Handling Global Conditions in Parameterized System Verification

Parosh Aziz Abdulla[1], Ahmed Bouajjani[2], Bengt Jonsson[1], and Marcus Nilsson[1]

[1] Dept. of Computer Systems, P.O. Box 325, S-751 05 Uppsala, Sweden
{parosh, bengt, marcusn}@docs.uu.se
[2] VERIMAG, Centre Equation, 2 av. de Vignate, 38610 Gieres, France
Ahmed.Bouajjani@imag.fr

Abstract. We consider symbolic verification for a class of parameterized systems, where a system consists of a linear array of processes, and where an action of a process may in general be guarded by both *local conditions* restricting the state of the process about to perform the action, and *global conditions* defining the *context* in which the action is enabled. Such actions are present, e.g., in idealized versions of mutual exclusion protocols, such as the bakery and ticket algorithms by Lamport, Burn's protocol, Dijkstra's algorithm, and Szymanski's algorithm. The presence of both local and global conditions makes the parameterized versions of these protocols infeasible to analyze fully automatically, using existing model checking methods for parameterized systems. In all these methods the actions are guarded only by local conditions involving the states of a finite set of processes.
We perform verification using a standard symbolic reachability algorithm enhanced by an operation to accelerate the search of the state space. The acceleration operation computes the effect of an arbitrary number of applications of an action, rather than a single application. This is crucial for convergence of the analysis e.g. when applying the algorithm to the above protocols.
We illustrate the use of our method through an application to Szymanski's algorithm.

1 Introduction

Much attention has recently been paid to extending the applicability of model checking to infinite-state systems. One reason why a program may have an infinite state space is that it operates on unbounded data structures. Examples of such systems include timed automata [ACD90], data-independent systems [Wol86], relational automata [Čer94], pushdown processes [BS95], and lossy channel systems [AJ96]. Another reason is that the program has an infinite control part. This is the case e.g. in Petri nets [Esp95,Jan90], and parameterized systems, in which the topology of the system is parameterized by the number of processes inside the system. In verification of parameterized systems, we are often interested in proving the correctness of the system regardless of the number of processes. Verification algorithms for systems consisting of an unbounded number of similar or identical finite-state processes include [GS92,AJ98,KMM$^+$97], and (using a manually supplied induction hypothesis) [CGJ95,KM89,WL89].

In this paper we consider algorithmic verification of a class of parameterized systems, intended to capture at least the behaviours of several mutual-exclusion algorithms that can be found in the literature. Examples of mutual exclusion algorithms that work for an arbitrary number of processes are: the bakery and ticket algorithms by Lamport, Burn's protocol, Dijkstra's algorithm, and Szymanski's algorithm. These algorithms are implemented on systems with an arbitrary number of processes with linearly ordered identities. The ordering of the processes may reflect the actual physical ordering (e.g. Szymanski's algorithm), or the values assigned to local variables inside processes (e.g. the ticket given to each process during the execution of Lamport's bakery protocol). A configuration of the system can be described as a string representing the local states of the processes. A common feature which places these protocols outside the scope of existing model checking methods, is that an action of a process is in general guarded by both *local* and *global* conditions on the processes. Local conditions restrict the state of the process which is about to perform the action. Global conditions define the *context* in which the action is allowed to occur. A *context* is typically stated as a formula which is quantified over the set of processes inside the system. Examples of contexts are "all processes with lower identities should have local states belonging to given set", or "there should be at least one process with a higher identity which has a local state included in a given set", etc. We propose a model which combines both types of conditions. An action involves the change of local state of a process, and may be conditioned on both the local state, and the context in which the action is performed.

To verify our protocols we perform a standard symbolic forward reachability analysis, using regular expressions to represent (possibly infinite) sets of configurations. It is well-known that checking most safety properties (including satisfiability of mutual exclusion) can be reduced to checking the reachability of a set of "bad" configurations (in our case specified as a regular expression). However, the presence of both local and global guards implies that the standard reachability algorithm will not terminate when applied to any of the earlier mentioned protocols. A main contribution of this paper is that we define an operation to accelerate the search through the state space. The acceleration operator computes the effect of an arbitrary number of applications of an action, rather than the effect of only a single application. This is crucial for obtaining termination during the analysis of any of the above protocols. Notice that the algorithm is incomplete and may in general still fail to terminate.

Related Work There are several results on verification of parameterized systems [GS92,AJ98,CGJ95,KMM$^+$97]. In all these works the actions are guarded only by local conditions involving the states of a finite set of processes. A work, which is close in spirit to ours is [KMM$^+$97]. The authors propose to use regular sets of strings to represent states of parameterized arrays of processes, and to represent the effect of performing an action by a predicate transformer (transducer). However, the work in [KMM$^+$97] considers only transducers that represent the effect of a single application of a transition. This means that their approach will not terminate if applied to reachability analysis for the protocols we consider in this paper. In contrast, we introduce a acceleration operator for actions with both local and global contexts, meaning that reachability analysis will terminate. Applications of acceleration operations are reported in the context of communicating finite state automata [BG96,BGWW97,BH97,ABJ98]. The acceleration operation is applied to transitions of different types than in our work, namely those that iterate a single loop in the control part of a program, rather than repetitive applications of a transition to different processes in the system. There has also been a number of case studies in verification of mutual exclusion protocols such as Burn's protocol [JL98] and Szymanski's algorithm [GZ98,MAB$^+$94, MP90]. The verification in each case is dependent on abstraction functions or lemmas explicitly provided by the user.

Outline In the next section, we define the class of system models that we consider and illustrate it by an idealized version of Szymanski's mutual exclusion algorithm. In Sect. 3 we define composition and acceleration of actions. In Sect. 4 we show how they can be used in verification of safety properties, illustrated by a verification of Szymanski's algorithm. Section 5 contains conclusions and some non-resolved problems.

2 Preliminaries

In this section, we will introduce a generic system model which is intended to capture the behaviour of idealized versions of many existing mutual exclusion protocols, e.g. Dijkstra's mutual exclusion problem, Lamport's bakery algorithm, Burn's protocol, Szymanski's algorithm. In our model, a program consists of an arbitrary number of identical processes, ordered in a linear array. The process behaviours are defined through a finite set of *actions*. An action represents a change of local state of a process. An action may be conditioned on both the local state of the process, and the *context* in which it may take place. The context represents a global condition on the local states of the rest of processes inside the system. The ordering of the processes may reflect the actual physical ordering (e.g. Szymanski's algorithm), or the values assigned to local variables inside processes (e.g. the ticket given to each process during the execution of Lamport's bakery protocol).

An idealized version of Szymanski's mutual exclusion algorithm can be given as follows. In the algorithm, an arbitrary number of processes compete for a critical section. The local state of each process i consists of a control state ranging over the integers from 1 to 7 and of two boolean flags, w_i and s_i. A process is in the critical section when the control state is 7. A pseudo-code version of the actions of any process i could look as follows:

```
1 :   await $\forall j : j \neq i : \neg s_j$
2 :   $w_i, s_i :=$ true, true
3 :   if $\exists j : j \neq i : (pc_j \neq 1) \wedge \neg w_j$
          then $s_i :=$ false; goto 4
          else $w_i :=$ false; goto 5
4 :   await $\exists j : j \neq i : s_j \wedge \neg w_j$ then $w_i, s_i :=$ false, true
5 :   await $\forall j : j \neq i : \neg w_j$
6 :   await $\forall j : j < i : \neg s_j$
7 :   $s_i :=$ false, goto 1
```

For instance, according to the code at line 6, if the control state of a process i is 6, and the value of s is *false* in all processes to the left, i.e. for all processes $j < i$, then the control state of i may be changed to 7. In a similar manner, according to the code at line 4, if the control state of a process i is 4, and if the context is that there is at least another process j (either to the right or to the left of i) where the value of s_j is *true* and the value of w_j is *false*, then the control state, w_i and s_i in i may be changed to 5, *false*, and *true*, respectively. In fact in almost all the protocols that we have considered, contexts are defined by existentially or universally quantified formulas restricting the local states of processes to the left or to the right. In our model we work with a particular subclass of regular languages, which can capture such contexts.

A *left context* is a regular language which can be accepted by a deterministic finite-state automaton with a unique accepting state, and where all outgoing transitions from the accepting state are self-loops. (transitions with identical source and target states). A *right context* is a language such that the language of reversed strings is a left context. The *tail* of a left context is the set of symbols that label self-loops from the accepting state. The tail of a right context is the tail of the left context which is its reverse language.

Examples of left contexts are regular expressions of the form

$$e_1 f_1 e_2 f_2 \cdots e_n f_n e_{n+1}$$

where each e_i is of form $(a_1 + \cdots + a_m)^*$, where each f_i is of form $(b_1 + \cdots + b_k)$ such that b_j does not occur in the expression e_i, for any $j = 1, \ldots, k$.

Now, we give the formal definition of our model. We use a finite set C of *colours* to model the local states of processes. A *program* is a triple $\mathcal{P} = \langle C, \phi_I, \mathcal{A} \rangle$ where

C	is a finite set of colours,
ϕ_I	is a regular expression denoting a set of *initial configurations* over C, and
$\mathcal{A}$	is a finite set of *actions*. An *action* is a triple of the form

$$\phi_L \ ; \ \tau(c, c') \ ; \ \phi_R$$

where ϕ_L is a left context, ϕ_R is a right context, and $\tau(c, c')$ is a an idempotent binary relation on C.

A *configuration* γ of $\mathcal{P}$ is a string $\gamma[1]\ \gamma[2]\ \cdots\ \gamma[n]$ over C, where $\gamma[i]$ denotes the local state of process i. For a regular expression ϕ, we use $\gamma \in \phi$ to denote that γ is a string in the language denoted by ϕ. For $i, j : 1 \leq i \leq j \leq n$, we use $\gamma[i\ ..\ j]$ to denote the substring $\gamma[i]\ \gamma[i+1]\ \cdots\ \gamma[j]$. An action

$$\alpha = \phi_L\ ;\ \tau(c, c')\ ;\ \phi_R$$

defines a relation α on configurations such that $\alpha(\gamma, \gamma')$ holds if γ and γ' are of equal length n, and there is an i with $1 \leq i \leq n$ such that $\tau(\gamma[i], \gamma'[i])$ holds, $\gamma[1\ ..\ i-1] = \gamma'[1\ ..\ i-1] \in \phi_L$, and $\gamma[i+1\ ..\ n] = \gamma'[i+1\ ..\ n] \in \phi_R$. Thus, an action corresponds to a (possibly nondeterministic) program statement in which the colour at one position i can be changed from some colour c to some colour c', provided that $\tau(c, c')$ holds and that the string to the left of i is in ϕ_L and that the string to the right of i is in ϕ_R. We write $\gamma_1 \longrightarrow \gamma_2$ to denote that $\alpha(\gamma_1, \gamma_2)$ for some action $\alpha \in \mathcal{A}$. We use α^* and $\xrightarrow{*}$ to denote the transitive closures of α and $\longrightarrow$ respectively. A configuration γ is said to be *reachable* if there is a configuration $\gamma_I \in \phi_I$ such that $\gamma_I \xrightarrow{*} \gamma$.

The *reachability problem* is defined as follows.

Instance A program $\mathcal{P}$ and a set of configurations of $\mathcal{P}$ represented by a regular expression ϕ_F.

Question Is any $\gamma \in \phi_F$ reachable?

In Fig. 1 we represent Szymanski's algorithm as a program in our framework. To simplify the notation, we introduce the following syntactical notations. We let a colour be a triple $\langle pc, w, s\rangle$, where $pc \in \{1, \ldots, 7\}$, and w and s are boolean. We use predicates to define colours. For example, the predicate $(\neg s)$ denotes the set of colours where the value of s is equal to *false*, that is the set $\{\langle pc, w, \mathit{false}\rangle\ :\ pc \in \{1, \ldots, 7\}$ and $w \in \{\mathit{true}, \mathit{false}\}\}$. We use the predicate *true* to denote the set of all colours. We use guarded commands to represent binary relations on colours. For instance, the command $(pc = 1) \longrightarrow pc := 2$ represents the relation $\{\langle\langle pc_1, w, s\rangle, \langle pc_2, w, s\rangle\rangle\ :\ (pc_1 = 1)$ and $(pc_2 = 2)\}$. Notice that e.g. at line 3 the left context $((pc = 1)\ \vee\ w)^*((pc \neq 1)\ \wedge\ \neg w)\mathit{true}^*$ is equivalent to $\mathit{true}^*((pc \neq 1)\ \wedge\ \neg w)\mathit{true}^*$; however, we use the previous expression in order to be consistent with the definition of a left context.

3 Acceleration of Actions

In this section we define an operation which computes the effect of an unbounded number of executions of an action.

For an action α, let α^* be the action constructed by repeating the action α an arbitrary number of times. More precisely, α^* denotes the set of pairs (γ, γ') of configurations such that there exists a sequence $\gamma_0\gamma_1\gamma_2\cdots\gamma_n$ of configurations with $n \geq 0$ such that $\gamma = \gamma_0$, $\gamma' = \gamma_n$, and such that $\alpha(\gamma_i, \gamma_{i+1})$ for $i = 0, 1, \ldots, n-1$. Similarly, we let α^+ be the action constructed by repeating the action α one or more times.

$$\begin{array}{ll}
1: & (\neg s)^* \ ; \ (pc=1) \longrightarrow pc := 2 \ ; \ (\neg s)^* \\
2: & \mathit{true}^* \ ; \ (pc=2) \longrightarrow pc, w, s := 3, \mathit{true}, \mathit{true} \ ; \ \mathit{true}^* \\
3: & ((pc=1) \ \vee \ w)^*((pc \neq 1) \ \wedge \ \neg w)\mathit{true}^* \ ; \ (pc=3) \longrightarrow pc, s := 4, \mathit{false} \ ; \ \mathit{true}^* \\
4: & \mathit{true}^* \ ; \ (pc=3) \longrightarrow pc, s := 4, \mathit{false} \ ; \ \mathit{true}^*((pc \neq 1) \ \wedge \ \neg w)((pc=1) \ \vee \ w)^* \\
5: & ((pc=1) \ \vee \ w)^* \ ; \ (pc=3) \longrightarrow pc, w := 5, \mathit{false} \ ; \ ((pc=1) \ \vee \ w)^* \\
6: & (\neg s \ \vee \ w)^*(s \ \wedge \ \neg w)\mathit{true}^* \ ; \ (pc=4) \longrightarrow pc, w, s := 5, \mathit{false}, \mathit{true} \ ; \ \mathit{true}^* \\
7: & \mathit{true}^* \ ; \ (pc=4) \longrightarrow pc, w, s := 5, \mathit{false}, \mathit{false} \ ; \ \mathit{true}^*(s \ \wedge \ \neg w)(\neg s \ \vee \ w)^* \\
8: & (\neg w)^* \ ; \ (pc=5) \longrightarrow pc := 6 \ ; \ (\neg w)^* \\
9: & (\neg s)^* \ ; \ (pc=6) \longrightarrow pc := 7 \ ; \ \mathit{true}^* \\
10: & \mathit{true}^* \ ; \ (pc=7) \longrightarrow pc, s := 1, \mathit{false} \ ; \ \mathit{true}^*
\end{array}$$

Fig. 1. Actions for Modelling Szymanski's Algorithm

We shall now characterize α^+ for any action α. A characterization of α^* can be obtained from a characteriztion of α^+ by taking the union with the identity relation.

Theorem 1. *Let α be an action of the form*

$$\alpha = \phi_L \ ; \ \tau(c, c') \ ; \ \phi_R$$

where $\tau(c, c')$ is idempotent, and where Σ_L (Σ_R) is the tail of ϕ_L (ϕ_R). Then α^+ consists of the set of pairs (γ, γ') of configurations of equal length (say n), such that there are indices i, j with $1 \leq i \leq j \leq n$ such that

1. $\gamma[1..i-1] = \gamma'[1..i-1] \in \phi_L$,
2. $\gamma[j+1..n] = \gamma'[j+1..n] \in \phi_R$,
3. $\tau(\gamma[i], \gamma'[i])$, $\tau(\gamma[j], \gamma'[j])$ *and for each k with $i < k < j$ we have* $\gamma[k] = \gamma'[k]$ *or* $\tau(\gamma[k], \gamma'[k])$.
4. *For each k with $i < k \leq j$ we have $\gamma[k] \in \Sigma_R$ or $\gamma'[k] \in \Sigma_R$.*
5. *For each k with $i \leq k < j$ we have $\gamma[k] \in \Sigma_L$ or $\gamma'[k] \in \Sigma_L$.*
6. *For all indices k_1, k_2 with $i \leq k_1 < k_2 \leq j$ we have $\gamma[k_1] \in \Sigma_L \ \vee \ \gamma[k_2] \in \Sigma_R$ and $\gamma'[k_1] \in \Sigma_L \ \vee \ \gamma'[k_2] \in \Sigma_R$.* □

In the symbolic reachability analysis (described in Sect. 4), we use regular expressions as representations of sets of configurations. The characterization of Theorem 1 can be used to model the effect of (repetitive applications of) actions on regular sets by using *finite-state transducers*. This approach is proposed in [KMM$^+$97], where however acceleration is not considered.

We recall that an action α denotes a set of pairs (γ, γ') of configurations. Equivalently, we can represent the action as a set of finite strings over $C \times C$, namely as the strings $(c_1, c_1')\ (c_2, c_2') \ \cdots \ (c_n, c_n')$ such that $(c_1 c_2 \cdots c_n, c_1' c_2' \cdots c_n') \in \alpha$. It is easy to see that each action can be represented by a finite-state transducer.

More importantly, for any action α the characterization of Theorem 1 can be used to find a representation of α^+ in a straight-forward way, since α^+ can be represented as a regular language over $C \times C$. As an example, in Fig. 2 we show the transducer which accepts α^+ where α is the action at line 3 in Fig.1. We note that the transducer in Fig. 2 need not use the full generality of the characterization of Theorem 1, since the alphabets Σ_L and Σ_R both are equal to the set of all colours.

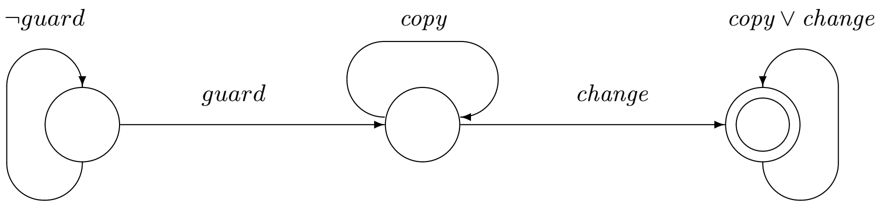


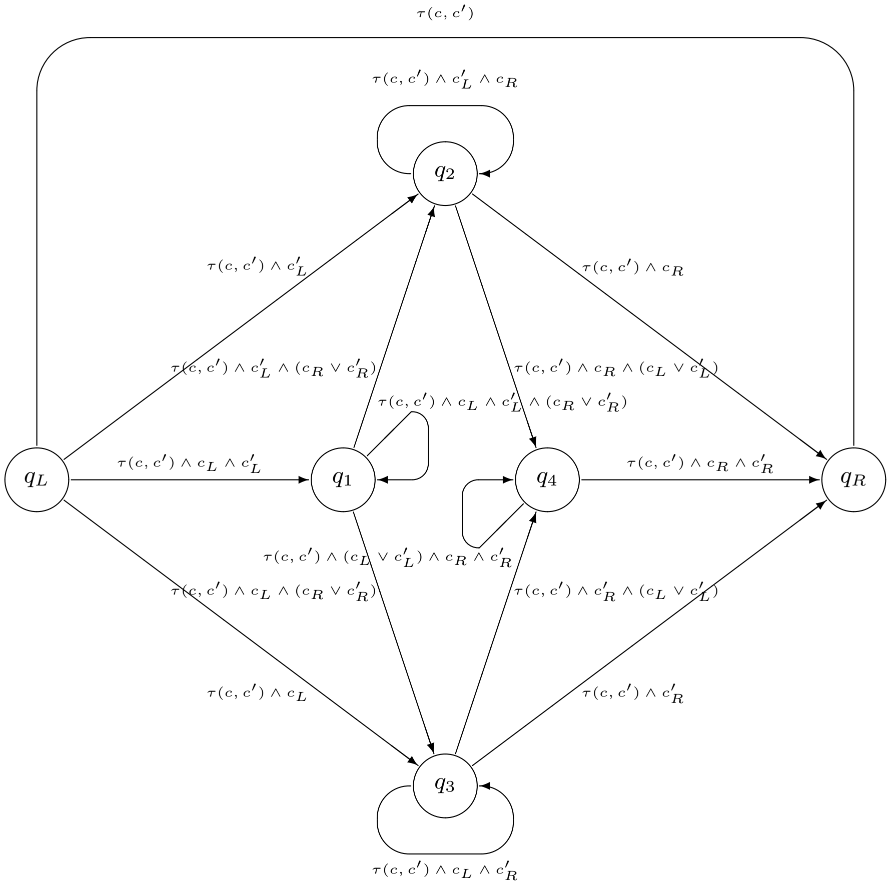
Fig. 2. Transducer for α^+ from line 3 of Fig. 1.

In the figure we use $\neg guard$ to denote pairs $\{(\langle pc, w, s\rangle, \langle pc, w, s\rangle) \;:\; pc \neq 1 \wedge \neg w\}$, we use *guard* to denote pairs $\{(\langle pc, w, s\rangle, \langle pc, w, s\rangle) \;:\; pc = 1 \vee w\}$, we use *copy* to denote pairs $(\langle pc, w, s\rangle, \langle pc, w, s\rangle)$ of identical tuples, and *change* to denote pairs $\{(\langle pc, w, s\rangle, \langle pc', w', s'\rangle) \;:\; pc = 3 \wedge pc' = 4 \wedge w' = w \wedge s' = \mathit{false}\}$ that represent a change of local control state.

For a regular expression ϕ and an action α, we use $\alpha^*(\phi)$ to denote the regular expression we get by computing (in the usual way) the product of ϕ and the transducer corresponding to α^*.

In order to illustrate that the conditions in Theorem 1 characterize a regular relation between configurations, we show a representation of this relation in terms of a finite-state transducer. We show the part which is inserted between the accepting state q_L of an automaton that copies strings in ϕ_L and the initial state q_R of an automaton that copies strings in ϕ_R. In Fig. 3, we show the general construction. Edges are labeled by predicates on pairs (c, c') of colours that are read. We use the abbreviations c_L for $c \in \Sigma_L$, c'_L for $c' \in \Sigma_L$, c_R for $c \in \Sigma_R$, and c'_R for $c' \in \Sigma_R$. In addition to the transitions in the figure, there are self-loop at states q_1, q_2, q_3, and q_4 labeled $c = c' \in \Sigma_L \cap \Sigma_R$. Informally, the states correspond to the following situations.

- q_1 corresponds to a state when the transducer has read an index where some change has occurred, but where so far there has been no index with change at which $c \notin \Sigma_L \;\vee\; c' \notin \Sigma_L$.
- q_2 corresponds to a state when the transducer has read an index where some change has occurred where $c \notin \Sigma_L$, but where so far there has been no index with change at which $c' \notin \Sigma_L$.
- q_3 corresponds to a state when the transducer has read an index where some change has occurred where $c' \notin \Sigma_L$, but where so far there has been no index with change at which $c \notin \Sigma_L$.



$\tau(c,c')$
$\tau(c,c') \wedge c'_L \wedge c_R$
q_2
$\tau(c,c') \wedge c'_L$
$\tau(c,c') \wedge c_R$
$\tau(c,c') \wedge c'_L \wedge (c_R \vee c'_R)$
$\tau(c,c') \wedge c_R \wedge (c_L \vee c'_L)$
$\tau(c,c') \wedge c_L \wedge c'_L \wedge (c_R \vee c'_R)$
q_L
$\tau(c,c') \wedge c_L \wedge c'_L$
q_1
q_4
$\tau(c,c') \wedge c_R \wedge c'_R$
q_R
$\tau(c,c') \wedge (c_L \vee c'_L) \wedge c_R \wedge c'_R$
$\tau(c,c') \wedge c_L \wedge (c_R \vee c'_R)$
$\tau(c,c') \wedge c'_R \wedge (c_L \vee c'_L)$
$\tau(c,c') \wedge c_L$
$\tau(c,c') \wedge c'_R$
q_3
$\tau(c,c') \wedge c_L \wedge c'_R$

Fig. 3. General transducer for α^+

- q_4 corresponds to a state when the transducer has read an index where some change has occurred where $c' \notin \Sigma_L$ and some index where some change has occurred where $c \notin \Sigma_L$.

4 Verification

In this section we show how the operation of acceleration, presented in the preceding section, can be used to enhance a standard version of symbolic forward reachability analysis, whose purpose is to compute a representation of the set of reachable configurations. The analysis algorithm maintains a set of reachable configurations, which is initially the set of initial configurations. In each step of the algorithm, the set of reachable configurations is extended with the configurations that can be reached by some action from a configuration in the current set. We use regular expressions to represent (potentially infinite) sets of configurations. As we shall illustrate later in the section, this algorithm will not terminate when applied to any of the protocols mentioned in the introduction. To solve this problem, we use the operation α^* (defined in Sect. 3) to accelerate the exploration of the state space. We recall that α^* computes the set of successors corresponding to an arbitrary number of applications of an action (rather than a single application).

Suppose we are given a program $\mathcal{P} = \langle C, \phi_I, \mathcal{A} \rangle$ and a regular expression ϕ_F, and that we want to check whether some configuration $\gamma_F \in \phi_F$ is reachable in $\mathcal{P}$. For the current discussion, let us represent the set of configurations maintained by the algorithm by a set V of regular expressions. The set V represents the union of the sets denoted by all regular expressions in V. Initially, $V = \{\phi_I\}$. The algorithm will now for each regular expression ϕ in V and each action α compute $\alpha^*(\phi)$ represented as a finite union of regular expressions. When a new expression ϕ is generated, it is compared with those which are already in V. If $\phi \subseteq \phi'$ for some $\phi' \in V$, then ϕ is discarded, since it will not add new configurations to the explored state space (it is actually sufficient that $\phi \subseteq \sum_{\phi' \in V} \phi'$ for ϕ to be safely discarded). In fact, we can also discard all $\phi' \in V$ with $\phi' \subseteq \phi$. It is also checked whether ϕ has a non-empty intersection with ϕ_F. If the intersection is non-empty, the algorithm terminates, reporting that some configuration in ϕ_F is reachable. Otherwise, the algorithm terminates when no new regular expressions can be generated. Obviously, our algorithm is incomplete in the sense that while it will always find reachable configurations in ϕ_F, it will not necessarily terminate if all configurations in ϕ_F are unreachable.

We illustrate this algorithm through an application to Szymanski's protocol. To simplify the notation we use the coding of colours shown in Table 1, so e.g. c_2 corresponds to the colour $\langle 2, \mathit{false}, \mathit{false} \rangle$. The set of initial configurations is represented by $\phi_0 = c_1^*$.

First, we observe that the above standard reachability algorithm will run into an infinite loop as follows. By applying action 1 to ϕ_0 we get $c_1^*\ c_2\ c_1^*$. Applying action 1 again gives $c_1^*\ c_2\ c_1^*\ c_2\ c_1^*$, etc.

Although the standard algorithm fails, using the acceleration operation leads to termination. In Table 2 we describe a simulation of our algorithm. We start from the set of initial configurations ϕ_0. For each regular expression ϕ_i and action α, we compute $\alpha^*(\phi_i)$ or $\alpha^+(\phi_i)$ and add the resulting regular expressions to the set of existing expressions. For instance, from ϕ_0, only action 1 is enabled, resulting in the configurations denoted by ϕ_1. Whenever an expression is entailed by another one (e.g. $\phi_7 \subseteq \phi_0$), we indicate that in the table. In such a case, the constraint (in this case ϕ_7) is discarded and not explored further. At ϕ_2, we pursue both α_3^+ and α_4^+, denoted $\alpha_3^+ \cup \alpha_4^+$, in one step. The algorithm terminates in 19 steps.

c_1	$\langle 1, \mathit{false}, \mathit{false} \rangle$
c_2	$\langle 2, \mathit{false}, \mathit{false} \rangle$
c_3	$\langle 3, \mathit{true}, \mathit{true} \rangle$
c_4	$\langle 4, \mathit{true}, \mathit{false} \rangle$
c_5	$\langle 5, \mathit{false}, \mathit{true} \rangle$
c_6	$\langle 6, \mathit{false}, \mathit{true} \rangle$
c_7	$\langle 7, \mathit{false}, \mathit{true} \rangle$

Table 1. Coding of colours in analysis of Szymanski's algorithm

5 Conclusions

In the paper, we have presented techniques for reachability analysis of parameterized systems where a configuration of the system can be described by a string representing the local states of the processes. We have found that naive symbolic reachability analysis does not converge for such systems, and propose to use acceleration of actions to obtain termination. We showed that using acceleration, symbolic reachability analysis terminates for an idealized version of Szymanski's algorithm. We have also analyzed corresponding versions of other mutual exclusion algorithms, including Burn's and Dijkstra's mutual exclusion algorithms, and the bakery and ticket algorithms by Lamport. For some of these algorithms, we use variants of the acceleration operation presented in this paper: we perform the acceleration on the action obtained by sequentially composing two actions, and we also define an acceleration operation on actions that involve two adjacent processes which can be guarded by left and right contexts.

We further note that we have considered idealized versions of the mutual exclusion algorithms. In most implementations of these algorithms, a global guard (such as e.g., $\forall j : j < i : \neg s_j$) is not atomic: in a more refined description of the algorithm this is a loop which checks the states of other processes. We have not considered how to treat the non-atomic versions of statements such as this one.

References

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *Proc. 10th Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318, 1998.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, Philadelphia, 1990.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes. In Bernhard Steffen, editor, *Proc. TACAS '98, 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312, 1998.

ϕ_0	c_1^*	α_1^*	ϕ_1	
ϕ_1	$(c_1+c_2)^*$	α_2^*	ϕ_2	
ϕ_2	$(c_1+c_2+c_3)^*$	$\alpha_3^+ \cup \alpha_4^+$	ϕ_3	
		α_5^+	ϕ_4	
ϕ_3	$(c_1+c_2+c_3)^* c_2 (c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^*$ $+$ $(c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^* c_2 (c_1+c_2+c_3)^*$	α_3^*	ϕ_8	
ϕ_4	$(c_1+c_3)^* c_5 (c_1+c_3)^*$	α_8^+	ϕ_5	
ϕ_5	$c_1^* c_6 c_1^*$	α_9^+	ϕ_6	
ϕ_6	$c_1^* c_7 c_1^*$	α_{10}^+	ϕ_7	
ϕ_7	$c_1^* c_1 c_1^*$			Subs ϕ_0
ϕ_8	$(c_1+c_2+c_3)^* c_2 (c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^*$ $+$ $(c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^* c_2 (c_1+c_2+c_3+c_4)^*$	α_4^*	ϕ_9	
ϕ_9	$(c_1+c_2+c_3+c_4)^* c_2 (c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^*$ $+$ $(c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^* c_2 (c_1+c_2+c_3+c_4)^*$	α_2^*	ϕ_{10}	
ϕ_{10}	$(c_1+c_2+c_3+c_4)^* (c_2+c_3) (c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^*$ $+$ $(c_1+c_2+c_3+c_4)^* c_4 (c_1+c_2+c_3+c_4)^* (c_2+c_3) (c_1+c_2+c_3+c_4)^*$	α_5^+	ϕ_{11}	
ϕ_{11}	$(c_1+c_3+c_4)^* c_5 (c_1+c_3+c_4)^* c_4 (c_1+c_3+c_4)^*$ $+$ $(c_1+c_3+c_4)^* c_4 (c_1+c_3+c_4)^* c_5 (c_1+c_3+c_4)^*$	α_6^*	ϕ_{12}	
ϕ_{12}	$(c_1+c_3+c_4)^* c_5 (c_1+c_3+c_4+c_5)^* (c_4+c_5) (c_1+c_3+c_4+c_5)^*$ $+$ $(c_1+c_3+c_4+c_5)^* (c_4+c_5) (c_1+c_3+c_4+c_5)^* c_5 (c_1+c_3+c_4)^*$	α_7^*	ϕ_{13}	
ϕ_{13}	$(c_1+c_3+c_4+c_5)^* c_5 (c_1+c_3+c_4+c_5)^* (c_4+c_5) (c_1+c_3+c_4+c_5)^*$ $+$ $(c_1+c_3+c_4+c_5)^* (c_4+c_5) (c_1+c_3+c_4+c_5)^* c_5 (c_1+c_3+c_4+c_5)^*$	α_8^+	ϕ_{14}	
ϕ_{14}	$(c_1+c_5+c_6)^* c_6 (c_1+c_5+c_6)^* (c_5+c_6) (c_1+c_5+c_6)^*$ $+$ $(c_1+c_5+c_6)^* (c_5+c_6) (c_1+c_5+c_6)^* c_6 (c_1+c_5+c_6)^*$	α_9^+	ϕ_{15}	
ϕ_{15}	$c_1^* c_7 (c_1+c_5+c_6)^* (c_5+c_6) (c_1+c_5+c_6)^*$	α_{10}	ϕ_{16}	
ϕ_{16}	$c_1^* c_1 (c_1+c_5+c_6)^* (c_5+c_6) (c_1+c_5+c_6)^*$	α_9	ϕ_{17}	
ϕ_{17}	$c_1^* c_1 c_7 (c_1+c_5+c_6)^*$	α_{10}	ϕ_{18}	
ϕ_{18}	$c_1^* c_1 c_1 (c_1+c_5+c_6)^*$	α_9^+	ϕ_{19}	
ϕ_{19}	$c_1^* c_1 c_1 c_7 (c_1+c_5+c_6)^*$			Subs ϕ_{17}

Table 2. Reachability Analysis of Szymanski's Algorithm

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc. 8th Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, 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*, Lecture Notes in Computer Science. 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*, number 1256 in Lecture Notes in Computer Science, 1997.

[BS95] O. Burkart and B. Steffen. Composition, decomposition, and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.

[Čer94] K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *Proc. ICALP '94*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer Verlag, 1994.

[CGJ95] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In Lee and Smolka, editors, *Proc. CONCUR '95, 6th Int. Conf. on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 395–407. Springer Verlag, 1995.

[Esp95] J. Esparza. Petri nets, commutative context-free grammers, and basic parallel processes. In *Proc. Fundementals of Computation Theory*, number 965 in Lecture Notes in Computer Science, pages 221–232, 1995.

[GS92] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[GZ98] E.P. Gribomont and G. Zenner. Automated verification of Szymanski's algorithm. In *Proc. TACAS '98, 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 424–438, 1998.

[Jan90] P. Jančar. Decidability of a temporal logic problem for Petri nets. *Theoretical Computer Science*, 74:71–93, 1990.

[JL98] E. Jensen and N. A. Lynch. A proof of Burn's n-process mutual exclusion algorithm using abstraction. In *Proc. TACAS '98, 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 409–423, 1998.

[KM89] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *Proc. 8th ACM Symp. on Principles of Distributed Computing, Canada*, pages 239–247, Edmonton, Alberta, 1989.

[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. 9th Int. Conf. on Computer Aided Verification*, volume 1254, pages 424–435, Haifa, Israel, 1997. Springer Verlag.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. chang, M. Colón, L. de Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STEP: the stanfor temporal prover. Draft Manuscript, June 1994.

[MP90] Z. Manna and A. Pnueli. An exercise in the verification of multi – process programs. In W.H.J. Feijen, A.J.M van Gasteren, D. Gries, and J. Misra, editors, *Beauty is Our Business*, pages 289–301. Springer-Verlag, 1990.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants (extended abstract). In Sifakis, editor, *Proc. Workshop on Computer Aided Verification*, number 407 in Lecture Notes in Computer Science, pages 68–80, 1989.

[Wol86] Pierre Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–193, Jan. 1986.