Regular Model Checking without Transducers

(On Efficient Verification of Parameterized Systems)

Parosh Aziz Abdulla[1] `parosh@it.uu.se`,
Giorgio Delzanno[2] `giorgio@disi.unige.it`,
Noomene Ben Henda[1] `Noomene.BenHenda@it.uu.se`, and
Ahmed Rezine[1] `Rezine.Ahmed@it.uu.se`

[1] Uppsala University, Sweden
[2] Università di Genova, Italy.

Abstract. We give a simple and efficient method to prove safety properties for parameterized systems with linear topologies. A process in the system is a finite-state automaton, where the transitions are guarded by both local and global conditions. Processes may communicate via broadcast, rendez-vous and shared variables. The method derives an over-approximation of the induced transition system, which allows the use of a simple class of regular expressions as a symbolic representation. Compared to traditional regular model checking methods, the analysis does not require the manipulation of transducers, and hence its simplicity and efficiency. We have implemented a prototype which works well on several mutual exclusion algorithms and cache coherence protocols.

1 Introduction

In this paper, we consider analysis of safety properties for *parameterized systems*. Typically, a parameterized system consists of an arbitrary number of finite-state processes organized in a linear array. The task is to verify correctness of the system regardless of the number of processes inside the system. Examples of parameterized systems include mutual exclusion algorithms, bus protocols, telecommunication protocols, and cache coherence protocols.

One important technique which has been used for verification of parameterized systems is that of *regular model checking* [19, 3, 6]. In regular model checking, states are represented by words, sets of states by regular expressions, and transitions by finite automata operating on pairs of states, so called *finite-state transducers*. Safety properties can be checked through performing reachability analysis, which amounts to applying the transducer relation iteratively to the set of initial states. The main problem with transducer-based techniques is that they are very heavy and usually rely on several layers of computationally expensive automata-theoretic constructions; in many cases severely limiting their applicability. In this paper, we propose a much more light-weight and efficient approach to regular model checking, and describe its application in the context of parameterized systems.

In our framework, a process is modeled as a finite-state automaton which operates on a set of local variables ranging over finite domains. The transitions of the automaton are conditioned by the local state of the process, values of the local variables, and by *global conditions*. A global condition is either *universally* or *existentially* quantified. An example of a universal condition is that all processes to the left of a given process i should satisfy a property θ. Process i is allowed to perform the transition only in the case where all processes with indices $j < i$ satisfy θ. In an existential condition we require that *some* (rather than *all*) processes satisfy θ. In addition, processes may communicate through broadcast, rendez-vous, and shared variables. Finally, processes may dynamically be created and deleted during the execution of the system.

The main idea of our method is to consider a transition relation which is an over-approximation of the one induced by the parameterized system. To do that, we modify the semantics of universal quantifiers by eliminating the processes which violate the given condition. For instance in the above example, process i is always allowed to take the transition. However, when performing the transition, we eliminate all processes which have indices $j < i$ and which violate the condition θ. The approximate transition system obtained in this manner is *monotonic* with respect to the subword relation on configurations (larger configurations are able to simulate smaller ones). In fact, it turns out that universal quantification is the only operation which does not preserve monotonicity and hence it is the only source of approximation in the model. Since the approximate transition relation is monotonic, it can be analyzed using symbolic backward reachability algorithm based on a generic method introduced in [1]. An attractive feature of this algorithm is that it operates on sets of configurations which are upward closed with respect to the subword relation. In particular, reachability analysis can be performed by computing predecessors of upward closed sets, which is much simpler and more efficient than applying transducer relations on general regular languages. Also, as a side effect, the analysis of the approximate model is always guaranteed to terminate. This follows from the fact that the subword relation on configurations is a *well quasi-ordering*. The whole verification process is fully automatic since both the approximation and the reachability analysis are carried out without user intervention. Observe that if the approximate transition system satisfies a safety property then we can safely conclude that the original system satisfies the property, too.

To simplify the presentation, we introduce the class of systems we consider in a stepwise manner. First, we consider a basic model where we only allow Boolean local variables together with local and global conditions. We describe how to derive the approximate transition relation and how to analyze safety properties for the basic model. Then, we introduce the additional features one by one. This includes using general finite domains, shared variables, broadcast and rendez-vous communication, dynamic creation and deletion of processes, and counters. For each new feature, we describe how to extend the approximate transition relation and the reachability algorithm in a corresponding manner.

Based on the method, we have implemented a prototype which works well on several mutual exclusion algorithms and cache coherence protocols, such as the Bakery and Szymanski algorithms, the Java Meta-locking protocol, the Futurebus+ protocol, German's directory-based protocol, etc.

Related work Several recent works have been devoted to develop regular model checking, e.g., [19, 9]; and in particular augmenting regular model checking with techniques such as widening [6, 27], abstraction [7], and acceleration [3]. All these works rely on computing the transitive closure of transducers or on iterating them on regular languages.
A technique of particular interest for parameterized systems is that of *counter abstraction*. The idea is to keep track of the number of processes which satisfy a certain property. In [15] the technique generates an abstract system which is essentially a Petri net. Counter abstracted models with *broadcast communication* are proved to be *well-structured* in [14]. In [10, 11] symbolic model checking based on real arithmetics is used to verify counter abstracted models of cache coherence protocols enriched with global conditions. The method works without guarantee of termination. The paper [24] refines the counter abstraction idea by truncating the counters at the value of 2, and thus obtains a finite-state abstract system. The method may require manual insertion of auxiliary program variables for programs that exploit knowledge of process identifiers (examples of such programs are the mutual exclusion protocols we consider in this paper). In general, counter abstraction is designed for systems with unstructured or clique architectures. Our method can cope with this kind of systems, since unstructured architectures can be viewed as a special case of linear arrays where the ordering of the processes is not relevant. In [18] and [26], the authors present a tool for the analysis and the verification of *linear parameterized hardware systems* using the *monadic second-order logic on strings*.

Other parameterized verification methods are based on reductions to finite-state models. Among these, the *invisible invariants* method [4, 23] exploits *cut-off* properties to check invariants for mutual exclusion protocols like the Bakery algorithm and German's protocol. The success of the method depends on the heuristic used in the generation of the candidate invariant. This method sometimes (e.g. for German's protocol) requires insertion of auxiliary program variables for completing the proof. In [5] finite-state abstractions for verification of systems specified in WS1S are computed on-the-fly by using the weakest precondition operator. The method requires the user to provide a set of predicates on which to compute the abstract model. Heuristics to discover *indexed predicates* are proposed in [20] and applied to German's protocol as well as to the Bakery algorithm. In contrast to these approaches, we provide a uniform approximation scheme which is independent on the analyzed system. *Environment abstraction* [8] combines predicate abstraction with the counter abstraction. The technique is applied to the Bakery and Szymanski algorithms. The model of [8] contains a more restricted form of global conditions than ours, and also does not include features such as broadcast communication, rendez-vous communication, and dynamic behaviour. Other approaches tailored to snoopy cache protocols modeled with broadcast communication are presented in [13, 21]. In [12] German's directory-based protocol is verified via a manual transformation into a snoopy protocol. It is important to remark that frameworks for finite-state abstractions [8] and those based on cutoff properties [4, 23] can be applied to parameterized systems where each component itself contains counters and other unbounded data structures. This allows for instance to deal with a model of the Bakery algorithm which is more concrete (precise) than ours.

Finally, in [25] a parameterized version of the Java Meta-locking algorithm is verified by means of an induction-based proof technique which requires manual strengthening of the mutual exclusion invariant.

In summary, our method provides a uniform simple abstraction which allows fully automatic verification of a wide class of systems. We have been able to verify all benchmarks available to us from the literature (with the exception of the Bakery protocol, where we can only model an abstraction of the protocol). The benchmarks include some programs, e.g. the German protocol and Java Meta-locking algorithm, which (to our knowledge) have previously not been possible to verify without user interaction or specialized heuristics. On the negative side, the current method only allows the verification of safety properties, while most regular model checking and abstraction-based techniques can also handle liveness properties.

Outline In the next Section we give some preliminaries and define a basic model for parameterized systems. Section 3 describes the induced transition system and introduces the coverability (safety) problem. In Section 4 we define the over-approximated transition system on which we run our technique. Section 5 presents a generic scheme for deciding coverability. In Section 6 we instantiate the scheme on the approximate transition system. Section 7 explains how we extend the basic model to cover features such as shared variables, broadcast and binary communications, and dynamic creation and deletion of processes. In Section 8 we report the results of our prototype on a number of mutual exclusion and cache coherence examples. Finally, in Section 9, we give conclusions and directions for future work. A detailed description of the case studies can be found in [2].

2 Preliminaries

In this section, we define a basic model of parameterized systems. This model will be enriched by additional features in Section 7.

For a natural number n, let $\overline{n}$ denote the set $\{1, \ldots, n\}$. We use $\mathcal{B}$ to denote the set $\{true, false\}$ of Boolean values. For a finite set A, we let $\mathbb{B}(A)$ denote the set of formulas which have members of A as atomic formulas, and which are closed under the Boolean connectives $\neg, \wedge, \vee$. A *quantifier* is either *universal* or *existential*. A universal quantifier is of one of the forms $\forall_{LR}$, $\forall_L$, $\forall_R$. An existential quantifier is of one of the forms $\exists_L$, $\exists_R$, or $\exists_{LR}$. The subscripts L, R, and LR stand for *Left*, *Right*, and *Left-Right* respectively. A *global condition* over A is of the form $\Box\theta$ where $\Box$ is a quantifier and $\theta \in \mathbb{B}(A)$. A global condition is said to be *universal* (resp. *existential*) if its quantifier is *universal* (resp. *existential*). We use $\mathbb{G}(A)$ to denote the set of global conditions over A.

Parameterized Systems A parameterized system consists of an arbitrary (but finite) number of identical processes, arranged in a linear array. Each process is a finite-state automaton which operates on a finite number of Boolean local variables. The transitions of the automaton are conditioned by the values of the local variables and by *global* conditions in which the process checks, for instance, the local states and variables of all processes to its left or to its right. A transition may change the value of any local variable inside the process. A parameterized system induces an infinite family of finite-state systems, namely one for each size of the array. The aim is to verify correctness of the systems for the whole family (regardless of the number of processes inside the system).

A *parameterized system* $\mathcal{P}$ is a triple (Q, X, T), where Q is a set of *local states*, X is a set of *local variables*, and T is a set of *transition rules*. A transition rule t is of the form

$$t : \begin{bmatrix} q \\ grd \rightarrow stmt \\ q' \end{bmatrix} \tag{1}$$

where $q, q' \in Q$ and $grd \rightarrow stmt$ is a *guarded command*. Below we give the definition of a guarded command. A *guard* is a formula $grd \in \mathbb{B}(X) \cup \mathbb{G}(X \cup Q)$. In other words, the guard grd constraints either the values of local variables inside the process (if $grd \in \mathbb{B}(X)$); or the local states and the values of local variables of other processes (if $grd \in \mathbb{G}(X \cup Q)$). A *statement* is a set of assignments of the form $x_1 = e_1; \ldots ; x_n = e_n$, where $x_i \in X$, $e_i \in \mathcal{B}$, and $x_i \neq x_j$ if $i \neq j$. A *guarded command* is of the form $grd \rightarrow stmt$, where grd is a guard and $stmt$ is a statement.

Remark We can extend the definition of the transition rule in (1) so that the grd is a *conjunction* of formulas in $\mathbb{B}(X) \cup \mathbb{G}(X \cup Q)$. All the definitions and algorithms which are later presented in this paper can easily be extended to the more general form. However, for simplicity of presentation, we only deal with the current form.

3 Transition System

In this section, we first describe the transition system induced by a parameterized system. Then we introduce the *coverability problem*.

Transition System A *transition system* $\mathcal{T}$ is a pair $(D, \Longrightarrow)$, where D is an (infinite) set of *configurations* and $\Longrightarrow$ is a binary relation on D. We use $\overset{*}{\Longrightarrow}$ to denote the reflexive transitive closure of $\Longrightarrow$. We will consider several transition systems in this paper.

First, a parameterized system $\mathcal{P} = (Q, X, T)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$ as follows. A configuration is defined by the local states of the processes, and by the values of the local variables. Formally, a *local variable state* v is a mapping from X to $\mathcal{B}$. For a local variable state v, and a formula $\theta \in \mathbb{B}(X)$, we evaluate $v \models \theta$ using the standard interpretation of the Boolean connectives. A *process state* u is a pair (q, v) where $q \in Q$ and v is a local variable state. Sometimes, abusing notation, we view a process state (q, v) as a mapping $u : X \cup Q \mapsto \mathcal{B}$, where $u(x) = v(x)$ for each $x \in X$, $u(q) = \mathit{true}$, and $u(q') = \mathit{false}$ for each $q' \in Q - \{q\}$. The process state thus agrees with v on the values of local variables, and maps all elements of Q, except q, to *false*. For a formula $\theta \in \mathbb{B}(X \cup Q)$ and a process state u, the relation $u \models \theta$ is then well-defined. This is true in particular if $\theta \in \mathbb{B}(X)$.

A *configuration* $c \in C$ is a sequence $u_1 \cdots u_n$ of process states. Intuitively, the above configuration corresponds to an instance of the system with n processes. Each pair $u_i = (q_i, v_i)$ gives the local state and the values of local variables of process i. Notice that if c_1 and c_2 are configurations then their concatenation $c_1 \bullet c_2$ is also a configuration.

Next, we define the transition relation $\longrightarrow$ on the set of configurations as follows. We will define the semantics of global conditions in terms of two quantifiers $\forall$ and $\exists$. For a configuration $c = u_1 \cdots u_n$ and a formula $\theta \in \mathbb{B}(X \cup Q)$, we write $c \models \forall\theta$ if $u_i \models \theta$ for each $i : 1 \leq i \leq n$; and write $c \models \exists\theta$ if $u_i \models \theta$ for some $i : 1 \leq i \leq n$. For a statement *stmt* and a local variable state v, we use $\mathit{stmt}(v)$ to denote the local variable state v' such that $v'(x) = v(x)$ if x does not occur in *stmt*; and $v'(x) = e$ if $x = e$ occurs in *stmt*. Let t be a transition rule of the form of (1). Consider two configurations $c = c_1 \bullet u \bullet c_2$ and $c' = c_1 \bullet u' \bullet c_2$, where $u = (q, v)$ and $u' = (q', v')$. We write $c \xrightarrow{t} c'$ to denote that the following three conditions are satisfied:

1. If $\mathit{grd} \in \mathbb{B}(X)$ then $v \models \mathit{grd}$, i.e., the local variables of the process in transition should satisfy *grd*.
2. If $\mathit{grd} = \Box\theta \in \mathbb{G}(X \cup Q)$ then one of the following conditions is satisfied:
   - $\Box = \forall_L$ and $c_1 \models \forall\theta$.
   - $\Box = \forall_R$ and $c_2 \models \forall\theta$.
   - $\Box = \forall_{LR}$ and $c_1 \models \forall\theta$ and $c_2 \models \forall\theta$.
   - $\Box = \exists_L$ and $c_1 \models \exists\theta$.
   - $\Box = \exists_R$ and $c_2 \models \exists\theta$.
   - $\Box = \exists_{LR}$ and either $c_1 \models \exists\theta$ or $c_2 \models \exists\theta$.

   In other words, if *grd* is a global condition then the rest of the processes should satisfy θ (in a manner which depends on the type of the quantifier).
3. $v' = \mathit{stmt}(v)$.

We use $c \longrightarrow c'$ to denote that $c \xrightarrow{t} c'$ for some $t \in T$.

Safety Properties In order to analyze safety properties, we study the *coverability problem* defined below. Given a parameterized system $\mathcal{P} = (Q, X, T)$, we assume that, prior to starting the execution of the system, each process is in an (identical) *initial* process state $u_{init} = (q_{init}, v_{init})$. In the induced transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$, we use *Init* to denote the set of *initial* configurations, i.e., configurations of the form $u_{init} \cdots u_{init}$ (all processes are in their initial states). Notice that this set is infinite.

We define an ordering on configurations as follows. Given two configurations, $c = u_1 \cdots u_m$ and $c' = u'_1 \cdots u'_n$, we write $c \preceq c'$ to denote the existance of a strictly monotonic[3] injection h from $\overline{m}$ to $\overline{n}$ such that $u_i = u'_{h(i)}$ for each $i : 1 \leq i \leq m$. A set of configurations $D \subseteq C$ is *upward closed* (with respect to $\preceq$) if $c \in D$ and $c \preceq c'$ implies $c' \in D$. For sets of configurations $D, D' \subseteq C$ we use $D \longrightarrow D'$ to denote that there are $c \in D$ and $c' \in D'$ with $c \longrightarrow c'$. The *coverability problem* for parameterized systems is defined as follows:

PAR-COV
Instance
- A parameterized system $\mathcal{P} = (Q, X, T)$.
- An upward closed set C_F of configurations.

Question *Init* $\xrightarrow{*} C_F$?

It can be shown, using standard techniques (see e.g. [28, 16]), that checking safety properties (expressed as regular languages) can be translated into instances of the coverability problem. Therefore, checking safety properties amounts to solving PAR-COV(i.e., to the reachability of upward closed sets).

4 Approximation

In this section, we introduce an over-approximation of the transition relation of a parameterized system.

In Section 3, we mentioned that each parameterized system $\mathcal{P} = (Q, X, T)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$. A parameterized system $\mathcal{P}$ also induces an *approximate* transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$, where the set C of configurations is identical to the one in $\mathcal{T}(\mathcal{P})$. We define $\rightsquigarrow = (\longrightarrow \cup \rightsquigarrow_1)$, where $\longrightarrow$ is the transition relation defined in Section 3, and $\rightsquigarrow_1$, which reflects the approximation of universal quantifiers, is defined as follows. For a configuration c, and a formula $\theta \in \mathbb{B}(X \cup Q)$, we use $c \ominus \theta$ to denote the maximal configuration c' (with respect to $\preceq$) such that $c' \preceq c$ and $c' \models \forall\theta$. In other words, we derive c' from c by deleting all process states which do not satisfy θ. Consider two configurations $c = c_1 \bullet u \bullet c_2$ and $c' = c'_1 \bullet u' \bullet c'_2$, where $u = (q, v)$ and $u' = (q', v')$. Let t be a transition rule of the form of (1), such that $grd = \Box\theta$ is a universal global condition. We write $c \overset{t}{\rightsquigarrow}_1 c'$ to denote that the following conditions are satisfied:

1. if $\Box = \forall_L$, then $c'_1 = c_1 \ominus \theta$ and $c'_2 = c_2$.
2. if $\Box = \forall_R$, then $c'_1 = c_1$ and $c'_2 = c_2 \ominus \theta$.
3. if $\Box = \forall_{LR}$, then $c'_1 = c_1 \ominus \theta$ and $c'_2 = c_2 \ominus \theta$.

[3] $h : \overline{m} \to \overline{n}$ strictly monotonic means: $i < j \Rightarrow h(i) < h(j)$ for all $i, j : 1 \leq i < j \leq m$.

4. $v' = stmt(v)$.

We use $c \rightsquigarrow c'$ to denote that $c \overset{t}{\rightsquigarrow} c'$ for some $t \in T$. We define the coverability problem for the approximate system as follows:

APRX-PAR-COV
Instance
- A parameterized system $\mathcal{P} = (Q, X, T)$.
- An upward closed set C_F of configurations.

Question $Init \overset{*}{\rightsquigarrow} C_F$?

Since $\longrightarrow \subseteq \rightsquigarrow$, a negative answer to APRX-PAR-COV implies a negative answer to PAR-COV.

5 Generic Scheme

In this section, we recall a generic scheme from [1] for performing symbolic backward reachability analysis.

Assume a transition system $(D, \Longrightarrow)$ with a set $Init$ of initial states. We will work with a set of constraints defined over D. A *constraint* ϕ denotes a potentially infinite set of configurations (i.e. $[\![\phi]\!] \subseteq D$). For a finite set Φ of constraints, we let $[\![\Phi]\!] = \bigcup_{\phi \in \Phi} [\![\phi]\!]$.

We define an *entailment relation* $\sqsubseteq$ on constraints, where $\phi_1 \sqsubseteq \phi_2$ iff $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$. For sets Φ_1, Φ_2 of constraints, abusing notation, we let $\Phi_1 \sqsubseteq \Phi_2$ denote that for each $\phi_2 \in \Phi_2$ there is a $\phi_1 \in \Phi_1$ with $\phi_1 \sqsubseteq \phi_2$. Notice that $\Phi_1 \sqsubseteq \Phi_2$ implies that $[\![\Phi_2]\!] \subseteq [\![\Phi_1]\!]$ (although the converse is not true in general).

For a constraint ϕ, we let $Pre(\phi)$ be a finite set of constraints, such that $[\![Pre(\phi)]\!] = \{c | \; \exists c' \in [\![\phi]\!] \, . \, c \Longrightarrow c'\}$. In other words $Pre(\phi)$ characterizes the set of configurations from which we can reach a configuration in ϕ through the application of a single transition rule. For our class of systems, we will show that such a set always exists and is in fact computable. For a set Φ of constraints, we let $Pre(\Phi) = \bigcup_{\phi \in \Phi} Pre(\phi)$. Below we present a scheme for a symbolic algorithm which, given a finite set Φ_F of constraints, checks whether $Init \overset{*}{\Longrightarrow} [\![\Phi_F]\!]$.

In the scheme, we perform a backward reachability analysis, generating a sequence $\Phi_0 \sqsupseteq \Phi_1 \sqsupseteq \Phi_2 \sqsupseteq \cdots$ of finite sets of constraints such that $\Phi_0 = \Phi_F$, and $\Phi_{j+1} = \Phi_j \cup Pre(\Phi_j)$. Since $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$, the procedure terminates when we reach a point j where $\Phi_j \sqsubseteq \Phi_{j+1}$. Notice that the termination condition implies that $[\![\Phi_j]\!] = (\bigcup_{0 \leq i \leq j} [\![\Phi_i]\!])$. Consequently, Φ_j characterizes the set of all predecessors of $[\![\Phi_F]\!]$. This means that $Init \overset{*}{\Longrightarrow} [\![\Phi_F]\!]$ iff $(Init \bigcap [\![\Phi_j]\!]) \neq \emptyset$.

Observe that, in order to implement the scheme (i.e., transform it into an algorithm), we need to be able to (i) compute Pre; (ii) check for entailment between constraints; and (iii) check for emptiness of $Init \bigcap [\![\phi]\!]$ for a given constraint ϕ. A constraint system satisfying these three conditions is said to be *effective*. Moreover, in [1], it is shown that termination is guaranteed in case the constraint system is *well quasi-ordered (WQO)* with respect to $\sqsubseteq$, i.e., for each infinite sequence $\phi_0, \phi_1, \phi_2, \ldots$ of constraints, there are $i < j$ with $\phi_i \sqsubseteq \phi_j$.

6 Algorithm

In this section, we instantiate the scheme of Section 5 to derive an algorithm for solving APRX-PAR-COV. We do that by introducing an effective and well quasi-ordered constraint system.

Throughout this section, we assume a parameterized system $\mathcal{P} = (Q, X, T)$ and the induced approximate transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$. We define a constraint to be a finite sequence $\theta_1 \cdots \theta_m$ where $\theta_i \in \mathbb{B}(X \cup Q)$. Observe that for any constraints ϕ_1 and ϕ_2, their concatenation $\phi_1 \bullet \phi_2$ is also a constraint. For a constraint $\phi = \theta_1 \cdots \theta_m$ and a configuration $c = u_1 \cdots u_n$, we write $c \models \phi$ to denote that there is a strictly monotonic injection h from $\overline{m}$ to $\overline{n}$ such that $u_{h(i)} \models \theta_i$ for each $i : 1 \leq i \leq m$. Given a constraint ϕ, we let $[\![\phi]\!] = \{c \in C | \; c \models \phi\}$. Notice that if $\phi = \theta_1 \cdots \theta_m$ and some θ_i is unsatisfiable then $[\![\phi]\!]$ is empty. Such a constraint can therefore be safely discarded in the algorithm.

An aspect of our constraint system is that each constraint characterizes a set of configurations which is upward closed with respect to $\preceq$. Conversely (by Higman's Lemma [17]), any upward closed set C_F of configurations can be characterized as $[\![\Phi_F]\!]$ where Φ_F is a finite set of constraints. In this manner, APRX-PAR-COV is reduced to checking the reachability of a finite set of constraints.

Below we show effectiveness and well quasi-ordering of our constraint system, meaning that we obtain an algorithm for solving APRX-PAR-COV.

Pre For a constraint ϕ', we define $Pre(\phi') = \bigcup_{t \in T} Pre_t(\phi')$, i.e., we compute the set of predecessor constraints with respect to each transition rule $t \in T$. In the following, assume t to be a transition rule of the form (1). To compute $Pre_t(\phi')$, we define first the function $[t]$ on $X \cup Q$ as follows: for each $x \in X$, $[t](x) = stmt(x)$ if x occurs in $stmt$ and $[t](x) = x$ otherwise. For each $q'' \in Q$, $[t](q'') = true$ if $q'' = q'$, and $false$ otherwise. For $\theta \in \mathbb{B}(X \cup Q)$, we use $\theta[t]$ to denote the formula obtained from θ by substituting all occurrences of elements in θ by their corresponding $[t]$-images.

Now, we define two operators, $\otimes$ and $\oplus$, which we use to capture the effects of universal and existential quantifiers when computing Pre. We use $\otimes$ to handle universal quantifiers. For a constraint $\phi = \theta_1 \cdots \theta_m$ and a $\theta \in \mathbb{B}(X \cup Q)$, we define $\phi \otimes \theta$ to be the constraint $(\theta_1 \wedge \theta) \cdots (\theta_m \wedge \theta)$. We use $\oplus$ to deal with existential quantifiers. For a constraint $\phi = \theta_1 \cdots \theta_m$ and a $\theta \in \mathbb{B}(X \cup Q)$, we define $\phi \oplus \theta$ to be the set of constraints which are of one of the following forms:

- $\theta_1 \cdots \theta_{i-1}(\theta_i \wedge \theta)\theta_{i+1} \cdots \theta_m$ where $1 \leq i \leq m$; or
- $(\theta_1 \wedge \neg\theta) \cdots (\theta_i \wedge \neg\theta)\theta(\theta_{i+1} \wedge \neg\theta) \cdots (\theta_m \wedge \neg\theta)$ where $0 \leq i \leq m + 1$.

In the first case, the constraint implies that there is at least one process satisfying θ. In the the second case, the constraint does not imply the existence of such a process, and therefore the formula θ is added explicitly to the representation of the constraint. Notice that in the second case the length of the resulting constraint is larger (by one) than the length of ϕ. This means that the lengths of the constraints which arise during the analysis are not *a priori* fixed. Nevertheless, termination is still guaranteed by well quasi-ordering of the constraints.

For a constraint ϕ' and a rule t of the form (1), we define $Pre_t(\phi')$ to be the set of all constraints ϕ such that ϕ (resp. ϕ') is of the form $\phi_1 \bullet \theta \bullet \phi_2$ (resp. $\phi'_1 \bullet \theta' \bullet \phi'_2$) and the following conditions are satisfied:

- If $grd \in \mathbb{B}(X)$ (i.e. grd is a local condition), then $\theta = \theta'[t] \wedge grd \wedge q$, $\phi_1 = \phi'_1$ and $\phi_2 = \phi'_2$;
- If $grd = \Box grd'$, where $grd' \in \mathbb{B}(X \cup Q)$, then $\theta = \theta'[t] \wedge q$ and depending on $\Box$ the following conditions hold:
  - If $\Box = \forall_L$ then $\phi_1 = \phi'_1 \otimes grd'$ and $\phi_2 = \phi'_2$.
  - If $\Box = \forall_R$ then $\phi_1 = \phi'_1$ and $\phi_2 = \phi'_2 \otimes grd'$.
  - If $\Box = \forall_{LR}$ then $\phi_1 = \phi'_1 \otimes grd'$ and $\phi_2 = \phi'_2 \otimes grd'$.
  - If $\Box = \exists_L$ then $\phi_1 \in \phi'_1 \oplus grd'$ and $\phi_2 = \phi'_2$.
  - If $\Box = \exists_R$ then $\phi_1 = \phi'_1$ and $\phi_2 \in \phi'_2 \oplus grd'$.
  - If $\Box = \exists_{LR}$ then either $\phi_1 \in \phi'_1 \oplus grd'$ and $\phi_2 = \phi'_2$; or $\phi_1 = \phi'_1$ and $\phi_2 \in \phi'_2 \oplus grd'$.

Entailment The following Lemma gives a syntactic characterization which allows computing of the entailment relation.

Lemma 1. *For constraints $\phi = \theta_1 \ldots \theta_m$ and $\phi' = \theta'_1 \ldots \theta'_n$, we have $\phi \sqsubseteq \phi'$ iff there exists a strictly monotonic injection $h : \overline{m} \to \overline{n}$ such that $\theta'_{h(i)} \Rightarrow \theta_i$ for each $i \in \overline{m}$.*

Proof. ($\Rightarrow$) Assume there is no such injection. We derive a configuration c such that $c \in [\![\phi']\!]$ and $c \notin [\![\phi]\!]$. To do that, we define the function g on $\overline{n}$ as follows: $g(1) = 1$, $g(i+1) = g(i)$ if $\theta'_i \not\Rightarrow \theta_{g(i)}$, and $g(i+1) = g(i) + 1$ if $\theta'_i \Rightarrow \theta_{g(i)}$. Observe that, since the above mentioned injection does not exist, we have either $g(n) < m$, or $g(n) = m$ and $\theta'_n \not\Rightarrow \theta_m$. We choose $c = u_1 \cdots u_n$, where u_i is defined as follows: (i) if $\theta'_i \not\Rightarrow \theta_{g(i)}$ let u_i be any process state such that $u_i \models \neg\theta_{g(i)} \wedge \theta'_i$; and (ii) if $\theta'_i \Rightarrow \theta_{g(i)}$ let u_i be any process state such that $u_i \models \theta'_i$.

($\Leftarrow$) Assume there exists a strictly monotonic injection $h : \overline{m} \to \overline{n}$ such that $\theta'_{h(i)} \Rightarrow \theta_i$ for each $i \in \overline{m}$. Let $c = u_1 \ldots u_p$ be a configuration in $[\![\phi']\!]$. It follows that there exists a strictly monotonic injection $h' : \overline{n} \to \overline{p}$ such that $u_{h'(i)} \models \theta'_i$ for each $i \in \overline{n}$. By assumption, for each $j \in \overline{m}$, we have $\theta'_{h(j)} \Rightarrow \theta_j$. Therefore, for each $j \in \overline{m}$, $u_{h' \circ h(j)} \models \theta_j$. It is straightforward to check that $h' \circ h$ is a strictly monotonic injection from $\overline{m}$ to $\overline{p}$. It follows that $c \in [\![\phi]\!]$.

Intersection with Initial States For a constraint $\phi = \theta_1 \ldots \theta_n$, we have $(Init \bigcap [\![\phi]\!]) = \emptyset$ iff $u_{init} \not\models \theta_i$ for some $i \in \overline{n}$.

Termination We show that the constraint system is *well quasi-ordered (WQO)* with respect to $\sqsubseteq$. $(A, \preceq)$ is obviously a *WQO* for any finite set A and any *quasi-order* $\preceq$ on A. Let A^* be the set of words over A, and $\preceq^*$ be the subword relation. Higman's Lemma [17] states that $(A^*, \preceq^*)$ is also a WQO. Take A to be the quotient sets of $\mathbb{B}(X \cup Q)$ under the equivalence relation. Let $\preceq$ be the implication relation on formulas in $\mathbb{B}(X \cup Q)$. By lemma 1, the relation $\sqsubseteq$ coincides with $\preceq^*$. We conclude that the constraint system is a WQO.

7 Extensions

In this section, we add a number of features to the model of Section 2. For each additional feature, we show how to modify the constraint system of Section 6 in a corresponding manner.

Shared Variables We assume the presence of a finite set S of Boolean *shared variables* that can be read and written by all processes in the system. A guard may constraint the values of both the shared and the local variables, and a statement may assign values to the shared variables (together with the local variables). It is straightforward to extend the definitions of the induced transition system and the symbolic algorithm to deal with shared variables.

Variables over Finite Domains Instead of Boolean variables, we can use variables which range over arbitrary finite domains. Below we describe an example of such an extension. Let Y be a finite set of variables which range over $\{0, 1, \ldots, k\}$, for some natural number k. Let $\mathbb{N}(A)$ be the set of formulas of the form $x \sim y$ where $\sim \in \{<, \leq, =, \neq, >, \geq\}$ and $x, y \in Y \cup \{0, 1, \ldots, k\}$. A guard is a formula $grd \in \mathbb{B}(X \cup \mathbb{N}(Y)) \cup \mathbb{G}(X \cup Q \cup \mathbb{N}(Y))$. In other words, the guard grd may also constraint the values of the variables in Y. Similarly, a statement may assign values in $\{0, 1, \ldots, k\}$ to variables in Y. A local variable state is a mapping from $X \cup Y$ to $\mathcal{B} \cup \{0, 1, \ldots, k\}$ respecting the types of the variables. The definitions of configurations, the transition relation, and constraints are extended in the obvious manner. Well quasi-ordering of the constraint system follows in a similar manner to Section 6, using the fact that variables in Y range over finite domains.

Broadcast In a broadcast transition, an arbitrary number of processes change states simultaneously. A *broadcast rule* is a sequence of transition rules of the following form

$$\begin{bmatrix} q_0 \\ grd_0 \rightarrow stmt_0 \\ q'_0 \end{bmatrix} \begin{bmatrix} q_1 \\ grd_1 \rightarrow stmt_1 \\ q'_1 \end{bmatrix}^* \begin{bmatrix} q_2 \\ grd_2 \rightarrow stmt_2 \\ q'_2 \end{bmatrix}^* \cdots \begin{bmatrix} q_m \\ grd_m \rightarrow stmt_m \\ q'_m \end{bmatrix}^* \quad (2)$$

where $grd_i \in \mathbb{B}(X)$ for each $i : 0 \leq i \leq m$. Below, we use t_i to refer to the i^{th} rule in the above sequence. The broadcast rule is deterministic in the sense that either $grd_i \wedge grd_j$ is not satisfiable or $q_i \neq q_j$ for each $i, j : 1 \leq i \neq j \leq m$. The broadcast is initiated by a process, called the *initiator*, which is represented by t_0 (i.e., the leftmost transition rule). This transition rule has the same interpretation as in Section 2. That is, in order for the broadcast transition to take place, the initiator should be in local state q_0 and its local variables should satisfy the guard grd_0. After the completion of the broadcast, the initiator has changed state to q'_0 and updated its local variables according to $stmt_0$. Together with the initiator, an arbitrary number of processes, called the *receptors*, change state simultaneously. The receptors are modeled by the transition rules $t_1, \ldots, t_m$ (each rule being marked by a * to emphasize that an arbitrary number of receptors may execute that rule). More precisely, if the local state of a process is q_i and its local variables satisfy grd_i, then the process changes its local state to q'_i and updates its local variables according to $stmt_i$. Notice that since the broadcast rule is deterministic, a receptor satisfies the precondition of at most one of the transition rules. Processes which do not satisfy the precondition of any of the transition rules remain passive during the broadcast. We define a transition relation $\longrightarrow_B$ to reflect broadcast transitions. The definition of $\longrightarrow_B$ can be derived in a straightforward manner from the above informal description. We extend the transition relation $\longrightarrow$ defined in Section 3, by taking its union with $\longrightarrow_B$. In a similar manner, we extend the approximate transition relation $\rightsquigarrow$ (defined in Section 4) by taking its union with $\longrightarrow_B$. This means that the introduction of broadcast transitions are interpreted exactly, and thus they do not add any additional approximation to $\rightsquigarrow$.

We use the same constraint system as the one defined for systems without broadcast; consequently checking entailment, checking intersection with initial states, and proving termination are identical to Section 6. Below we show how to compute *Pre*. Consider a constraint $\phi' = \theta'_1 \cdots \theta'_n$ and a broadcast rule b of the above form. We define $Pre_b(\phi')$ to be the set of all constraints of the form $\theta_1 \cdots \theta_n$ such that there is $i : 1 \leq i \leq n$ and the following properties are satisfied:

- $\theta_i = \theta'_i[t_0] \wedge grd_0 \wedge q_0$. This represents the predecessor state of the initiator.
- For each $j : 1 \leq j \neq i \leq n$, one of the following properties is satisfied:
  - $\theta_j = \theta'_j \wedge \neg((q_1 \wedge grd_1) \vee (q_2 \wedge grd_2) \vee \cdots \vee (q_m \wedge grd_m))$. This represents a passive process (a process other than the initiator, is allowed to be passive if it does not satisfy the preconditions of any of the rules).
  - $\theta_j = \theta'_j[t_k] \wedge grd_k \wedge q_k$, for some $k : 1 \leq k \leq m$. This represents a receptor.

Binary Communication In *binary communication* two processes perform a *rendez-vous* changing states simultaneously. A rendez-vous rule consists of two transition rules of the from

$$\begin{bmatrix} q_1 \\ grd_1 \rightarrow stmt_1 \\ q'_1 \end{bmatrix} \begin{bmatrix} q_2 \\ grd_2 \rightarrow stmt_2 \\ q'_2 \end{bmatrix} \qquad (3)$$

where $grd_1, grd_2 \in \mathbb{B}(X)$. Binary communication can be treated in a similar manner to broadcast transitions (here there is exactly one receptor). The model definition and the symbolic algorithm can be extended in a corresponding way.

Dynamic Creation and Deletion We allow dynamic creation and deletion of processes. A *process creation* rule is of the form

$$\begin{bmatrix} \cdot \\ grd \rightarrow \cdot \\ q' \end{bmatrix} \qquad (4)$$

where $q' \in Q$ and $grd \in \mathbb{B}(X)$. The rule creates a new process whose local state is q' and whose local variables satisfy grd. The newly created processes may be placed anywhere inside the array of processes.

We define a transition relation $\longrightarrow_D$ to reflect process creation transitions as follows. For configurations c and c', and a process creation rule d of the form of (4), we define $c \xrightarrow{d}_D c'$ to denote that c' is of the form $c'_1 \bullet u' \bullet c'_2$ where $c = c'_1 \bullet c'_2$, $u' = (q', v')$ and $v' \models grd$. We use the same constraint system as the one defined for systems without process creation and deletion. We show how to compute *Pre*. Consider a constraint ϕ' and a creation rule d of the form of (4). We define $Pre_d(\phi')$ to be the set of all constraints ϕ such that ϕ' (resp. ϕ) is of the form $\phi'_1 \bullet \theta' \bullet \phi'_2$ (resp. $\phi'_1 \bullet \phi'_2$) and $\theta'[t] \wedge grd$ is satisfiable. Notice that $\theta'[t]$ does not change the values of the local variables in θ'.

A *process deletion* rule is of the form

$$\begin{bmatrix} q \\ grd \rightarrow \cdot \\ \cdot \end{bmatrix} \tag{5}$$

where $q \in Q$ and $grd \in \mathbb{B}(X)$. The rule deletes a single process whose local state is q provided that the guard grd is satisfied. The definitions of the transition system and the symbolic algorithm can be extended in a similarly to the case with process creation rules. We omit the details here due to shortage of space.

Counters Using deletion, creation, and universal conditions we can simulate counters, i.e., global unbounded variables which range over the natural numbers. For each counter c, we use a special local state q_c, such that the value of c is encoded by the number of occurrences of q_c in the configuration. Increment and decrement operations can be simulated using creation and deletion of processes in local state q_c. Zero-testing can be simulated through universal conditions. More precisely, $c = 0$ is equivalent to the condition that there is no process in state q_c. This gives a model which is as powerful as Petri nets with inhibitor arcs (or equivalently counter machines). Observe that the approximation introduced by the universal condition means that we replace zero-testing (in the original model) by resetting the counter value to zero (in the approximate model). Thus, we are essentially approximating the counter machine by the corresponding lossy counter machine (see [22] for a description of lossy counter machines). In fact, we can equivalently add counters as a separate feature (without simulation through universal conditions), and approximate zero-testing by resetting as described above.

8 Experimental Results

Based on our method, we have implemented a prototype tool and run it on a collection of mutual exclusion and cache coherence protocols. The results, using a Pentium M 1.6 Ghz with 1G of memory, are summarized in Tables 1 and 2. For each of the mutual exclusion protocols, we consider two variants; namely one with dynamic creation and deletion of processes (marked with a * in Table 1), and one without. Full details of the examples can be found in [2]. For each example, we give the number of iterations performed by the reachability algorithm, the

	# iter	# constr	t(ms)
Bakery	2	2	4
Bakery*	2	2	4
Burns	14	71	230
Burns*	9	21	32
Java M-lock	5	24	30
Java M-lock*	5	17	30
Dijkstra	13	150	1700
Dijkstra*	8	57	168
Szymanski	17	334	3880
Szymanski*	17	334	4080

Table 1. Mutual exclusion algorithms

	# iter	# constr	t(ms)
Synapse	3	3	4
Berkeley	2	6	8
Mesi	3	8	8
Moesi	1	12	12
Dec Firefly	3	11	16
Xerox P.D	3	20	52
Illinois	5	33	80
Futurebus	7	153	300
German	44	14475	3h45mn

Table 2. Cache coherence protocols

largest number of constraints maintained at any point during the execution of the algorithm, and the time (in milliseconds). The computation for each example required less than 15MB of memory.

9 Conclusion and Future Work

We have presented a method for verification of parameterized systems where the components are organized in a linear array. We derive an over-approximation of the transition relation which allows the use of symbolic reachability analysis defined on upward closed sets of configurations. Based on the method, we have implemented a prototype which performs favorably compared to existing tools on several protocols which implement cache coherence and mutual exclusion.

One direction for future research is to apply the method to other types of topologies than linear arrays. For instance, in the cache coherence protocols we consider, the actual ordering on the processes inside the protocol has no relevance. These protocols fall therefore into a special case of our model where the system can be viewed as set of processes (without structure) rather than as a linear array. This indicates that the verification algorithm can be optimized even further for such systems. Furthermore, since our algorithm relies on a small set of properties of words which are shared by other data structures, we believe that our approach can be lifted to a more general setting. In particular we aim to develop similar algorithms for systems whose behaviours are captured by relations on trees and on general forms of graphs.

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. LICS '96*, pages 313–321, 1996.
2. P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Regular model checking without transducers. Technical Report 2006-052, Uppsala University, Dec. 2006.
3. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR '02*, pages 116–130, 2002.

4. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV '01*, pages 221–234, 2001.
5. K. Baukus, Y. Lakhnech, and K. Stahl. Parameterized verification of a cache coherence protocol: Safety and liveness. In *VMCAI '02*, pages 317–330, 2002.
6. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. CAV '03*, volume 2725 of *LNCS*, pages 223–235, 2003.
7. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. CAV '04*, LNCS, pages 372–386, Boston, July 2004.
8. E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proc. VMCAI '06*, volume 3855 of *LNCS*, pages 126–141, 2006.
9. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *Proc. CAV' 01*, volume 2102 of *LNCS*, 2001.
10. G. Delzanno. Automatic verification of cache coherence protocols. In Emerson and Sistla, editors, *Proc. CAV '00*, volume 1855 of *LNCS*, pages 53–68, 2000.
11. G. Delzanno. Verification of consistency protocols via infinite-state symbolic model checking. In *Proc. FORTE/PSTV 2000*, pages 171–186, 2000.
12. E. Emerson and V. Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME 2003*, pages 247–262, 2003.
13. E. Emerson and V. Kahlon. Model checking guarded protocols. In *Proc. LICS '03*, pages 361–370, 2003.
14. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS '99*, pages 352–359, 1999.
15. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
16. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *FMSD*, 2(2):149–164, 1993.
17. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.
18. P. Kelb, T. Margaria, M. Mendler, and C. Gsottberger. MOSEL: A flexible toolset for monadic second-order logic. In *Proc. TACAS '97*, pages 183–202, 1997.
19. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS, Volume 256*, pages 93–112, 2001.
20. S. K. Lahiri and R. E. Bryant. Indexed predicate discovery for unbounded system verification. In *CAV 2004*, pages 135–147, 2004.
21. M. Maidl. A unifying model checking approach for safety properties of parameterized systems. In *Proc. CAV '01*, pages 324–336, 2001.
22. R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science, Volume 297*, pages 337–354, 2003.
23. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proc. TACAS '01*, pages 82–97, 2001.
24. A. Pnueli, J. Xu, and L. Zuck. Liveness with (0,1,infinity)-counter abstraction. In *Proc. CAV '02*, volume 2404 of *LNCS*, 2002.
25. A. Roychoudhury and I. Ramakrishnan. Automated inductive verification of parameterized protocols. In *Proc. CAV '01*, pages 25–37, 2001.
26. C. Topnik, E. Wilhelm, T. Margaria, and B. Steffen. jMosel: A Stand-Alone Tool and jABC Plugin for M2L(Str). In *Model Checking Software: 13th International SPIN Workshop*, volume 3925 of *LNCS*, pages 293–298, 2006.
27. T. Touili. Regular Model Checking using Widening Techniques. *ETCS*, 50(4), 2001. Proc. VEPAS'01.
28. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86*, pages 332–344, June 1986.