Handling Liveness Properties in (ω-)Regular Model Checking

Ahmed Bouajjani [1]

LIAFA
University of Paris 7,
Paris, France

Axel Legay, Pierre Wolper [2,3,4]

Institut Montefiore
Université de Liège,
Liège, Belgium

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 liveness properties could also be checked within this framework, little has been done about working out the corresponding details, and experimentally evaluating the approach. This paper addresses these issues 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 solves the problem resulting from the fact that infinite computations of unbounded configurations might never contain the same configuration twice, thus making cycle detection problematic. Several experiments showing the applicability of the approach were successfully conducted.

Key words: ($\omega-$)Regular Model Checking, Liveness.

1 Introduction

Regular model-checking [3,5,6,9,14] is a general approach to analyzing infinite-state systems in which states are represented by words, the transition relation is represented by a finite-state transducer, and reachable states are

[1] Email: abou@liafa.jussieu.fr

[2] Email: legay@montefiore.ulg.ac.be

[3] Axel Legay is supported by a F.R.I.A grant.

[4] Email: pw@montefiore.ulg.ac.be

computed by iterating this transducer with the help of appropriate acceleration techniques. 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 safety properties model checking can be reduced to a state reachability problem, but for properties that include a liveness component, the best that can be done is to reduce the (linear-time) model-checking problem to emptiness of a Büchi automaton [10], which means checking for repeated reachability rather than reachability. As already shown in [3,16], this is conceptually possible in the context of regular model checking (when the considered transducers represent length-preserving transformations of finite words), but the corresponding details and pragmatics have, so far, not been adequately addressed. Doing so is one of the objectives of this paper. Another objective of the paper is to provide a general specification framework and generic analysis techniques covering the case of finite-word configurations (which correspond to a variety of models such as pushdown systems, FIFO-channel systems, parametric networks of identical processes, and even integer counter systems) as well as the case of infinite-word configurations (which allows for instance to reason about timed or hybrid systems manipulating real-valued variables).

For an infinite-state system whose states are represented by finite (or even infinite [6]) words, a computation is an infinite sequence of such words. To define a property of such a computation, one has the choice between moving within a configuration (horizontally) or between configurations (vertically). One thus naturally thinks of a two-dimensional logic to describe properties of such computations. 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 computation properties. On the computations we are considering, word automata move either horizontally or vertically and clearly both are needed to define meaningful properties. One could consider arbitrary alternation between both directions, but in practice, one alternation is sufficient. Though generalization is possible, we thus limited our study to horizontal properties defined in terms of vertical ones, which make sense for parametric systems in which a vertical slice corresponds to the computation of one component of the system; as well as to vertical properties defined in terms of horizontal ones, which makes sense for systems where words are used to encode a queue content or the value of an integer [5,6]. 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 configurations, and finally checking for the reachability of configurations appearing in such cycles. When dealing with systems whose configurations are finite words and whose transition relation is length-preserving, an accepting computation of the Büchi automaton will always contain the same configuration twice and hence an identifiable cycle. However, when dealing with configurations whose length can grow or that are infinite, there might very well be an accepting computation of the Büchi automaton in which the same configuration never appears twice.

To cope with this, we look for configurations that are not necessarily identical, but such that one entails the other in the sense that any computation 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 configurations of the system.

The nice twist is that the computation of the symbolic representation of the simulation relation is, in fact, the computation of the limit of a sequence of finite-state transducers, for which the acceleration techniques introduced in the context of regular model-checking can also be used. However, in several cases we have considered, 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 configurations in a finite number of classes, and hence that existing accepting computations will necessarily be found, which might not be the case when the number of simulation equivalence classes is infinite.

Finally, we conducted a number of experiments to establish the feasibility of automatically verifying liveness properties of infinite-state systems in the purely automata-theoretic framework of regular model-checking. Liveness properties of parametric systems, of programs using integer variables, and of hybrid systems were successfully checked.

Related works: There exists a variety of earlier work on the verification of liveness properties for infinite-state systems. In [8,7,17], methods based on combining abstraction techniques and finite-state model-checking are proposed for the verification of liveness properties of parametric networks of identical processes. In contrast with these methods, our approach is not limited to the case of parametric networks.

Very recently, Abdulla et al. developed independently an approach similar to ours based on a specification logic combining S1S and linear-time temporal logic [1]. The techniques they propose are however different and are only applicable in the case of parametric systems. In fact, the logic they develop can only express properties of parametric systems and cannot express for instance global properties of infinite-state systems such as counter systems (an example of such properties is given in Example 4.2). Moreover, their techniques assume length-preserving systems, and they did not address neither the case of non-length preserving ones (such as push-down systems, FIFO-channel systems, counter systems, etc) nor the case of ω-regular model checking, and therefore they cannot deal for instance with timed or hybrid systems as we can do.

Other results in the literature are based on automatic techniques for the synthesis of ranking functions. These results address mainly the problem of checking termination of some classes of (infinite-state) programs / extended automata. [11,12]. The proposed techniques exploit the specific nature of the considered data domains, which are mainly numerical data domains such as integer variables with linear tests and updates. While these methods can be more efficient in particular cases, the aim of our work is to provide generic techniques which are applicable regardless of the types of the variables and data structures being used.

2 Preliminaries

In this section, we briefly recall the basic automata-theoretic definitions that will be used in this paper.

A finite-state automaton on finite words is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite alphabet, Q is a set of *states*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a *transition function* ($\delta : Q \times \Sigma \rightarrow Q$ if the automaton is deterministic), and $F \subseteq Q$ is a set of *accepting* states. A triple (s, a, s') such that $s' \in \delta(s, a)$ is said to be a transition labeled by a. A finite sequence (word) $w = a_1a_2 \ldots a_k$ of elements of Σ is *accepted* by the automaton A if there exists a sequence of states $s_0 \ldots s_k$ such that $\forall 1 \leq i \leq k : \; s_i \in \delta(s_{i-1}, a_i)$ ($s_i = \delta(s_{i-1}, a_i)$ for a deterministic automaton), $s_0 = q_0$, and $s_k \in F$. The set of words accepted by A is called the *language accepted by* A, and is denoted by $L(A)$.

An *infinite word* (or *ω-word*) w over an alphabet Σ is a mapping from the natural numbers to Σ. The set of infinite words over Σ is denoted Σ^ω. A *Büchi automaton* is syntactically identical to a finite-word automaton. A *run* π of a Büchi automaton $A = (\Sigma, Q, q_0, \delta, F)$ on an ω-word w is a mapping $\pi : \mathbb{N} \rightarrow Q$ such that $\pi(0) = q_0$, and for all $i \geq 0$, $\pi(i{+}1) \in \delta(\pi(i), w(i))$ (nondeterministic automaton) or $\pi(i + 1) = \delta(\pi(i), w(i))$ (deterministic automaton).

Let $\mathit{inf}(\pi)$ denote the set of states that occur infinitely often in a run π. A run π is said to be *accepting* if $\mathit{inf}(\pi) \cap F \neq \emptyset$. An ω-word w is *accepted* by a Büchi automaton if that automaton admits at least one accepting run on w. The language $L_\omega(A)$ *accepted* by a Büchi automaton A is the set of ω-words it accepts. A language $L \subseteq \Sigma^\omega$ is *ω-regular* if it can be accepted by a Büchi automaton. 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. We will thus restrict ourselves to *weak* automata [15].

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

Not all omega-regular languages can be accepted by weak deterministic Büchi automata, nor even by weak nondeterministic automata, but they are sufficient for handling many applications. In particular they are as expressive as the first-order linear arithmetics of integers and reals [4], which allows to deal for instance with models such as timed automata, linear hybrid automata, as well as their extensions with integer counters [6]. Furthermore, there are algorithmic advantages to working with weak automata: weak deterministic automata can be directly complemented by inverting their accepting and non-accepting states; and there exists a simple determinization procedure for weak automata, which produces Büchi automata that are deterministic, but generally not weak. Nevertheless, if the represented language can be accepted by a weak deterministic automaton, the result of the determinization procedure can easily transformed into a weak automaton [4].

3 Systems models, Regular and ω-Regular Model Checking

In this section, we present the automata based encoding of systems used in this paper. We adopt the concepts of regular model checking ([3]), representing system configurations by finite or infinite (see [6]) words. Precisely, a *system* is defined to be a triple $M = (\Sigma, \phi_I, R)$ where

- Σ is a finite alphabet, over which the system configurations are encoded as finite (infinite) words;
- ϕ_I is a set of initial configurations represented by a finite (ω-)automaton over Σ;
- R is a transition relation represented by a finite-state (ω-)automaton over $\Sigma \times \Sigma$, which will be referred to as a *transducer* over Σ. Note that with this definition of a transducer, the configurations of an execution of a *length-preserving*. This is less restrictive that might appear since initial configurations can always be arbitrarily padded and one can work with a set of initial configurations that contains all possible paddings; however, this coding technique has an impact on the verification of liveness properties (see Section 6).

In the finite-word case, an execution of the system is an infinite sequence of same-length finite words over Σ. This model has often been used to represent parametric systems [3] or systems with integer variables ([5]).

Example 3.1 Let us consider a simple example of parametric network of identical processes implementing a token passing algorithm. Each process can be in one of the two states T (has the token) or N (does not have the token), and an action of passing the token from left to right can be encoded using the regular relation $((T,T)+(N,N))^*(T,N)(N,T)((T,T)+(N,N))^*$.

In the infinite-word case (ω-regular model checking [6]), an execution of the system is an infinite sequence of infinite words over Σ. This model can be used for systems involving integer and real variables, such as hybrid systems. When dealing with infinite word configurations, we will restrict transducers to be weak deterministic Büchi automata as is done in [6].

So far, work on ($\omega-$)regular model checking has focused on two problems: computing the transitive closure R^* of the relation R, and computing the image $R^*(\phi)$ of a given initial set of states ϕ. Here, we will assume that we have a technique for computing both R^* and $R^*(\phi)$ (see [3,5,6,9] for examples of such techniques) and we will show how the verification of liveness properties can be reduced to these problems.

4 System Properties

In this section, we consider the definition of properties we want to verify. We consider two classes of properties. The first class examines computations of the global system. This class of properties can be used for expressing properties on the configurations of systems such as pushdown systems, FIFO-channel systems, counter systems, hybrid systems, etc. The second class is oriented towards parametric systems and examines first the computations of the individual processes of the system. Boolean combinations of properties in the two classes of properties can also be considered. These combinations are typically useful in expressing liveness properties under fairness conditions.

4.1 Global System Properties

If configurations are looked at as a whole — which is the only reasonable possibility when they represent for instance numbers (integer or reals), stack or queue contents, etc— it makes sense to define properties of executions in terms of properties of configurations.

Definition 4.1 Let $M=(\Sigma,\phi_I,R)$ be a system, a *configuration property* is a set $cop \subseteq \Sigma^*$ (resp. $cop \subseteq \Sigma^\omega$ when considering infinite-words) . Given a set of configuration properties $COP=\{cop_1,\ldots,cop_k\}$, a *global system property* is a set $gsp \subseteq (2^{COP})^\omega$, i.e. a set of infinite sequences of subsets of COP. An execution $\sigma = w_0,w_1,w_2,w_3\ldots$ satisfies a global system property gsp, $\sigma \models gsp$, if $\mathbf{cop}(w_0)\mathbf{cop}(w_1)\ldots \in gsp$, where $\mathbf{cop}(w)=\{cop_i \in COP \mid w \models cop_i\}$.

We will consider global system properties that are defined by Büchi automata and configuration properties expressed by finite-word automata. This model captures all the properties that are expressible in linear-time temporal logic, using configuration properties as propositions.

Example 4.2 Consider a system that manipulate two integer variables: x and y. The following property $\Box[(x > 0) \Rightarrow \Diamond(y = 5)]$ is a global system property.

4.2 Local-oriented System Properties

These properties can only be checked on parametric systems, they are used in order to express liveness properties of individual processes of such systems. In our model, a computation of a parametric system is represented by an infinite sequence of identical length finite words. Each position in these words corresponds to a process and the infinite sequences of identically positioned letters in an execution represents a process execution. We thus use the following notation and definitions.

Definition 4.3 Consider an execution $\sigma = w_0, w_1, w_2, w_3 \ldots$ of a system $M = (\Sigma, \phi_I, R)$. The jth local projection $\Pi_j(\sigma)$ is the infinite word $w_0(j)w_1(j)w_2(j)\ldots$, where $w(j)$ represents the jth letter of the work w.

Definition 4.4 A *local execution property* is a set $\ell ep \subseteq \Sigma^\omega$. A local execution property ℓep is *satisfied by an execution σ at position j*, $\Pi_j(\sigma) \models \ell ep$, if $\Pi_j(\sigma) \in \ell ep$.

Local execution properties can naturally be defined by using a linear-temporal logic, but we will assume that logic-expressed properties have been translated to automata [10].

Definition 4.5 Given a set of local execution properties $LEP = \{\ell ep_1, \ldots \ell ep_k\}$, a *local-oriented system property* is a set $\ell osp \subseteq (2^{LEP})^*$, i.e. a set of finite sequences of subsets of LEP. An execution σ satisfies a local-oriented system property ℓosp if $\mathbf{lep}(\Pi_1(\sigma))\mathbf{lep}(\Pi_2(\sigma))\ldots\mathbf{lep}(\Pi_n(\sigma)) \in \ell osp$, where n is the common length of the words in σ, and $\mathbf{lep}(\Pi_i(\sigma)) = \{\ell ep_i \in LEP \mid \Pi_i(\sigma) \models \ell ep_i\}$.

Example 4.6 Consider the parametric system defined in Example 3.1. The fact that whenever a process is in state N, it will eventually move to state T ($\Box(N \Rightarrow \Diamond T)$ in linear-time temporal logic) is a local execution property. That this property holds for each process is then a local-oriented system property.

4.3 Boolean Combinations of Local-Oriented System and Global System Properties

Liveness properties of systems need sometimes be expressed as Boolean combinations of local-oriented/global system properties. Typically, in parametric systems, this is the case for properties corresponding to the pattern: under some fairness conditions some liveness requirement must hold. In the case of other types of system, such as systems manipulating sequential data structures (pushdown systems, FIFO-channel systems, programs with linked lists, arrays, etc) or numerical variables (integer counters, real-valued clocks, stopwatches, etc), both fairness conditions and liveness requirements are global system properties since local-oriented properties are not meaningful in these cases.

5 Checking Properties of length-preserving systems

In this section, we will describe how we verify global and local-oriented system properties on length-preserving systems. Due to space limitations, the verification of boolean combination is only described in the full version of the paper [13].

5.1 Checking Global System Properties

To check that a length-preserving system $M = (\Sigma, I, R)$ satisfies a global system property *gsp* defined over a set of configuration properties $COP = \{cop_1, \ldots cop_k\}$, we check for the absence of executions of M that do not satisfy *gsp*. This is done by augmenting the transition system M in such a way that its executions are only those that are runs of the automaton defining $A_{\neg gsp}$. The augmented transition system is defined as $M_a = (\Sigma_a, I_a, R_a)$. M_a is obtained by taking a "special" product between the initial system M and the automaton $A_{\neg gsp}$. There are a lot of technical points in this construction, and due to space limitation, we have deferred them to the full version ([13]).

After constructing M_a, the next step is to check whether there is a run of the transition system M_a that is accepting for the automaton $A_{\neg gsp}$. This is done by checking whether there is an accepting configuration (i.e. a configuration in where the automaton $A_{\neg gsp}$ is in an accepting state), nontrivially reachable from itself, and reachable from an initial configuration. This condition is indeed necessary and sufficient because the system is length-preserving, which means that one cannot find an infinite path that never visits the same configuration twice.

The computation checking the condition above can be organized as follow. Let *accept* be the set of accepting configurations (i.e configurations of the augmented system in where $A_{\neg gsp}$ is in an accepting state), let R_a^+ be the non-reflexive transitive closure of R_a and Id the identity relation. Then augmented configurations from which there exists a nontrivial loop are those in the domain of $R_a^+ \cap Id$ (with $R^+ = R^* \circ R$). Such reachable accepting configurations are thus those in

$$R_a^*(I_a) \cap accept \cap domain(R_a^+ \cap Id),$$

and the property is satisfied iff this set if empty.

5.2 Checking Local-oriented System Properties

Checking that a system $M = (\Sigma, I, R)$ satisfies a local-oriented system property ℓosp defined over a set of local execution properties $LEP = \{\ell ep_1, \ldots \ell ep_k\}$, is done by searching for an execution of the system that satisfies the negation $\neg \ell osp$ of the property. We proceed by augmenting the system M into a system $M_a = (\Sigma_a, I_a, R_a)$.

Let $T_R = (\Sigma \times \Sigma, S_R, s_{0R}, \delta_R, F_R)$ be the finite automaton defining the transition relation R of M, $A_{\neg \ell osp} = (2^{LEP}, S_{\neg \ell osp}, s_{0 \neg \ell osp}, \delta_{\neg \ell osp}, F_{\neg \ell osp})$ be the finite-word automaton accepting the finite sequences that do not satisfy ℓosp, $A_{\ell ep_i} = (\Sigma, S_{\ell ep_i}, s_{0 \ell ep_i}, \delta_{\ell ep_i}, F_{\ell ep_i})$ for $1 \leq i \leq k$ be the complete (but not necessary deterministic) Büchi infinite-word automata defining the local execution properties, and $A_{\neg \ell ep_i} = (\Sigma, S_{\neg \ell ep_i}, s_{0 \neg \ell ep_i}, \delta_{\neg \ell ep_i}, F_{\neg \ell ep_i})$ automata for the negation of these properties. The latter are needed since, the automata $A_{\ell ep_i}$ being nondeterministic, the fact that they have a nonaccepting computation does not indicate that the corresponding property does not hold.

Since, *a priori*, we do not know which local execution property will be satisfied at which position of the configuration, each of the automata $A_{\ell ep_i}$ and $A_{\neg \ell ep_i}$ has to be run at each position. So, we need to extend our alphabet in such a way that each position in the configuration is also labeled by a state of each of the $A_{\ell ep_i}$ and $A_{\neg \ell ep_i}$. Furthermore, for each position in configurations, 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 position by an element of 2^{LEP} corresponding exactly to the properties ℓep_i that are satisfied. This labeling will remain unchanged from configuration to configuration and will enable us to run the automaton $A_{\neg \ell osp}$. The next step is to check whether there is a run of the transition system M_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 configuration, 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 are thus faced 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_a

The augmented alphabet is

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

Two subsets of LEP are introduced in the alphabet: the second is 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 of not. The augmented transducer,

T_{R_a}, can then be defined as follows.

- Its alphabet is $\Sigma_a \times \Sigma_a$
- Its set of states and accepting states are respectively $S_{R_a} = S_R$ and $F_{R_a} = F_R$, its initial state is $s_{0R_a} = s_{0R}$.
- The transition relation is defined by (assuming nondeterministic automata)

$$s'_{R_a} \in \delta(s_{R_a}, ((a_1, s_{\ell ep_1}1, \ldots, s_{\ell ep_k}1, s_{\neg\ell ep_1}1, \ldots, s_{\neg\ell ep_k}1, \mathbf{lep}_1, \mathbf{lep}_{F1}, \rho_1), \\ (a_2, s_{\ell ep_1}2, \ldots, s_{\ell ep_k}2, s_{\neg\ell ep_1}2, \ldots, s_{\neg\ell ep_k}2, \mathbf{lep}_2, \mathbf{lep}_{F2}, \rho_2)))$$

  iff
  · $s'_{R_a} \in \delta_R(s_{R_a}, (a_1, a_2))$ and $s_{\ell ep_i}2 \in \delta_{\ell ep_i}(s_{\ell ep_i}1, a_1)$, $s_{\neg\ell ep_i}2 \in \delta_{\neg\ell ep_i}(s_{\neg\ell ep_i}1, a_1)$, for $1 \leq i \leq k$,
  · $\mathbf{lep}_1 = \mathbf{lep}_2$,
  · if $\mathbf{lep}_{F1} = LEP$, then $\mathbf{lep}_{F2} = \emptyset$ and $\rho_2 = reset$, or $\mathbf{lep}_{F2} = \mathbf{lep}_{F1}$ and $\rho_2 = noreset$, otherwise, $\mathbf{lep}_{F2} = \mathbf{lep}_{F1} \cup \{\ell ep_i \in \mathbf{lep}_1 \mid s_{\ell ep_i}1 \in F_{\ell ep_i}\} \cup \{\ell ep_i \notin \mathbf{lep}_1 \mid s_{\neg\ell ep_i}1 \in F_{\neg\ell ep_i}\}$ 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 [5].
- The set of accepting states is F_R.

The set of initial configurations of M_a are those of the form

$$(a_1, s_0\ell ep_1, \ldots, s_0\ell ep_k, s_0\neg\ell ep_1, \ldots, s_0\neg\ell ep_k, \mathbf{lep}_1, \emptyset, noreset)$$
$$(a_2, s_0\ell ep_1, \ldots, s_0\ell ep_k, s_0\neg\ell ep_1, \ldots, s_0\neg\ell ep_k, \mathbf{lep}_2, \emptyset, noreset)$$
$$\ldots$$
$$(a_n, s_0\ell ep_1, \ldots, s_0\ell ep_k, s_0\neg\ell ep_1, \ldots, s_0\neg\ell ep_k, \mathbf{lep}_n, \emptyset, noreset),$$

where $w = a_1a_2a_3\ldots a_n$ is a word in I, and $\mathbf{lep}_1\mathbf{lep}_2\ldots\mathbf{lep}_n \models \neg\ell osp$.

If we define accepting configurations to be those in which for every position the last part ρ of the label is *reset* [6], checking for the existence of an accepting execution can be done by checking if

$$R_a^*(I_a) \cap accept \cap domain(R_a^+ \cap Id),$$

is empty. In this case, the property is satisfied, else it is not.

6 Checking Properties of Non Length-Preserving Systems and Infinite-Words

In this section, we consider the problem of checking global system properties for finite-word systems which are not length-preserving and for infinite-word

[5] This makes it possible to wait until the required acceptance conditions have been satisfied at each position and then to reset everywhere simultaneously

[6] which implies that all relevant automata have seen an accepting state since the last "reset"

systems. As mentioned in Section 3, for the purpose of computing reachable configurations, non length-preserving systems can be handled as length-preserving ones by the use of padding. We can thus still use the constructions of Section 5.1 for obtaining an augmented system M_a that checks for a global system property. However, it is no longer true that an infinite computation will always repeatedly visit the same configurations, and we have to adapt the criterion given in Section 5.1. For infinite words, the situation is similar: the construction stays basically the same, though we have to deal with some additional technical difficulties due to the fact that configurations are infinite (see ([13] for details) and we also have to adapt the criterion that checks for loops.

Since we cannot reduce the problem of deciding if M_a has an infinite accepting computation to the problem of finding reachable accepting loops, our approach is to search for reachable configurations c from which it is possible to nontrivially reach some configuration c' such that (1) the path from c to c' visits a repeating state of $A_{\neg gsp}$, and (2) c' has at least the same computation paths as c. To check the condition (2), we actually check for a stronger condition which is the fact that c' must *simulates* c. In what follows, we will only consider infinite-words, but these results can easily be transposed to the finite-word case.

Definition 6.1 The greatest simulation relation over configurations of M_a which is compatible with the configuration properties in a set COP is the relation S defined as the limit of the following decreasing sequence of relations, where $w|_\Sigma$ denotes the projection of the word $w \in \Sigma_a$ over the alphabet Σ.

$$S_0 = \{(w_1, w_2) \in \Sigma_a^\omega \times \Sigma_a^\omega \ \mid \mathbf{cop}(w_1|_\Sigma) = \mathbf{cop}(w_2|_\Sigma)\}$$
$$S_{k+1} = \{(w_1, w_2) \in S_k \ : \ \forall w_1'.((w_1, w_1') \in R_a \Rightarrow \exists w_2'.(w_2, w_2') \in R_a \wedge (w_1', w_2') \in S_k)\}$$

The greatest simulation equivalence over M_a which is compatible with COP is the relation $\widetilde{S} = S \cap S^{-1}$.

First, we have the following result:

Proposition 6.2 *Let* accept *be the set of all augmented configurations where the automaton $A_{\neg gsp}$ is in some accepting state. Then, it can be seen that M_a has an accepting infinite computation if the following condition holds:*

$$R_a^*(I_a) \cap domain[((R_a^* \cap (\Sigma_a^\omega \times accept)) \circ R_a^+) \cap S] \neq \emptyset \tag{1}$$

The problem now is to compute the relation S. Observe that S_0 can be defined straightforwardly as a regular relation and that S_{k+1}, for every $k \geq 0$, is defined in terms of the relations R_a and S_k using boolean operations and projection (corresponding to existential quantification). Therefore, given transducers representing R_a and S_k, it is possible to compute effectively a transducer representing S_{k+1}. The main issue is whether the iterative computation of S terminates.

If the computation terminate then S has a finite-index simulation, i.e., a finite number of equivalence classes. This means that each infinite path of the system must visit infinitely often some of the equivalence classes. Therefore, we have the following result (which is detailed in the full version [13])).

Theorem 6.3 *Assume that the system M_a has a finite-index simulation. Then, M_a has an accepting infinite computation if and only if the condition (1) holds.*

In case M_a does not have a finite-index simulation, we can use approximations of S. Let us consider first the case of upper-approximations.

Proposition 6.4 *If there exists some $k \geq 0$, such that*

$$R_a^*(I_a) \cap \mathit{domain}[((R^* \cap (\Sigma_a^\omega \times \mathit{accept})) \circ R_a^+) \cap S_k] = \emptyset$$

then the system M_a has no infinite accepting computation, which means that M_a satisfies the property gsp.

Lower-approximations can also be useful to decide if the system does not satisfy a property.

Proposition 6.5 *Let $L \subseteq S$. Checking that*

$$R_a^*(I_a) \cap \mathit{domain}[((R^* \cap (\Sigma_a^\omega \times \mathit{accept})) \circ R_a^+) \cap L] \neq \emptyset$$

allows us to deduce that the system M_a has an infinite accepting computation, which means that M_a does not satisfy the property gsp.

To compute a lower-approximation of S, we proceed as follows: Instead of computing the decreasing sequence of relations $(S_i)_{i\geq 0}$, we compute the *increasing* sequence of their negations $(\neg S_i)_{i\geq 0}$. The advantage of doing that is that we can apply at each step of the iterative computation *widening* techniques such as those defined in [6] which allows us to speed up the fixpoint computation and, in many cases, to make it terminate. Then, the computed sequence of relations is actually an increasing sequence of ω-regular relations $(U_i)_{i\geq 0}$ such that for every $i \geq 0$, $U_i \supseteq \neg S_i$, and therefore, the limit of this sequence U is in general an ω-regular upperapproximation of $\neg S$. This means that the set $\neg U$ is a lower-approximation of S. Notice that [5,6] provide (sufficient) conditions which allows us to check that the computed set is precise.

7 Experimental Results

The techniques and algorithms presented in this paper have been tested on several examples covering different classes of systems and properties. Details about the considered models and the corresponding experiments are reported in the full version ([13]). We give hereafter a brief synopsis of these results.

First, we considered several examples of parametric networks corresponding to mutual exclusion protocols including the Bakery algorithm and the token ring protocol. For these systems, we have been able to check automatically livelock freedom properties.

Next, we have been able to check termination or non-termination of (multi-loop) programs manipulating integer variables.

Finally, we addressed the problem of verifying a liveness property of a system manipulating counters as well as (continuous time) clocks. One of the examples we considered is a simplified model of the IEEE 1394 root contention protocol.

References

[1] P. A. Abdulla and B. Jonsson and M. Nilsson and J. d'Orso and M. Saksena, *Regular Model Checking for S1S + LTL*.

[2] A. Bouajjani and A. Collomb-Annichini and Y. Lakhnech and M. Sighireanu", *Analyzing Fair Parametric Extended Automata*, Lecture Notes in Computer Science, volume 2126, year 2001.

[3] A. Bouajjani and B. Jonsson and M. Nilsson and Tayssir Touili, *Regular Model Checking*, Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00), Lecture Notes in Computer Science, volume 1855, year 2000, pages 403–418.

[4] B. Boigelot and S. Jodogne and P. Wolper", *On the Use of Weak Automata for Deciding Linear Arithmetic with Integer and Real Variables*, Proc. International Joint Conference on Automated Reasoning (IJCAR), Lecture Notes in Computer Science, volume 2083, year 200, pages 611–625.

[5] B. Boigelot and A. Legay and P. Wolper, *Iterating Transducers in the Large*, Proc. 15th Int. Conf. on Computer Aided Verification, Boulder, USA, Lecture Notes in Computer Science, volume 2725, year 2003, pages 223–235.

[6] B. Boigelot and A. Legay and P. Wolper, *Omega-Regular Model Checking*, Proc. 10th Int. Conf. on Tools and techniques , Barcelona, Spain, Lecture Notes in Computer Science, volume 2988, year 2004, pages 561–575.

[7] Y. Fang and N. Piterman and A. Pnueli and L. Zuck, *Liveness with Incomprehensible Ranking*, Proc. 10th Int. Conf. on Tools and techniques, Barcelona, Spain, Lecture Notes in Computer Science, volume 2988, year, pages 482–496.

[8] K. Baukus and Y. Lakhnech and K. Stahl, *Verifying Universal Properties of Parametrized Networks*, FTRTFT'00, Lecture Notes in Computer Science volume 1926, year 2000.

[9] D. Dams and Y. Lakhnech and M. Steffen, *Iterating Transducers*, Proceedings 13th International Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science, volume 2102, year 2001, pages 286–297.

[10] M. Y. Vardi and P. Wolper, *Automata-Theoretic Techniques for Modal Logics of Programs*, Journal of Computer and System Science, volume 32, number 2, pages 183-221, year 1986.

[11] M. Colon and H. Sipma, *Practical Methods for Proving Program Termination*, Proc. 14th Int. Conf. on Computer Aided Verification, Copenhagen, Denmark, Lecture Notes in Computer Science, volume 2404, year 2002, 442-454.

[12] D.Dams and R.Gerth and O.Grumberg, *A heuristic for the automatic generation of ranking functions*, Proceedings of WAVe00, URL: citeseer.nj.nec.com/article/dams00heuristic.html.

[13] A.Bouajjani and A.Legay and P.Wolper, *Handling Liveness Properties in (ω-)Regular Model Checking: full version*, Technical Report, URL: http://www.montefiore.ulg.ac.be/ legay/WWW/papers/infinity04.ps.

[14] Y. Kesten and O. Maler and M. Marcus and A. Pnueli and E. Shahar, *Symbolic Model Checking with Rich Assertional Languages*, Proceedings of 9th International Conference on Computer-Aided Verification (CAV'97), Lecture Notes in Computer Science, volume 1254.

[15] D. E. Muller and A. Saoudi and P. E. Schupp, *Alternating automata, the weak monadic theory of the tree and its complexity*, Proc. 13th Int. Colloquium on Automata, Languages and Programming, year 1986.

[16] A. Pnueli and E. Shahar", *Liveness and Acceleration in Parameterized Verification*, Proceedings of the 12th International Conference on Computer-Aided Verification (CAV'00), Lecture Notes in Computer Science, volume 1855, year 2000, pages 328-343.

[17] A. Pnueli and J. Xu and L. Zuck, *Liveness with (0,1,infinite)-Counter Abstraction*, Proc. 14th Int. Conf. on Computer Aided Verification, Copenhagen, Denmark, Lecture Notes in Computer Science, volume 2404, year 2002, 107-122.