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 supported by a B.A.E.F. grant.

The present article is an improved version of [Boigelot et al. 2003], [Boigelot et al. 2004], and [Legay 2008].

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 1529-3785/YY/00-0001 $5.00
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 \cite{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 \cite{Touili2001} 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 \(T^{2i}\) with \(T^{2i+1}\) 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 these 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 sys-
tems 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 suf-
ficient 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 determin-
istic 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 (omega)Regular Reach-
ability 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 through-
out the rest of the paper. Section 3 introduces counter-word automata, a class
of counter automata that will be used by our preciseness technique. Section 4
presents the (omega)Regular Model Checking framework as well as the problems we
want to solve. Sections 5, 6, 7, 8, and 9 describe our main results. Implementation
and experiments are discussed in Section 10. Finally, Sections 11 and 12 contain
a comparison with other works on the same topic and several directions for future
research, respectively.

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 N, and N_0 is used for N \ {0}.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
2.1 Relations

Consider a set $S$, a set $S_1 \subseteq S$, and two binary relations $R_1, R_2 \subseteq S \times S$. The identity relation on $S$, denoted $R^0_{id}$ (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 $\{s' \in S_1 | (\exists s \in S_1)((s,s') \in R_1)\}$. The composition of $R_1$ with $R_2$, denoted $R_2 \circ R_1$, is the set $\{(s, s') | (\exists s'')((s, s'') \in R_1 \land (s'', s') \in R_2)\}$. The $i$th power of $R_1$ ($i \in \mathbb{N}_0$), denoted $R^i_1$, is the relation obtained by composing $R_1$ with itself $i$ times. The zero-power of $R_1$, denoted $R^0_1$, corresponds to the identity relation. The transitive closure of $R_1$, denoted $\bar{R}^+_1$, is given by $\bigcup_{i=1}^{\infty} R^i_1$, its reflexive transitive closure, denoted $R^\omega_1$, is given by $R^+_1 \cup R^{\infty}_{id}$. The domain of $R_1$, denoted $\text{Dom}(R_1)$, is given by $\{s \in S | (\exists s' \in S)((s, s') \in R_1)\}$.

2.2 Words and Languages

An alphabet is a (nonempty) finite set of distinct symbols. A finite word of length $n$ over an alphabet $\Sigma$ is a mapping $w : \{0, \ldots, n-1\} \rightarrow \Sigma$. An infinite word, also called $\omega$-word, over $\Sigma$ 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 $\Sigma$ are denoted by $\Sigma^*$ and by $\Sigma^\omega$, respectively. We define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A finite-word (respectively infinite-word) language over $\Sigma$ is a (possibly infinite) set of finite (respectively, infinite) words over $\Sigma$. Consider $L_1$ and $L_2$, two finite-word (resp. 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 $\Sigma$ that do not belong to $L_1$.

We also 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 $\Sigma$.

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

$$L_1 \times L_2 = \{(w(0), w(0)'), \ldots (w(n), w(n)') | w = w(0)w(1)\cdots w(n) \in L_1 \land w' = w(0)'w(1)'\cdots w(n)' \in L_2\}.$$ 

- If $L_1$ and $L_2$ are $\omega$-languages, the synchronous product $L_1 \times L_2$ of $L_1$ and $L_2$ is defined as follows

$$L_1 \times L_2 = \{(w(0), w(0))'(w(1), w(1))'\cdots | w = w(0)w(1)\cdots \in L_1 \land w' = w(0)'w(1)'\cdots \in L_2\}.$$ 

The language $L_1 \times L_2$ is defined over the alphabet $\Sigma^2$. 

---

\[1\] 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 \( w_1 \times w_2 \) to denote the unique word in \( L_1 \times L_2 \).

Definition 2.2. Suppose \( L \) a language over the alphabet \( \Sigma^n \) 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

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

2.3 Automata

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

- \( Q \) is a finite set of states,
- \( \Sigma \) is a finite alphabet,
- \( Q_0 \subseteq Q \) is the set of initial states,
- \( \Delta \subseteq Q \times \Sigma \times Q \) 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 \( A = (Q, \Sigma, Q_0, \Delta, F) \) be an automaton. If \((q_1, a, q_2) \in \Delta\), 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 \( q_2 \in \Delta(q_1, a) \) instead of \((q_1, a, q_2) \in \Delta\). Two transitions \((q_1, a, q_2), (q_3, b, q_4) \in \Delta\) are consecutive if \( q_2 = q_3 \). Given two states \( q, q' \in Q \) and a finite word \( w \in \Sigma^* \), we write \((q, w, q') \in \Delta^* \) 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 \((q_i, w_i, q_{i+1}) \in \Delta \) 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 \((q, a, q' \in \Delta^* \). 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 \((q, a, q') \in \Delta \). An automaton can easily be completed by adding an extra nonaccepting state.

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

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.
On (Omeg-)Regular Model Checking

The automaton $A$ may behave nondeterministically 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 $(q_1, a, q_2) \in \Delta$, 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 $(q_1, a, q_2) \in \Delta$. Operations on languages directly translate to operations on automata, and so do the notations.

One can decide whether 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 [Müller 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 $\omega$-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 \( A_1 = (Q_1, \Sigma_1, Q_{01}, \Delta_1, F_1) \) and \( A_2 = (Q_2, \Sigma_2, Q_{02}, \Delta_2, F_2) \), 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 \( L_{q_1}^{B_1}(A_1) = L_{q_2}^{B_2}(A_2) \).

Definition 2.7. Given two automata \( A_1 = (Q_1, \Sigma, Q_{01}, \Delta_1, F_1) \) and \( A_2 = (Q_2, \Sigma, Q_{02}, \Delta_2, F_2) \), a relation \( R \subseteq Q_1 \times Q_2 \) is an isomorphism between \( A_1 \) and \( A_2 \) if and only if

- \( R \) is a bijection,
- for each \( a \in (\Sigma \cup \{\epsilon\}) \) and \( q_1, q_2 \in Q_1 \), \( (q_1, a, q_2) \in \Delta_1 \) \iff \( (R(q_1), a, R(q_2)) \in \Delta_2 \),
- for each \( (q, q') \in R \), \( q \in Q_{01} \) \iff \( q' \in Q_{02} \),
- for each \( (q, q') \in R \), \( q \in F_1 \) \iff \( q' \in F_2 \).

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 \( \Sigma^n \), we denote by \( |L| \) the finite-word (respectively, infinite-word) relation over \( \Sigma^n \) consisting of the set of tuples \( (w_1, w_2, \ldots, w_n) \) such that \( w_1 \times w_2 \times \ldots \times w_n \) is in \( L \). The arity of such a relation is \( n \). Note that for \( n = 1 \), we have that \( L = |L| \). 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 \( \Sigma^n \) is \((\omega-)regular\) if there exists a \((\omega-)regular\) language \( L \) over \( \Sigma^n \) such that \( |L| = R \).
We now introduce transducers that are automata for representing \((\omega-)\)regular relations over \(\Sigma^2\).

**Definition 2.8.** A transducer over \(\Sigma^2\) is an automaton \(T\) over \(\Sigma^2\) given by \((Q, \Sigma^2, Q_0, \triangle, F)\), where
- \(Q\) is the *finite* set of states,
- \(\Sigma^2\) 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 \(\Sigma\), the transducer representing the identity relation over \(\Sigma^2\) is denoted \(T^\Sigma_{id}\) (or \(T_{id}\) when \(\Sigma\) 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 2.9.** 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 \(\Sigma\) 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 transducer that represents the relation \(R_2 \circ R_1\). We denote by \(T^i_1\) \((i \in \mathbb{N}_0)\) the transducer that represents the relation \(R^i_1\). 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 \(\Sigma\) 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 $\Sigma^2$ and let $A$ be a finite-word automaton (respectively, Büchi) automaton defined over $\Sigma$. We observe that $T_2 \circ T_1 = \pi_2[(T_1 \times T_2^\Sigma) \cap (T_1^\Sigma \times T_2)]$ and $T(A) = \pi_1[(A^2 \times \Sigma) \cap T]$, where $A^2$ is an automaton accepting $\Sigma^*$ (respectively, $\Sigma^\omega$). 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 transducers 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. COUNTER AUTOMATA

We introduce counter-word automata, a class of automata whose states are augmented by a vector of counters. Counter-word automata are intended to be used in our procedure for checking the preciseness of an extrapolation. All the concepts presented in this section are thus developed for this purpose.

3.1 Definitions

We start with the definition of a counter automaton.

**Definition 3.1.** A counter-word automaton (counter automaton for short) over an alphabet $\Sigma$ is a tuple $A_c = (n, c, Q, \Sigma, Q_0, \Delta, F)$, where

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

Let $A_c = (n, c, Q, \Sigma, Q_0, \Delta, F)$ be a counter automaton. If $(q_1, (a, v), q_2) \in \Delta$, 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** $v$. 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 $A_c = (n, c, Q, \Sigma, Q_0, \Delta, F)$, the **maximal increment value** of $A_c$ is the smallest $d \in \mathbb{N}$ such that $\Delta \subseteq Q \times (\Sigma \times [0, d]^{n}) \times Q$. Counter automata being finite structures, the maximal increment value can always be computed by enumerating the elements of the 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 “+”.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
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 3.2.** Let \( A_c = (n, c, Q, \Sigma, Q_0, \triangle, F) \) be a counter automaton. The counterless automaton corresponding to \( A_c \) is the finite automaton \( A = (Q, \Sigma, Q_0, \triangle', F) \), where

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

**Definition 3.3.** 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 \( A_c = (n, c, Q, \Sigma, Q_0, \triangle, F) \) 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. \( (\forall 0 \leq i \leq m - 1), \rho(i + 1) = (q_{i+1}, v_{i+1}) \) if and only if \( \rho(i) = (q_i, v_i) \) and there exists \( (q_i, (w(i), v), q_{i+1}) \in \triangle \) with \( v_{i+1} = v_i + v \).

Let \( \rho(m - 1) = (q_f \times \{v\}) \). If \( q_f \in F \), then we say that \( \rho \) is an accepting run and that \( w \) is accepted by \( A_c \) with the counter valuation \( v \). Otherwise \( \rho \) 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 \( A_c \) 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. \( (\forall 0 \leq i \leq m - 1), \rho(i + 1) = (q_{i+1}, v_{i+1}) \) if and only if \( \rho(i) = (q_i, v_i) \) and there exists \( (q_i, (w(i), v), q_{i+1}) \in \triangle \) with \( v_{i+1} = v_i + v \).

Contrary to the finite-word case, it is generally not possible to associate a counter valuation to \( \rho \). 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 3.4.** 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 $A_c = (n, Q, \Sigma, Q_0, \Delta, F)$ be a run-bounded weak Büchi counter automaton and $\rho$ be one of its runs. We say that $\rho$ is an accepting run and that $w$ is accepted by $A_c$ with the counter valuation $v$ if and only if $\inf(\rho) \cap (F \times \{v\}) \neq \emptyset$, where $\inf(\rho)$ is the set of configurations that appear infinitely often in $\rho$. Otherwise $\rho$ 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 3.5.** The counter language of a counter automaton $A_c$, denoted $L(A_c)$, is the set of pairs $(w, v)$ such that $w$ can be accepted by $A_c$ with counter valuation $v$.

Observe that the class of counter-word automata is particular with respect to existing classes of counter automata\(^2\) such as reversal bounded counter automata (Ibarra 1978), constraint automata (Henglein and Rehof 1998), Parikh automata (Klaedtke and Rueß 2003), or weighted automata (Mohri 2003). 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\(^3\) generally leads to more powerful models\(^4\) for which most problems are undecidable. The expressiveness of those models is not needed for the practical applications we considered in the paper.

### 3.2 Graph-Based Operations

In this section, the operations of intersection and composition defined for finite automata are extended to counter automata. We have the following definitions.

**Definition 3.6.** Let $A_{c_1} = (n_1, c_1, Q_1, \Sigma, Q_{01}, \Delta_1, F_1)$ and $A_{c_2} = (n_2, c_2, Q_2, \Sigma, Q_{02}, \Delta_2, F_2)$ 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 $A = (n_1 + n_2, c_1 \times c_2, Q, \Sigma, Q_0, \Delta, F)$ with $L(A) = L(A_{c_1}) \cap L(A_{c_2})$ and $L(A_c) = \{(w, v) \in \Sigma^* \times \mathbb{N}^{n_1+n_2} \mid (\exists (v_1, v_2) \in L(A_{c_1}))(\exists (v, v_2) \in L(A_{c_2}))(v = v_1 \times v_2)\}$.

**Definition 3.7.** Let $T_{c_1} = (n_1, c_1, Q_1, \Sigma^2, Q_{01}, \Delta_1, F_1)$ and $T_{c_2} = (n_2, c_2, Q_2, \Sigma^2, Q_{02}, \Delta_2, F_2)$ 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 $T_c = (n_1 + n_2, c_1 \times c_2, Q, \Sigma^2, Q_0, \Delta, F)$, with $L(T_c) = L(T_2 \circ T_1)$ and $L(T_c) = \{(w, v) \in \Sigma^* \times \mathbb{N}^{n_1+n_2} \mid (\exists (w_1, v_1) \in L(T_{c_1}))(\exists (w_2, v_2) \in L(T_{c_2}))(v = v_1 \times v_2 \land w = w_2 \circ w_1)\}$.

**Definition 3.8.** Let $T_1 = (Q_1, \Sigma^2, Q_{01}, \Delta_1, F_1)$ be a finite-word (respectively, run-bounded weak Büchi) transducer, and $A_{c_2} = (n_2, c_2, Q_2, \Sigma, Q_{02}, \Delta_2, F_2)$ be a finite-word (respectively, run-bounded weak Büchi) counter automaton. The

---

\(^2\)As an example, we cannot test the values of the counters.

\(^3\)As an example, one could associate constraints on each transition.

\(^4\)As an example, models that can recognize nonregular languages (Klaedtke and Rueß 2003).
counter-image of $A_{c_2}$ by $T_1$, denoted $T_1(A_{c_2})$, is the finite-word (respectively, Büchi) counter automaton $A_c = (n, c_2, Q, \Sigma, Q_0, \Delta, F)$, where $L(A_c) = L(T_1(A_{c_2}))$ and $L(A_c) = \{(w, v_2) \in \Sigma^\omega \times \mathbb{N}^n \mid (\exists w_1 \in L(T_1))(\exists(w_2, v_2) \in L(A_{c_2}))(w = w_2 \circ w_1)\}$. 

3.3 Counter-Based Operations

Let $A_c$ be a $n$-dimensional counter automaton over the alphabet $\Sigma$, 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 3.9. Let $A_c = (n, c, Q, \Sigma, Q_0, \Delta, F)$ be a counter automaton whose maximal increment value is $d$. The extended automaton corresponding to $A_c$ is the finite automaton $A = (Q, \Sigma', Q_0, \Delta', F)$, where

$-\Sigma' = \Sigma \times [0, d]^n$, and

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

A $n$-dimensional counter automaton over an alphabet $\Sigma$ 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 $\Sigma$ 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 $L((A_c)^e) = (\Sigma \times [0, d]^n)\omega$. If $A_c$ is a run-bounded weak Büchi counter automaton, then it is universal if and only if $L((A_c)^e) = (\Sigma \times [0, d]^n)\omega(\Sigma \times 0)^\omega$.

Definition 3.10. 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 3.11. The extended intersection/union of two run-bounded weak Büchi counter automata is a run-bounded weak Büchi counter automaton.

Definition 3.12. Let $A = (Q, \Sigma, Q_0, \Delta, F)$ be a finite-word (respectively, Büchi automaton), the counter-zero automaton corresponding to $A$ is the one-dimensional counter automaton $A_c = (1, c, Q, \Sigma, Q_0, \Delta', F)$, where

$-\Delta' = \{(q, (a, 0), q') \in Q \times (\Sigma \times 0) \times Q \mid (q, a, q') \in \Delta\}$.

The problem of testing the equivalence between counter languages is known to be undecidable for many classes of counter automata [Ibarra 1978], but decidability 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 3.13.** Let $A_{c_1}$ and $A_{c_2}$ be two finite-word (respectively, Büchi) counter automata of same dimension. If $L(A_{c_1}^e) = L(A_{c_2}^e)$, then $L(A_{c_1}) = L(A_{c_2})$.

There are situations where $L(A_{c_1}^e) \neq L(A_{c_2}^e)$, while $L(A_{c_1}) = L(A_{c_2})$.

**Example 3.14.** Consider the two finite-word counter automata $A_{c_1}$ and $A_{c_2}$ given in Figure 2. The automaton $A_{c_1}^e$ does not accept the same language as $A_{c_2}^e$. However, $L(A_{c_1}) = L(A_{c_2})$.

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

**Definition 3.15.** Let $A_c = (n, c, Q, Q_0, \Delta, F)$ 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 $A'_c = (n-1, c', Q, Q_0, \Delta', F)$, where $c' = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$, $L(A_c) = L(A'_c)$, and $L(A'_c) = \{(w, c_1 \times c_2) \in \Sigma^\infty \times N^{n-1} \mid \exists c_3 \in N \exists (w, c_1 \times c_3 \times c_2) \in L(A_c)\}$.

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 3.16.** 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 $(A_c)^{c_i > c_j}$ to denote $((\ldots((A_c)^{c_1 > c_2})^{c_1 > c_3})^{c_1 > c_n})^{c_1 > c_p}$.
Let $A_c$ be a finite-word (respectively, run-bounded weak Büchi) $n$-dimensional counter automaton over $\Sigma$ and whose maximal increment value is $d$. A way to compute $(A_c)^{c_i > c_j}$ 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 $A^U$. 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 $c_i > c_j$. However, since there is no bound on the difference between the values of $c_i$ and $c_j$ before the 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 having to work with 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 will see in Section 9, imposing this synchronization is sufficient for the applications we will consider. We have the following definition.

Definition 3.17. Let $A_c = (n, c, Q, \Sigma, Q_0, \Delta, F)$ be a finite-word (respectively, run-bounded weak Büchi) counter automaton and a synchronization bound $M \in \mathbb{N}$. Let $\Delta c_i(\sigma)$ denotes the difference between the value associated to the counter $c_i$ in the last and in the first state of the subrun $\sigma$ of a run $\rho$ 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 $\rho$ on $w$, we have $\|\Delta c_i(\sigma) - \Delta c_j(\sigma)\| \leq M$.

Definition 3.18. The finite-word (respectively, run-bounded weak) counter automaton $A^{MU} = (n, c, Q, \Sigma, Q_0, \Delta, F)$ 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 $c_j$. Clearly, 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 3.18 is to make sure that accepting runs are removed from $A_c$ only if they do not satisfy the constraints over $c_i$ and $c_j$.

4. THE ($\omega$)-REGULAR MODEL CHECKING 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 $\omega$-Regular Model Checking [Boigelot et al. 2004]. Formally, a finite automata-based representation of a system can be defined as follows.

**Definition 4.1.** A ($\omega$-)regular system for a system $T = (S, S_0, R)$ is a triple $M = (\Sigma, A, T)$, where

- $\Sigma$ 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 $\Sigma$ that represents $S_0$;
- $T$ is a deterministic finite-word (respectively, deterministic weak Büchi) transducer over $\Sigma^2$ 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 $\Sigma$. 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 $\Sigma$. Each state of an instance of the system can then be encoded as a finite word $w = w(0) \ldots w(n-1)$ over $\Sigma$, where $w(i-1)$ encodes the current state of the $i$th 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 4.2.** 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:
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. \((N,N)^*(T,N)(N,N)^*\) to describe the move of the token from \( w(i) \) to \( w(i+1) \) (with \( 0 \leq i \leq n-2 \)), 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 \( \Sigma \). The \( \omega \)-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] or in [Bouajjani et al. 2004; Legay 2007]).

It is known that verifying properties of systems in the \( \omega \)-Regular Model Checking framework generally reduces to solving the \( \omega \)-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 4.3.** 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 \( \omega \)-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.

The \( \omega \)-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.

5. **ON SOLVING \( \omega \)-REGULAR REACHABILITY PROBLEMS**

Among the techniques to solve the \( \omega \)-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], and for lossy queues.
in [Abdulla and Jonsson 1996]. 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.

In this paper, we propose a new generic technique to solving the ($\omega$-)Reachability Problems. We use the following definition.

**Definition 5.1.** Given a possibly infinite sequence $A^1, A^2, \ldots$ of automata, the limit of this sequence is an automaton $A^\ast$ such that $L(A^\ast) = \bigcup L(A^i)$.

Consider a transducer $T$ and an automaton $A$. We first observe that the computations of both $T^\ast$ and $T^\ast(A)$ can be reduced to the computation of the limit of a possibly infinite sequence of automata. Indeed, computing $T^\ast$ amounts to compute the limit of $T_{id}, T^1, T^2, T^3, \ldots$, and computing $T^\ast(A)$ amounts to compute the limit of $A, T^1(A), T^2(A), T^3(A), \ldots$. 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.

The presentation of our solution is organized as follows. Section 6 discusses the choice of the sampling sequence. Section 7 presents a methodology to detect increments. Section 8 presents several extrapolation algorithms. Finally, Section 9 introduces criteria to determine the correctness of the extrapolation. An implementation of those results as well as some experiments are presented in Section 10.

**6. CHOOSING THE SAMPLING SEQUENCE**

Choosing the sampling sequence is a rather 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 6.1.** 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 $L(T^\ast) = \bigcup_{k \geq 0} L(T^{s_k})$ and, similarly, $L(T^\ast(A)) = \bigcup_{k \geq 0} L(T^{s_k}(A))$.

**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, $(\forall j \leq i)(L(T^j) \subseteq L(T^i))$.
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 6.2.** Figure 3 shows the minimal transducer of Example 2.9 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 \( \{2, 7\} \).

7. DETECTING INCREMENTS

We consider a finite sequence \( A^1, A^2, A^3, \ldots, A^n \) of finite automata that are either all finite-word automata or all weak Büchi automata. Those automata are assumed to be deterministic and minimal. 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.

For each \( i > 0 \), let \( A^i = (Q^i, \Sigma, q^i_0, \delta^i, F^i) \). 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 which identical languages are accepted. Precisely, we compute a forward equivalence relation \( E^i_f \subseteq Q^i \times Q^{i+1} \) between \( A^i \) and \( A^{i+1} \). Since we are dealing with deterministic minimal automata, the forwards equivalence \( E^i_f \) is one-to-one (though not total) and can easily be computed by partitioning the states of the joint automaton \( (Q^i \cup Q^{i+1}, \Sigma, q^i_0, \delta^i \cup \delta^{i+1}, F^i \cup F^{i+1}) \) 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 7.1.** 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 \( E^i_b \subseteq Q^i \times Q^{i+1} \) between \( A^i \) and \( A^{i+1} \). Since \( A^i \) and \( A^{i+1} \) are deterministic and minimal, the backwards equivalence \( E^i_b \) can be computed by forward propagation, starting from the pair \( (q^i_0, q^{i+1}_0) \) 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 7.2.** Note that because the automata are minimal, the parts of \( A^i \) and \( A^{i+1} \) linked by \( E^i_b \) are isomorphic, outgoing transitions being ignored.

We now define a notion of finite-state increment between two successive automata, in terms of the relations \( E^i_f \) and \( E^i_b \).
Fig. 3. Transducer of Example 6.2 at powers of two.
Definitions 7.3 and 7.4 are illustrated in Figure 4.

\[ Q \]

The set \( Q \) is the following definition.

\[ A \]

The set \( A \) can be partitioned into \( \{ Q_b, Q_f \} \), such that

- The set \( Q_f \) 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 \( Q_b \) contains the remaining states.

(2) The set \( Q^{i+1} \) can be partitioned into \( \{ Q_H^{i+1}, Q_{T0}^{i+1}, Q_T^{i+1} \} \), where

- The head part \( Q_H^{i+1} \) is the image by \( E_b^i \) of the set \( Q_b \);
- The tail part \( Q_T^{i+1} \) is the image by \( E_f^i \) of the set \( Q_f \), dismissing the states that belong to \( Q_H^{i+1} \) (the intention is to have an unmodified head part);
- The increment \( Q_{T0}^{i+1} \) contains the states that do not belong to either \( Q_H^{i+1} \) or \( Q_T^{i+1} \).

Definitions 7.3 and 7.4 are illustrated in Figure 4.

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.

\[ E \]

Let \( S_j = A^i, A^{i+1}, \ldots, A^{i+k} \) and for each \( 0 \leq j \leq k \), let \( A^{i+j} = (Q^{i+j}, \Sigma, \delta^{i+j}, F^{i+j}) \) be a finite-word (respectively, weak Büchi) automata. For each \( 0 \leq j < k \), let \( E_b^{i+j} \) and \( E_f^{i+j} \) be respectively, the backward and the forward equivalences computed between \( A^i \) and \( A^{i+j} \). The automaton \( A^{i+j} \) is incrementally larger than \( A^i \) if the relations \( E_b^{i+j} \) and \( E_f^{i+j} \) cover all the states of \( A^i \). In other words, for each \( q \in Q^i \), there must exist \( q' \in Q^{i+j} \) such that \((q, q') \in E_b^{i+j} \cup E_f^{i+j} \).

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 \( E_b^i \).

Definitions 7.3 and 7.4 are illustrated in Figure 4.

![Partitioning automata states.](image-url)
forward equivalences computed between \( A^{i+j} \) and \( A^{i+j+1} \). The sequence \( S_I \) is an \textit{incrementally growing} sequence if

- For each \( 0 \leq j \leq k \), \( A^{i+j} \) is minimal;
- For each \( 0 \leq j \leq k-1 \), \( A^{i+j+1} \) is incrementally larger than \( A^{i+j} \);
- For each \( 1 \leq j \leq k-1 \), the \textit{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. For \( 2 \leq j \leq n \), the tail part \( Q_T^{i+j} \) of \( A^{i+j} \) will then consist of \( j-1 \) copies of its head increment \( Q_{I_0}^{i+j} \) plus a part that we will name the \textit{tail-end set}. Precisely, \( Q_T^{i+j} \) can be partitioned into \( \{ Q_{I_1}^{i+j}, Q_{I_2}^{i+j}, \ldots, Q_{I_{j-1}}^{i+j}, Q_{I_j}^{i+j} \} \), where

- For each \( 1 \leq \ell \leq j-1 \), the \textit{tail increment} \( Q_{I_\ell}^{i+j} \) is the image by the relation \( E_{I_\ell}^{i+j-1} \circ E_{I_\ell}^{i+j-2} \circ \cdots \circ E_{I_\ell}^{i+j-\ell} \) of the head increment \( Q_{I_0}^{i+j-\ell} \);
- The \textit{tail-end set} \( Q_{I_j}^{i+j} \) contains the remaining elements of \( Q_T^{i+j} \).

Given an automaton \( A^{i+j} \) in the sequence \( S_I \), we define its \textit{growing decomposition} w.r.t. \( S_I \), denoted \( GROW_{S_I}(A^{i+j}) \), to be the ordered list \( \{ Q_H^{i+j}, Q_{I_0}^{i+j}, \ldots, Q_{I_{j-1}}^{i+j}, Q_{I_j}^{i+j} \} \). It is easy to see that the head increment \( Q_{I_0}^{i+j} \) of \( A^{i+j} \) and all its tail increments \( Q_{I_\ell}^{i+j}, \ell \in [1, j-1] \) appearing in its tail part \( Q_T^{i+j} \) are images of the head increment \( Q_{I_0}^{i+j} \) detected between \( A^i \) and \( A^{i+1} \) by a combination of forward and backward equivalences. This observation extends to all the automata in \( S_I \). Consequently the transition graphs internal\(^\text{5}\) to all increments of all the automata in the sequence are isomorphic to that of \( Q_{I_0}^{i+1} \), and hence are isomorphic to each other. In the rest of the thesis, this isomorphism relation between two increments is called the \textit{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 \( Q_{I_0}^{i+j+1} \) is the image by \( E_b^{i+j} \) of the head part \( Q_H^{i+j} \). Consequently, 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 \textit{head isomorphism relation};
- The tail-end set \( Q_{I_j}^{i+j+1} \) is the image by \( E_f^{i+j} \) of the tail-end set \( Q_{T_j}^{i+j} \). Consequently, 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 \textit{tail-end set isomorphism relation}.

The situation is illustrated in Figure 5.

---

\(^{5}\)The transition graph only contains transitions between states of the increment.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
The property does not hold for $Q$ defined from states 3 by

\[ \{ \delta_{i+1}, \delta_{i+2}, \delta_{i+3} \} \]

The definition easily generalizes to increments of different automata.

**Definition 7.6.** Let $A^{i+k} = (Q^{i+k}, \Sigma, q_0^{i+k}, \delta^{i+k}, F^{i+k})$ be the last automaton of an incrementally growing sequence of automata $S_f = A^i, A^{i+1}, \ldots, A^{i+k}$. Assume that $\text{GROW}(S_f)(A^{i+k}) = \{ Q_h^{i+k}, \{ Q_{i-1}^{i+k}, \ldots, Q_{i-1}^{i+k}, Q_{i+1}^{i+k} \} \}$. Then, an increment $Q_{i-1}^{i+k} (0 \leq \alpha \leq k - 1)$ is said to be communication equivalent to an increment $Q_{i+1}^{i+k}$ ($0 \leq \beta \leq k - 1$) if and only if, for each pair of corresponding states (by the increment isomorphism) $(q, q')$, $q \in Q_{i-1}^{i+k}$ and $q' \in Q_{i+1}^{i+k}$, and $a \in \Sigma$, we have that, either

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

The definition easily generalizes to increments of different automata.

**Example 7.7.** Consider the automaton of Figure 6, 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 $Q_{I_2}$ 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.
8. EXTRAPOLATION ALGORITHMS

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

**Definition 7.8.** 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 \( GROW(S_I)(A^{i+k-1}) = \{Q^{i+k-1}, \{Q_0^{i+k-1}, \ldots, Q_{l-2}^{i+k-1}\}, Q_{T_I}^{i+k-1}\} \) and \( GROW(S_I)(A^{i+k}) = \{Q^{i+k}, \{Q_0^{i+k}, \ldots, Q_{l-1}^{i+k}\}, Q_{T_I}^{i+k}\} \). 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) \((q, q')\), \( q \in Q^{i+k-1}_H \) and \( q' \in Q^{i+k}_H \), and \( a \in \Sigma \), we have that, either

\[-\delta^{i+k-1}(q, a) \in Q^{i+k-1}_H \quad \text{and} \quad \delta^{i+k}(q', a) \in Q^{i+k}_H, \quad \text{hence leading to corresponding states by the existing head isomorphism between} \quad Q^{i+k-1}_H \quad \text{and} \quad Q^{i+k}_H, \quad \text{or} \]

\[-\delta^{i+k-1}(q, a) \quad \text{and} \quad \delta^{i+k}(q', a) \quad \text{are both undefined, or} \]

\[-\delta^{i+k-1}(q, a) = q^{i+k-1}_f \in Q^{i+k-1}_{T_f} \quad \text{and} \quad \delta^{i+k}(q', a) = q^{i+k}_f \in Q^{i+k}_{T_f}, \quad \text{hence leading to corresponding states by the existing tail-end set isomorphism between} \quad Q^{i+k-1}_{T_f} \quad \text{and} \quad Q^{i+k}_{T_f}, \quad \text{or} \]

\[-\delta^{i+k-1}(q, a) \in Q^{i+k-1}_{I_x} \quad \text{and} \quad \delta^{i+k}(q', a) \in Q^{i+k}_{I_x}, \quad \text{hence leading to corresponding states by the existing increment isomorphism between} \quad Q^{i+k-1}_{I_x} \quad \text{and} \quad Q^{i+k}_{I_x} \quad (0 \leq x \leq k-1). \]

**Fig. 6.** An automaton for Definition 7.6.

8. 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 \( S_I = A^{e_0}, \ldots, A^{e_k} \). The “candidate” extrapolation for \( A^1, A^2, \ldots \) is then given by the extrapolation of the sequence \( S_I \). Let \( A^{e_0} = A^{e_k} \) 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 \( Q_0^{e_0} \), and define its outgoing transitions in order to make this extra increment communication equivalent to \( Q_0^{e_0} \). By repeatedly applying this 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 \( i \geq 0, A^{e_0}, A^{e_1}, \ldots, A^{e_{k-1}}, A^{e_0}, A^{e_1}, \ldots, A^{e_i} \) grows incrementally;
- For each \( i > 0, A^{e_i} \) is communication stable with \( A^{e_0} \); and
- For each \( i > 0, \) the head increment detected between \( A^{e_{i-1}} \) and \( A^{e_i} \) is communication equivalent to \( Q_0^{e_0} \).
The limit $A^e_\ast$ 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 $A^e_\ast$. For technical reasons, the cases of finite-word and weak Büchi automata are considered separately.

8.1 Finite-word Automata

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

Consider the automaton $A^e_0$ with $GROW(S_I)(A^e_0) = \{Q^e_0, \{Q^e_{I_0}, \ldots, Q^e_{I_{k-1}}\}, Q^e_0T_f\}$. Suppose the existence of a transition labeled by $a$ from a state $x$ of $Q^e_{I_0}$ to a state $x'$ of $Q^e_{I_2}$. Since, the increment $Q^e_{I_1}$ added between $A^e_0$ and $A^e_1$ is communication equivalent to $Q^e_{I_0}$, there must exist a transition $t$ labeled by $a$ from the state isomorphic to $x$ in $Q^e_{I_0}$ to the state isomorphic to $x'$ in $Q^e_{I_2}$. 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 $Q^e_{I_2}$. 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 $Q^e_{I_0}$. Due to the communication equivalence property, a similar principle has to be applied for outgoing transitions from $Q^e_{I_0}$. 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 $A^e_\ast$ can be built from $A^e_0$ with the construction underlined in the following proposition.

**Proposition 8.1.** Let $A^e_0$ defined over $\Sigma$ be a minimal finite-word automaton which is the last automaton of an incrementally growing sequence of automata $S_I$. Assume that $GROW(S_I)(A^e_0) = \{Q^e_0, \{Q^e_{I_0}, \ldots, Q^e_{I_{k-1}}\}, Q^e_0T_f\}$. One can compute a finite-word automaton $A^e_\ast$ that represents the limit of the extrapolated sequence of origin $A^e_0$.

**Proof.** Let $\delta$ be the transition relation of $A^e_0$. The automaton $A^e_\ast$ can be built from $A^e_0$ by augmenting $\delta$ using the following rule:

For each state $q \in Q^e_{I_0} \cup Q^e_{I_2}$ and $a \in \Sigma$, if $\delta(q, a)$ leads to a state $q'$ in an increment $Q_{I_\ell}$, $1 \leq \ell \leq k-1$, then for each $0 \leq \ell < j$, add a transition.
Example 8.2. Consider the minimal finite-word automaton $A^c_0$ given in Figure 8(a), with $Q^c_{R_0} = \{0\}$, $Q^c_{I_0} = \{1\}$, $Q^c_{I_1} = \{2\}$, $Q^c_{I_2} = \{3\}$, $Q^c_{T_3} = \{4\}$, and $Q^c_{T_5} = \{5, 6\}$. Applying the construction of Proposition 8.1 to $A^c_0$ gives the automaton $A^c_\ast$ in Figure 8(b).

We now show that it is possible to add a counter $c$ to $A^c_\ast$ in such a way that when a word is accepted, the value of $c$ is the smallest index $i$ of the automaton $A^c_i$ 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 9 we sketch the construction for the automaton given in Figure 9.

**Proposition 8.3.** 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 $\text{GROW}(S_I)(A^{e_0}) = \{Q_{e_0}^H, Q_{I_0}, \ldots, Q_{I_{k-1}}, Q_{e_0}^T\}$ 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 $A^{e_0 c}$ such that (1) $L(A^{e_0 c}) = \bigcup_{i \geq 0} L(A^{e_i})$, (2) for each $(w, i) \in L(A^{e_0 c})$, $w \in L(A^{e_i})$, and (3) for each $i \geq 0$, $w \in L(A^{e_i})$, $0 \leq j \leq i$ exists such that $(w, j) \in L(A^{e_0 c})$.

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

1. Start with $\Delta = \{\emptyset\}$;
2. For each $(q, a, q') \in \delta$, add $(q, (a, 0), q')$ to $\Delta$;
3. For each state $q \in Q_{e_0}^H \cup Q_{I_0}$ 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 $\Delta$ a transition $(q, (a, j - \ell), q'')$, where $q''$ is the state corresponding to $q'$ (by the increment isomorphism) in $Q_{I_{\ell}}^H$.

Let $A^{e_0 c}$ 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 $(w, j) \in L(A^{e_0 c}) \setminus L(A^{e_0 c})$. 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 8.4.** Figure 10 presents the result of applying the construction of Proposition 8.3 to Automaton $A^{e_0}$ of Example 8.2.

### 8.2 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 8.1.
EXAMPLE 8.5. Consider the minimal weak Büchi automaton $A_{\omega}$ given in Figure 11(a), with $Q_0^e = \{0\}$, $Q_{I_0}^e = \{1\}$, $Q_{I_1}^e = \{2\}$, $Q_{I_2}^e = \{3\}$, and $Q_{I_3}^e = \{4, 5\}$. Applying the construction of Proposition 8.1 to $A_{\omega}$ gives the automaton $A^{e^*}$ in Figure 11(b). This automaton accepts the word $x\omega$ which cannot be accepted by one of the automata $A^{e^*}$ in the extrapolated sequence of origin $A_{\omega}$.

The example above shows that applying the construction of Proposition 8.1 to $A_{\omega}$ may introduce new cycles from states of $Q_{I_0}^e$ to themselves. Since the accepting runs of the $A^{e^*}$ 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 8.6. Let $A_{\omega}$ defined over $\Sigma$ be a minimal weak Büchi automaton which is the last element of an incrementally growing sequence of automata $S_I$. Assume that $GROW_{(S_I)}(A_{\omega}) = \{Q_{I_0}^e, \ldots, Q_{I_{k-1}}^e, Q_{I_k}^e\}$. One can compute a weak Büchi automaton $A^{e^*}$ that represents the limit of the extrapolated sequence of origin $A_{\omega}$.

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

1. Build an isomorphic copy $A_{I_0, \text{copy}}$ of the automaton formed by the states in $Q_{I_0}^e$, the transitions between them, and the outgoing transitions from these states to states in $Q_{I_0}^e$, $Q_{I_1}^e$, ..., $Q_{I_{k-1}}^e$, and $Q_{I_k}^e$;

2. Make all the states of $A_{I_0, \text{copy}}$ nonaccepting;

3. For each state $q \in Q_{I_0}^e \cup Q_{I_k}^e$ and $a \in \Sigma$, if $\delta(q, a)$ leads to a state $q'$ in an increment $Q_{I_j}^e$, $1 \leq j \leq k - 1$, 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 $Q_{I_j}^e$. Also, add a transition $(q, a, q''')$, where $q'''$ is the state corresponding to $q'$ in $A_{I_0, \text{copy}}$;

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

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
Example 8.7. The automaton in Figure 12 is the result of applying the construction of Proposition 8.6 to Automaton $A^{\omega}$ of Example 8.5.
PROPOSITION 8.8. Let $A^{e_1}$ be the result of applying the construction of Proposition 8.6 to $A^{o_0}$, the last automaton of a finite incrementally growing sequence of deterministic weak Büchi automata. The automaton $A^{e_1}$ may not be weak deterministic.

PROOF. Consider the minimal weak Büchi automaton $A^{o_0}$ given in Figure 13(a), with $Q_H^{o_0} = \{6, 4\}$, $Q_{I_0}^{o_0} = \{7\}$, $Q_{I_0}^{c} = \{5\}$, and $Q_{T_0}^{o_0} = \{0, 1, 2, 3, 8\}$. Applying the construction of Proposition 8.6 to $A^{o_0}$ gives the nondeterministic weak Büchi automaton $A^{e_1}$ in Figure 13(b). In this automaton, the state labeled by 9 is the duplication of $Q_{I_0}^{c}$. The result of determinizing $A^{e_1}$ is the deterministic co-Büchi automaton $A^{e_2}$ that is given in Figure 13(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 $A^{e_1}$ 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 8.9. Let $A^{o_0} = (Q, \Sigma, Q_0, \delta, F)$ be a minimal weak Büchi automaton which is the last element of an incrementally growing sequence of automata $S_I$. Assume that $GROW(S_I)(A^{o_0}) = \{Q_0^{o_0}, \{Q_{I_0}^{o_0}, \ldots, Q_{I_k}^{o_0}, \delta, F\} \}$ and let $A^{e_0}, A^{e_1}, \ldots$ be the extrapolated sequence of origin $A^{o_0}$. One can compute a run-bounded weak Büchi counter automaton $A^{e_*}$ such that (1) $L(A^{e_*}) = \bigcup_{i \geq 0} A^{e_i}$, (2) for each $(w, i) \in \mathcal{L}(A^{e_1})$, $w \in L(A^{e_1})$, and (3) for each $w \in L(A^{e_1})$, $j \leq i$ exists such that $(w, j) \in \mathcal{L}(A^{e_1})$.

PROOF. Let $\delta$ be the transition relation of $A^{o_0}$. The one-dimensional counter automaton $A^{e_1}$ is given by $(1, c, Q', \Sigma, Q_0, \triangle, F)$, with $Q$ and $\triangle$ defined as follows:

1. Start with $\triangle = \{0\}$;
2. For each $(q, a, q') \in \delta$, add $(q, (a, 0), q')$ to $\triangle$;
3. Build an isomorphic copy $A_{I_0}^{copy}$ of the automaton formed by the states in $Q_{I_0}^{o_0}$, the transitions between them, and the outgoing transitions from these states to states in $Q_{I_1}^{o_0}, Q_{I_2}^{o_0}, \ldots, Q_{I_{k-1}}^{o_0}$, and $Q_{T_0}^{o_0}$. 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 $q \in Q_{I_0}^{o_0} \cup Q_{H_0}^{o}$ and $a \in \Sigma$, if $\delta(q, a)$ leads to a state $q'$ in an increment $Q_{I_j}^{o_0}$, $1 \leq j \leq k - 1$, 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 $Q_{I_\ell}^{o_0}$. 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 $Q_{I_\ell}^{o_0}$. Also, add a transition $(q_{copy}, (a, j), q''')$, where $q'''$ is the state corresponding to $q'$ in $A_{I_0}^{copy}$. 

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
Let $A_{e0}$ be the counter-zero automaton corresponding to $A^{e0}$. From the observations above, we directly see that for each $i \in \mathbb{N}_0$ $w \in L(A^i) \setminus L(A^0)$, $1 \leq j \leq i$ exists such that $(w, j) \in \mathcal{L}(A_{e0}) \setminus \mathcal{L}(A_{e0}).$

**Example 8.10.** Figure 14 presents the result of applying the construction of Proposition 8.9 to Automaton $A_{e0}$ of Example 8.5.

9. SAFETY AND PRECISION

After having constructed a finite automaton $A_{e1}$ representing the extrapolation of a sequence $A^1$, $A^2$, ... 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 ($\omega$)-Regular Reachability Problems.

**Remark 9.1.** As we already mentioned in the introduction, the ability to extrapolate an infinite sequence of automata has other applications than solving the ($\omega$)-Regular Reachability Problems [see Bouajjani et al. 2004; Cantin et al. 2008] for examples]. Depending on the problem being considered, we may have to use other correctness criteria than those that are proposed in this paper.

9.1 Transitive Closure of a Transducer

Consider a reflexive deterministic finite-word (respectively, deterministic weak Büchi) transducer $T$ and let $T^{e0}$ be the last element of an incrementally growing sampling sequence $S_i$ of powers of $T$. Assume that $T^{e0}$ is the origin of an extrapolated sequence $T^{e0}, T^{e1}, \ldots$. The limit of this sequence is the transducer $T^{e*}$ with $L(T^{e*}) = \bigcup_{i>0} L(T^{e_i})$ that has been computed by applying the construction of Proposition 8.1 (respectively, Proposition 8.6) to $T^{e0}$. We provide sufficient criteria to test whether $L(T^{e*}) = L(T^{e*}).$

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

**Proposition 9.2.** 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 $L(T_1^e) \subseteq L(T_2)$.
PROOF. We show by induction that for each \( i > 0 \), \( L(T_i^n) \subseteq L(T_2) \). The base cases, i.e., \( L(T_1^n) \subseteq L(T_2) \) 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 \( L(T_i^n) \subseteq L(T_2) \). Indeed, \( L(T_1^n) = L(T_i^{-1} \circ T_1) \subseteq L(T_2 \circ T_1) \subseteq L(T_2 \circ T_2) \subseteq L(T_2) \).

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 9.2 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 9.2 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 8.8). 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 Buchi 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 \( T^{e^i} \) with \( j < i \), then it also holds for \( T^{e^i} \), we are done. For this, we propose the following theorem.

**Theorem 9.3.** Let \( T \) and \( T^{e^i} \) 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

\[
\forall w, \forall i > 0 \left[ w \in L(T^{e^i}) \setminus L(T^{e^0}) \Rightarrow \exists j, j' < i, w \in L(T^{e^j} \circ T^{e^j'}) \right],
\]

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 9.3 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.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
Lemma 9.4. Let $T^{c_o}$ be the last element of an incrementally growing sampling sequence $S_i$ of transducers, and $T^{c_0}$ be the counter-zero automaton corresponding to $T^{c_0}$. Assume that $T^{c_0}$ is the origin of an extrapolated sequence $T^{c_0}, T^{c_1}, \ldots$ and let $T^{c_1}, T^{c_2}, T^{c_3}$ be three copies of the counter transducer $T^{c_0}$ which is obtained by applying the construction of Proposition 8.3 (respectively, Proposition 8.9) to $T^{c_0}$. If

$$\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})) = \mathcal{L}(T^{c_0}) \setminus \mathcal{L}(T^{c_0}),$$

then

$$\forall w, \forall i > 0 \ [w \in L(T^{c_0}) \setminus L(T^{c_0}) \Rightarrow \exists 0 \leq j, j' < i, w \in L(T^{c_0} \circ T^{c_0})].$$

Proof. Observe that the counter language of $\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})$ is the counter language of $T^{c_0}$ from where one has removed all the pairs $(w, i)$ for which there is not $(w', j < i), (w'', j' < i) \in \mathcal{L}(T^{c_0})$ with $w = w' \circ w''$. For each $i$ and each word $w$, if $w \in L(T^{c_0}) \setminus L(T^{c_0})$ then, by Proposition 8.3 (respectively, Proposition 8.9), there exists $k > 0 \in \mathbb{N}$ such that $(w, k') \in \mathcal{L}(T^{c_0}) \setminus \mathcal{L}(T^{c_0})$. Since Condition (2) holds, there exist $j, j' \in \mathbb{N}$ with $j, j' < k < i$ and two words $w', w''$ such that $(w', j) \in \mathcal{L}(T^{c_0})$ and $(w'', j') \in \mathcal{L}(T^{c_0})$, with $w = w' \circ w''$. By Proposition 8.3 (respectively, Proposition 8.9), $w' \in L(T^{c_0})$ and $w'' \in L(T^{c_0})$.

We can now state our main result.

Theorem 9.5. Let $T$ be a transducer, $T^{c_0}$ the last element of an incrementally growing sampling sequence $S_i$ of powers of $T$, and $T^{c_0}$ the counter-zero automaton corresponding to $T^{c_0}$. Assume that $T^{c_0}$ is the origin of an extrapolated sequence $T^{c_0}, T^{c_1}, \ldots$ and let $T^{c_0}$ be the transducer that has been obtained by applying the construction of proposition 8.1 (respectively, Proposition 8.6) to $T^{c_0}$. Let $T^{c_1}, T^{c_2}, T^{c_3}$ be three copies of the counter transducer $T^{c_0}$ which is obtained by applying the construction of Proposition 8.3 (respectively, Proposition 8.9) to $T^{c_0}$. If $\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})) = \mathcal{L}(T^{c_0}) \setminus \mathcal{L}(T^{c_0})$, then $L(T^{c_0}) \subseteq L(T^{c_0})$.

Proof. By Proposition 8.1 (respectively, Proposition 8.6), we have $L(T^{c_0}) = \bigcup_{i=0}^{\infty} L(T^{c_i})$. According to Lemma 9.4, since

$$\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})) = \mathcal{L}(T^{c_0}) \setminus \mathcal{L}(T^{c_0}),$$

we have

$$\forall w, \forall i > 0 \ [w \in L(T^{c_0}) \setminus L(T^{c_0}) \Rightarrow \exists 0 \leq j, j' < i, w \in L(T^{c_0} \circ T^{c_0})].$$

It follows from Theorem 9.3 that $L(T^{c_0}) \subseteq L(T^{c_0})$.

Condition (2) can be implemented as follows:

— Observe that, since $\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}}))$ is disjoint from, checking $\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})) = \mathcal{L}(T^{c_0}) \setminus \mathcal{L}(T^{c_0})$ is equivalent to check $\mathcal{L}(\pi_{\{c_2, c_3\}}((T^{c_1} \cap c(T^{c_2} \cup c T^{c_3}))^{c_1 > \{c_2, c_3\}})) \cup c T^{c_0} = \mathcal{L}(T^{c_0})$.
which avoid to compute \( \mathcal{L}(T_{c_1}^e) \setminus \mathcal{L}(T_{c_2}^e) \). Computing \( \mathcal{L}(T_{c_1}^e) \setminus \mathcal{L}(T_{c_2}^e) \) 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.

—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 3. Observe that if \( T \) is weak, then the counter automaton for \( T_{c_1}^e \cap_c (T_{c_2}^e \circ_c T_{c_3}^e) \) is run-bounded weak.

—We do not compute the one-counter automaton for \( (T_{c_1}^e \cap_c (T_{c_2}^e \circ_c T_{c_3}^e)) \cup \{c_2, c_3\} \), but a \( M \)-synchronized counter automaton whose language and counter languages may be subsets of those of \( (T_{c_1}^e \cap_c (T_{c_2}^e \circ_c T_{c_3}^e)) \) and \( \{c_2, c_3\} \). We follow the methodology described in Section 3, and compute the extended-intersection between the automaton \( T_{c_1}^e \cap_c (T_{c_2}^e \circ_c T_{c_3}^e) \) 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 \( \Sigma^2 \) is the maximal increment value of \( T_{c_1}^e \). The extended alphabet of \( T_e \) is \( \Sigma \times [0, d] \), and the one of \( T_{c_1}^e \cap_c (T_{c_2}^e \circ_c T_{c_3}^e) \) is thus \( \Sigma^2 \times [0, d]^3 \) (see constructions for \( \circ_c \) and \( \cap_c \)). In our experiments, we worked with counter automata whose extended alphabet is \( \Sigma^2 \times [0, d]^3 \), and such that \( c_1 \) is \( M \)-synchronized with respect to \( c_2 \) and \( c_3 \), with \( M = 2 \times d \). This choice turned out to be the best compromise for our experimental results [Legay 2007; T(O)RMC ], where we clearly observed a synchronization between the counters.

—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 (see Proposition 3.13).

Observe that, if \( L(T^e) = L(T_{c_1}^e) \), then the transducers \( T_{c_i}^e \) \((i \geq 0)\) may constitute new elements in an extension of the sampling sequence \( T_{c_1} \), i.e., if \( S_T = T_{c_0}, T_{c_1}, \ldots, T^e \) with \( T^e \cap T_{c_0} \), then the extension is \( T_{c_1}, T_{c_2}, \ldots, T_{c_{k+1}}, T_{c_{k+2}}, \ldots \), with \( T_{c_{k+i}} = T_{c_i} \) 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 \( T_{c_i}^e \) in the extended sampling sequence is the composition of two transducers \( T_{c_j}^e \) and \( T_{c_{i'}}^e \) that appear before in this sequence. If more than one composition is needed, then the condition may not be satisfied even if \( L(T_{c_1}^e) = L(T^e) \). Condition (1) can be adapted to work with other sampling sequences. This is illustrated with the following example.

**Example 9.6.** 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

\[
\forall w, \forall i > 0 \left[ w \in L(T_{c_i}^e) \setminus L(T_{c_0}^e) \Rightarrow \exists 0 \leq j_1, \ldots, j_n < i, w \in L(T_{c_{j_1}} \circ \cdots \circ T_{c_{j_n}}) \right], 
\]

rather than to test whether Condition (1) holds.

Theorem 9.5 easily extends to other sampling sequences.
9.2 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 \( A^0 \) 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 \( A^0 \) is the origin of an extrapolated sequence \( A^0, A^1, \ldots \). The limit of this sequence is the automaton \( A^\omega \) with \( L(A^\omega) = \bigcup_{i=0}^\infty L(A^i) \) that has been computed by applying the construction of Proposition 8.1 (respectively, Proposition 8.6) to \( A^0 \). We provide sufficient criteria to test whether \( L(T^*(A)) = L(A^\omega) \).

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

**Proposition 9.7.** Let \( A_1 \) and \( A_2 \) be two automata defined over the same alphabet \( \Sigma \) and with \( L(A_1) \subseteq L(A_2) \). Let \( T \) be a reflexive transducer over \( \Sigma^2 \). If \( L(T(A_1)) \subseteq L(T^*(A_1)) \subseteq L(A_2) \) 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 \( L(T^j(A_1)) \subseteq L(A_2) \). Indeed, \( L(T^j(A_1)) = L(T(T^{j-1}(A_1))) \subseteq L(T(A_2)) \subseteq L(A_2) \). The first inclusion holds by induction and the second because \( L(T(A_2)) \subseteq L(A_2) \).

Proposition 9.7 states that checking whether \( A^\omega \) is a safe extrapolation of \( \bigcup_{i=0}^\infty T_i(A) \) can be done by checking whether \( L(T(A^\omega)) \subseteq L(A^\omega) \). It is worth mentioning that this criterion is only sufficient. Indeed, their could exist a word \( w \in L(A^\omega) \) such that \( w \notin L(T^*(A)) \) and \( w \notin L(T^*(A^\omega)) \).

We now turn to determine whether \( A^\omega \) is a precise extrapolation of \( T^*(A) \), i.e., whether \( L(A^\omega) \subseteq L(T^*(A)) \). As in Section 9.1, we use an inductive argument, which is formalized with the following theorem.

**Theorem 9.8.** Let \( T \) be a transducer and \( A, A^\omega \) be two automata. Let \( A^0 = T^0(A) \), and consider an infinite sequence of automata \( A^0, A^1, \ldots \), with \( L(A^\omega) = \bigcup_{i=0}^\infty L(A^i) \). If

\[
\forall w, \forall i > 0 \ [w \in L(A^i) \setminus L(A^0) \Rightarrow \exists j < i, w \in L(T(A^j))] \tag{4}
\]

then \( L(A^\omega) \subseteq L(T^*(A)) \).

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

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
We now go one step further and reduce the verification of Condition (4) to simple automata-based manipulations.

**Lemma 9.9.** Let $T$ be a reflexive transducer and $A$ be an automaton. Let $A^{co}$ 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 $A^{co}$ is the origin of an extrapolated sequence $A^{co}, A^{c_1}, \ldots$ and let $A_{c_1}^{e}, A_{c_2}^{e}$ be two copies of the counter automaton $A_{c}^{e}$ that is obtained by applying the construction of Proposition 8.3 (respectively, Proposition 8.9) to $(A^{co}, GROW(S_I)(A^{co}))$. Let $A_{c}^{e}$ be the counter-zero automaton corresponding to $A^{co}$. If

$$L(\pi(\neq c_2) \{[A_{c_1}^{e} \cap c \ T(A_{c_2}^{e})]^{c_1 \ast c_2}]\}) = L(A_{c}^{e}) \ \setminus \ L(A_{c}^{co}), \quad (5)$$

then

$$\forall w, \forall i > 0 \ [w \in L(A^{c_1}) \ \setminus \ L(A^{co}) \ \Rightarrow \ \exists j < i, w \in L(T(A^{c_1}))].$$

**Proof.** Observe that the counter language of $\pi(\neq c_2) \{[A_{c_1}^{e} \cap c \ T(A_{c_2}^{e})]^{c_1 \ast c_2}]\}$ is the counter language of $A_{c}^{e}$ from where one has removed all the pairs $(w, i)$ for which there is no pair $(w', j < i) \in L(A_{c}^{e})$ 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 $w \in L(A_{c}^{e}) \ \setminus \ L(A^{co})$ then, by Proposition 8.3 (respectively, Proposition 8.9), there exists $k > 0 \in \mathbb{N}$ such that $(w, k \leq i) \in L(A_{c}^{e})$. Since Condition (5) holds, there exists $j \in \mathbb{N}$ with $j < i \leq k$ and a word $w'$ such that $(w', j) \in L(T_{c}^{e})$ with $w = L(T(A_{w'}))$. By Proposition 8.3 (respectively, Proposition 8.9), $w' \in L(A^{c_1})$ and $w \in L(T(A^{c_1}))$. \qed

Finally, we obtain our main result.

**Theorem 9.10.** Let $T$ be a reflexive transducer and $A$ be an automaton. Let $A^{co}$ 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 $A^{co}$ is the origin of an extrapolated sequence $A^{co}, A^{c_1}, \ldots$ Let $A_{c}^{e}$ be the automaton that has been obtained by applying the construction of Proposition 8.1 (respectively, Proposition 8.6) to $A^{co}$, and let $A_{c_1}^{e}, A_{c_2}^{e}$ be two copies of the counter automaton $A_{c}^{e}$ that is obtained by applying the construction of Proposition 8.3 (respectively, Proposition 8.9) to $A^{co}$. Let $A_{c}^{e}$ be the counter-zero automaton corresponding to $A^{co}$. If

$$L(\pi(\neq c_2) \{[A_{c_1}^{e} \cap c \ T(A_{c_2}^{e})]^{c_1 \ast c_2}]\}) = L(A_{c}^{e}) \ \setminus \ L(A_{c}^{co}),$$

then $L(A_{c}^{e}) \subseteq L(T^{\ast}(A))$.

**Proof.** By Proposition 8.1 (respectively, Proposition 8.6), we have $L(A_{c}^{e}) = \bigcup_{i=0}^{\infty} L(A^{c_i})$. According to Lemma 9.9, since

$$L(\pi(\neq c_2) \{[A_{c_1}^{e} \cap c \ T(A_{c_2}^{e})]^{c_1 \ast c_2}]\}) = L(A_{c}^{e}),$$

we have

$$\forall w, \forall i > 0 \ [w \in L(A^{c_1}) \ \setminus \ L(A^{co}) \ \Rightarrow \ \exists j < i, w \in L(T(A^{c_1}))].$$

It follows from Theorem 9.9 that $L(A_{c}^{e}) \subseteq L(T^{\ast}(A))$. \qed
Theorem 9.10 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, we 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 \( S_I = A^{e_0}, A^{e_1}, \ldots, A^{e_k} \) with \( A^{e_k} = A^{e_0} \), then the extension is \( A^{e_0}, A^{e_1}, \ldots, A^{e_k}, A^{e_{k+1}}, \ldots \), with \( A^{e_{k+i}} = A^{e_i} \) 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 9.11.** 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

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

(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.

10. IMPLEMENTATION AND EXPERIMENTS

This section briefly discusses an implementation of our results as well as the experiments that have been conducted.

10.1 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 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.

10.2 The T(O)RMC Toolset

The results presented in this paper have been implemented in the $T(O)RMC$ (states for Tool for $(\omega)$-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 $(\omega)$-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 [Boigelot 2003], 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 $(\omega)$-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 $S = A^1, A^2, \ldots$. 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(Ω)RMC assumes that $A^\omega_r$ 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(Ω)RMC behaves as follows:

1. T(Ω)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(Ω)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(Ω)RMC goes back to point (1) and consider a prefix of a longer size. If yes, then T(Ω)RMC extrapolates $S_I$ and produces an automaton $A^\omega_r$. This extrapolation is followed by a call to the function $CHECK$ on $A^\omega_r$. If the function returns yes, then the computation terminates, and $A^\omega_r$ 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.

### 10.3 A brief Overview of the Experiments

The T(Ω)RMC toolset has been applied to more than 100 case studies. This section only briefly recaps the classes of problems for which T(Ω)RMC has been used so far. Details about the experiments (including performances in terms of time and memory, which vary from examples to examples) can be found in Chapters 7 and 13 of [Legay 2007].

We first used T(Ω)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(Ω)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(Ω)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. 2007; 2008]. T(Ω)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].

The main goal of T(Ω)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(Ω)RMC is slower than...
tools that are specific to solving such model checking problems for the arithmetic domain (e.g., FAST [Bardin et al. 2006], LIRA [Becker et al. 2007], 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.

11. A BRIEF COMPARISON WITH OTHER WORKS

In this section, we briefly compare our approach with other generic techniques for solving the (\(\omega\)-)Regular Reachability Problems.

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 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. proposed several simulation-based techniques that, given a finite-word transducer \(T\), compute a finite-state representation for \(T^+\). The core idea of those techniques is to iteratively compute the successive unions \(T^{\leq 1}, T^{\leq 2}, T^{\leq 3}, \ldots\) (where \(T^{\leq i} = \bigcup_{n=1}^{i} T^n\)) and collapsing progressively their states according to an equivalence relation, which is induced by the simulation relations. 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 [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].

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. There is no methodology to test whether the extrapolation is precise or not. It is however easy to see that our preciseness criterion directly adapts to Touili’s extrapolation procedure.

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. A drawback with this approach is that it requires the addition of witness variables that may break the regularity of the set of reachable states. 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.

Finally, even if they do not consider exactly the same problem as us, it is relevant to mention a series of recent work [Bouajjani et al. 2004; Bouajjani et al. 2005] that combine abstraction-based techniques with automata-based constructions to verifying reachability properties. Those works have been shown to be particularly efficient for parametric and queue systems [Bouajjani et al. 2004] as well as for systems manipulating pointers [Bouajjani et al. 2005]. On the other hand, one dedicated abstraction is needed for each class of system, while our extrapolation-based technique is designed to be applicable on any system that can be represented by a (ω-)regular system.

12. CONCLUSION AND FUTURE WORK

In this paper, we have introduced an extrapolation-based technique for solving the (ω-)Regular Reachability Problems. The approach consists in computing the limit of an infinite sequence of minimal finite-word (respectively, minimal weak Büchi) automata by extrapolating a finite sampled prefix of this sequence, i.e., selected automata from a prefix of the sequence. The technique does not guarantee that a result will be obtained, and correctness of the guessed extrapolation needs to be checked once it is obtained. Our results have been implemented in a tool called T(O)RMC, which has been applied to several case studies.

One possible direction for future work would be to extend the increment detection procedure described in Section 7. Indeed, as it is illustrated with the following example, the procedure is not able to detect all possible forms of increment.

**Example 12.1.** Consider the finite-word automata given in Figure 15. The automaton $A_2$ differs from the automaton $A_1$ by the addition of an increment, which is represented by state 1. If we compare $A_2$ and $A_3$, we see the addition of one more increment. Clearly, $A_3$ differs from $A_1$ by the addition of two increments represented by states 1 and 2. Unfortunately, in $A_3$, the increment detected between $A_2$ and $A_3$ (state 2 of $A_3$) is the origin of a transition whose destination is the increment detected between $A_1$ and $A_2$ (state 1 of $A_3$). Such a situation cannot be
captured with the technique introduced in Section 7.

We could also investigate whether it is possible to detect the repetition of different increment patterns in the same automaton. As an example, the automata representing $ab$, $aabb$, $aaabbb$, ... differ by the repetitions of the symbols $a$ and $b$. If we separately close those repetitions, we will obtain an automaton that represents $a^+b^+$. This language, which is an over approximation of the “correct” closure (i.e., $a^n b^n (n \in \mathbb{N})$), may be sufficient for practical applications. Another interesting direction would be to extend our results to other classes of automata, which includes tree and pushdown automata.

Another interesting direction would be to extend our results to other classes of systems such as visibly pushdown systems [Alur and Madhusudan 2004]. We could isolate a class of systems for which we can always compute a safe and precise extrapolation.

Finally, it would be of interest to extend (ω-)Regular Model Checking to the verification of Open systems. As opposed to state-transition systems, open systems are systems whose behavior depends on an external environment. In a series of fairly recent papers, symbolic games [Abdulla et al. 2003; de Alfaro et al. 2001; Bouyer et al. 2005] have been proposed as a general framework to specifying finite-state Open systems [Adler et al. 2006; de Alfaro and Henzinger 2001; de Alfaro et al. 2005]. We believe that our work could help to extending this approach to infinite-state open systems.

Thanks

We thank Bernard Boigelot for a fruitful collaboration on preliminary versions of this work. 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., and d’Orso, J. 2003. Deciding monotonic games. In Proc. 12th Int. Conference for Computer Science Logic (CSL). Lecture Notes in Computer Science, vol. 2803. Springer, 1–14.

Abdulla, P. A., Bouajjani, A., Jonsson, B., and Nilsson, M. 1999. Handling global conditions in parameterized system verification. In Proc. 11th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1633. Springer, 134–145.

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 Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, 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 Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 2725. Springer, 236–248.

Abdulla, P. A., Jonsson, B., Nilsson, M., d’Orso, J., and Saksena, M. 2004. Regular model checking for $\mu$l(mso). In Proc. 16th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 3114. Springer, 348–360.

Adler, B. T., de Alfaro, L., da Silva, L. D., Faella, M., Legay, A., Raman, V., and Roy, P. 2006. Ticc: A tool for interface compatibility and composition. In Proc. 18th Int. Conference on ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
Boigelot, B., Rassart, S., and Wolper, P. 1998. On the expressiveness of real and integer arithmetic automata (extended abstract). In Proc. 25th Int. Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 1443. Springer, 152–163.

Boigelot, B. and Wolper, P. 2002. Representing arithmetic constraints with finite automata: An overview. In Proc. 18th Int. Conference on logic Programming (ICLP). Lecture Notes in Computer Science, vol. 2401. Springer, 1–19.

Bouajjani, A., Espanza, J., and Maler, O. 1997. Reachability analysis of pushdown automata: Application to model-checking. In Proc. 8th Int. Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 1243. Springer, 135–150.

Bouajjani, A. and Habermehl, P. 1997. Symbolic reachability analysis of fifo channel systems with nonregular sets of configurations (extended abstract). In Proc. 24th Int. Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, 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 Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, 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 Int. Symposium on Static Analysis (SAS). Lecture Notes in Computer Science, vol. 4134. Springer, 52–70.

Bouajjani, A., Habermehl, P., and Vojnar, T. 2004. Abstract regular model checking. In Proc. 16th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 3114. Springer, 372–386.

Bouajjani, A., Jonsson, B., Nilsson, M., and Touili, T. 2000. Regular model checking. In Proc. 12th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1855. Springer-Verlag, 403–418.

Bouajjani, A., Legay, A., and Wolper, P. 2004. Handling liveness properties in (omega-)regular model checking. In Proc. 6th Int. Workshop on Verification of Infinite State Systems (INFINITY). Electronic Notes in Theoretical Computer Science, vol. 138(3). Elsevier Science Publishers.

Bouajjani, A. and Touili, T. 2002. Extrapolating tree transformations. In Proc. 14th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 2404. Springer, 539–554.

Bouyer, P., Cassez, F., Fleury, E., and Larsen, K. G. 2005. Synthesis of optimal strategies using hytech. In Proc. 1st Int. Workshop on Game in Design and Verification (GDV04). Electronic Notes in Theoretical Computer Science 119, 11–31.

Cantin, F., Legay, A., and Wolper, P. 2007. Computing convex hulls by automata iteration. In Proc. 1st Int. Conference automata: From Mathematics to Applications (AUTOMATHA). To appear.

Cantin, F., Legay, A., and Wolper, P. 2008. Computing convex hull by automata iteration. In Proc. 13th Int. Conference on Implementation and Application on Automata (CIAA). Lecture Notes in Computer Science, 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.

de Alfaro, L., da Silva, L. D., Faella, M., Legay, A., Roy, P., and Sorea, M. 2005. Sociable interfaces. In Proc 5th Int. Conference on Frontiers of Combining Systems (FROCO5). Lecture Notes in Computer Science, vol. 3717. Springer, 81–105.

de Alfaro, L. and Henzinger, T. A. 2001. Interface theories for component-based design. In Proc. 1st Int. Workshop on Embedded Systems (EMSOFT). Lecture Notes in Computer Science, vol. 2211. Springer, 148–165.

de Alfaro, L., Henzinger, T. A., and Majumdar, R. 2001. Symbolic algorithms for infinite-state games. In Proc. 12th Int. Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 2154. Springer, 536–550.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
On (Omega-)Regular Model Checking

Finkel, A. and Leroux, J. 2002. How to compose presburger-accelerations: Applications to broadcast protocols. In Proc. 22nd Int. Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). Lecture Notes in Computer Science, 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 Int. Workshop on Verification of Infinite State Systems (INFINITY). Electronic Notes in Theoretical Computer Science, vol. 9. Elsevier Science Publishers.

Habermehl, P. and Vojnar, T. 2004. Regular model checking using inference of regular languages. In Proc. 6th Int. Workshop on Verification of Infinite State Systems (INFINITY). Electronic Notes in Theoretical Computer Science, vol. 2556. Elsevier Science Publishers.

Henglein, F. and Rehof, J. 1998. Constraint automata and the complexity of recursive subtype entailment. In Proc. 25th Int. Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 1443. Springer, 616–627.

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 Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 1785. Springer, 220–234.

Kesten, Y., Maier, O., Marcus, M., Pnueli, A., and Shahar, E. 1997. Symbolic model checking with rich assertional languages. In Proc. 9th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1254. Springer, 424–435.

Kesten, Y., Pnueli, A., Shahar, E., and Zuck, L. D. 2002. Network invariants in action. In Proc. 13th Int. Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 2421. Springer, 101–115.

Klaedtke, F. and Rueß, H. 2003. Monadic second-order logics with cardinalities. In Proc. 30th Int. Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 2719. Springer-Verlag, 681–696.

LASH. The Li`ege 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 Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 5123. Springer, 548–551.

Löding, C. 2001. Efficient minimization of deterministic weak ω—automata. Information Processing Letters 79, 3, 105–109.

Mohri, M. 2003. Edit-distance of weighted automata: General definitions and algorithms. International Journal on Foundations of Computer Science 14, 6, 957–982.

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-Verlag, 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 Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1855. Springer, 328–343.

Rivest, R. L. and Shapiro, 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.

ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.
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 determination 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.

T(O)RMC. The T(O)RMC toolset. Available at http://www.montefiore.ulg.ac.be/~legay/TORMC/index-tormc.html.

Touili, T. 2001. Regular model checking using widening techniques. Electronique Notes in Theoretical Computer Science 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 Conference 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. 23rd Int. Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). Lecture Notes in Computer Science, 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 Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 3440. Springer, 45–60.

Vardhan, A. AND Viswanathan, M. 2006. Lever: A tool for learning based verification. In Proc. 19th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, 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 (extended abstract). In Proc. 2nd Int. Symposium on Static Analysis (SAS). Lecture Notes in Computer Science, vol. 983. Springer, 21–32.

Wolper, P. AND Boigelot, B. 1998. Verifying systems with infinite but regular state spaces. In Proc. 10th Int. Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1427. Springer-Verlag, 88–97.

Wolper, P. AND Boigelot, B. 2000. On the construction of automata from linear arithmetic constraints. In Proc. 6th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 1785. Springer, 1–19.