checking cases (parametric systems, FIFO-queue systems, ...) [RMC ; Vardhan and Viswanathan 2006]. T(O)RMC relies on LASH for automata manipulations. The LASH toolset is oriented towards experimentation. It is thus less efficient for manipulating automata representing sets of real/integer numbers than LIRA and FAST that are oriented towards performances. Observe also that automata-based tools are generally less efficient than those tools which are working with specific representations such as octagons [?] or polyhedron [?].

B.8 Other applications

The ability of T(O)RMC to compute the limit of an infinite sequence of automata has other applications that are beyond the scope of existing regular model checking approaches. As an example, the tool has been used in a semi-algorithm to compute the convex hull of a set of integer vectors [Cantin et al. 2007; 2008]. The method in [Cantin et al. 2007; 2008] consists in computing a sequence of automata representing approximations of the convex hull and using the extrapolation technique of Sections 4, 5, 6 to compute the limit of this sequence[11]. The convex hull can then be directly computed from this limit in the form of an automaton-based representation of the corresponding set of real vectors. T(O)RMC was also used to compute a symbolic representation of the simulation relation between the states of several classes of infinite-state systems [Bouajjani et al. 2004].

B.9 Conclusion and observations about preciseness

We have seen that parametric and FIFO systems share similar behaviors and can be handled with the technique presented in the paper. Integer and linear hybrid systems also exhibit similar behaviors. For those systems, we can compute a safe extrapolation of the set of reachable states, but we cannot apply our preciseness check. This suggest that our criterion will work for systems whose sampling sequence is linear. In [Legay 2007] (Section 6.5.3), we have proposed another preciseness criterion, which works well for integer and linear hybrid systems[12]. The idea behind this criterion is to associate to each state of the system an integer variable that counts the number of applications of the reachability relation needed to reach this state from the initial set. Using this variable, we can propose a preciseness criterion whose induction is based on the number of applications of the reachability relation rather than on the sampling sequence. We have not been able to combine the encoding of an integer variable with the one of a FIFO or a parametric system in such

[11] The preciseness criterion used in [Cantin et al. 2008] is not the one proposed in the present paper.

[12] This criterion can handle all the integer and linear hybrid systems described in the experiments above. Of course, this has a cost since add one more integer variable in the configuration will increase the size of the automata and the transducer (see [Legay 2007]).

a way that the sequence of automata can be extrapolated. As a consequence, the check that works well for integer systems does not work for FIFO and parametric systems.