

A FRAMEWORK TO HANDLE LINEAR TEMPORAL PROPERTIES IN (ω -)REGULAR MODEL CHECKING

AHMED BOUAIJANI, AXEL LEGAY, AND PIERRE WOLPER

LIAFA, University of Paris 7, 2 place Jussieu, 75251 Paris cedex 5, France
e-mail address: abou@liafa.jussieu.fr

Carnegie Mellon University, Computer Science Department, Pittsburgh, PA
e-mail address: legay@montefiore.ulg.ac.be, alegay@cs.cmu.edu

University of Liège, Montefiore Institute, B28; 4000 Liège, Belgium
e-mail address: pw@montefiore.ulg.ac.be

ABSTRACT. Since the topic emerged several years ago, work on regular model checking has mostly been devoted to the verification of state reachability and safety properties. Though it was known that temporal properties could also be checked within this framework, little has been done about working out the corresponding details. This paper addresses this issue in the context of regular model checking based on the encoding of states by finite or infinite words. It works out the exact constructions to be used in both cases, and proposes a partial solution to the problem resulting from the fact that infinite computations of unbounded configurations might never contain the same configuration twice, thus making cycle detection problematic.

1. INTRODUCTION

(ω -)Regular model-checking [KMM⁺97, BJNT00, DLS02, AJNd03, BLW03, BLW04a, BHV04, Leg07] is the name of a family of techniques in which states are represented by words, sets of states by finite automata on these objects, and transitions by finite automata operating on pairs of state encodings, i.e. finite-state transducers. In this context, the problem of computing the set of reachable states of a system can be reduced to the one of computing the iterative closure of the finite-state transducer representing its transition relation. Given the expressiveness of the framework these acceleration techniques cannot be perfectly general and exact but, in many meaningful cases, they are able to compute a regular representation (or approximation) of the reachable states of infinite-state systems. However, computing reachable states is not quite model-checking. For reachability properties model checking can be reduced to a state reachability problem, but for properties that include a linear-time temporal component [Pnu77, LPZ85, Var88, GO03], the best that can be done is to reduce the linear-time model-checking problem to emptiness of a Büchi

2000 ACM Subject Classification: F.1.1, F.3.1, F.4.1, F.4.3.

Key words and phrases: Regular Model Checking, Infinite-state Systems, Linear Temporal Logic.
An extended abstract of this paper appeared as [BLW04b].

automaton [VW86], which represents all the executions of the system that do not satisfy the property. If this automaton is empty, then the system satisfies the property, else the property is not satisfied. In this framework, one thus has to check for repeated reachability rather than reachability.

In this paper, we consider the specification and the verification of linear temporal properties in the (ω) -regular model checking framework¹. The main objective of the paper is to provide a general specification framework and generic analysis techniques covering various classes of systems that can be encoded in this framework.

In (ω) -regular model checking, an execution of the system is represented as an infinite sequence of words. To define a temporal property of such an execution, one has the choice between moving within a state or between states. One thus naturally thinks of a two-dimensional logic to describe properties of such execution. However, rather than focusing on the fine points of a logic for defining properties, we have chosen to concentrate on the computational aspects of verification, and use finite (or infinite) word automata as a basis for defining temporal properties. On the executions we are considering, finite automata move either horizontally or vertically and clearly both are needed to define meaningful properties. One could consider arbitrary alternation between both directions. However, one alternation is sufficient since one can reduce the case with several alternations to the case with one alternation. We thus limited our study to horizontal properties defined in terms of vertical ones, which only makes sense for parametric systems, i.e. systems that are the result of the parallel composition of an arbitrary number of identical finite-state processes, in which a vertical slice corresponds to the execution of one process of the system; as well as to vertical properties defined in terms of horizontal ones, which makes sense for any (ω) -regular system. For both of these cases, we fully worked out how to augment the transducer representing the system transitions in order to obtain a transducer encoding the Büchi automaton resulting from combining the system with the property.

Once the transition relation of the Büchi automaton has been obtained, checking the automaton for nonemptiness is done by computing the iterative closure of this relation, finding nontrivial cycles between states, and finally checking for the reachability of states appearing in such cycles. When dealing with systems where the number of successors of each state is bounded, an accepting execution of the Büchi automaton will always contain the same state twice and hence an identifiable cycle. However, when dealing with states whose length can grow or that are infinite, there might very well be an accepting computation of the Büchi automaton in which the same state never appears twice.

To cope with this, we look for states that are not necessarily identical, but such that one entails the other in the sense that any execution possible from one is also possible from the other. The exact notion of entailment we use is simulation. For that, we compute symbolically the greatest simulation relation on the states of the system.

The nice twist is that the computation of the symbolic representation of the simulation relation, in fact, the computation of the limit of a sequence of finite-state automata, for which the acceleration techniques introduced in [BLW03, BLW04a, Leg07] can be used. However, there are also several cases where this computation converges after a finite number of steps, which has the added advantage of guaranteeing that the induced simulation equivalence relation partitions the set of configuration in a finite number of classes, and hence that existing accepting computations will necessarily be found, which might not be

¹In the rest of the paper, we use “ (ω) -regular model checking” to denote either “regular model checking” or “ ω -regular model checking”, depending on whether states are encoded by finite or infinite words.

the case when the number of simulation equivalence classes is infinite.

Structure of the paper. The paper is divided as follows. In Section 2, we recall the elementary definitions on automata theory that will be used throughout the rest of the paper. Section 3 presents the (ω -)regular model checking framework as well as a methodology to reason on infinite executions. In Sections 4, 5, 6, and 7, the verification of several classes of linear temporal properties in the (ω -)regular model checking framework is considered. Finally, Sections 8 and 9 conclude the paper with 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 concepts and definitions that will be used throughout the rest of this paper.

2.1. Relations. Given a set S , a set $S_1 \subseteq S$, and two binary² relations $R_1, R_2 \subseteq S \times S$. The identity relation on S , denoted R_{id}^S (or R_{id} when S is clear from the context) is the set $\{(s, s) | s \in S\}$. The *image* of S_1 by R_1 , denoted $R_1(S_1)$, is the set $\{s' \in S_1 | (\exists s \in S_1)((s, s') \in R_1)\}$. The *composition* of R_1 with R_2 , denoted $R_2 \circ R_1$, is the set $\{(s, s') | (\exists s'')((s, s'') \in R_1 \wedge (s'', s') \in R_2)\}$. The *i th power* of R_1 ($i \in \mathbb{N}_0$), denoted R_1^i , is the relation obtained by composing R_1 with itself i times. The *zero-power* of R_1 , denoted R_1^0 , corresponds to the identity relation. The *transitive closure* of R_1 , denoted R_1^+ , is given by $\bigcup_{i=1}^{+\infty} R_1^i$, its *reflexive transitive closure*, denoted R_1^* , is given by $R_1^+ \cup R_{id}^S$. The *domain* of R_1 , denoted $Dom(R_1)$, is given by $\{s \in S | (\exists s' \in S)((s, s') \in R_1)\}$. Finally, the *inverse* of R_1 , denoted R_1^{-1} , is the set $\{(s', s) | (s, s') \in R_1\}$.

2.2. Words and Languages. An *alphabet* is a (nonempty) finite set of distinct symbols. A *finite word* of length n over an alphabet Σ is a mapping $w : \{0, \dots, n-1\} \rightarrow \Sigma$. An *infinite word*, also called ω -word, over Σ is a mapping $w : \mathbb{N} \rightarrow \Sigma$. We denote by the term *word* either a finite word or an infinite word, depending on the context. The *length* of the finite word w is denoted by $|w|$. A finite word w of length n is often represented by $w = w(0) \cdot \dots \cdot w(n-1)$. An infinite word w is often represented by $w(0)w(1) \cdot \dots$. The sets of finite and infinite words over Σ are denoted by Σ^* and by Σ^ω , respectively. A *finite-word* (resp. *infinite-word*) *language* over Σ is a (possibly infinite) set of finite (resp. infinite) words over Σ .

We assume the reader to be familiar with the operation of union, intersection, concatenation and complement for languages. We introduce *synchronous product* and *projection*, which are two operations needed to define relations between languages.

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

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

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

²The term “binary” will be dropped in the rest of the paper.

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

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

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

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

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

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

2.3. Automata.

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

- Q is a *finite* set of *states*,
- Σ 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 a finite 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, \dots, q_{k-1} and $w_0, \dots, w_{k-2} \in \Sigma$ 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 $(q, a, q') \in \Delta$. A finite automaton can easily be completed by adding an extra nonaccepting state.

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

We distinguish between *finite-word automata* that are finite automata accepting finite words,

and *Büchi automata* that are finite 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 (resp. Büchi) automaton is said to be *regular* (resp. ω -*regular*).

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 finite automaton is denoted as a tuple $(Q, \Sigma, q_0, \delta, F)$, where q_0 is the unique initial state and $\delta : Q \times \Sigma \rightarrow Q$ 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.

Finite-word automata own the following properties :

- Finite-word automata are closed under determinization, complementation, union, projection, and intersection.
- Testing whether the language accepted by a finite-word automaton is included in the one accepted by another automaton is decidable.
- Finite-word automata admit a minimal form, which is unique up to isomorphism.

Though the union and intersection 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 [Var07] for a survey). To circle this problem, we will restrict ourselves to *weak* automata [MSS86] 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, \dots, 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, \dots, Q_m such that for every $q \in Q_i$ and $q' \in Q_j$ for which, for some $a \in \Sigma$, $q' \in \delta(q, a)$ ($q' = \delta(q, a)$ in the deterministic case), $Q_j \leq Q_i$.

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

Not all ω -regular languages can be accepted by deterministic weak Büchi automata, nor even by nondeterministic weak automata. However, there are algorithmic advantages to working with weak automata: deterministic weak automata can be complemented simply by inverting their accepting and non-accepting states; and there exists a simple determinization procedure for weak automata [Saf92], 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 [BJW01] 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 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 stop. 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 [BJW01].

Weak Büchi automata own the following properties :

- (Deterministic) weak Büchi automata are closed under union, intersection, and complementation.
- The projection of a deterministic weak Büchi automaton is a weak Büchi automaton whose deterministic version may not be weak.
- Testing whether the language accepted by a weak Büchi automaton is included in the one accepted by another automaton is decidable.
- Deterministic weak Büchi automata admit a minimal form, which is unique up to isomorphism.

2.4. Transducers. In this paper, we will consider relations that are defined over sets of words. We use the following definitions taken from [Nil01]. For a finite-word (resp. infinite-word) language L over Σ^n , we denote by $\lfloor L \rfloor$ the finite-word (resp. infinite-word) relation over Σ^n consisting of the set of tuples (w_1, w_2, \dots, w_n) such that $w_1 \bar{\times} w_2 \bar{\times} \dots \bar{\times} w_n$ is in L . The arity of such a relation is n . Note that for $n = 1$, we have that $L = \lfloor L \rfloor$. The relation R_{id} is the *identity relation*, i.e. $R_{id} = \{(w_1, w_2, \dots, w_n) | w_1 = w_2 = \dots = w_n\}$. A relation R defined over Σ^n is (ω -)regular if there exists a (ω -)regular language L over Σ^n such that $\lfloor L \rfloor = R$.

We now introduce finite transducers that are finite automata for representing (ω -)regular relations.

Definition 2.6. A finite transducer over Σ^2 is a finite automaton T over Σ^2 given by $(Q, \Sigma^2, Q_0, \Delta, F)$, where

- Q is the *finite* set of *states*,
- Σ^2 is the *finite* alphabet,
- $Q_0 \subseteq Q$ is the set of *initial states*,
- $\Delta : Q \times \Sigma^2 \times Q$ is the *transition relation*, and
- $F \subseteq Q$ is the set of accepting states (the states that are not in F are the *non accepting* states).

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

Given two transducers T_1 and T_2 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_1^i ($i \in \mathbb{N}_0$) the transducer that represents the relation R_1^i . The *transitive closure* of T is $T^+ = \bigcup_{i=1}^{\infty} T^i$; its *reflexive transitive closure* is $T^* = T^+ \cup T_{id}$. The transducer T is *reflexive* if and only if $L(T_{id}) \subseteq L(T)$. Given an automaton A 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 $R(S)$. Finally, T_1^{-1} is the *inverse* transducer for T_1 , *i.e.* the transducer that represents R_1^{-1} .

We conclude the section with the following properties :

- The composition of two finite-word transducers is a finite-word transducer.
- The composition of two deterministic weak Büchi transducer is a weak Büchi transducer whose deterministic version may not be weak.
- The image of a finite-word automaton by a finite-word transducer is a finite-word automaton.
- The image of a deterministic weak Büchi automaton by a deterministic weak Büchi transducer is a weak Büchi automaton whose deterministic version may not be weak.
- The inverse of a (deterministic) finite-word (resp. (deterministic) weak Büchi) transducer is a (deterministic) finite-word (resp. (deterministic) weak Büchi) transducer.

3. SYSTEMS MODELS AND (ω)-REGULAR MODEL CHECKING

3.1. The Framework. In this section, we recall the definition of state-transition system, that is the abstraction formalism which is generally used to describe programs. We then present an automata-based encoding of state-transition systems. Finally, the properties of this encoding are discussed.

3.1.1. State-transition Systems. Systems are often modeled as *state-transition systems*.

Definition 3.1. A state-transition system is a tuple (S, S_0, R) , where

- S is a (possibly infinite) set of *states*,
- $S_0 \subseteq S$ is a (possibly infinite) set of *initial states*, and
- $R \subseteq S \times S$ is a (possibly infinite) *reachability relation* that describes the transitions between the states of the system.

Let $\mathcal{T} = (S, S_0, R)$ be a state-transition system. If $(s, s') \in R$, then we say that there is a *transition* from s (the *origin*) to s' (the *destination*). Given two states $s, s' \in S$, we write $s \rightarrow_R s'$ if and only if $(s, s') \in R$. A state $s' \in S$ is said to be *reachable* from a state $s \in S$ if there exists $k > 0$ and states $s_0, s_1, s_2, \dots, s_{k-1} \in S$ such that $s_0 = s$, $s_{k-1} = s'$ and $s_i \rightarrow_R s_{i+1}$, for all $0 \leq i < k - 1$. The fact that (s, s') belongs to the reflexive transitive closure R^* of R is denoted by $s \rightarrow_R^* s'$. A state $s \in S$ is *reachable* if it is reachable from a state in S_0 . The set of all reachable states of \mathcal{T} is denoted $S_R^{\mathcal{T}}$. The *state space* $(S_R^{\mathcal{T}}, R_R^{\mathcal{T}})$ of \mathcal{T} is the (possibly infinite) graph whose nodes are the reachable states of \mathcal{T} , and whose edges $R_R^{\mathcal{T}}$ are given by $R \cap (S_R^{\mathcal{T}} \times S_R^{\mathcal{T}})$. We say that \mathcal{T} is *finite* if $S_R^{\mathcal{T}}$ is finite, it is *infinite* otherwise.

\mathcal{T} is said to be *locally-finite* if and only if for each state $s \in S_R^T$, the set $R^*(s)$ is finite. If there exists a state $s \in S_R^T$ such that $R^*(s)$ is infinite, then \mathcal{T} is said to be *locally-infinite*. Observe that a locally-finite system is not necessarily finite. As an example, a system with an infinite initial set of states is by definition infinite, but may be locally-finite since each initial state may only reach a finite set of states. A finite *execution* π of \mathcal{T} is a mapping $\pi : \{0, \dots, n-1\} \rightarrow S$ such that $\pi(0) \in S_0$ and for all $0 \leq i < n-1$, $\pi(i) \rightarrow_R \pi(i+1)$. A finite execution is often represented by $\pi = \pi(0)\pi(1)\pi(2)\dots\pi(n-1)$. An infinite execution π of \mathcal{T} is a mapping $\pi : \mathbb{N} \rightarrow S$ such that $\pi(0) \in S_0$ and for all $i \geq 0$, $\pi(i) \rightarrow_R \pi(i+1)$. An infinite execution is often represented by $\pi = \pi(0)\pi(1)\pi(2)\dots$. In the rest of this paper, we consider systems whose executions are *all* infinite.

One distinguishes between two types of properties.

- (1) *Reachability properties.* We assume that a reachability property φ is described as a set of state $S_\varphi \subseteq S$. The system \mathcal{T} satisfies φ iff $S_R^T \subseteq S_\varphi$. Verifying reachability properties thus reduces to computing the set of reachable states.
- (2) *Linear temporal properties.* We assume that a linear temporal property φ is described as a set of executions π_φ , which can be represented by a Büchi automaton. The system \mathcal{T} satisfies φ iff each of its executions belongs to π_φ . In general, the verification of linear temporal properties does not reduce to the computation of the set of reachable states of the system.

3.1.2. *(ω)-Regular Model Checking.* In this paper, we suppose that states of state-transition systems 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* [KMM⁺97, WB98]. If the states are encoded by infinite words, then sets of states can be represented by deterministic weak Büchi automata and relations between states by deterministic weak Büchi transducers. This setting is referred to as *ω -regular model checking* [BLW04a]. Formally, a finite automata-based representation of a state-transition system can be defined as follows.

Definition 3.2. A *(ω -)regular system* for a state-transition system $\mathcal{T} = (S, S_0, R)$ is a triple $M = (\Sigma, A_{S_0}, T_R)$, where

- Σ is a finite alphabet over which the states are encoded as finite (resp. infinite) words;
- A_{S_0} is a deterministic finite-word (resp. deterministic weak Büchi) automaton over Σ that represents S_0 ;
- T_R is a deterministic finite-word (resp. deterministic weak Büchi) transducer over Σ^2 that represents R .

States being represented by words, the notion of set of states, initial states, reachability relation, computation, reachable state, locally-finite for (ω -)regular systems are defined identically to those of the corresponding state-transition system. There are many state-transition

systems whose sets of states cannot be encoded by (ω -)regular languages³. Consequently, there are many state-transition systems for which there exists no corresponding (ω -)regular system.

In the finite-word case, an execution of the system is an infinite sequence of same-length finite words over Σ . The regular model checking framework was first used to represent parametric systems [AJMd02, BT02, KMM⁺97, ABJN99, BJNT00, KPSZ02]. The framework can also be used to represent various other models, which includes linear integer systems [WB95, WB00], FIFO-queues systems [BG96], *XML* specifications [BHRV06, Td06], and heap analysis [BHMV05, BHRV06].

As an illustration we give details on how to represent parametric systems. Let P be a process represented by a finite state-transition 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 given as an alphabet Σ . Each state of an instance of the system can then be encoded as a finite word $w = w(0) \dots w(n-1)$ over Σ , where $w(i-1)$ encodes the current state of the i th copy of P . Sets of states of several instances can thus be encoded together by finite-word automata. Observe that the states of an instance S_n are all encoded with words of the same length. Consequently, relations between states in S_n can be represented by binary finite-word relations, and eventually by transducers.

Example 3.3. Consider a simple example of parametric network of identical processes implementing a token ring algorithm. Each of these processes can be either in idle or in critical mode, depending on whether or not it owns the unique token. Two neighboring processes can communicate with each other as follows: a process owning the token can give it to its right-hand neighbor. We consider the alphabet $\Sigma = \{N, T\}$. Each process can be in one of the two following states : T (has the token) or N (does not have the token). Given a word $w \in \Sigma^*$ with $|w| = n$ (meaning that n processes are involved in the execution), we assume that the process whose states are encoded in position $w(0)$ is the right-hand neighbor of the one whose states are encoded in position $w(n-1)$. The transition relation can be encoded as the union of two *subreachability* relations that are the following:

- $(N, N)^*(T, N)(N, T)(N, N)^*$ to describe the move of the token from $w(0)$ to $w(n-1)$,
and
- $(N, T)(N, N)^*(T, N)$ 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 described by TN^* .

In the infinite-word case, an execution of the system is an infinite sequence of infinite words over Σ . The ω -regular model checking framework has been used for handling systems with both integer and real variables [BW02, BJW05], such as linear hybrid systems with a constant derivative (see examples in [ACH⁺95] or in [BLW04b, Leg07]).

Verifying reachability properties of state-transition systems using their (ω -)regular representation can easily be conducted with simple automata-based manipulations, assuming the existence of finite-word (resp. weak Büchi) automata for representing both the set of

³Indeed, there are uncountably many subsets of an infinite set of states, but only countably many finite strings of bits.

reachable states and the property. Computing an automaton that represents the set of reachable states can be reduced to the $(\omega\text{-})$ regular reachability problems defined hereafter.

Definition 3.4. Let A be a deterministic finite-word (resp weak Büchi) automaton, and T be a deterministic finite-word (resp. weak Büchi) transducer. The $(\omega\text{-})$ regular reachability problems for A and T are the following:

- (1) *Computing $T^*(A)$* : the goal is to compute a finite-word (resp. 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

Being able to compute $T^*(A)$ is clearly enough for verifying reachability properties. On the other hand, we will see that the computation of T^* is generally incontrovertible when considering the verification of temporal properties. In the rest of this paper, we propose techniques that reduce the verification of several classes of linear temporal properties to the resolution of the $(\omega\text{-})$ regular reachability problems over an augmented system.

3.2. On Solving $(\omega\text{-})$ Regular Reachability Problems. Among the techniques to compute $T^*(A)$ and T^* , 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 [BG96, BH97], for systems with integers and reals in [Boi99, BW02, BHJ03], for pushdown systems in [FWW97, BEM97], and for lossy queues in [AJ96]. Generic techniques consider automata-based representations and provide algorithms that operate directly on these representations, mostly disregarding the domain for which it is used. There are various generic techniques to computing $T^*(A)$ and T^* when considering T and A to be finite-word automata [KMM⁺97, BJNT00, DLS02, AJNd03, BLW03, BHV04, Nil05, Leg07]. The ω -regular reachability problems can be addressed with the technique introduced in [BLW04a].

3.3. Convention, Concepts, and Observations. This section introduces some concepts and observations that will be used throughout the rest of the paper. We first introduce *Büchi $(\omega\text{-})$ regular systems*.

Definition 3.5. A Büchi $(\omega\text{-})$ regular system is a tuple (M, F) , where $M = (\Sigma, A_{S_0}, T_R)$ is a $(\omega\text{-})$ regular system, and F is a deterministic finite-word (resp. deterministic weak Büchi) automaton called the *Büchi acceptance condition*.

The notions of set of states, initial states, reachability relation, computation, reachable state, and locally-finite for Büchi $(\omega\text{-})$ regular system (M, F) are defined exactly as those of its underlying $(\omega\text{-})$ regular system $M = (\Sigma, A_{S_0}, T_R)$. An infinite computation $\pi = \pi(0)\pi(1)\dots$ of (M, F) is *accepting* iff there are infinitely many i such that $\pi(i) \in L(F)$. We say that (M, F) is *empty* if all its infinite executions are non-accepting. In the rest of the paper, we abuse the notations and write $(\Sigma, A_{S_0}, T_R, F)$ instead of (M, F) .

We now reason on infinite executions. Consider a $(\omega\text{-})$ regular system $M = (\Sigma, A_{S_0}, T_R)$ that encodes a state-transition system $\mathcal{T} = (S, S_0, R)$. The fact that T_R is structure-preserving

does not imply that M is locally-finite. Indeed, as it is illustrated with the following example, each state of \mathcal{T} can potentially be associated to an infinite set of encodings.

Example 3.6. Following the framework of [WB00], the digit 5 can be encoded in base 2 as 0101, or as 00101, or as 000101, ..., and in fact by any word in the set 0^+101 .

By definition, parametric systems are always locally-finite. Indeed, the number of finite-state processes is fixed during the whole execution. This forbid to visit an infinite number of different states. Most of other classes of infinite-state systems can either be locally-finite or not, depending on their specifications.

Example 3.7. An integer system that continuously adds 1 to a variable x up to a constant value is locally-finite. However, if there is no bound on the value of x , then the system is not locally-finite.

Unfortunately, testing whether a system is locally-finite or not is an undecidable problem. As a consequence only partial solutions can be proposed. In the rest of this section, we propose such a solution that is based on a reduction to the (ω -)regular reachability problems over an augmented system. Our solution is formalized with the following theorem.

Theorem 3.8. Consider a state-transition system $\mathcal{T} = (S, S_0, R)$ and the following sets

- $S_0^a = S_0 \times \{0\}$,
- $R^a = \{((s, i), (s', i + 1)) \mid (i \in \mathbb{N})((s, s') \in R)\}$,
- $S_{lf} = \{s \in S \mid \exists i, \forall (j > i), \neg \exists s'((s, 0), (s', j)) \in (R^a)^*\}$.

If $S_0^a \subseteq S_{lf}$, then \mathcal{T} is locally-finite.

Proof. Direct by construction. □

The procedure sketched above requires to compute the set S_{lf} . In the (ω -)regular model checking framework, this computation can easily be performed when both $(R^a)^*$ and S_0^a represent solutions of Presburger arithmetic formulas [WB00, BJW05].

Remark 3.9. The solution above is not a panacea. Indeed, it is known that one can compute a regular representation of the transitive closure of the relation $\{(x, 2x)\}$ in basis 2, but the transitive closure of the relation $\{((x, y), (2x, y + 1))\}$ is not regular.

4. LINEAR TEMPORAL PROPERTIES IN REGULAR MODEL CHECKING

4.1. Definitions. In this section we propose a methodology to verify linear temporal properties of state-transition systems that are represented in the regular model checking framework. Our first step is a symbolic representation for linear temporal properties in this framework. We propose the following definitions.

Definition 4.1. Given an alphabet Σ , a *state property* is a set $cop \subseteq \Sigma^*$ that can be represented by a finite-word automaton.

Definition 4.2. Let COP be a finite set of state properties. A *global system property* over COP is a set $gsp \subseteq (2^{COP})^\omega$, i.e. a set of infinite sequences of state properties, that can be represented by a Büchi automaton.

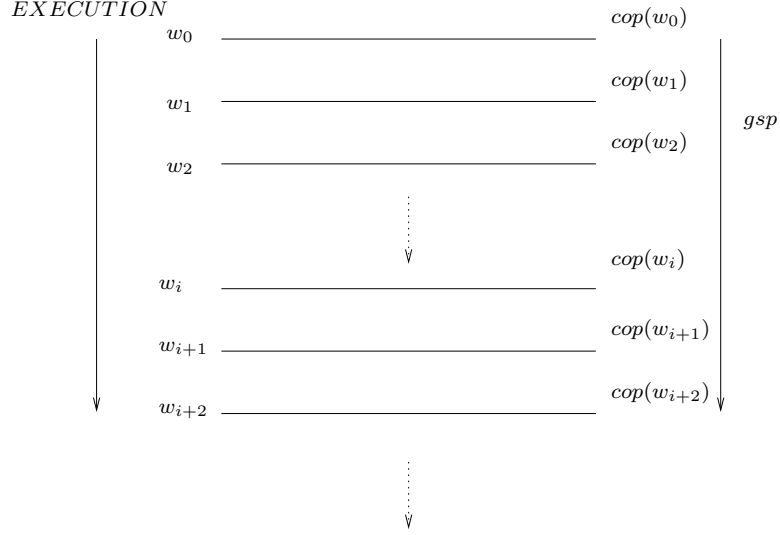


Figure 1: Global system properties: an illustration.

Assume a set of state properties COP , and a global system property gsp defined over COP . An execution $\pi = w_0w_1w_2w_3w_4 \dots$ of a regular system M satisfies gsp , denoted $\pi \models gsp$, if and only if $\mathbf{cop}(w_0)\mathbf{cop}(w_1)\dots \in gsp$, where $\mathbf{cop}(w) = \{cop_i \in COP \mid w \models cop_i\}$. We say that M satisfies gsp , denoted $M \models gsp$, if and only if all its executions satisfy the property.

Global system properties are vertical properties defined in term of horizontal ones (the state properties). The concept is illustrated in Figure 1.

Remark 4.3. Any *Linear Temporal Logic property*⁴ (LTL in short) whose atomic propositions are symbolically represented by sets of states is thus a global system property. The set of LTL properties whose atomic propositions are represented by sets of states is a strict subset of the set of global system properties.

4.2. Verification. Assume a regular system $M = (\Sigma, A_{S_0}, T_R)$, a set of state properties $COP = \{cop_1, \dots, cop_k\}$, and a global system property gsp defined over COP . Suppose that each $cop_i \in COP$ is represented by a complete deterministic finite-word automaton $A_{cop_i} = (Q_{cop_i}, \Sigma, q_{0_{cop_i}}, \delta_{cop_i}, F_{cop_i})$. We extend the automata theoretic approach of [VW86] towards a semi-algorithm to test whether M satisfies gsp . Our approach consists in three successive steps that are the following:

- (1) Computing a complete Büchi automaton $A_{\neg gsp} = (Q_{\neg gsp}, 2^{COP}, q_{0_{\neg gsp}}, \Delta_{\neg gsp}, F_{\neg gsp})$ representing the negation of the property gsp , i.e. $(2^{COP})^\omega \setminus gsp$;
- (2) Building a Büchi regular system $M_{\neg gsp}^a = (\Sigma^a, A_{S_0}^a, T_R^a, F^a)$ whose accepting executions correspond to those of M that are accepted by $A_{\neg gsp}$;
- (3) Testing whether $M_{\neg gsp}^a$ is empty or not. By construction, M satisfies gsp if and only if $M_{\neg gsp}^a$ is empty.

⁴We assume the reader is familiar with the syntax, the semantic, and the notations of the linear temporal logic introduced in [Pnu77]. We recall the shortcuts for the temporal operators that are \square for “always”, \diamond for eventually, and \bigcirc for “next”.

The property gsp being (by definition) representable by a Büchi automaton, one can always compute the automaton $A_{\neg gsp}$. We now focus on the two other problems. The system $M_{\neg gsp}^a$ can be built by taking the product between the states of M and those of $A_{\neg gsp}$. Given $w, w' \in \Sigma^*$ and $q_{\neg gsp}, q'_{\neg gsp} \in Q_{\neg gsp}$, the product must ensure that one can move from the pair $(w, q_{\neg gsp})$ to the pair $(w', q'_{\neg gsp})$ if and only if (1) $(w, w') \in T_R$, and (2) $(q_{\neg gsp}, \mathbf{cop}(w), q'_{\neg gsp}) \in \Delta_{\neg gsp}$. Since the set of states of M may be infinite, we have to work with a symbolic representation of \mathbf{cop} . We propose to represent \mathbf{cop} implicitly by associating to each pair (w, q) the set COP_i such that $\mathbf{cop}(w) = COP_i$. Hence a state of the product is now a triple $(w, q_{\neg gsp}, COP_i)$ such that (1) $w \in \Sigma^*$, (2) $q_{\neg gsp} \in Q_{\neg gsp}$, and (3) $\mathbf{cop}(w) = COP_i$. Each triple $(w, q_{\neg gsp}, COP_i)$ has to be encoded by a finite word over an extended alphabet. The solution is to label the last symbol of w with COP_i and $q_{\neg gsp}$, and the other symbols by \perp . Hence, we define the augmented alphabet to be

$$\Sigma^a = \Sigma \times (Q_{\neg gsp} \cup \{\perp\}) \times (2^{COP} \cup \{\perp\}).$$

Given a word $w^a \in (\Sigma^a)^*$, we denote by $\Pi_\Sigma(w^a)$, the word $w \in \Sigma^*$ obtained from w^a by removing all the symbols that do not belong to Σ . As an example, given $w^a = (w(0), \perp, \perp)(w(1), \perp, \perp) \cdots (w(n-1), q, \lambda)$ with $q \in Q_{\neg gsp}$, $\lambda \in 2^{COP}$, $\Pi_\Sigma(w^a) = w(0)w(1) \cdots w(n-1)$.

An execution $\pi^a = w_0^a w_1^a w_2^a \dots$ of $M_{\neg gsp}^a$ is an infinite sequence of finite words over Σ^a . This sequence has to satisfy the following four requirements:

- (1) For each $i \geq 1$ $(\Pi_\Sigma(w_{i-1}^a), \Pi_\Sigma(w_i^a)) \in T_R$, which ensures that the transitions of $M_{\neg gsp}^a$ are compatible with the transition relation of M ;
- (2) For each $i \geq 0$, $w_i^a \in (\Sigma \times \perp \times \perp)^*(\Sigma \times Q_{\neg gsp} \times 2^{COP})$;
- (3) For each $i \geq 0$ and $w_i^a = (w_i(0), \perp, \perp)(w_i(1), \perp, \perp) \cdots (w_i(n-1), q_{i-\neg gsp}, COP_i)$, $\mathbf{cop}(\Pi_\Sigma(w_i^a)) = COP_i$;
- (4) For each $i \geq 1$ and $w_{i-1}^a = (w_{i-1}(0), \perp, \perp)(w_{i-1}(1), \perp, \perp) \cdots (w_{i-1}(n-1), q_{i-1-\neg gsp}, COP_{i-1})$, and $w_i^a = (w_i(0), \perp, \perp)(w_i(1), \perp, \perp) \cdots (w_i(n-1), q_{i-\neg gsp}, COP_i)$, we have $(q_{i-1-\neg gsp}, COP_{i-1}, q_{i-\neg gsp}) \in \Delta_{\neg gsp}$, this to ensure that the infinite sequence of labellings from 2^{COP} and $Q_{\neg gsp}$ form a run of the automaton $A_{\neg gsp}$.

We have to build automata for $A_{S_0}^a$, F^a , and T_R^a in such a way that the four requirements above are satisfied.

Let $T_R = (Q_R, \Sigma^2, q_{0R}, \delta_R, F_R)$, the transducer $T_R^a = (Q_R^a, (\Sigma^a)^2, q_{0R}^a, \Delta_R^a, F_R^a)$ is built as follows:

- The set of states Q_R^a is $Q_R^a = Q_R \times \prod_{1 \leq i \leq k} Q_{cop_i} \times \{0, 1\}$, the last Boolean being used to remember if non \perp labellings have been seen and $\prod_{1 \leq i \leq k} Q_{cop_i}$ is used to run the automata representing the state properties, this to ensure that each state of $M_{\neg gsp}^a$ is associated to the set of state properties it satisfies;
- The initial state is $q_{0R}^a = (q_{0R}, q_{0cop_1}, \dots, q_{0cop_k}, 0)$;
- The transition relation Δ_R^a is defined by

$$(q'_R, q'_{cop_1}, \dots, q'_{cop_k}, b') \in \Delta_R^a((q_R, q_{cop_1}, \dots, q_{cop_k}, b), ((a_1, \alpha_1, \lambda_1), (a_2, \alpha_2, \lambda_2)))$$

if and only if

- $q'_R \in \delta_R(q_R, (a_1, a_2))$ and $q'_{cop_i} = \delta_{cop_i}(q_{cop_i}, a_1)$, for $1 \leq i \leq k$,
- $b' = 1$ if and only if $\lambda_1, \alpha_1, \lambda_2$, and α_2 are not equal to \perp and, in this case, $\alpha_2 \in \Delta_{\neg gsp}(\alpha_1, \lambda_1)$, which checks that we have a run of $A_{\neg gsp}$ and, for $1 \leq i \leq k$,

$q'_{cop_i} \in F_{cop_i}$ if and only if $cop_i \in \lambda_1$, which checks that the label λ_1 matches the result of running the automata A_{cop_i} on the state (this justify the need for each A_{cop_i} to be deterministic and complete);

- The set of accepting states F_R^a is defined as $F_R \times \prod_{1 \leq i \leq k} Q_{cop_i} \times \{1\}$.

The definition of T_R^a ensures that requirements (1) (3) and (4) are satisfied.

The set of initial states of M_{-gsp}^a contains states of the following form:

$$(w(0), \perp, \perp) \cdots (w(n-2), \perp, \perp)(w(n-1), q_{0-gsp}, \lambda),$$

where $w(0) \cdots w(n-1) \in L(A_{S_0})$ and λ is any element of 2^{COP} . This definition combined with the one of T_R^a ensures that the second requirement on the executions of M_{-gsp}^a is always satisfied. The set of initial states can be represented by a finite-word automaton $A_{S_0}^a$ that is given by $A_{S_0} \bar{\times} A_{\perp}$, where A_{\perp} is the automaton representing the set $(\perp \times \perp)^*(q_{0-gsp} \times 2^{COP})$.

The set of accepting states of M_{-gsp}^a is defined as follows:

$$(\Sigma \times \{\perp\} \times \{\perp\})^*(\Sigma \times F_{-gsp} \times 2^{COP}).$$

We directly see that this set can be represented by a finite-word automaton F^a .

Theorem 4.4. *The Büchi regular system M_{-gsp}^a has an accepting execution of the form $\pi^a = \pi^a(0)\pi^a(1) \dots$ if and only if the execution $\pi = w_0w_1w_2 \dots$ of M , where $w_i = \Pi_{\Sigma}(\pi^a(i))$ ($\forall i$), does not satisfy *gsp*.*

Proof. Follows from the construction above. □

The next step is to test whether M_{-gsp}^a is empty or not. If M is locally-finite, then M_{-gsp}^a is also locally-finite and checking the emptiness of M_{-gsp}^a can be reduced to solving the regular reachability problems. We have the following result.

Proposition 4.5. *If M_{-gsp}^a is locally finite, then it is empty if and only if*

$$L((T_R^a)^*(A_{S_0}^a) \cap F^a \cap \Pi_{\neq 2}((T_R^a)^+ \cap T_{id})) = \emptyset.$$

Proof. We first observe that $L(F^a \cap \Pi_{\neq 2}((T_R^a)^+ \cap T_{id}))$ contains the set of states accepted by F^a that are reachable from themselves by following at least a transition. We prove the two directions.

- (1) If M_{-gsp}^a is not empty, then there is at least a state w in F^a that is reachable from itself and from w_0 accepted by $A_{S_0}^a$. There exists thus an accepting execution π that passes infinitely often by w .
- (2) If M_{-gsp}^a is not empty, then there is an accepting infinite execution π that pass infinitely often by states accepted by F^a . Since M_{-gsp}^a is locally-finite, π can only meet a finite subset of the states accepted by F^a . Consequently, π reaches infinitely often a state w accepted by F^a .

□

If $M_{\neg gsp}^a$ is not locally-finite, then we cannot reduce the problem of deciding if it has an infinite accepting execution to the one of finding reachable accepting loops. Indeed, in this case, an infinite execution could never visit the same state twice. Therefore, our approach is to search for a reachable state w from which it is possible to nontrivially reach some state w' such that (1) the path from w to w' visits a repeating state of $A_{\neg gsp}$, and (2) w' has at least the same execution paths as w . To check the condition (2), we check actually for a stronger condition which is the fact that w' must *simulate* w .

We define the *greatest simulation relation* over $M_{\neg gsp}^a$ which is compatible with the set of state properties COP to be the relation Sim defined as the limit of the (possibly infinite) decreasing sequence of relations $Sim_0, Sim_1, Sim_2, \dots$ with

$$Sim_0 = \{(w_1^a, w_2^a) \mid w_1^a \bar{\times} w_2^a \in (\Sigma^a \times \Sigma^a)^* \wedge \mathbf{cop}(\Pi_{\Sigma}(w_1^a)) = \mathbf{cop}(\Pi_{\Sigma}(w_2^a))\} \quad (4.1)$$

$$Sim_{k+1} = Sim_k \cap \{(w_1^a, w_2^a) \in Sim_k \mid \forall w_3^a. ((w_1^a, w_3^a) \in T_R^a \Rightarrow \exists w_4^a. (w_2^a, w_4^a) \in T_R^a \wedge (w_3^a, w_4^a) \in Sim_k)\}, \forall k \in \mathbb{N} \quad (4.2)$$

The *complement* of Sim , denoted $\neg Sim$, is the set $\{(w_1^a, w_2^a) \mid (w_1^a \bar{\times} w_2^a \in (\Sigma^a \times \Sigma^a)^* \wedge ((w_1^a, w_2^a) \notin Sim))\}$. The *greatest simulation equivalence* over $M_{\neg gsp}^a$ which is compatible with COP is the relation $\widetilde{Sim} = Sim \cap Sim^{-1}$. Observe that Sim_0 can be represented by a finite-word transducer over the alphabet $(\Sigma^a)^2$. Since, for each $k \geq 0$, the relation Sim_{k+1} is defined in terms of the relations $[L(T_R^a)]$ and Sim_k using Boolean operations and projections (needed to apply the quantifiers), it can be represented by a finite-word transducer over the alphabet $(\Sigma^a)^2$. Moreover, if Sim_k can be represented by a transducer, then its complement and inverse can also be represented in the same way.

Assume that Sim and $(\Sigma^a)^*$ are respectively represented by a transducer T_{Sim} and an automaton $A^{(\Sigma^a)^*}$. We have the following result.

Proposition 4.6. *If*

$$L((T_R^a)^*(A_{S_0}^a) \cap \Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{\times} F^a) \cap T_{Sim})) \neq \emptyset,$$

then $M_{\neg gsp}^a$ has an infinite execution that does not satisfy gsp .

Proof. The set $L(\Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{\times} F^a) \cap T_{Sim}))$ is the set of states w from which it is possible to reach an accepting state w' such that w' simulates w . Since w' simulates w , one can reach from w' another accepting state w'' that simulates w' and, inductively, there exists an execution that infinitely often goes through an accepting state. \square

Remark 4.7. In the present context, checking that

$$L((T_R^a)^*(A_{S_0}^a) \cap \Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{\times} F^a) \cap T_{Sim})) \neq \emptyset$$

is mostly a finner way than checking

$$L((T_R^a)^*(A_{S_0}^a) \cap F^a \cap \Pi_{\neq 2}((T_R^a)^+ \cap T_{id})) \neq \emptyset$$

for discovering bugs.

The main issue is now to determine whether the iterative computation of Sim terminates. We consider the three following cases.

4.2.1. *Exact Analysis.* We say that M_{-gsp}^a has a *finite-index simulation* if the simulation equivalence \widetilde{Sim} has a finite number of equivalence classes. The following lemma is quite straightforward.

Lemma 4.8. *The iterative computation of the simulation relation Sim terminates if and only if M_{-gsp}^a has a finite-index simulation.*

If M_{-gsp}^a has a finite-index simulation equivalence, then every infinite execution of M_{-gsp}^a must visit infinitely often some of the equivalence classes. Therefore, we have the following proposition.

Proposition 4.9. *Assume that the system M_{-gsp}^a has a finite-index simulation. M_{-gsp}^a has an accepting execution if and only if*

$$L((T_R^a)^*(A_{S_0}^a) \cap \Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{\times} F^a) \cap T_{Sim})) \neq \emptyset.$$

However, the system M_{-gsp}^a is in general not finite-index simulation. Moreover this property is undecidable. Therefore, we adopt an approach based on the use of over/lower approximations of Sim .

4.2.2. *Using over approximations:* For each $k \in \mathbb{N}$, the relation \widetilde{Sim}_k is an over approximation of Sim , i.e. $Sim \subseteq Sim_k$, and the induced equivalence \widetilde{Sim}_k is finite-index. Given a transducer T_{Sim_k} representing Sim_k for $k \geq 0$. If the following condition holds

$$L((T_R^a)^*(A_{S_0}^a) \cap \Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{\times} F^a) \cap T_{Sim_k})) = \emptyset,$$

then we can deduce that M_{-gsp}^a has no accepting execution, which means that M_{-gsp}^a satisfies the property gsp .

4.2.3. *Using lower approximations:* Instead of computing the decreasing sequence of relations $(Sim_i : i \in \mathbb{N})$, we can compute the *increasing* sequence of their negations $(\neg Sim_i : i \in \mathbb{N})$. Then, the computed sequence of relations is actually an increasing sequence of relations $(N_i : i \in \mathbb{N})$ such that for every $i \geq 0$, $N_i = \neg Sim_i$. Since each N_i can be represented by a transducer, we can use the extrapolation-based technique of [BLW03, BLW04a, Leg07]. The technique can compute an automaton that represents an *extrapolation* N^{e*} of the limit $\bigcup_{i=0}^{+\infty} N_i$ by observing finite prefixes of the sequence N_0, N_1, N_2, \dots . A sufficient criterion to test whether this extrapolation is safe (does it contain the limit?) consists in applying one more time the construction that builds Sim_{k+1} from Sim_k to the complement of N^{e*} , and then check if the complement of the result we obtain is included in N^{e*} . We have not been able to find a technique to test the preciseness (is it exactly the limit?) of our result⁵. Consequently, we can use the technique of [BLW03, BLW04a, Leg07] to compute an upper approximation N^{e*} of the limit of the sequence $(N_i : i \in \mathbb{N})$. The negation of N^{e*} , denoted $\neg N^{e*}$, is a lower approximation of S . Let $T_{\neg N^{e*}}$ be the transducer representing $\neg N^{e*}$. If the following condition holds

⁵The preciseness criteria introduced in [BLW03, BLW04a, Leg07] do not apply in the present context. Indeed those criteria are based on the property that each element in the sequence that is extrapolated can be obtained from the previous one by applying an operation of composition between two (ω) -regular relations. Unfortunately, moving from Sim_k to Sim_{k+1} cannot be expressed as the composition of two (ω) -regular relations. Indeed, the application of the operator \neg (implicitly contained in the definition of \forall) requires the use of second order variables.

$$L((T_R^a)^*(A_{S_0}^a) \cap \Pi_{\neq 2}((T_R^a)^+ \cap (A^{(\Sigma^a)^*} \bar{x} F^a) \cap T_{\neg Ne*})) \neq \emptyset,$$

then we can deduce that $M_{\neg gsp}^a$ has an infinite accepting execution, which means that $M_{\neg gsp}^a$ does not satisfy the property gsp .

Remark 4.10. The technique of [BLW03, BLW04a, Leg07] has been implemented in a tool called T(O)RMC [T(O)] (states for “a Tool for (Omega-)Regular Model Checking). In the experiments (see [Leg07]), we observed that T(O)RMC almost always produces extrapolations that are both safe and precise. This gives us some confidence on the fact that the safe lower approximation we compute is in fact the simulation itself.

5. LINEAR TEMPORAL PROPERTIES IN ω -REGULAR MODEL CHECKING

5.1. Definitions. We extend the concept of global system properties from regular to ω -regular systems. For this, we simply encode state-properties as sets of infinite words rather than sets of finite words. We propose the following definitions.

Definition 5.1. Given an alphabet Σ , a ω -state property is a set $cop \subseteq \Sigma^\omega$ that can be represented by a deterministic weak Büchi automaton.

The choice of using deterministic weak automata to represent ω -state properties is for technical reasons that will be clarified in the next section.

Definition 5.2. Let COP be a finite set of ω -state properties defined over an alphabet Σ . An ω -global system property over COP is a set $gsp \subseteq (2^{COP})^\omega$, i.e. a set of infinite sequences of ω -state properties, that can be represented by a Büchi automaton.

Assume a set of ω -state properties COP , and a global system property gsp defined over COP . An execution $\pi = w_0 w_1 w_2 w_3 w_4 \dots$ of an ω -regular system M satisfies gsp , denoted $\pi \models gsp$, if and only if $\mathbf{cop}(w_0)\mathbf{cop}(w_1)\dots \in gsp$, where $\mathbf{cop}(w) = \{cop_i \in COP \mid w \models cop_i\}$. We say that M satisfies gsp , denoted $M \models gsp$, if and only if all its executions satisfy the property.

5.2. Verification. Assume an ω -regular system $M = (\Sigma, A_{S_0}, T_R)$, a set of ω -state properties $COP = \{cop_1, \dots, cop_k\}$, and an ω -global system property gsp defined over COP . Suppose that the negation of gsp can be represented by a Büchi automaton $A_{\neg gsp} = (Q_{\neg gsp}, 2^{COP}, q_{0\neg gsp}, \Delta_{\neg gsp}, F_{\neg gsp})$, and that each $cop_i \in COP$ can be represented by a complete deterministic weak Büchi automaton $A_{cop_i} = (Q_{cop_i}, \Sigma, q_{0cop_i}, \delta_{cop_i}, F_{cop_i})$.

To test whether M satisfies gsp , we proceed as in Section 4.2 and build a Büchi ω -regular system $M_{\neg gsp}^a = (\Sigma^a, A_{S_0}^a, T_R^a, F^a)$ whose executions correspond to those of M that do not satisfy gsp . We then check whether $M_{\neg gsp}^a$ is empty or not. We already provided partial solutions to test whether a Büchi regular system is empty or not, and those solutions directly extend to Büchi ω -regular systems. In the rest of this section, we mainly focus on the construction of $M_{\neg gsp}^a$.

The main difference between the present case and the one in Section 4.2 is that since we are

working with infinite-words, we cannot encode the current state of the Büchi automaton A_{-gsp} and the current set of ω -state properties satisfied only in one position of each word of M . Therefore, we include this information everywhere (in each position) of the word. We must also ensure that this information is the same for each position (which is needed to be coherent with the definition of product between M and A_{-gsp}). We use the following augmented alphabet:

$$\Sigma^a = \Sigma \times Q_{-gsp} \times 2^{COP}.$$

Let $T_R = (Q_R, \Sigma^2, Q_{0R}, \delta_R, F_R)$, the possibly nondeterministic transducer $T_R^a = (Q_R^a, (\Sigma^a)^2, Q_{0R}^a, \Delta_R^a, F_R^a)$ is built as follows:

- The set of states Q_R^a is $Q_R^a = Q_R \times \prod_{1 \leq i \leq k} Q_{cop_i} \times Q_{-gsp} \times 2^{COP}$. Instead of the Boolean variable, we have to store the state A_{-gsp} and the set $COP_i \in 2^{COP}$ in each state of T_R^a ;
- The set of initial states Q_{0R}^a contains elements of the form $(q_{0R}, q_{0cop_1}, \dots, q_{0cop_k}, q_{0-gsp}, \lambda)$, where λ is any element in 2^{COP} ;
- The transition relation Δ_R^a is defined by $(q'_R, q'_{cop_1}, \dots, q'_{cop_k}, \alpha_1, \lambda_1) \in \Delta_R^a((q_R, q_{cop_1}, \dots, q_{cop_k}, \alpha_1, \lambda_1), ((a_1, \alpha_1, \lambda_1), (a_2, \alpha_2, \lambda_2)))$ if and only if
 - $q'_R \in \delta_R(q_R, (a_1, a_2))$ and $q'_{cop_i} = \delta_{cop_i}(q_{cop_i}, a_1)$, for $1 \leq i \leq k$,
 - $\alpha_2 \in \delta_{-gsp}(\alpha_1, \lambda_1)$, which checks that we have a run of A_{-gsp} ;
- The set of accepting states F_R^a contains states of the form $(q_R, q_{cop_1}, \dots, q_{cop_k}, \alpha_1, \lambda_1)$ with for $1 \leq i \leq k$, $q'_{cop_i} \in F_{cop_i}$ iff $cop_i \in \lambda_1$.

Observe that, since T_R is deterministic weak and the ω -state properties are represented by deterministic weak automata, the transducer T_R^a is also deterministic weak.

The initial states of M_{-gsp}^a are those of the following form:

$$(w(0), q_{0-gsp}, \lambda)(w(1), q_{0-gsp}, \lambda) \dots$$

where $w(0)w(1)\dots \in L(A_{S_0})$, and λ is any element of 2^{COP} .

The set of accepting states of M_{-gsp}^a are those of the following form:

$$(\Sigma \times q_{-gsp} \times COP_i)(\Sigma \times q_{-gsp} \times COP_i) \dots$$

where $COP_i \in 2^{COP}$ and $q_{-gsp} \in F_{-gsp}$. We directly see that the sets of initial and accepting states can be represented by deterministic weak automata.

Theorem 5.3. *The Büchi regular system M_{-gsp}^a has an accepting execution of the form $\pi^a = \pi^a(0)\pi^a(1)\dots$ if and only if the execution $\pi = w_0w_1w_2\dots$ of M , where $w_i = \Pi_\Sigma(\pi^a(i))$ ($\forall i$), does not satisfy gsp .*

Proof. Follows from the construction above. □

As already mentioned, testing the emptiness of M_{-gsp}^a can be done with the techniques developed in Section 4.2. In the present situation, the definition of *greatest simulation relation* over M_{-gsp}^a is given by the limit of the (possibly infinite) decreasing sequence of relations Sim_0, Sim_1, \dots defined as follows:

$$Sim_0 = \{(w_1^a, w_2^a) \mid w_1^a \bar{\times} w_2^a \in (\Sigma^a \times \Sigma^a)^\omega \wedge \mathbf{cop}(\Pi_\Sigma(w_1^a)) = \mathbf{cop}(\Pi_\Sigma(w_2^a))\} \quad (5.1)$$

$$Sim_{k+1} = Sim_k \cap \{(w_1^a, w_2^a) \in Sim_k \mid \forall w_3^a. ((w_1^a, w_3^a) \in T_R^a \Rightarrow \exists w_4^a. (w_2^a, w_4^a) \in T_R^a \wedge (w_3^a, w_4^a) \in Sim_k)\}, \forall k \in \mathbb{N} \quad (5.2)$$

To compute a lower approximation of Sim we can again use the extrapolation-based technique of [BLW03, BLW04a, Leg07]. In the present case, the technique requires that each of the Sim_k can be represented by a deterministic weak automaton. It is easy to see that Sim_0 can be represented by a deterministic weak Büchi automaton. However, the fact that Sim_k is represented by a deterministic weak Büchi automaton does not necessarily imply that Sim_{k+1} can be represented in the same way. Indeed, building Sim_{k+1} from Sim_k requires projection operations, and there is not guarantee that the resulting automaton can be turned to a weak deterministic one.

6. LINEAR TEMPORAL PROPERTIES FOR PARAMETRIC SYSTEMS : PARAMETRIZATION

Suppose that we are working with a regular system representing a parametric system. Global system properties allow to express *communal temporal properties* of parametric systems, *i.e.* properties such as “if a process is in a state s_1 , then finally some (possibly different) process will reach a state s_2 . However, global system properties cannot express *individual temporal properties*, *i.e.* properties such as “if the process i is in a state s_1 , then finally the process i (the same process) will reach a state s_2 ”. Indeed, global system properties can only reason on the whole execution of a system, while individual temporal properties require to reason on the execution of one of the processes. In this section, we define a new class of temporal properties that allows to express individual temporal properties of parametric systems.

6.1. Definitions. In our model, an execution of a parametric system is represented by an infinite sequence of identical length finite words (see Example 3.3 and [Leg07] for more details). Each position in these words corresponds to the state of a process, also called a *local state*, and the infinite sequences of identically positioned letters in an execution represents a process execution. We thus use the following notations and definitions.

Definition 6.1. Consider an execution $\pi = w_0 w_1 w_2 w_3 \dots$ of a regular system $M = (\Sigma, A_{S_0}, T_R)$. The j th local projection $\Pi_j(\pi)$ is the infinite word $w_0(j) w_1(j) w_2(j) \dots$.

Given an execution $\pi = w_0 w_1 w_2 w_3 \dots$ of a parametric system, the j th local projection $\Pi_j(\pi)$ corresponds to the execution of the j th process.

Definition 6.2. Given an alphabet Σ , a *local execution property* is a set $lep \subseteq \Sigma^\omega$ that can be represented by a Büchi automaton.

A local execution property lep is *satisfied by an execution π of a parametric system at position j* , denoted $\Pi_j(\pi) \models lep$, if and only if $\Pi_j(\pi) \in lep$.

We are now ready to define a logic suited for parametric systems.

Definition 6.3. Given a set of local execution properties $LEP = \{lep_1, \dots, lep_k\}$, a *local-oriented system property* is a set $losp \subseteq (2^{LEP})^*$, *i.e.* a set of finite sequences of subsets of LEP , that can be represented by a finite-word automaton.

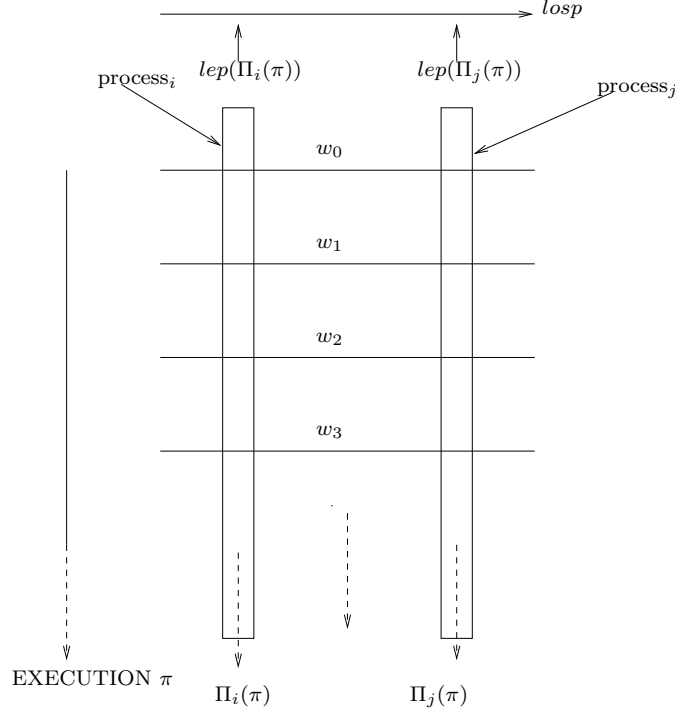


Figure 2: Local-oriented system properties: an illustration.

Assume a local-oriented system property $losp$ defined over LEP . An execution π of a parametric system M satisfies $losp$, denoted $\pi \models losp$, if and only if $\mathbf{lep}(\Pi_1(\pi)) \mathbf{lep}(\Pi_2(\pi)) \cdots \mathbf{lep}(\Pi_n(\pi)) \in losp$, where n is the common length of the words in π , and $\mathbf{lep}(\Pi_i(\pi)) = \{lep_i \in LEP \mid \Pi_i(\pi) \models lep_i\}$. We say that M satisfies $losp$, denoted $M \models losp$, if and only if all its executions satisfy the property.

Local-oriented system properties are thus horizontal properties defined in term of vertical ones (the local execution properties). The concept is illustrated in Figure 2.

Example 6.4. Consider the parametric system defined in Example 3.3. Given a natural i and a state N , the Boolean proposition $N[i]$ is true if and only if the i -th process involved in the computation (i.e. the one whose state is encoded in the i -th letter of the word describing the global state) is in state N . The fact that whenever a process i is in state $N[i]$, it will eventually move to state $T[i]$ ($\square(N[i]) \Rightarrow \diamond T[i]$) using the well-known notations for LTL is a local execution property. That this property holds for each process ($\forall i(\square(N[i]) \Rightarrow \diamond T[i])$) is then a local-oriented system property. It is easy to see that this property is trivially satisfied by the system. Indeed, the transition relation does not allow for a process to keep the token indefinitely.

We can generalize the previous example as follows : “Any local-oriented system property can be described with a formula in monadic second-order logic whose atomic propositions are Büchi automata ”.

6.2. Verification. Consider a regular system $M = (\Sigma, A_{S_0}, T_R)$ that represents a parametric system, a set of local execution properties $LEP = \{\ell ep_1, \dots, \ell ep_k\}$, and a local-oriented system property ℓosp defined over LEP . Suppose that for $1 \leq i \leq k$, ℓep_i is represented by a Büchi automaton $A_{\ell ep_i} = (Q_{\ell ep_i}, \Sigma, q_{0\ell ep_i}, \Delta_{\ell ep_i}, F_{\ell ep_i})$, which is assumed to be *complete*. We extend the automata theoretic approach of [VW86] towards a semi-algorithm to test whether M satisfies ℓosp . Our approach consists in three successive steps that are the following:

- (1) Computing a deterministic finite-word automaton $A_{\neg \ell osp} = (Q_{\neg \ell osp}, 2^{LEP}, q_{0\neg \ell osp}, \delta_{\neg \ell osp}, F_{\neg \ell osp})$, which is the finite-word automaton accepting the finite sequences that do not satisfy ℓosp , *i.e.* sequences in $\neg \ell osp = (2^{LEP})^* \setminus \ell osp$;
- (2) Building a Büchi regular system $M_{\neg \ell osp}^a = (\Sigma^a, A_{S_0}^a, T_R^a, F^a)$ whose accepting executions correspond to those of M that are accepted by $A_{\neg \ell osp}$;
- (3) Testing whether $M_{\neg \ell osp}^a$ is empty or not.

The property ℓosp being (by definition) representable by a finite-word automaton, one can always compute the automaton $A_{\neg \ell osp}$. Computing $M_{\neg \ell osp}^a$ is a much harder endeavor for which we propose the following solution.

For each automaton $A_{\ell ep_i}$, we assume the existence of a *complete* automaton $A_{\neg \ell ep_i} = (Q_{\neg \ell ep_i}, \Sigma, q_{0\neg \ell ep_i}, \Delta_{\neg \ell ep_i}, F_{\neg \ell ep_i})$ whose accepted language is the complement of the one of $A_{\ell ep_i}$. Consider an execution π^a of $M_{\neg \ell osp}^a$. Since, *a priori*, we do not know which local execution property will be satisfied by which process, each of the automata $A_{\ell ep_i}$ and $A_{\neg \ell ep_i}$ has to be run in parallel⁶ with the local executions of the processes involved in π . So, we need to extend the alphabet of M in such a way that each local state is now also labeling by a state of each of the $A_{\ell ep_i}$ and $A_{\neg \ell ep_i}$. For each $1 \leq i \leq k$, running $A_{\neg \ell ep_i}$ is necessary since the automaton $A_{\ell ep_i}$ being nondeterministic, the fact that it has a nonaccepting run does not indicate that the corresponding property does not hold.

Furthermore, in each local state, each property $\ell ep_i \in LEP$ might be satisfied ($A_{\ell ep_i}$ has an accepting run), or might not be satisfied ($A_{\neg \ell ep_i}$ has an accepting run). We make a note of these facts by also labeling each local state by an element of 2^{LEP} corresponding exactly to the properties ℓep_i that are satisfied. This labeling will remain unchanged from state to state and will enable us to run the automaton $A_{\neg \ell osp}$. The next step is to check whether there is an execution of $M_{\neg \ell osp}^a$ that is accepting for suitable automata $A_{\ell ep_i}$ and $A_{\neg \ell ep_i}$. Precisely, at a given position j in the state, the run of the automaton $A_{\ell ep_i}$ has to be accepting if $\ell ep_i \in \mathbf{lep}_j$ and the run of $A_{\neg \ell ep_i}$ has to be accepting if $\ell ep_i \notin \mathbf{lep}_j$, where \mathbf{lep}_j is the element of 2^{LEP} labeling that position. We face thus with the problem of checking not one, but several Büchi conditions, *i.e.* a generalized Büchi condition. To do this, we use the fact that a generalized Büchi automaton has an accepting run exactly when it has an accepting run that goes sequentially through each of the accepting sets. We now define $M_{\neg \ell osp}^a$. The augmented alphabet is

$$\Sigma^a = \Sigma \times \prod_{1 \leq i \leq k} Q_{\ell ep_i} \times \prod_{1 \leq i \leq k} Q_{\neg \ell ep_i} \times 2^{LEP} \times 2^{LEP} \times \{\text{reset}, \text{noreset}\}.$$

We thus have two subsets of LEP , the second being used to remember if suitable automata checking for properties ℓep_i (or $\neg \ell ep_i$) have seen an accepting state; the last component of the labeling indicates whether the second of these subsets has just been reset or not. We

⁶This can be achieved since the automata are complete.

denote by $\Pi_\Sigma(w^a)$, the word $w \in \Sigma^*$ obtained from w^a by removing all the symbols that do not belong to Σ .

An execution $\pi^a = w_0^a w_1^a w_2^a \dots$ of M_{-gsp}^a is an infinite sequence of finite words over Σ^a that has to satisfy three requirements:

- (1) For each $i \geq 1$ ($\Pi_\Sigma(w_{i-1}^a), \Pi_\Sigma(w_i^a)$) $\in T_R$, which ensures that the transitions of M_{-gsp}^a are compatible with the transition relation of M ;
- (2) For each position in a state, the labeling by states of the A_{lep_i} form a run of these automata;
- (3) The labeling of each position by elements of 2^{LEP} stays the same when moving from one state to the next one.

We have to build $A_{S_0}^a$, F^a , and T_R^a in such a way that the three requirements above are satisfied.

Let $T_R = (Q_R, \Sigma^2, q_{0R}, \delta_R, F_R)$. The possibly nondeterministic transducer $T_R^a = (Q_R^a, (\Sigma^a)^2, q_{0R}^a, \Delta_R^a, F_R^a)$ is built as follows:

- Its set of states and accepting states are respectively $Q_R^a = Q_R$ and $F_R^a = F_R$, its initial state is $q_{0R}^a = q_{0R}$;
- The transition relation is defined by (assuming nondeterministic automata)

$$(q_R^a)' \in \Delta(q_R^a, ((a_1, q_{1lep_1}, \dots, q_{1lep_k}, q_{1-lep_1}, \dots, q_{1-lep_k}, \mathbf{lep}_1, \mathbf{lep}_{F_1}, \rho_1), (a_2, q_{2lep_1}, \dots, q_{2lep_k}, q_{2-lep_1}, \dots, q_{2-lep_k}, \mathbf{lep}_2, \mathbf{lep}_{F_2}, \rho_2)))$$

if and only if

- for $1 \leq i \leq k$, $(q_R^a)' \in \Delta_R^a(q_R^a, (a_1, a_2))$ and $q_{2lep_i} \in \delta_{lep_i}(q_{1lep_i}, a_1)$, $q_{2-lep_i} \in \delta_{-lep_i}(q_{1-lep_i}, a_1)$,
- $\mathbf{lep}_1 = \mathbf{lep}_2$,
- if $\mathbf{lep}_{F_1} = LEP$, then $\mathbf{lep}_{F_2} = \emptyset$ and $\rho_2 = reset$, or $\mathbf{lep}_{F_2} = \mathbf{lep}_{F_1}$ and $\rho_2 = noreset$, otherwise, $\mathbf{lep}_{F_2} = \mathbf{lep}_{F_1} \cup \{lep_i \in \mathbf{lep}_1 \mid q_{lep_i} \in F_{lep_i}\} \cup \{lep_i \notin \mathbf{lep}_1 \mid q_{-lep_i} \in F_{-lep_i}\}$ and $\rho_2 = noreset$.

Note that at a given position, when all required accepting conditions have been satisfied, the choice to reset or not is nondeterministic, which makes it possible to wait until the required acceptance conditions have been satisfied at each position and then to reset everywhere simultaneously;

- The set of accepting states F_R^a is F_R .

The initial states of M_{-gsp}^a are those of the following form:

$$\begin{aligned} & (w(0), q_{0lep_1}, \dots, q_{0lep_k}, q_{0-lep_1}, \dots, q_{0-lep_k}, \mathbf{lep}_1, \emptyset, noreset) \\ & (w(1), q_{0lep_1}, \dots, q_{0lep_k}, q_{0-lep_1}, \dots, q_{0-lep_k}, \mathbf{lep}_2, \emptyset, noreset) \\ & \dots \\ & (w(n-1), q_{0lep_1}, \dots, q_{0lep_k}, q_{0-lep_1}, \dots, q_{0-lep_k}, \mathbf{lep}_n, \emptyset, noreset), \end{aligned}$$

where $w(0) \dots w(n-1) \in L(A_{S_0})$ and $\mathbf{lep}_1 \mathbf{lep}_2 \dots \mathbf{lep}_n \in \neg loSp$.

The accepting states in the language of the automaton F^a are those in which for every position the last part ρ of the label is *reset*, which implies that all relevant automata have seen an accepting state since the last “reset”.

Theorem 6.5. *The Büchi regular system M_{-loSp}^a has an accepting execution of the form $\pi^a = \pi^a(0)\pi^a(1)\dots$ if and only if the execution $\pi = w_0 w_1 w_2 \dots$ of M , where $w_i = \Pi_\Sigma(\pi^a(i))$ ($\forall i$), does not satisfy $\neg loSp$.*

Proof. Follows from the construction above. \square

The system M being locally-finite, M_{-losp}^a is also locally-finite. We thus have the following result that shows that checking the emptiness of M_{-losp}^a can be reduced to solving the regular reachability problems.

Proposition 6.6. *The Büchi regular system $M_{-losp}^a = (\Sigma^a, A_{S_0}^a, T_R^a, F^a)$ is empty if and only if*

$$L((T^a)^*(A_{S_0}^a) \cap F^a \cap \prod_{\neq 2}((T^a)^+ \cap T_{id})) = \emptyset.$$

Proof. Same as Proposition 4.5. \square

7. BOOLEAN COMBINATIONS AND MULTIPLE ALTERNATIONS FOR PARAMETRIC SYSTEMS

7.1. Boolean Combinations. It is easy to see that one can verify Boolean combinations of global and local-oriented system properties (each property being a literal). Indeed, any Boolean combination can be turned into another combination that only uses the connectors for the disjunction (\vee) and the negation (\neg). Properties being defined by finite-word and Büchi automata, one can always compute their negation. Verifying the disjunction of several properties is direct by definition.

7.2. Multiple Alternations for Parametric Systems. In some situations, it is also interesting to consider properties with *multiple alternations* between local-oriented and global system properties. By multiple alternations, we mean local-oriented properties that reference global system properties and vice-versa. We will not formally characterize the way alternations can occur, but rather illustrate the concept with several examples. Multiple-alternation properties will be specified by combining the notations introduced in Sections 4.1 and 6.1. The semantic of multiple-alternation properties easily follows from those notations.

We now propose several examples that illustrate how multiple-alternation properties can be reduced to properties with a simple alternation on an augmented system, a problem for which this paper provided verification procedures. We consider a parametric system, and assume that each of its processes can be in one of the two following states $\{C, T\}$. The following property is a local-oriented system property:

$$\forall i \square (C[i] \Rightarrow \diamond T[i]). \tag{7.1}$$

Indeed, we could think that this property is a local oriented system property. However, due to the presence of the \exists quantifier, $\square(C[i] \Rightarrow \diamond(\exists j \neq i)T[j])$ can reference several processes and is thus not a local execution property.

The solution we propose is to reduce the property above to a local execution property over an augmented system. This is done by introducing new Boolean variables in the specification of each process. Those variables can be arbitrarily true or false in any moment of an execution. Let us go back to our example and assume that we add to each process a Boolean variable “a” that behaves as described above. We use $a[i]$ to denote that the

variable a is true for the process i in the current state, and $\neg a[i]$ to denote that it is false⁷. In this case Property 7.9 can be rewritten as

$$\forall i \Box(C[i] \Rightarrow \Diamond a[i]) \wedge \quad (7.2)$$

$$\Box \forall i(a[i] \Leftrightarrow (\exists j \neq i)T[j]) \wedge \quad (7.3)$$

$$\Box \forall i(a[i] \vee \neg a[i]). \quad (7.4)$$

Clearly, $\phi_1 \equiv \forall i \Box(C[i] \Rightarrow \Diamond a[i])$ is a local-oriented system property, and $\phi_2 \equiv \Box \forall i(a[i] \Leftrightarrow (\exists j \neq i)T[j])$ and $\phi_3 \equiv \Box \forall i(a[i] \vee \neg a[i])$ are global system properties. In order to test whether a system M satisfies ϕ_1 , we simply test whether there exists an execution π of M that satisfies the negation of ϕ_1 , assuming that π satisfies both ϕ_2 and ϕ_3 . Concretely, ϕ_2 and ϕ_3 impose conditions on π .

We now give two other illustrating examples.

Example 7.1. Consider the following property:

$$\Box(\forall i \Diamond T[i] \wedge \exists j C[j]). \quad (7.5)$$

This property cannot be expressed neither by a local-oriented system property nor by a global system property. The solution is again to reduce the extended state property to a state property over an augmented system. We introduce a Boolean variable “a” that can be either true or false in each state. Using variable “a”, Property 7.5 can be rewritten as a conjunction of local-oriented and global system properties.

$$\Box(\forall i a[i] \wedge \exists j C[j]) \wedge \quad (7.6)$$

$$\forall i \Box(a[i] \Rightarrow \Diamond T[i]) \wedge \quad (7.7)$$

$$\Box \forall i(a[i] \vee \neg a[i]). \quad (7.8)$$

Of course, we can have several alternations in the same formula. In such situations, construction has to be applied for each alternation. Consider the following example.

Example 7.2. Consider the following property φ_1 :

$$\forall i \Box(C[i] \Rightarrow \Diamond(\exists j \neq i) \text{Buchi}_\varphi[j]), \quad (7.9)$$

where $\text{Buchi}[j]$ is a *Büchi modality* which is true if and only if the j -th process satisfies the local execution property φ described by the Büchi automaton Buchi_φ .

Property φ_1 cannot be expressed neither by a local-oriented system property nor by a global system property. The solution is to introduce two Boolean variables “a” and b . Using those variables, φ_1 can be rewritten as the property φ_2 defined as follows:

⁷When we add a Boolean variable, we extend the alphabet on which processes’s states are encoded. As an example, if the set of states was given by $\Sigma = \{C, T\}$ before the variable a is added, it becomes $\Sigma_{\{a\}} = \Sigma \times \{-a, a\} = \{(C, \neg a), (C, a), (T, \neg a), (T, a)\}$ after the addition occurs. As a consequence, any automaton defined over Σ must take this extension into account, which is done by duplicating each of its transitions. As an example, a transition labeled by T is duplicated into two transitions, one labeled by (T, a) and the other one by $(T, \neg a)$. To not lengthen the presentation, we will assume this translation to be implicit, and we write $a[i]$ for $(T, a)[i] \vee (C, a)[i]$ and T for $(T, a)[i] \vee (T, \neg a)[i]$.

$$\forall i \Box (C[i] \Rightarrow \Diamond a[i]) \wedge \quad (7.10)$$

$$\Box \forall i (a[i] \Leftrightarrow (\exists j \neq i) b[j]) \wedge \quad (7.11)$$

$$\forall i \Box (b[i] \Rightarrow \text{Buchi}_\varphi[i]) \wedge \quad (7.12)$$

$$\Box \forall i (a[i] \vee \neg a[i]) \wedge \quad (7.13)$$

$$\Box \forall i (b[i] \vee \neg b[i]). \quad (7.14)$$

By observing that $\forall i \Box (b[i] \Rightarrow \text{Buchi}_\varphi[i])$ is a local-oriented property (The set of executions that satisfy b can easily be described with a Büchi automaton), we conclude that φ_2 is a Boolean combination of local-oriented and global system properties.

There are also alternations that we have not been able to handle. As an example, we cannot treat a property that has two free-variables or a second order variable under the scope of a temporal LTL operator. Such an observation was made for a similar logic in [AJN⁺04, AJNS04].

8. RELATED WORK ON VERIFYING TEMPORAL PROPERTIES IN (ω -)REGULAR MODEL CHECKING

The problem of verifying linear temporal properties in the framework of regular model checking has been first addressed in [BJNT00, PS00, Sha01]. However, the treatment of this problem in these papers was preliminary and somewhat adhoc for very particular kinds of properties of parametric systems.

In [AJN⁺04, AJNS04], Abdulla et al. independently⁸ proposed an approach based on a specification logic called LTL(MSO), which combines the monadic second order logic MSO and the linear-time temporal logic LTL. Properties written in the LTL(MSO) logic are local-oriented system properties, where the local system properties are LTL properties that can make assumptions on the executions of the other processes up to some restrictions. The work in [AJN⁺04, AJNS04] has been implemented as an extension of the RMC toolset [RMC]. This implementation has been shown to be very efficient when being applied to several parametric systems (see [Nil05, Sak08] for many experimental results). On the other hand, we observed that the LTL(MSO) logic has mainly be designed for parametric systems and is not suited (and sometimes not powerful enough) to express very simple properties of many other interesting classes of systems such as systems with integer variables (when considering a non-unary encoding). The verification procedure in [AJN⁺04, AJNS04] is only dedicated to regular systems that are locally-finite and the ω -regular framework is not considered. Finally, unlike our local-oriented properties, the LTL(MSO) logic cannot be used to express properties which are Boolean combinations of properties written in logics that are more expressive/concise than LTL (e.g. PTL [GO03, LPZ85], ETL [Wol82], or μ TL [Var88]).

In [VSVA05], Agha et al. proposed to use learning-based algorithms [Ang87] to verify global system properties of regular systems. The technique they proposed relies on the computation of several fixed point operators which are used to test whether a Büchi regular system is empty or not. The technique has been implemented in the LEVER toolset [VV06] and applied to several case studies. The experimental results, which are presented in [Var06], are quite impressive. On the other hand, the use of learning algorithms to make fixed point

⁸The approach in [AJN⁺04, AJNS04] has been proposed in the same period of time as our early work [BLW04b], whose present paper is an extension.

computation terminating requires to enrich the systems with two extra variables. This is a clear restriction since it is known that there are many systems for which the set of reachable states is regular before the variables have been introduced, but not after. The work in [VSVA05] also lacks of a clear description of the encoding of linear temporal properties in the regular framework, which is one of the main contribution of our work. Finally, we mention that [VSVA05] does not consider the ω -regular framework.

9. CONCLUSION AND FUTURE WORK

We have presented a general framework for specifying and verifying a large class of temporal properties for systems represented in the (ω) -regular model checking framework. The verification techniques we provide are based on reductions to automata-based techniques for computing the limit of an infinite sequence of automata. We plan to implement our results in the T(0)RMC toolset [T(O)].

Aside from the implementation point of view, we want to pursue several research directions. One of those directions is to extend our results to the verification of computational tree logics properties. It would also be of interest to propose criteria to check whether the extrapolation of the simulation with the technique of [BLW03, BLW04a, Leg07] is precise. Developing a methodology to decide whether FIFO-Queue and pushdown systems are locally-finite is another topic of interests. We would also like to give a formal characterization of what are the allowed alternations between local-oriented and global system properties. This will have to be done with the help of a logic. As a first step, we observe that we can easily extend the LTL(MSO) logic of [AJN⁺04, AJNS04] with Büchi modalities. Finally, adapting our concept of local-oriented system property to the frameworks introduced in [ALRd06, ADR07] is another promising direction for future research.

ACKNOWLEDGEMENT

We thank Julien d’Orso, Marcus Nilsson, and Mayank Saksena for answering many email questions on their work.

REFERENCES

- [ABJN99] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parameterized system verification. In *Proc. 11th Int. Conference on Computer Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 1999.
- [ACH⁺95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
- [ADR07] P. A. Abdulla, G. Delzanno, and A. Rezzina. Parameterized verification of infinite-state processes with global conditions. In *Proc. 19th Int. Conference on Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2007.
- [AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, June 1996.
- [AJMd02] P. A. Abdulla, B. Jonsson, P. Mahata, and J. d’Orso. Regular tree model checking. In *Proc. 14th Int. Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 555–568. Springer, 2002.
- [AJN⁺04] P. A. Abdulla, B. Jonsson, M. Nilsson, J. d’Orso, and M. Saksena. Regular model checking for ltl(mso). In *Proc. 16th Int. Conference on Computer Aided Verification (CAV)*, volume 3114 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2004.

- [AJNd03] P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d’Orso. Algorithmic improvements in regular model checking. In *Proc. 15th Int. Conference on Computer Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 236–248. Springer, 2003.
- [AJNS04] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *Proc. 15th Int. Conference on Concurrency Theory (CONCUR)*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.
- [ALRd06] P. A. Abdulla, A. Legay, A. Rezzine, and J. d’Orso. Tree regular model checking: A simulation-based approach. *Journal of Logic and Algebraic Programming*, 2006.
- [Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th Int. Conference on Concurrency Theory (CONCUR)*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, July 1997.
- [BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds (extended abstract). In *Proc. 8th Int. Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1996.
- [BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo channel systems with non-regular sets of configurations (extended abstract). In *Proc. 24th Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, pages 560–570. Springer, 1997.
- [BHJ03] B. Boigelot, F. Herbreteau, and S. Jodogne. Hybrid acceleration using real vector automata (extended abstract). In *Proc. 15th Int. Conference on Computer Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 193–205. Springer, 2003.
- [BHMV05] A. Bouajjani, P. Habermehl, P. Moro, and T. Vojnar. 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)*, volume 3440 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2005.
- [BHRV06] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking of complex dynamic data structures. In *Proc. 13th Int. Symposium on Static Analysis (SAS)*, volume 4134 of *Lecture Notes in Computer Science*, pages 52–70. Springer, 2006.
- [BHV04] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. 16th Int. Conference on Computer Aided Verification (CAV)*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.
- [BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proc. 12th Int. Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer-Verlag, 2000.
- [BJW01] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. Int. Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Siena, Italy, June 2001. Springer-Verlag.
- [BJW05] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic*, 6(3):614–633, 2005.
- [BLW03] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *Proc. 15th Int. Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science*, pages 223–235. Springer, 2003.
- [BLW04a] B. Boigelot, A. Legay, and P. Wolper. Omega-regular model checking. In *Proc. 10th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 561–575. Springer, 2004.
- [BLW04b] A. Bouajjani, A. Legay, and P. Wolper. Handling liveness properties in (ω -)regular model checking. In *Proc. 6th Int. Workshop on Verification of Infinite State Systems (INFINITY)*, volume 138(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2004.
- [Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. Collection des publications de la Faculté des Sciences Appliquées de l’Université de Liège, Liège, Belgium, 1999.

- [BT02] A. Bouajjani and T. Touili. Extrapolating tree transformations. In *Proc. 14th Int. Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2002.
- [BW02] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In *Proc. 18th Int. Conference on Logic Programming (ICLP)*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2002.
- [DLS02] D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *Journal of Logic and Algebraic Programming (JLAP)*, 52-53:109–127, 2002.
- [FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY)*, volume 9 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1997.
- [GO03] P. Gastin and D. Oddoux. Ltl with past and two-way very-weak alternating automata. In *Proc. Mathematical Foundations of Computer Science 2003, 28th International Symposium (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 439–448. Springer, 2003.
- [KMM⁺97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proc. 9th Int. Conference on Computer Aided Verification (CAV)*, volume 1254 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 1997.
- [KPSZ02] Y. Kesten, A. Pnueli, E. Shahar, and L. D. Zuck. Network invariants in action. In *Proc. 13th Int. Conference on Concurrency Theory (CONCUR)*, volume 2421 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2002.
- [Leg07] A. Legay. *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, 2007. to appear.
- [LPZ85] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proc. Int. Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.
- [MSS86] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*, pages 275–283, Rennes, 1986. Springer-Verlag.
- [Nil01] M. Nilsson. Regular model checking. Master’s thesis, Uppsala University, 2001.
- [Nil05] M. Nilsson. *Regular Model Checking*. PhD thesis, Uppsala University, 2005.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
- [PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Proc. 12th Int. Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2000.
- [RMC] The regular model checking tool (RMC). Available at <http://www.it.uu.se/research/docs/fm/apv/rmc>.
- [Saf92] S. Safra. Exponential determinization for ω -automata with strong-fairness acceptance condition. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, Victoria, May 1992.
- [Sak08] M. Saksena. *Verifying Absence of infinite Loops in Parameterized Protocols*. PhD thesis, Uppsala University, 2008.
- [Sha01] E. Shahar. *Tools and Techniques for Verifying Parametrized Systems*. PhD thesis, Weizmann Institute of Science, 2001.
- [Td06] T. Touili and J. d’Orso. Regular hedge model checking. In *Proc. 4th Int. IFIP Conference on Theoretical Computer Science (TCS06)*, 2006.
- [T(O)] The T(O)RMC toolset. Available at <http://www.montefiore.ulg.ac.be/~legay/TORMC/index-tormc.html>.
- [Var88] M. Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th Int. Symposium on Principles of Programming Languages (POPL)*, pages 250–259. ACM, 1988.
- [Var06] A. Vardhan. *Learning to Verify Systems*. PhD thesis, University of Illinois, 2006.
- [Var07] M. Y. Vardi. From church and prior to psl, 2007. Available at <http://www.cs.rice.edu/~vardi/papers/index.html>.

- [VSVA05] A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. 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)*, volume 3440 of *Lecture Notes in Computer Science*, pages 45–60. Springer, 2005.
- [VV06] A. Vardhan and M. Viswanathan. Lever: A tool for learning based verification. In *Proc. 18th Int. Conference on Computer Aided Verification (CAV)*, volume 4144 of *Lecture Notes in Computer Science*, pages 471–474. Springer, 2006.
- [VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. 2nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 332–344. IEEE Computer Society, 1986.
- [WB95] P. Wolper and B. Boigelot. An automata-theoretic approach to presburger arithmetic constraints (extended abstract). In *Proc. 2nd Int. Symposium on Static Analysis (SAS)*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 1995.
- [WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th Int. Conference on Computer Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 88–97. Springer-Verlag, 1998.
- [WB00] P. Wolper and B. Boigelot. 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)*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.
- [Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982.