Regular Model Checking

Ahmed Bouajjani[1], Bengt Jonsson[2], Marcus Nilsson[⋆2], and Tayssir Touili[1]

[1] Liafa, Univ. Paris 7, Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France
{Ahmed.Bouajjani,Tayssir.Touili}@liafa.jussieu.fr
[2] Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden
{bengt, marcusn}@docs.uu.se

Abstract. We present *regular model checking*, a framework for algorithmic verification of infinite-state systems with, e.g., queues, stacks, integers, or a parameterized linear topology. States are represented by strings over a finite alphabet and the transition relation by a regular length-preserving relation on strings. Major problems in the verification of parameterized and infinite-state systems are to compute the set of states that are reachable from some set of initial states, and to compute the transitive closure of the transition relation. We present two complementary techniques for these problems. One is a direct automata-theoretic construction, and the other is based on widening. Both techniques are incomplete in general, but we give sufficient conditions under which they work. We also present a method for verifying ω-regular properties of parameterized systems, by computation of the transitive closure of a transition relation.

1 Introduction

This paper presents *regular model checking*, intended as a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. Regular model checking considers systems whose states can be represented as finite strings of arbitrary length over a finite alphabet. This includes parameterized systems consisting of an arbitrary number of homogeneous finite-state processes connected in a linear or ring-formed topology, and systems that operate on queues, stacks, integers, and other data structures that can be represented by sequences of symbols.

Regular model checking can be seen as symbolic model checking, in which regular sets words over a finite alphabet is used as a symbolic representation of sets of states, and in which regular length-preserving relations between words, usually in the form of finite-state transducers, represent transition relations. This framework has been advocated by, e.g., Kesten et al. [KMM+97] and by Boigelot and Wolper [WB98], as a uniform framework for analyzing several classes of parameterized and infinite-state systems, where automata-theoretic algorithms

⋆ supported in part by the ASTEC competence center, and by the Swedish Board for Industrial and Technical Development (NUTEK)

for manipulation of regular sets can be exploited. Such algorithms have been implemented, e.g., in the Mona [HJJ+96] and MoSel [KMMG97] packages.

A major problem in regular model checking is that the standard iteration-based methods, used for finite-state systems (e.g., [BCMD92]), to compute, e.g., the set of reachable states, are guaranteed to terminate only if there is a bound on the distance (in number of transitions) from the initial configurations to any reachable configuration. An analogous observation holds if we perform a reachability backwards, from a set of "unsafe" configurations, as in [KMM+97]. In general, a parameterized or infinite-state system does not have such a bound. To explore the entire state-space, one must therefore be able to calculate the effect of arbitrarily long sequences of transitions. For instance, consider a transition of a parameterized system in which a process passes a token to its neighbor. The effect of an arbitrarily long sequence of such transitions will be to pass the token to any other process through an arbitrary sequence of neighbors.

The problem of calculating the effect of arbitrarily long sequences of transitions has been addressed for certain classes of systems, e.g., systems with unbounded FIFO channels [BG96,BGWW97,BH97,ABJ98], systems with push-down stacks [BEM97,Cau92,FWW97], systems with counters [BW94,CJ98], and certain classes of parameterized systems [ABJN99]. A more uniform calculation of the *transitive closure* of a transition relation was presented in our previous work [JN00] for the case that the transition relation satisfies a condition of *bounded local depth*. This construction was used to verify safety properties of several parameterized algorithms and algorithms that operate on unbounded FIFO channels or on counters.

In this paper, we develop the regular model checking paradigm further. We give a simple and uniform presentation of the program model, which is simpler than that used in [JN00]. The main part of the paper considers the problem of calculating the set of configurations reachable from a set of initial configurations via a transition relation, and the problem of computing the transitive closure of a transition relation. We present two complementary techniques to attack these problems:

- The first technique is an automata-theoretic construction, which uses the standard subset-construction and minimization techniques from automata theory. The construction succeeds if the resulting automaton is finite-state. It generalizes the transitive closure construction in [JN00].
- The second technique is based on computing fixpoints using widening, in the spirit of [CC77]. We propose exact widening techniques for regular languages and show how to use them for reachability analysis and for computing transitive closures.

As another main contribution, we show how to verify *liveness properties* (or, more generally, ω-regular properties). When applied to parameterized systems, our method allows to prove, e.g., absence of starvation in resource allocation algorithms. In general, it allows to prove a liveness property, given an arbitrary number of *parameterized fairness requirements*. For instance, in a parameterized algorithm there is typically a fairness requirement associated with each process.

Our method is based on a reduction of the model-checking problem of ω-regular properties to the problem of finding fair loops, which can be detected after having computed the transitive closure of a transition relation.

Both the automata-theoretic and the widening-based techniques presented in this paper have been implemented and experimented on several examples of parameterized and infinite-state systems, including different parameterized algorithms for mutual exclusion (e.g., the Bakery algorithm by Lamport, Burns protocol, Dijkstra algorithm), unbounded FIFO-channel systems (e.g., Alternating-bit protocol), and counter systems (e.g., Sliding Window protocol with unbounded sequence numbers).

Outline In the next section, we present the framework of regular model checking, and define the verification problems that are considered in the paper. In Section 3 we present an automata-theoretic construction for computing the transitive closure of a transition relation. The construction can also be used to compute reachability sets. Sect. 4 presents widening techniques for the same problem. A method for model checking of ω-regular properties is given in Sect. 5, and concluding remarks are given in Sect, 6.

Related Work Previous work on the general aspects of regular model checking, and on analyzing classes of systems, e.g., pushdown systems, parameterized systems, systems with FIFO channels, or with counters, has already been mentioned earlier in the introduction. The acceleration techniques presented in this paper are able to emulate the acceleration operations for FIFO channels reported in [BG96].

Widening techniques have been introduced in [CC77] in order to speed up the computation of fixpoints in the framework of abstract interpretation. Many works have proposed widening operators for different kinds of systems, e.g., widening operators defined on representations based on convex polyhedra for use in the analysis of systems operating on integers or reals (e.g., [CH78,Hal93]), or widening operators on automata [LHR97]. All these techniques compute upper approximations of the desired fixpoint.

Other researchers, e.g., [FO97,Sis97], use regular sets in a deductive framework, where basic manipulations on regular sets are performed automatically. These methods are based on proving an invariant given by the user or by some invariant generation technique, but are not fully automatic. In [FO97], the authors show how to check that a given regular set is the reachability set of special kind of relations. We use their result to prove that our widening technique is exact. However, they do not provide any technique to find automatically the fixpoint.

2 Preliminaries

2.1 Model

We introduce the program model used in regular model checking.

Let Σ be a finite alphabet. As usual, Σ^* is the set of finite words over Σ. A relation R on Σ^* is *regular* and *length-preserving* if w and w' are of equal length whenever $(w, w') \in R$, and the set $\{(a_1, a'_1) \cdots (a_n, a'_n) \; : \; (a_1 \cdots a_n, a'_1 \cdots a'_n) \in R\}$ is a regular subset of $(\Sigma \times \Sigma)^*$. In the following, we will implicitly understand that a regular relation is also length-preserving. A regular relation can be conveniently recognized by a finite-state transducer over $(\Sigma \times \Sigma)$.

Definition 1. A *program* is a triple $\mathcal{P} = \langle \Sigma, \phi_I, R \rangle$ where

Σ is a finite alphabet,
ϕ_I is a regular set over Σ, denoting a set of *initial configurations*, and
R is a regular relation on Σ^*, sometimes called the *transition relation*. $\square$

A *configuration* w of a program $\mathcal{P}$ is a word $a_1 \; a_2 \; \cdots \; a_n$ over Σ. We denote by $R_{id} = \{(w, w') \; : \; w = w'\}$ the identity relation on configurations. Regular relations can be composed to yield new regular relations. For two regular relations R and R', their union $R \cup R'$, intersection $R \cap R'$, sequential (or relational) composition $R \circ R'$, and concatenation $R \cdot R'$ are regular.

2.2 Modeling parameterized and Infinite-state Systems

We give two examples of different classes of systems that can be modeled in the framework of the preceding subsection. More examples can be found in [JN00].

Parameterized Systems. Consider a parameterized system consisting of an arbitrary number of homogeneous finite-state processes connected in a linear topology. A configuration of this system can be represented by a word over an alphabet consisting of the set of states for each process, where the length of the word equals the number of processes. A particular example of such a system is an array of processes that pass a token from the left to the right. Each process can be in one of two states, $\perp$ or t, where $\perp$ denotes that the process does not have the token, and t denotes that the process has the token. We model this system as the program $\mathcal{P} = \langle \Sigma, \phi_I, R \rangle$ where

- $\Sigma = \{\perp, t\}$ is the set of states of a process,
- $\phi_I = t\perp^*$ is the set of initial configurations, in which the leftmost process has the token, and
- $R = (\perp,\perp)^* \cdot (t,\perp) \cdot (\perp,t) \cdot (\perp,\perp)^* \quad \cup \quad (\perp,\perp)^* \cdot ((\perp,\perp) \cup (t,t)) \cdot (\perp,\perp)^*$ is the union of two relations, of which the first denotes the passing of the token from a process to its right neighbor, and the second denotes an idling computation step. Note that the transition relation is implicitly constrained by the invariant that there is exactly one token in the system.

Systems Communicating over Unbounded FIFO Channels As another example, consider systems of finite-state processes that communicate over unbounded FIFO channels. Assume for simplicity that there is only one FIFO channel, containing a sequence of messages in the finite set $\mathcal{M}$. Let Q be the finite set of combinations of control states of the processes. A configuration of this system can be modeled as a word of the form

$$q \ \perp\perp\cdots\perp \ m_1 m_2 \cdots m_n \ \perp\perp\cdots\perp$$

where $q \in Q$ is the current control state, and $m_1 m_2 \cdots m_n$ is the sequence of messages in the channel. In order to allow messages to be added to and removed from the channel, we add an arbitrary amount of "padding symbols" $\perp$ before and after the sequence of messages. Thus, we can model this system as the program $\mathcal{P} = \langle \Sigma, \phi_I, R \rangle$, where

- $\Sigma = Q \cup \mathcal{M} \cup \{\perp\}$,
- ϕ_I is the regular set $q_i \perp^*$ of configurations with no message in the channel, and the initial control state q_i,
- R is the union of several relations: one for each operation in the system. An operation which sends a message $m \in \mathcal{M}$ while making a transition from control state q to q' is modeled by the relation

$$(q, q') \ \cdot \ (\perp, \perp)^* \ \cdot \ (R_{id} \cap \mathcal{M}^2)^* \cdot \ (\perp, m) \ \cdot \ (\perp, \perp)^*$$

  Receive operations are modeled in a similar way.

2.3 Verification

Let R be a regular relation. We use R^+ to denote the transitive closure of R, and R^* to denote the reflexive and transitive closure of R, and R^{-1} to denote the inverse of R. For a regular set ϕ of configurations, $R(\phi)$ denotes the image $\{w' \ : \ \exists w \in \phi . (w, w') \in R\}$ of ϕ under R.

In this paper, we will consider two verification problems:

- *Computing Reachability Sets:* Given a regular set ϕ of configurations and a regular relation R, compute the reachability set $R^*(\phi)$. This can be used for checking whether some configuration in a set ϕ_F is reachable from a set ϕ_I of initial configurations, either by checking whether $\phi_F \cap R^*(\phi_I) = \emptyset$ (forward reachability analysis), or whether $\phi_I \cap (R^{-1})^*(\phi_F) = \emptyset$ (backward reachability analysis). If R is a union $R_1 \cup \cdots \cup R_k$ of several regular relations, then one can also compute the set of reachable configurations stepwise, by repeatedly extending the set ϕ of currently reached configurations by $R_i^*(\phi)$ for some i.
- *Computing Transitive Closure:* Given a regular relation R, compute its transitive closure R^+. The transitive closure can be used for finding loops of parameterized systems. The relation $R^+ \cap R_{id}$ represents the identity relation on the set of configurations that can reach themselves via a non-empty sequence of computation steps. To obtain the loops that can actually occur in an execution, we intersect this set with the set of reachable configurations, computed as a reachability set. The transitive closure can also be used for computing the reachability set.

A straight-forward approach to computing $R^*(\phi)$ is to compute a sequence of unions $\cup_{i=0}^{n} R^i(\phi)$ for $n = 1, 2, 3, \ldots$ until a fixpoint is reached, i.e., $\cup_{i=0}^{n} R^i(\phi) = \cup_{i=0}^{n+1} R^i(\phi)$ for some n. This approach is guaranteed to terminate for finite-state systems [BCMD92], but in general not for parameterized and infinite-state systems, as observed, e.g., in [ABJN99]. The analogous observation holds for the approach of computing R^+ through the sequence of unions $\cup_{i=1}^{n} R^i$ for $n = 1, 2, 3, \ldots$ until convergence.

We present two complementary techniques which are applicable to both of the above problems.

- A direct automata-theoretic construction for computing $R^*(\phi)$ or R^+, which is based on the transducer for R. It is presented in Section 3.
- A technique based on widening, in the spirit of [CC77] is presented in Section 4. The technique is based on observing successive approximations to $R^*(\phi)$, and trying to guess automatically the limit. The guess can always be checked to be an upper-approximation, and to be *exact* under some conditions. The technique can also be applied to the computation of R^*, by viewing it as a regular set over $\Sigma \times \Sigma$.

3 Automata Theoretic Construction of the Transitive Closure

In this section, we present a technique for computing R^+, which attempts to compute a minimal deterministic transducer that recognizes R^+ using the standard subset-construction and minimization techniques from elementary automata theory. The technique can also, with obvious modifications, be used for computing $R^*(\phi)$. We will here concentrate on the transitive closure construction, and only briefly summarize the reachability set construction at the end of this section.

Our transitive closure construction is not guaranteed to terminate, in particular if R^+ is not regular. However, if R has bounded *local depth*, a concept introduced in [JN00], a slightly modified version of our construction will yield a finite transducer.

3.1 The Transducer Construction

For the remainder of this section, let R be a regular relation on Σ, represented as a finite-state transducer $R = \langle Q, q_0, \delta, F\rangle$ where Q is the set of states, q_0 is the initial state, $\delta : (Q \times (\Sigma \times \Sigma)) \mapsto 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. For each pair (w, w') in R^+, there is a sequence $w^0, w^1, \ldots, w^m$ of configurations such that $w = w^0$, $w' = w^m$ and $(w^{i-1}, w^i) \in R$ for $0 < i \leq m$. Let w^i be the word $a_1^i a_2^i \cdots a_n^i$ of length n. Then $(w^{i-1}, w^i) \in R$ means that there is a run $q_0^i q_1^i \cdots q_n^i$ of R which accepts the word $(a_1^{i-1}, a_1^i)\ (a_2^{i-1}, a_2^i) \cdots (a_n^{i-1}, a_n^i)$. Let us organize these runs into a matrix of form:

$$\begin{array}{ccccccc}
q_0^1 & \xrightarrow{(a_1^0,a_1^1)} & q_1^1 & \xrightarrow{(a_2^0,a_2^1)} & q_2^1 \cdots q_{n-1}^1 & \xrightarrow{(a_n^0,a_n^1)} & q_n^1 \\
q_0^2 & \xrightarrow{(a_1^1,a_1^2)} & q_1^2 & \xrightarrow{(a_2^1,a_2^2)} & q_2^2 \cdots q_{n-1}^2 & \xrightarrow{(a_n^1,a_n^2)} & q_n^2 \\
 & \vdots & & \vdots & & \vdots & \\
q_0^m & \xrightarrow{(a_1^{m-1},a_1^m)} & q_1^m & \xrightarrow{(a_2^{m-1},a_2^m)} & q_2^m \cdots q_{n-1}^m & \xrightarrow{(a_n^{m-1},a_n^m)} & q_n^m
\end{array}$$

with m rows, where each row shows a run of the transducer R with n transitions.

The first step in our construction of R^+ is to regard the above matrix as a single run of another transducer, whose states are columns (i.e., sequences) of form $q^1 q^2 \cdots q^m$ for $m \geq 1$, and whose transitions represent the relationship between adjacent columns in a matrix of the above form. More precisely, define the *column transducer for* R^+ as the tuple $\langle Q^+, q_0^+, \Delta, F^+ \rangle$ where

- Q^+ is the set of non-empty sequences of states of R,
- q_0^+ is the set of non-empty sequences of initial states of R,
- $\Delta : (Q^+ \times (\Sigma \times \Sigma)) \mapsto 2^{Q^+}$ is defined as follows: for any columns $r^1 r^2 \cdots r^m$ and $q^1 q^2 \cdots q^m$, and pair (a, a'), we have $r^1 r^2 \cdots r^m \in \Delta(q^1 q^2 \cdots q^m, (a, a'))$ iff there are $a^0, a^1, \ldots, a^m$ with $a = a^0$ and $a' = a^m$ such that $r^i \in \delta(q^i, (a^{i-1}, a^i))$ for $1 \leq i \leq m$.

It is not difficult to show that the column transducer for R^+ accepts exactly the relation R^+. The only problem is that it has infinitely many states. We will therefore determinize it using the standard subset-construction, in the hope of decreasing the number of states. Let x, y range over columns and X, Y over sets of columns. The subset construction applied to the column transducer for R^+ yields the automaton, $\langle 2^{Q^+}, q_0^+, \wp\Delta, \mathcal{F} \rangle$ whose states are sets of columns of states of R, whose initial state is the set q_0^+ of columns of initial states, whose set $\mathcal{F}$ of accepting states is the set of states with at least one column in F^+, and whose transition function $\wp\Delta$ is the lifting of Δ to sets: $\wp\Delta(X, (a, a')) = \cup_{x \in X} \Delta(x, (a, a'))$. We note that for each pair (a, a'), the relation $\{(x, y) : y \in \Delta(x, (a, a'))\}$ is regular, so that a transducer which implements the function $\wp\Delta$ can be constructed directly from the definition of Δ. It follows that if X is a regular set of columns, then $\wp\Delta(X)$ is also a regular set of columns, which can be constructed by composing the automaton for X with that for $\wp\Delta$ and projecting onto the "next column".

In most cases, the subset construction does not yield a finite automaton. We therefore try to make it smaller by identifying equivalent sets of columns during the construction (cf. the minimization of deterministic finite automata). For a set X of columns (i.e., a state in the subset-automaton), let $\mathsf{suff}(X)$ denote the set of suffixes of X, i.e., the set of words $\pi \in (\Sigma \times \Sigma)^*$ such that $\wp\Delta(X, \pi) \in \mathcal{F}$. Two sets X, Y of columns are *equivalent* if $\mathsf{suff}(X) = \mathsf{suff}(Y)$. We employ a simple (and incomplete) technique to detect equivalent sets, based on *saturation*. The basic idea is to extend (saturate) each set X of columns by additional columns x such that $\mathsf{suff}(\{x\}) \subseteq \mathsf{suff}(X)$. Hopefully, two equivalent sets of columns will become identical after saturation, in which case they will identified during the subset construction.

Let us present the saturation rule. A state q in the original transducer R is a *copying state* if $\mathsf{suff}(\{q\}) \subseteq R_{id}$, i.e., if its suffixes only perform copying of words. We use the following saturation rule:.

- if $xy \in X$ or if $xqqy \in X$, where x and y are (possibly empty) columns and q is a copying state, then add xqy to X.

It is not difficult to see that $\mathsf{suff}(\{xqy\}) \subseteq \mathsf{suff}(\{xy\})$ and that $\mathsf{suff}(\{xqy\}) \subseteq \mathsf{suff}(\{xqqy\})$, implying the soundness of the saturation rule. Let $\lceil X \rceil$ denote the saturation of X, i.e., the least set of columns containing X which is closed under the above rule. To saturate a regular set of columns is an easy operation on the automaton which represents the set.

It follows from the above construction that the transducer obtained after saturation recognizes R^+. We summarize the discussion in the following theorem.

Theorem 1. *Let $R = \langle Q, q_0, \delta, F \rangle$ be a finite-state transducer. Then the deterministic (possibly infinite-state) transducer $\langle \mathcal{Q}, \lceil q_0^+ \rceil, \tilde{\Delta}, \tilde{\mathcal{F}} \rangle$, where*

- *$\mathcal{Q} \subseteq 2^{Q^+}$ is the set of saturated regular subsets of Q^+,*
- *$\tilde{\Delta} : (\mathcal{Q} \times (\Sigma \times \Sigma)) \mapsto \mathcal{Q}$ is defined by $\tilde{\Delta}(X, (a, a')) = \lceil \wp\Delta(X, (a, a')) \rceil$,*
- *$\tilde{\mathcal{F}}$ are the sets with at least one sequence in F^+.*

accepts the relation R^+. □

It follows that if the set set of reachable states in the above automaton is finite, then R^+ is regular. We can then, using standard techniques, obtain a minimal deterministic finite-state transducer which recognizes R^+.

3.2 A sufficient condition for Termination

In [JN00], it was shown that R^+ is regular under some sufficient conditions on a regular relation R. We will restate these conditions and note that the construction given in the previous section, with slight modifications, will terminate under these conditions. It should be noted that these conditions are merely sufficient conditions for regularity, and that there are many other situations in which our construction of R^+ yields a finite-state transducer.

Let R be of the form $\phi_L \cdot R_l \cdot \phi_R$ where

- $\phi_L \subseteq R_{id}$ is a *left context*, i.e., it copies a regular language which can be accepted by a deterministic finite automaton which has a unique accepting state, and which has no transitions from an accepting to a non-accepting state,
- R_l is any regular relation,
- $\phi_R \subseteq R_{id}$ is a *right context*, i.e., the mirror image of some left context.

We say that R has *local depth k* if for each $(w, w') \in R^+$ there is a sequence $w = w^0, w^1, \ldots, w^m = w'$ such that for each $1 \leq i \leq m$ we can find indices l_i and u_i such that

- $(a_1^{i-1} a_2^{i-1} \cdots a_{l_i-1}^{i-1}\ ,\ a_1^i a_2^i \cdots a_{l_i-1}^i) \in \phi_L$,
- $(a_{u_i+1}^{i-1} a_{u_i+2}^{i-1} \cdots a_n^{i-1}\ ,\ a_{u_i+1}^i a_{u_i+2}^i \cdots a_n^i) \in \phi_R$, and
- $(a_{l_i}^{i-1} a_{l_i+1}^{i-1} \cdots a_{u_i}^{i-1}\ ,\ a_{l_i}^i a_{l_i+1}^i \cdots a_{u_i}^i) \in R_l$.

where $w^i = a_1^i a_2^i \cdots a_n^i$ and such that for each position p (with $1 \leq p \leq n$), there are at most k indices i with $1 \leq i \leq m$ such that $l_i \leq p \leq u_i$. Intuitively, the decomposition of R into $\phi_L \cdot R_l \cdot \phi_R$ serves to structure R as a local rewriting R_l in a "context", represented by ϕ_L and ϕ_R. A relation with local depth k never needs to rewrite any element of a word more than k times to relate two words.

Theorem 2. *[JN00] If R has local depth k, for any k, the transitive closure R^+ is regular and can be recognized by a transducer of size exponential in k.* □

The proof of the above theorem divides the set of columns into a finite set of equivalence classes. We will try to apply the same reasoning to the saturated columns in the construction of Theorem 1.

Let a *copied state* be a state whose prefixes are a subset of the identity relation, in analogy with copying states. The key of the proof is the observation that all sets X of columns generated in the subset construction satisfy the following closure property:

- if $xqy \in X$, where x and y are (possibly empty) columns and q is a copied state, then $xzy \in X$ for any $z \in q^*$.

It can be shown that the saturated columns corresponding to sequences of configurations satisfying the conditions of local depth k is built up from sets of the form

$$X_0 q_0 X_1 q_1 \cdots X_{n-1} q_{n-1} X_n$$

with $n \leq k$ and where each q_i with $0 \leq i < n$ is a state in the transducer for R_l and each X_i with $0 \leq i \leq n$ is one of the following sets, where q_L is a copied state and q_R is a copying state:

1. q_R^* *2.* $(q_L + q_R)^*$ *3.* $(q_L + q_R)^* q_R (q_L + q_R)^*$
4. $q_R(q_L + q_R)^*$ *5.* $(q_L + q_R)^* q_R$ *6.* $q_R(q_L + q_R)^* q_R$ *7.* q_R^+

To obtain the termination result, we therefore restrict the columns in the construction to have up to k states from the transducer for R_l. The conditions on R ensure that it is enough to consider such columns.

3.3 Computing Reachable Configurations

We will finally sketch the modifications for computing instead the set $R^*(\phi)$, where ϕ is a regular set of configurations. Assume that ϕ is recognized by a finite automaton. In the construction of Section 3.1, a run of ϕ will replace the transducer run in the first row of the matrix, with the obvious modifications. The end result, corresponding to Theorem 1, is a (possibly infinite-state) automaton that recognizes $R^*(\phi)$, with the starting state $\lceil p_0 q_0^* \rceil$, where p_0 is the starting state of an automaton that recognizes ϕ and q_0 is the starting state of a transducer for R.

4 Widening based techniques

We present in this section techniques for computing the effect of iterating a regular relation. These techniques are based on using *exact widening* operations in order to speed up the calculation of a regular fixpoint.

Roughly speaking, our techniques consist in (1) guessing automatically the image of iterating a relation starting from some given regular set, and (2) deciding whether this guess is correct.

The guessing technique we use consists in, given a relation R and a set ϕ, comparing the sets ϕ and $R(\phi)$ in order to detect some "growth" (e.g., $R(\phi) = \phi \cdot \Lambda$ for some regular Λ), and then extrapolate by guessing that each application of R will have the same effect (e.g., we guess that the limit could be $\phi \cdot \Lambda^*$). Then, we apply a simple fixpoint test which ensures (in general) that the guessed set is an upper approximation of the set $R^*(\phi)$. This means that using our widening techniques we can capture in one step (at least) all the reachable configurations from ϕ by R^*. Moreover, we show that under some conditions on R, the fixpoint test allows in fact to *decide* the *exactness* of our guess (i.e., that the guessed set in precisely $R^*(\phi)$).

4.1 Widening on regular sets

Widening is applied during the iterative construction of the set of reachable configurations in order to help termination. Given a set of configurations $\phi \subseteq \Sigma^*$ and a relation R, a widening step consists in guessing the result of iterating R starting from ϕ by comparing ϕ with $R(\phi)$ (in general, this guess can be made by considering the sets $R^i(\phi)$ up to some finite bound k). Once this guess is made, the obtained set is added to the computed set of configurations and the exploration of the configuration space is continued.

Let $\phi \subseteq \Sigma^*$ and let R be a regular relation on Σ^*. Our widening principle consists in checking whether there are regular sets ϕ_1, ϕ_2, and Λ such that the following two conditions are satisfied

C1: $\phi = \phi_1 \cdot \phi_2$ and $R(\phi) = \phi_1 \cdot \Lambda \cdot \phi_2$,
C2: $\phi_1 \cdot \Lambda^* \cdot \phi_2 = R(\phi_1 \cdot \Lambda^* \cdot \phi_2) \cup \phi$.

and, if C1 and C2 hold, in adding $\phi_1 \cdot \Lambda^* \cdot \phi_2$ to the computed set of configurations.

Intuitively, condition C1 means that the effect of applying R to ϕ is to "add" Λ between ϕ_1 and ϕ_2. Notice that when ϕ_1 or ϕ_2 is equal to $\{\epsilon\}$, this corresponds respectively to the case where a growth occurs to the left or to the right of ϕ. Condition C2 implies that $R^*(\phi) \subseteq \phi_1 \cdot \Lambda^* \cdot \phi_2$. Indeed, C2 means that $\phi_1 \cdot \Lambda^* \cdot \phi_2$ is a fixpoint of $\mathcal{F} = \lambda X.\ \phi \cup R(X)$ and $R^*(\phi)$ is the least fixpoint of $\mathcal{F}$. Hence, by adding $\phi_1 \cdot \Lambda^* \cdot \phi_2$ to the computed set of configurations, we capture at least all the reachable configurations from ϕ by iterating R.

The inclusion $\phi_1 \cdot \Lambda^* \cdot \phi_2 \subseteq R^*(\phi)$ is not guaranteed in general by C2 (for any kind of relation R). Nevertheless, we show in the next section that for a large class of relations, condition C2 *guarantees* the *exactness* of our technique, i.e., it computes exactly the set $R^*(\phi)$.

The application of the widening principle depends on the quality of the "automatic guessing" part which is implicit in C1. Given two regular sets ϕ and $\phi' = R(\phi)$, we have to *find* regular sets ϕ_1, ϕ_2 and Λ such that C1 holds, and check that for these sets the condition C2 also holds. We use techniques on automata allowing to extract from ϕ and ϕ' these sets. Roughly speaking, these techniques consist in finding cuts in the automata of ϕ and ϕ' that delimit ϕ_1, ϕ_2 and Λ. To reduce the number of choices to examine, we adopt a heuristic which consists in considering cuts at entering vertices to strongly connected components. This heuristic behaves well because almost always the automata we need are of simple forms (e.g., their loops are only self-loops) and the Λ is very often a nonempty word (or a sum of nonempty words $w_1 + \cdots + w_n$), or a set of the form $w \cdot \Lambda'$ or $\Lambda' \cdot w$ where w is a nonempty word.

4.2 Simple rewriting relations

We introduce in this section a class of relations for which it can be shown that our widening technique is exact.

A regular relation R is *unary* if it is a subset of $\Sigma \times \Sigma$. A unary relation is *acyclic* if there is no $a \in \Sigma$ such that $(a, a) \in R^+$. A regular relation R is *binary* if it is a subset of $\Sigma^2 \times \Sigma^2$. A binary relation R is a *permutation* if it is a set of pairs of form (ab, ba) where $a, b \in \Sigma$ and $a \neq b$. A permutation is *antisymmetric* if there are no $a, b \in \Sigma$ such that both $(ab, ba) \in R$ and $(ba, ab) \in R$.

A regular relation is defined to be *simple* if it is a finite union of relations of form $\phi_L \cdot R_l \cdot \phi_R$, such that

1. In each sub-relation of form $\phi_L \cdot R_l \cdot \phi_R$, the "contexts" ϕ_L and ϕ_R are regular subsets of R_{id}, and R_l is either unary or binary,
2. The union of all unary R_l's is acyclic,
3. The union of all binary R_l's is antisymmetric.

The class of simple relations is noncomparable with the class of relations of bounded local depth. We can now prove the following theorem, stating that conditions C1 and C2 give an exact guess for simple relations.

Theorem 3. *If R is a simple relation, and if there are regular sets ϕ_1, ϕ_2, and Λ such that conditions C1 and C2 are satisfied, then $R^*(\phi) = \phi_1 \cdot \Lambda^* \cdot \phi_2$.*

To prove this theorem, we need to introduce the notion of *nœtherian* relations. We say that a relation R is *nœtherian* if for every word $w \in \Sigma^*$, there is no infinite sequence $w_0, w_1, w_2, \ldots$ such that $w_0 = w$ and for every $i \geq 0$, $(w_i, w_{i+1}) \in R$ (i.e., no word can be rewritten an infinite number of times). Notice that a length preserving relation R is nœtherian if and only if $R_{id} \cap R^+ = \emptyset$. We can prove the following fact:

Proposition 1. *For every simple relation R, both R and R^{-1} are nœtherian.*

Then, Theorem 3 can be deduced from Proposition 1 and the following result by Fribourg and Olsen on nœtherian relations:

Proposition 2 ([FO97]). *Let R be a relation. If R^{-1} is nœtherian, then for every $\phi, \phi' \subseteq \Sigma^*$, $\phi' = R(\phi') \cup \phi$ if and only if $\phi' = R^*(\phi)$.*

4.3 Constructing transitive closures

Given a length preserving relation R, widening can also be used to compute the transitive closure of R. Indeed, R can be seen as a language over $\Sigma \times \Sigma$ and R^+ can be computed as the limit of the sequence $(R^i)_{i>0}$. Our procedure starts from R and computes iteratively the sets R^i for increasing i's. Each step consists in applying the operation $\lambda X.\ R \circ X$. During this iterative computation, widening operations can be applied in order to jump to the limit. Now, from the definition of nœtherian relations, it is easy to see that:

Lemma 1. *For every nœtherian relation R, the relation $\{((w, w_1), (w, w_2)) : w_2 \in R(w_1)\}$ is nœtherian.*

From Lemma 1 and Proposition 1, we deduce that our widening technique is also *exact* when computing the transitive closure of a simple relation R, starting from R and applying iteratively $\lambda X.\ R \circ X$. Notice that R^* can be computed in the same manner, starting from R_{id} instead of R.

4.4 Example

We have applied our widening-based techniques to several examples of parameterized systems including the mutual exclusion protocols of Szymanski, Burns, Dijkstra, the Bakery protocol of Lamport, as well as the Token passing protocol. All these examples can be modeled as simple relations, and our procedure terminates for all of them and computes the exact set of reachable configurations. Moreover, our techniques allow also to construct the transitive closures of the relations modeling these systems.

We show here the application of our techniques on the example of the token passing protocol described in section 2.2. It is easy to see that the relation R in this model is indeed simple.

First, let us consider the problem of computing the reachability set. Our procedure starts from the initial set of configurations $\phi_I = t\bot^*$ and computes the set $R(\phi_I) = \bot t \bot^*$. At this point, it checks that condition C1 holds since $R(t\bot^*) = \Lambda \cdot t\bot^*$ where $\Lambda = \bot$. Then, it checks that condition C2 also holds:

$$R(\bot^* t \bot^*)\ \cup\ t\bot^* = \bot^* \bot t \bot^*\ \cup\ t\bot^* = \bot^* t \bot^*$$

Hence, we can apply an exact widening step by adding $\bot^* t \bot^*$ to the set of reachable configurations. By doing this, our procedure terminates (since no new configurations can be added) and we get the result:

$$R^*(t\bot^*) = \bot^* t \bot^*$$

Now, let us consider the problem of constructing the relation R^+. Our procedure starts from the relation

$$R = (\bot, \bot)^* \cdot (t, \bot) \cdot (\bot, t) \cdot (\bot, \bot)^*$$

(We omit here the part of the relation corresponding to idle transitions.) The first step is to compute R^2 which is:

$$R^2 = (\perp,\perp)^* \cdot (t,\perp) \cdot (\perp,\perp) \cdot (\perp,t) \cdot (\perp,\perp)^*$$

Then, it can be checked that C1 holds because $R = \phi_1 \cdot \phi_2$ and $R^2 = \phi_1 \cdot \Lambda \cdot \phi_2$ with $\phi_1 = (\perp,\perp)^* \cdot (t,\perp)$, $\phi_2 = (\perp,t) \cdot (\perp,\perp)^*$, and $\Lambda = (\perp,\perp)$. It can also be checked that C2 holds:

$$(R \circ (\phi_1 \cdot (\perp,\perp)^* \cdot \phi_2)) \cup R = \phi_1 \cdot (\perp,\perp)^* \cdot (\perp,\perp) \cdot \phi_2 \cup \phi_1 \cdot \phi_2 = \phi_1 \cdot (\perp,\perp)^* \cdot \phi_2$$

and hence, our procedure gives the result:

$$R^+ = (\perp,\perp)^* \cdot (t,\perp) \cdot (\perp,\perp)^* \cdot (\perp,t) \cdot (\perp,\perp)^*$$

5 Model Checking of ω-regular Properties

In this section we will show how to reduce the problem of verifying a property specified by a Büchi automaton to the problem of computing the transitive closure. A related technique is presented by Pnueli and Shahar [PS00]. Our technique is based on the observation that the problem of detecting infinite sequences reduces to that of detecting loops. This is true because the transition relation is length preserving which implies that each state, which is a word of a certain length, can only reach a finite set of states. For a program $\mathcal{P} = \langle \Sigma, \phi_I, R \rangle$, we can check for loops by checking the emptiness of the set

$$R^*(\phi_I) \ \cap \ R^+ \ \cap \ R_{id}$$

We can use this idea to verify that a program satisfies an ω-regular property under a set of fairness requirements, as follows. We use the standard technique [VW86] of encoding the negation of the property to be checked as a Büchi automaton. We also encode each fairness constraint as a Büchi automaton. Actually, we can handle parameterized fairness requirements, using the position as the parameter: simply associate one Büchi automaton with each position in the word, which expresses the fairness constraint for that position. Now construct the product of the program with the Büchi automaton for the negation of the property, and the Büchi automata for the fairness requirements. We must now check whether this product has a reachable "fair loop" in which each Büchi automaton visits an accepting state.

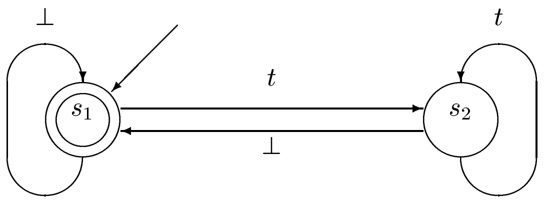
To check for fair loops, we first construct the set of reachable configurations of the product. Then, for each Büchi automaton, we add an observer. This is a bit which can be initialized to *false* in a reachable state and thereafter detects whether the Büchi automaton has visited some accepting state during a sequence of transitions. More precisely, the transition relation is extended so that it sets an observer bit to *true* whenever the corresponding Büchi automaton reaches an accepting state; an observer bit can never become *false* after being set to true.

Let R_{aug} be the so constructed transition relation containing both Büchi automata and their observer bits. Fair loops can now be detected by checking whether R_{aug}^{+} relates a reachable state with all observer bits being *false* with the same reachable state but with all observer bits being *true*.

We illustrate this method by verifying a liveness property for the token array system, given in Sect. 2.2, extended with fairness constraints. We will verify that every process eventually gets the token. The negation of this property is "some process never gets the token", which can be expressed by a Büchi automaton accepting an infinite sequence of states of a process where the token is never obtained, i.e., an infinite sequence of the symbol $\perp$. We encode this automaton by adding one boolean variable r which is true at the position at which the Büchi automaton reads symbols and by constraining the transition relation and the set of initial configurations so that

- r is true at exactly one position in the word,
- the truth value of r never changes in any position, and
- the token is never passed to the position where r holds.

We also impose for each process the fairness constraint that "the process may not hold the token indefinitely". For each position, this fairness constraint can be expressed by the Büchi automaton



These Büchi automata, one for each position in the word, are encoded by an extra variable s, initialized to s_1 and ranging over the set of automaton states $\{s_1, s_2\}$. The transition relation is extended so that the variable s simulates the state of the above automaton. Let $\hat{\mathcal{P}} = \langle \hat{\Sigma}, \hat{\phi}_I, \hat{R} \rangle$ denote the resulting program obtained by adding the variables r and s as described above.

Finally, a boolean observer s_{obs} is added to each position and the transition relation is changed so that s_{obs} becomes true when s becomes the accepting state s_1. Let R_{aug} be the resulting transition relation.

Let the variable $\hat{a}$ range over $\hat{\Sigma}$, the alphabet of the program $\hat{\mathcal{P}}$. We can now check for *fair* infinite runs that violate the original property ("every process eventually gets the token") by checking emptiness of the set

$$\hat{R}^*(\hat{\phi_I}) \;\cap\; R_{aug}^{+} \;\cap\; (\hat{a} = \hat{a}' \;\wedge\; \neg s_{obs} \;\wedge\; s'_{obs})^*$$

Note that the set $\hat{R}^*(\hat{\phi_I})$ can be computed as a reachability set. We have applied the construction given in Sect. 3 to construct a transducer for R_{aug}^{+} successfully in our implementation.

6 Conclusions

We have presented *regular model checking*, a framework for algorithmic verification of parameterized and infinite-state systems. To solve verification problems in this framework, we need to reason about the effect of iterating regular relations an unbounded number of times. It is well-known that the verification of safety properties reduces to reachability analysis. Moreover, the verification problem of any ω-regular properties, including liveness properties, can be reduced to the construction of the transitive closure of a regular length-preserving relation.

We have investigated properties of transitive closures, which lead to the development of new techniques for the construction of transitive closures. We have also presented widening-based techniques for constructing transitive closures and for reachability analysis. These techniques can be combined for computing sets of reachable configurations and transitive closures during verification.

Acknowledgments We are grateful to Amir Pnueli for fruitful discussions. Collaboration was supported by a travel grant from the ARTES network for real time research and graduate education in Sweden.

References

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc.* 10^{th} *CAV*, volume 1427 of *LNCS*, pages 305–318, 1998.

[ABJN99] Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parameterized system verification. In *Proc.* 11^{th} *CAV*, volume 1633 of *LNCS*, pages 134–145, 1999.

[BCMD92] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *Proc. CONCUR '97*. LNCS 1243, 1997.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc.* 8^{th} *CAV*, volume 1102 of *LNCS*, pages 1–12. Springer Verlag, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. of the Fourth International Static Analysis Symposium*, LNCS. Springer Verlag, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. In *Proc. ICALP '97*, volume 1256 of *LNCS*, 1997.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc.* 6^{th} *CAV*, volume 818 of *LNCS*, pages 55–67. Springer Verlag, 1994.

[Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, Nov. 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified model for static analysis of programs by construction or approximation of fixpoints. In *Proc.* 4^{th} *POPL*, pages 238–252, 1977.

[CH78] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78*. ACM, 1978.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV'98*. LNCS 1427, 1998.

[FO97] L. Fribourg and H. Olsén. Reachability sets of parametrized rings as regular languages. In *Proc. 2nd INFINITY'97*, volume 9 of *Electronical Notes in Theoretical Computer Science*. Elsevier Science Publishers, July 1997.

[FWW97] A. Finkel, B. Willems, , and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *Proc. Infinity'97, Electronic Notes in Theoretical Computer Science*, Bologna, Aug. 1997.

[Hal93] N. Halbwachs. Delay Analysis in Synchronous Programs. In *CAV'93*. LNCS 697, 1993.

[HJJ$^+$96] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Proc. TACAS '95,* 1^{th} *TACAS*, volume 1019 of *LNCS*, 1996.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *Proc. TACAS '00,* 6^{th} *TACAS*, LNCS, 2000. to appear.

[KMM$^+$97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proc.* 9^{th} *CAV*, volume 1254, pages 424–435, Haifa, Israel, 1997. Springer Verlag.

[KMMG97] P. Kelb, T. Margaria, M. Mendler, and C. Gsottberger. Mosel: A flexible toolset for monadic second–order logic. In *Proc. TACAS '97, Enschede (NL)*, volume 1217 of *LNCS (LNCS)*, pages 183–202, Heidelberg, Germany, March 1997. Springer–Verlag.

[LHR97] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *24th POPL*, Paris, January 1997.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *Proc.* 12^{th} *CAV*, 2000. In this Volume.

[Sis97] A. Prasad Sistla. Parametrized verification of linear networks using automata as invariants. In O. Grumberg, editor, *Proc.* 9^{th} *CAV*, volume 1254 of *LNCS*, pages 412–423, Haifa, Israel, 1997. Springer Verlag.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc.* 1^{st} *LICS*, pages 332–344, June 1986.

[WB98] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. 10th CAV*, volume 1427 of *LNCS*, pages 88–97, Vancouver, July 1998. Springer Verlag.