Systematic Acceleration in Regular Model Checking

Bengt Jonsson and Mayank Saksena

Dept. of Information Technology, P.O. Box 337, S-751 05 Uppsala, Sweden
{bengt,mayanks}@it.uu.se

Abstract. Regular model checking is a form of symbolic model checking technique for systems whose states can be represented as finite words over a finite alphabet, where regular sets are used as symbolic representation. A major problem in symbolic model checking of parameterized and infinite-state systems is that fixpoint computations to generate the set of reachable states or the set of reachable loops do not terminate in general. Therefore, *acceleration* techniques have been developed, which calculate the effect of arbitrarily long sequences of transitions generated by some action. We present a systematic method for using acceleration in regular model checking, for the case where each transition changes at most one position in the word; this includes many parameterized algorithms and algorithms on data structures. The method extracts a maximal (in a certain sense) set of actions from a transition relation. These actions, and systematically obtained compositions of them, are accelerated to speed up a fixpoint computation. The extraction can be done on any representation of the transition relation, e.g., as a union of actions or as a single monolithic transducer. Using this approach, we are for the first time able to verify completely automatically both safety and absence of starvation properties for a collection of parameterized synchronization protocols from the literature; for some protocols, we obtain significant improvements in verification time. The results show that symbolic state-space exploration, without using abstractions, is a viable alternative for verification of parameterized systems with a linear topology.

1 Introduction

A major approach in algorithmic verification of parameterized and infinite-state systems is to extend the paradigm of symbolic model checking [17] by appropriate symbolic representations; examples include Petri nets, timed automata, systems with unbounded communication channels, integers and reals. One direction is *regular model checking*, which considers systems whose states can be represented as finite words over a finite alphabet; regular sets are used to represent sets of states and transition relations. Regular model checking has been proposed as a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems [26,32,16,4].

In symbolic model checking of parameterized and infinite-state systems, a major problem is that fixpoint computations that generate the set of reachable

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 131–144, 2007.

states or the set of reachable loops (for verifying liveness properties) do not terminate in general, since there is no uniform bound on the distance (in number of transitions) from an initial configuration to any reachable configuration. To make fixpoint computations converge more frequently, *acceleration* techniques have been developed, which calculate the effect of arbitrarily long sequences of transitions generated by some action (i.e., a subset of the transition relation). This has been done, e.g., for systems with unbounded FIFO channels [11,12,14,1], systems with counters [13,18], and for parameterized systems [6]. Acceleration is typically applied to small actions, e.g., corresponding to a single program statement or simple loop, since acceleration of larger actions or the entire transition relation is often intractable. Fixpoint computations can be sped up by using accelerated actions in each iteration, thereby allowing the fixpoint computation to converge in many practical cases (e.g., [1]).

For regular model checking, methods have been developed for computing the set of reachable configurations or reachable loops [25,16,19,8]. These algorithms typically work well for small system models, but have difficulties to cope with large transition relations. For instance, the automata-theoretic approach for parameterized systems [3] transforms verification of a liveness property into the problem of finding reachable loops for a system with a rather large transition relation. There has been no systematic way to to extract actions for acceleration from such a transition relation, and therefore liveness properties for several parameterized mutual exclusion protocols have not been proven automatically by this class of techniques.

In this paper, we present a systematic approach for using acceleration to speed up fixpoint computations in regular model checking. We consider *unary* systems, in which each computation step changes at most one position in the word; many models of parameterized algorithms and algorithms on data structures are unary. Our approach is based on accelerating a class of actions (called *separable*) which can be efficiently accelerated. We present techniques for

(a) systematically extracting a set of separable actions which is maximal in the sense that any other separable action is included in some extracted one; the extraction can be done on any representation of the transition relation, e.g., as a union of actions or as a single monolithic transducer,
(b) systematically composing actions to form separable actions that represent the effect of several transitions; such compositions are analogous to program loops; many verification examples require the acceleration of such compositions, rather than single actions, for termination.

We have implemented our approach in the context of our LTL(MSO) model checker for parameterized systems [3], and verified safety and liveness properties of several idealized parameterized protocols from the literature, including parameterized algorithms for mutual exclusion (e.g., the Bakery algorithm by Lamport, algorithms by Burns, Szymanski, and Dijkstra). The most important result is that, for the first time, liveness properties have been successfully verified for all

of these algorithms; previous approaches have not been successfully applied to all of them. One should also note that our verification, following the automata theoretic approach, does not employ any form of abstraction: it computes an exact representation of the set of reachable states and reachable loops.

Related Work. Works on acceleration techniques in other contexts include techniques for systems with FIFO channels [1,12,14] and systems with counter variables [9,32]. Finkel, Leroux and colleagues have presented a systematic framework for acceleration techniques for programs with a finite number of variables, typically ranging over integers [10,23]. Their approach cannot be used for regular model checking, in which systems can not be modeled by a fixed number of integer variables. For regular model checking, Pnueli and Shahar [28] show how specific acceleration schemes can be defined in a version of S1S. They did not consider composition of actions, which is necessary in many cases, and they have reported verification of liveness for only one example, after applying a manually supplied abstraction. In our earlier work [6], we proved safety properties of several parameterized protocols by accelerating individual actions; this approach did not consider composition of actions and would therefore not have been able to verify liveness properties.

Proving liveness properties of parameterized systems has been considered also in other approaches. Pnueli, Xu, and Zuck [29] use a version of counter abstraction to prove absence of starvation properties for Szymanski's algorithm and the Bakery algorithm. Their abstractions are rather coarse, and lose information so that, e.g., safety properties can no longer be checked. Fang, Piterman, Pnueli, and Zuck [22,21] infer a ranking function and helpful directions of a certain form, by generalizing from the verification of finite instances. These approaches require that a system can be verified using assertions of a certain form. In our earlier work [5], we proved liveness properties by backwards reachability analysis from "terminated" configurations; this technique can be combined with other techniques for proving liveness, but can not be used to find counterexamples (bugs); our technique is based on state-space exploration, which is guaranteed to report counterexamples when they exist.

Abdulla et al. [2] verify safety properties of parameterized protocols by over-approximation of backwards reachable states; their approach can not be used for proving liveness properties. Other works apply abstraction [15] or regular inference [24] directly on the automata that represent reachable states or the transition relation.

Outline. In the next section, we introduce the framework of regular model checking and the fixpoint computations that are our concern. Section 3 presents our technique for extracting parts of a transition relation for acceleration. In Section 4, we present how to use our systematic acceleration in the verification of liveness properties. Experimental results from our implementation, and comparisons with other results are presented in Section 5. Section 6 presents conclusions and future work directions.

2 The Regular Model Checking Framework

Let Σ be a finite alphabet. A relation $\mathcal{R}$ on Σ^* (the set of finite words over Σ) is *length-preserving* if w and w' are of equal length whenever $(w, w') \in \mathcal{R}$. In this paper, we will only consider length-preserving relations on Σ^*. A relation $\mathcal{R}$ on Σ^* is *regular* if the set $\{(a_1, a_1') \cdots (a_n, a_n') \mid (a_1 \cdots a_n, a_1' \cdots a_n') \in \mathcal{R}\}$ is a regular subset of $(\Sigma \times \Sigma)^*$. A regular relation Σ^* can be represented by a finite-state transducer, i.e., a finite automaton over $(\Sigma \times \Sigma)$.

Regular relations are closed under union $\cup$, intersection $\cap$, relational composition $\circ$, as well as concatenation $\cdot$ defined by $\mathcal{R} \cdot \mathcal{R}' \triangleq \{(w_1 \cdot w_1', w_2 \cdot w_2') \mid w_1 \ \mathcal{R} \ w_2$ and $w_1' \ \mathcal{R}' \ w_2'\}$. For a (regular) set $\mathcal{S}$ of words, let $\mathcal{S} \circ \mathcal{R}$ denote the (regular) set $\{w \mid \exists w' \in \mathcal{S}.\ w' \ \mathcal{R} \ w\}$. We use $\mathcal{R}^+$ to denote the (not necessarily regular) transitive closure of $\mathcal{R}$; and $\mathcal{R}^*$ the reflexive-transitive closure. We denote by $Id = \{(w, w') \mid w = w'\}$ the identity relation on Σ^*.

Definition 1. A *regular transition system* (RTS for short) over Σ is a pair $(\mathcal{I}, \mathcal{R})$, where

$\mathcal{I}$ is a regular set over Σ, denoting a set of *initial configurations*, and
$\mathcal{R}$ is a regular relation on Σ^*, denoting the *transition relation*.

A *fair regular transition system* (FRTS for short) over Σ is a tuple $(\mathcal{I}, \mathcal{R}, \mathcal{F})$, where $(\mathcal{I}, \mathcal{R})$ is an RTS and $\mathcal{F}$ is a regular set over Σ, denoting the set of *accepting configurations*. Transition relations and regular sets are typically represented by transducers and automata, or by regular expressions. □

A *configuration* w of an RTS $(\mathcal{I}, \mathcal{R})$ is a word $a_1\ a_2\ \cdots\ a_n \in \Sigma^*$. A *computation* of $(\mathcal{I}, \mathcal{R})$ is a finite or infinite sequence $w^0, w^1, w^2, \ldots$ of configurations such that $w^0 \in \mathcal{I}$ and $w^i \mathcal{R} w^{i+1}$ for all adjacent pairs of configurations. A configuration is *reachable* if it occurs in some computation. An infinite computation $w^0, w^1, w^2, \ldots$ of a FRTS is *accepting* if $w^i \in \mathcal{F}$ for infinitely many i.

Many parameterized systems with linear or ring-shaped topologies can be modeled as regular transition systems, by letting each position in a configuration model the local state of a system component. As an example of a parameterized system, we describe the mutual exclusion algorithm by Burns. In the algorithm, an arbitrary number of processes compete for a critical section. The processes are numbered, say from 1 to N. The *local state* of each process consists of a control state ranging over the integers from 1 to 7 and one Boolean flag, $flag$. A pseudo-code description of the behavior of process number i is shown in Figure 1. For instance, according to the code on line 4, if the control state of a process i is 4, and if the value of $flag$ is 1 for some process $j < i$, then the control state of i may be changed to 1; otherwise to 5. Line 7 represents the critical section.

To model Burns' algorithm as an RTS, we let Σ be the set of possible local states, e.g., represented as tuples $\langle pc, flag \rangle$. A system configuration is a word in Σ^*. The effect of line j can be represented by a regular relation α_j. For instance, α_1 corresponds to $Id \cdot [(pc = 1) \longrightarrow (pc := 2, flag := 0)] \cdot Id$ where the notation $(pc = 1) \longrightarrow (pc := 2, flag := 0)$ represents the relation

```
1:  flag[i] := 0
2:  if ∃j < i : flag[j] = 1 then goto 1
3:  flag[i] := 1
4:  if ∃j < i : flag[j] = 1 then goto 1
5:  await ∀j > i : flag[j] ≠ 1
6:  flag[i] := 0
7:  goto 1
```

Fig. 1. Burns' mutual exclusion algorithm

$\{(\langle pc_1, flag_1\rangle, \langle pc_2, flag_2\rangle) \mid pc_1 = 1, pc_2 = 2 \text{ and } flag_2 = 0\}$. To distinguish between branches, let α_{ja}, α_{jb} denote the *if* and *else* branch of α_j, for $j = 2, 4$.

It is also possible to model programs that operate on linear unbounded data structures such as queues, stacks, integers, etc. For instance, a stack can be modeled by letting each position in the word represent a position in the stack. The stack should initially contain an arbitrary but bounded number of empty stack positions, which are "statically allocated". We can then faithfully model all finite computations of the system, by initially allocating sufficiently many empty stack positions. We will consider two verification problems:

Reachability: Compute the set of reachable states of a given RTS $(\mathcal{I}, \mathcal{R})$, i.e., the set $\mathcal{I} \circ \mathcal{R}^*$. The problem of verifying any safety property can in the standard way be reduced to that of computing the set of reachable states of a suitable RTS.

Repeated reachability: Does a given FRTS $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ have an infinite accepting computation? The problem of verifying a liveness properties can, using the classical automata-theoretic framework [31] adapted to regular model checking [3], be reduced to the problem of repeated reachability of a suitable FRTS. A repeated reachability problem can be checked by computing the transitive closure of a transition relation, to be described in Section 4.

In general, these problems are undecidable, but techniques have been developed which are complete for certain classes of RTSs, and also verify examples from the literature (e.g., [25,8]).

3 Verification Using Acceleration

We can attempt to compute both reachable and repeatedly reachable configurations by standard fixpoint iterations. Let us describe this for the case of reachability. A naive computation of the set $\mathcal{I} \circ \mathcal{R}^*$ of reachable states is to compute the sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \cdots$, where $\mathcal{C}_0 = \mathcal{I}$ and $\mathcal{C}_{i+1} = \mathcal{C}_i \cup (\mathcal{C}_i \circ \mathcal{R})$, until a fixpoint is reached, i.e., $\mathcal{C}_{k+1} = \mathcal{C}_k$ for some k. This approach is guaranteed to terminate for finite-state systems, but not in general for parameterized and infinite-state systems, since there is no uniform bound on the number of computation steps needed to reach any particular configuration. For RTSs, $\mathcal{I} \circ \mathcal{R}^*$

and $\mathcal{R}^+$ are in general not computable, but incomplete techniques have been developed [7,16,19], which are guaranteed to complete under conditions which are typically satisfied when $\mathcal{R}$ is "simple", but not when $\mathcal{R}$ is the entire transition relation of an RTS. We therefore present a method to compute $\mathcal{I} \circ \mathcal{R}^*$ or $\mathcal{R}^+$ by decomposing $\mathcal{R}$ into "simple" parts, compute the transitive closure of each part, and then use the results in a refined fixpoint computation.

To this end, let an *action* of the RTS $(\mathcal{I}, \mathcal{R})$ be any subset of $\mathcal{R}$. We use α to range over actions. By *acceleration*, we mean to compute α^+ from α. The fixpoint computation described in the previous paragraph is modified by instead defining $\mathcal{C}_{i+1}$ as the result of choosing an appropriate $\alpha_i \subseteq R^+$, and letting $\mathcal{C}_{i+1} = \mathcal{C}_i \cup (\mathcal{C}_i \circ \alpha_i^+)$. The test for convergence remains the same: is $\mathcal{C}_i = \mathcal{C}_i \cup (\mathcal{C}_i \circ \mathcal{R})$? The main problem is to decide how to choose the sequence of actions $\alpha_0\ \alpha_1\ \cdots$ to accelerate, in order to converge at $\mathcal{I} \circ \mathcal{R}^*$.

We will consider the class of *unary* RTS, in which each computation step changes at most one position in a configuration. This class contains many parameterized synchronization algorithms. For unary RTSs, there is a particular class of actions (called *separable*) which can be accelerated efficiently.

Definition 2. *A regular relation $\mathcal{R}$ is* unary *if w and w' differ in at most one position whenever $w\ \mathcal{R}\ w'$. A RTS $(\mathcal{I}, \mathcal{R})$ is* unary *if $\mathcal{R}$ is unary. A unary relation is* separable *if it is of form $\phi_L \cdot \tau \cdot \phi_R$, where $\phi_L, \phi_R \subseteq Id$, and τ is a relation on Σ. We call ϕ_L the* left context *and ϕ_R the* right context *of $\phi_L \cdot \tau \cdot \phi_R$.*

Separable unary actions are interesting, because there are efficient techniques for accelerating them, which are complete when ϕ_L and ϕ_R satisfy certain conditions that hold for a majority of separable unary actions encountered in practice [6,25], and yield good underapproximations otherwise. Our verification strategy is therefore to generate a sequence $\alpha_0\ \alpha_1 \cdots$ of separable unary actions to drive the above modified fixpoint computation. To avoid overapproximation, we must obviously require $\alpha_i \subseteq \mathcal{R}^*$ for each i. To make the fixpoint computation as powerful as possible, we will generate as "large" actions as possible. By this, we will mean that any unary separable action in $\mathcal{R}$ is subsumed. We would also like to require the same for any composition of such actions, but this is not possible, since if α and α' are separable unary actions, then in general $\alpha \circ \alpha'$ is not unary and $\alpha \cup \alpha'$ is not separable. We therefore define restricted versions of these operations, *separable composition* $\circ_s$ and *separable union* $\cup_s$, as follows

$$(\phi_L \cdot \tau \cdot \phi_R)\ \circ_s\ (\phi'_L \cdot \tau' \cdot \phi'_R) \ \triangleq\ (\phi_L \cap \phi'_L) \cdot \tau \circ \tau' \cdot (\phi_R \cap \phi'_R)$$
$$(\phi_L \cdot \tau \cdot \phi_R)\ \cup_s\ (\phi'_L \cdot \tau' \cdot \phi'_R) \ \triangleq\ (\phi_L \cap \phi'_L) \cdot \tau \cup \tau' \cdot (\phi_R \cap \phi'_R)$$

where the changes in α and α' are constrained to occur in the same position. The resulting actions are separable, and can be efficiently accelerated.

Definition 3. *Let $\mathcal{R}$ be a regular relation. A set of actions $\mathcal{A}$ is* separable-complete with respect to $\mathcal{R}$*, if it satisfies:*
(U) For any sequence $\alpha_1, \ldots, \alpha_n$ of separable unary actions, where $\alpha_j \subseteq \mathcal{R}$ for $j \in [1, n]$, there is an action $\alpha \in \mathcal{A}$ such that

$$(\alpha_1 \circ_s \ldots \circ_s \alpha_n)^+ \subseteq \alpha^+$$

If condition (U) is true for $n \leq k$, for some bound k, the set is separable-complete up to k*, and k is called the* composition depth*.* □

As a special case, if $\mathcal{A}$ is separable-complete up to 1, then any separable unary action $\alpha' \subseteq \mathcal{R}$ is subsumed by some $\alpha \in \mathcal{A}$.

Let us see why separable-completeness is relevant for Burns' algorithm. Imagine that we are computing $\mathcal{I} \circ \mathcal{R}^*$ for Burns' algorithm, using a fixpoint computation. Consider a configuration where there are arbitrarily many processes on line 2, each with α_{2b} enabled. It is then possible for any single process to proceed to line 5, via lines 3 and 4. However, whenever α_3 is executed by some process i, all processes $j < i$ are blocked. Hence, in order for arbitrarily many processes to move from 2 to 5, they must act sequentially from higher to lower index. It follows that we need the accelerated sequential composition $(\alpha_{2b} \circ_s \alpha_3 \circ_s \alpha_{4b})^+$, to capture this behaviour; a fixpoint computation using only $\alpha_{2b}^+, \alpha_3^+, \alpha_{4b}^+$ would need unboundedly many computation steps. If (U) were true, we would have an action with $\alpha^+ \supseteq (\alpha_{2b} \circ_s \alpha_3 \circ_s \alpha_{4b})^+$, allowing us to compute the set of reachable configurations.

We are now ready to present our technique for generating actions to be accelerated in the fixpoint computation; it will automatically generate a finite set of actions which is separable-complete.

Generation Procedure. Our procedure for generating a sequence of actions that satisfy condition (U) has three steps.

1. We obtain any finite set of separable actions $\mathcal{A}'$ such that $\mathcal{R} = \cup \mathcal{A}'$.
One way to do this is to extract such actions from a representation of $\mathcal{R}$ as a minimal deterministic automaton $\mathcal{T} = \langle S, \Sigma \times \Sigma, s_0, \delta, F \rangle$, as follows. Let $\mathcal{T}(s, Q)$ equal $\mathcal{T}$ but with $s_0 = s$ and $F = Q$. Then $\mathcal{R}$ is the union of actions $\{\phi_L \cdot \tau \cdot \phi_R\}$ where $\phi_L = \mathcal{T}(s_0, \{s\}) \cap Id$, and $\phi_R = \mathcal{T}(t, F) \cap Id$, and $\tau = \delta(s, t)$ for states $s, t \in S$ (and $\phi_L, \phi_R, \tau \neq \emptyset$).

2. We thereafter transform $\mathcal{A}'$ so that it has the property that any separable unary action $\alpha \subseteq \mathcal{R}$ is *in* (i.e., a subset of) the separable union of some actions in $\mathcal{A}'$. For this purpose, we define two operations on separable unary actions:

$$(\phi_L \cdot \tau \cdot \phi_R) \sqcap_L (\phi'_L \cdot \tau' \cdot \phi'_R) \triangleq (\phi_L \cap \phi'_L) \cdot (\tau \cap \tau') \cdot (\phi_R \cup \phi'_R)$$
$$(\phi_L \cdot \tau \cdot \phi_R) \sqcap_R (\phi'_L \cdot \tau' \cdot \phi'_R) \triangleq (\phi_L \cup \phi'_L) \cdot (\tau \cap \tau') \cdot (\phi_R \cap \phi'_R)$$

Closing the set of actions under the operations $\sqcap_L$ and $\sqcap_R$ achieves the goal. As an optimization, we delete actions that are then subsets of other actions.

3. Finally, we close the set of actions $\mathcal{A}'$, from previous step, under $\cup_s$. Again, as an optimization, we delete actions that become subsets of other actions.

We motivate step **2** for Burns' algorithm. Suppose step **1** is applied to a deterministic representation of $\mathcal{R}$. We get $\mathcal{A}' \supseteq \{\alpha, \alpha'\}$, with $\alpha = \phi_{L4a} \cdot (\tau_3 \cup \tau) \cdot Id$, and $\alpha' = \phi_{L4b} \cdot (\tau_3 \cup \tau') \cdot Id$, for some τ, τ'. The desired property is false: α_3 is not in the separable union of α, α' (nor of $\mathcal{A}'$). The left context of α_3 has been

divided. However, $\alpha_3 = (\alpha \sqcap_R \alpha')$, giving the desired property. Without step **2**, our procedure underapproximates α_3 and sequential compositions involving α_3.

The generated actions are separable-complete up to 1 by construction (by steps **2** and **3**). Let us now establish that they are even separable-complete. We use the following lemma, which establishes how $\circ_s$ and $\cup_s$ are related.

Lemma 1. *Let $\mathcal{A}' = \{\alpha_1, \ldots, \alpha_m\}$ be a set of separable unary actions, with $\alpha_j = \phi_L^j \cdot \tau_j \cdot \phi_R^j$, for $j \in [1, m]$. Let $\sigma = \alpha_{i_1} \circ_s \alpha_{i_2} \circ_s \ldots \circ_s \alpha_{i_n}$ be any composition such that each $\alpha \in \mathcal{A}'$ occurs at least once. Then:*

$$\sigma \subseteq \phi_L^1 \cap \ldots \cap \phi_L^m \cdot (\tau_1 \cup \ldots \cup \tau_m)^+ \cdot \phi_R^1 \cap \ldots \cap \phi_R^m \qquad \square$$

Theorem 1. *The set of actions generated by steps* **1**–**3** *is separable-complete.*

Proof. Given any sequence $\alpha_1, \ldots, \alpha_n$, where $\alpha_j \subseteq \mathcal{R}$ for $j \in [1, n]$. Let us denote the fact that the generated actions have composition depth 1 by (U_1). By (U_1), there are actions $\alpha'_1, \ldots, \alpha'_n$ generated by our procedure such that $\alpha_j \subseteq \alpha'_j$, for each j. Again by (U_1), there exists a generated $\alpha = \phi_L \cdot \tau \cdot \phi_R$ such that $\alpha \supseteq \alpha'_1 \cup_s \ldots \cup_s \alpha'_n$. Now, by the lemma, $\alpha'_1 \circ_s \ldots \circ_s \alpha'_n \subseteq \phi_L \cdot \tau^+ \cdot \phi_R$. Finally, $(\phi_L \cdot \tau^+ \cdot \phi_R)^+ \subseteq \alpha^+$. $\square$

Note on complexity. Our procedure is essentially conjoining the guards of the actions; so an upper bound of the number of obtained actions is $2^{|\mathcal{A}'|}$, where $\mathcal{A}'$ is the least set satisfying the property of step **2**. For our benchmark (see Section 5), the actions can only be composed in a monotonic order, so the bound is only $|\mathcal{A}'|^2$. Nonetheless, in practice, we may choose to combine actions under $\cup_s$ a fixed number of iterations in step **3**, obtaining $\mathcal{A}$ with composition depth k.

4 Verifying Liveness

In this section, we describe how to verify liveness properties, which are reduced to the repeated reachability problem of a suitable FRTS. In particular, we describe how liveness properties of parameterized algorithms are verified using our *LTL*(*MSO*) model checker [3].

Recall that the falsification of a liveness property can be reduced to checking whether an FRTS has an infinite accepting run. Since the transition relation is length-preserving, so that each computation can visit only a finite set of configurations, this problem can be solved by repeated reachability, i.e., by checking whether there exists a reachable loop containing some configuration from $\mathcal{F}$. This is equivalent to checking whether there is a reachable configuration w in $\mathcal{F}$ such that $(w, w) \in Id \cap \mathcal{R}^+$, which can be checked as follows [27].

(1) Compute the set of *reachable configurations* $Inv = \mathcal{I} \circ \mathcal{R}^*$, as described in Section 3.
(2) Let $Inv_{\mathcal{F}} = \{(w, w') \mid w \in Inv \cap \mathcal{F}, (w, w') \in \mathcal{R}\}$, i.e., the *relation* containing all pairs of consecutive reachable configurations, where the first satisfies $\mathcal{F}$.

(3) Compute the relation $Inv_{\mathcal{F}} \circ \mathcal{R}^*$ as a fixpoint, which in the acceleration-based version constructs the sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \cdots$, where $\mathcal{C}_0 = Inv_{\mathcal{F}}$ and $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \mathcal{C}_i \circ \alpha_i^+$ for a suitable action $\alpha_i \subseteq \mathcal{R}^+$, until $\mathcal{C}_i = \mathcal{C}_i \cup \mathcal{C}_i \circ \mathcal{R}$.
(4) If the fixpoint computation in (3) converges, a repeatedly reachable configuration w exists if and only if $(Inv_{\mathcal{F}} \circ \mathcal{R}^*) \cap Id$ is non-empty.

Note that if $\mathcal{C}_i \cap Id$ is non-empty for some approximation $\mathcal{C}_i$, we can abort the fixpoint computation of (3), and report that $\mathcal{C}_i$ contains a repeatedly reachable configuration.

The reachability phase (1) computes a fixpoint on sets of configurations, while the repeated reachability phase (3) computes a fixpoint on relations of configurations; the latter is significantly more difficult to compute.

We next show how this procedure specializes to verifying absence of starvation for parameterized systems. A typical liveness property, absence of starvation, is of form $\Box \forall i\, (\phi(i) \longrightarrow \Diamond \psi(i))$, where i ranges over processes modeled by positions in the configuration. For instance, for Burns' algorithm we check the property $\Box \forall i\, ((\mathrm{pc}[i] = 1 \wedge i = 0) \longrightarrow \Diamond \mathrm{pc}[i] = 7)$. This property is proven assuming *weak process fairness*, i.e., that in an infinite computation, each process is infinitely often either blocked or progressing, which can be expressed as $\forall i \Box\Diamond(\alpha(i) \vee \neg En(\alpha(i)))$, where $\alpha(i)$ is a disjunction of all actions process i can take, and $En(\alpha(i))$ is true if and only if process i is not blocked.

To verify absence of starvation using the automata-theoretic approach [31,3], the transition relation, fairness requirements and the negation of the liveness properties are conjoined and compiled into an FRTS, which accepts all fair computations of the system which violate the liveness property, i.e., satisfy $\Diamond \exists i\, (\phi(i) \wedge \Box \neg \psi(i))$. The negation of the liveness property is transformed into an extra boolean component $b_{violate}(i)$ in the local state of each position i, such that if $b_{violate}(i)$ is true, then process i satisfies $\Box \neg \psi(i)$. Process i may non-deterministically set $b_{violate}(i)$ to true. The weak fairness requirement is transformed into an extra boolean component $b_{fair}(i)$ in the local state of each position i and the set $\mathcal{F}$ of accepting configurations in which $b_{fair}(i)$ is 1 for all i and $b_{violate}(i)$ is 1 for some i. All components $b_{fair}(i)$ are set to 0 immediately after some configuration in $\mathcal{F}$ was visited, and each $b_{fair}(i)$ is thereafter set to 1 whenever process i satisfies $\alpha(i) \vee \neg En(\alpha(i))$. The repeated reachability problem becomes to check whether there is an infinite computation which first visits a configuration where $b_{violate}(i)$ is 1 for some i, and thereafter infinitely often visits a configuration in $\mathcal{F}$.

The above procedure can be adapted to this setting by inserting a step $(1')$ between steps (1) and (2), which computes the set $Inv' = [Inv \wedge \exists i(\phi(i) \wedge b_{violate}(i))] \circ \mathcal{R}'^*$, where $\mathcal{R}'$ is $\mathcal{R}$ constrained to follow the semantics of $b_{violate}$, as described above. For the remaining steps, Inv' and $\mathcal{R}'$ take the roles of Inv and $\mathcal{R}$. For step (3), we further constrain $\mathcal{R}'$ to follow the semantics of b_{fair}. We have also added the following optimizations to our model checker.

– *Separating updates of* $b_{fair}(i)$. We separate the updates of $b_{fair}(i)$ into one action that sets it when $\alpha(i)$ is taken, and one action that sets it when

$\neg En(\alpha(i))$; this equivalent modeling makes the acceleration work more efficiently, since actions remain unary.

- *One violating witness.* We constrain the transition relation so that $b_{violate}(i)$ can be true for *at most* one process i; this simplifies the transition relation. Note that this does not forbid other processes from violating the property.

5 Experimental Results

We have implemented the systematic acceleration method described in this paper in our *LTL*(*MSO*) model checker [3], and used it to generate the set of reachable states, as described in Section 3, and to check absence of individual starvation under weak fairness for parameterized synchronization algorithms from the literature, as described in Section 4. The models are described in detail in [27], and are available at `http://user.it.uu.se/~mayanks/systematic/`. For the Bakery algorithm, we verified the property $\mathsf{Ba} \triangleq \Box\forall i\,(\mathrm{q}[i] = w \longrightarrow \Diamond \mathrm{q}[i] = cs)$. All other checked liveness properties were of form $\Box\forall i\,(\phi(i) \longrightarrow \Diamond\psi(i))$, where $\psi(i)$, defined as $\mathrm{pc}[i] = cs$, represents that process i is in the critical section, and where $\phi(i)$ expresses that process i intends to reach the critical section, and that also it is reasonable to suspect that process i is guaranteed to succeed in doing so. For each choice of $\phi(i)$ our implementation either reports a success in verification, or a counterexample. We checked several properties, whose $\phi(i)$ are given below, named after the initial letters of their corresponding protocols; Bakery, Burns, Szymanski, Dijkstra.

$$
\begin{array}{lll}
\mathsf{Bu}_1 : \mathrm{pc}[i] = 1 \wedge i = 0 & & \mathsf{Sz}_1 : \mathrm{pc}[i] = 1 \\
\mathsf{Bu}_2 : \mathrm{pc}[i] = 1 \wedge i \neq 0 & & \mathsf{Sz}_2 : \mathrm{pc}[i] = 2 \\
\mathsf{Bu}_3 : \mathrm{pc}[i] \neq 1 \wedge i \neq 0 & & \mathsf{Sz}_3 : \mathrm{pc}[i] \neq 1 \\
\mathsf{Bu}_4 : i = 0 & & \\
\mathsf{Di}_1 : \mathrm{p}[i] \wedge \mathrm{flag}[i] \neq 0 \wedge \forall j \neq i \,.\, \mathrm{pc}[j] \neq 3 & & \\
\mathsf{Di}_2 : \mathrm{p}[i] \wedge \mathrm{flag}[i] = 0 \wedge \forall j \neq i \,.\, \mathrm{pc}[j] \neq 3 & &
\end{array}
$$

We used composition depth k as a parameter, successively using higher values if the verification did not succeed within a certain time bound. The times are given for the best values of k, not including "too low k" time. All protocols worked with some $k \in [2, 5]$. Dijkstra's protocol needed 5, and Szymanski's protocol was significantly slower with $k > 2$, due to its actions using many different guards. If the generated actions become separable-complete for parameter k, using a higher value is not significantly slower, as testing for separable-completeness is quick. By Lemma 1, we need not consider k higher than the number of actions generated in step **2** of the generation procedure – that k gives the best approximation of $\mathcal{R}^+$, but can be suboptimal with respect to time. To handle the fact that one action of Dijkstra's protocol is not unary, we extended the composition techniques to a class of non-unary actions in the most straight-forward way. The experiments were run on a PC with a 2.4 GHz processor and 1 GB of RAM.

Table 1. Liveness (to the left) and safety (to the right) results

Property	This work	[27,3]	[5]
Ba	13	23	36
Bu_1	98		450
Bu_2	56 (f)		
Bu_3	60 (f)		
Bu_4	144		
Sz_1	540 (f)		
Sz_2	1369		435
Sz_3	1635		
Di_1	244		3311
Di_2	1031 (f)		

Protocol	This work	[27,3]
Bakery	4	5
Burns	15	39
Szymanski	19	34
Dijkstra	25	38

Results and Comparison with Related Work. Our verification results are presented in Table 1. The table contains time measured in seconds for the analysis, but does not include the translation from LTL(MSO) formulas into FRTSs. False properties, for which a counterexample was found, are marked "(f)". In the table, we compare our times with the works [27,3,5], as they use similar techniques, and were in fact timed on the same system. We also present related work, in alphabetical order with respect to authors. Note that works [27,3,28] could only have succeeded to verify Burns' and Dijkstra's protocols if the right sequential compositions were included; but they are difficult to find manually.

[2] Abdulla et al. use overapproximation for safety properties, obtaining times an order of magnitude better than ours (0.004–3.9 seconds), but the technique can not be extended to liveness properties.

[5] These techniques compute states which are guaranteed to satisfy $\psi(i)$ using backwards reachability, thus avoiding the repeated reachability problem. However, they are not able to produce counterexamples, and are sometimes slower (due to requiring many accelerations).

[15] Bouajjani et al. verify liveness of Bakery, as well as safety of all listed protocols, using counter-example guided abstractions, in 0.06–0.73 seconds.

[21,22,20] Fang et al. verify the Bakery protocol using automatically generated ranking functions, but do not report running times.

[27,3] The works of Nilsson et al. [27,3] report times for essentially the same technique, so we gave the best time for each protocol. The verification setting is as ours, but without the systematic addition of sequential compositions.

[28] Pnueli and Shahar, use user defined accelerations to verify safety properties of Szymanski's protocol in 0.2 seconds, as well as some protocols not in our benchmark.

[29] Pnueli et al. verify liveness of the Bakery and Szymanski protocols using manually supplied counter abstractions, in 1 and 96 seconds respectively. Their modeling of Szymanski's protocol is slightly different from ours, so we can not say which of the true properties were checked.

Using the techniques of this paper, we can compute an exact representation of the reachable loops for all the above protocols. It has, to our knowledge, never been done for Burns' and Dijkstra's protocols before.

6 Conclusions and Future Work

We have presented a systematic method for using acceleration to speed up fixpoint computations in regular model checking. The method is defined for unary transition relations, and is independent of how the transition relation is represented. We show how to accelerate a set of actions which is maximal in a certain sense, in order to make the verification as powerful as possible. Using this approach, we have succeeded in verifying safety and liveness of parameterized synchronization protocols, whose verification has not been reported before.

Our work shows that acceleration-based symbolic state-space exploration can be used efficiently also in regular model checking, thus extending this approach from other classes of systems (e.g., [1,12,14,32,10,23]). Future work includes extending the approach to non-unary transition relations, in order to handle, e.g., systems with synchronous communication between adjacent processes.

References

1. Abdulla, P., Collomb-Annichini, A., Bouajjani, A., Jonsson, B.: Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design 25(1), 39–65 (2004)
2. Abdulla, P., Delzanno, G., Henda, N.B., Rezine, A.: Regular model checking without transducers. In: Proc. TACAS '07 (to appear, 2007)
3. Abdulla, P., Jonsson, B., Nilsson, M., d'Orso, J., Saksena, M.: Regular model checking for MSO + LTL. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 348–360. Springer, Heidelberg (2004)
4. Abdulla, P., Jonsson, B., Nilsson, M., Saksena, M.: A survey of regular model checking. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 35–48. Springer, Heidelberg (2004)
5. Abdulla, P., Jonsson, B., Saksena, M., Rezine, A.: Proving liveness by backwards reachability. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 95–109. Springer, Heidelberg (2006)
6. Abdulla, P.A., Bouajjani, A., Jonsson, B., Nilsson, M.: Handling global conditions in parameterized system verification. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 134–145. Springer, Heidelberg (1999)
7. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Brim, L., Jančar, P., Křetínský, M., Kucera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 116–130. Springer, Heidelberg (2002)
8. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Algorithmic improvements in regular model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 236–248. Springer, Heidelberg (2003)
9. Annichini, A., Asarin, E., Bouajjani, A.: Symbolic techniques for parametric reasoning about counter and clock systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 419–434. Springer, Heidelberg (2000)

10. Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P.: Flat acceleration in symbolic model checking. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 474–488. Springer, Heidelberg (2005)
11. Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 1–12. Springer, Heidelberg (1996)
12. Boigelot, B., Godefroid, P., Willems, B., Wolper, P.: The power of QDDs. In: Van Hentenryck, P. (ed.) SAS 1997. LNCS, vol. 1302, Springer, Heidelberg (1997)
13. Boigelot, B., Wolper, P.: Symbolic verification with periodic sets. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 55–67. Springer, Heidelberg (1994)
14. Bouajjani, A., Habermehl, P.: Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. Theoretical Computer Science 221(1-2), 211–250 (1999)
15. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 372–386. Springer, Heidelberg (2004)
16. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000)
17. Burch, J., Clarke, E., McMillan, K., Dill, D.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98, 142–170 (1992)
18. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and presburger arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, Springer, Heidelberg (1998)
19. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. J. Log. Algebr. Program. 52-53, 109–127 (2002)
20. Fang, Y., McMillan, K.L., Pnueli, A., Zuck, L.D.: Liveness by invisible invariants. In: FORTE, pp. 356–371 (2006)
21. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.: Liveness with incomprehensible ranking. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 482–496. Springer, Heidelberg (2004)
22. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.: Liveness with invisible ranking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 223–238. Springer, Heidelberg (2004)
23. Finkel, S., Leroux, J.: How to compose presburger-accelerations: Applications to broadcast protocols. In: Agrawal, M., Seth, A.K. (eds.) FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002)
24. Habermehl, P., Vojnar, T.: Regular model checking using inference of regular languages. Electr. Notes Theor. Comput. Sci. 138(3), 21–36 (2005)
25. Jonsson, B., Nilsson, M.: Transitive closures of regular relations for verifying infinite-state systems. In: Schwartzbach, M.I., Graf, S. (eds.) ETAPS 2000 and TACAS 2000. LNCS, vol. 1785, Springer, Heidelberg (2000)
26. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theoretical Computer Science 256, 93–112 (2001)
27. Nilsson, M.: Regular Model Checking. PhD thesis, Uppsala University (2005)
28. Pnueli, A., Shahar, E.: Liveness and acceleration in parameterized verification. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 328–343. Springer, Heidelberg (2000)

29. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infinity)-counter abstraction. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, Springer, Heidelberg (2002)
30. Szymanski, B.K.: Mutual exclusion revisited. In: Proc. Fifth Jerusalem Conference on Information Technology, pp. 110–117. IEEE Computer Society Press, Los Alamitos (1990)
31. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. LICS '86, 1^{st} LICS, pp. 332–344 (June 1986)
32. Wolper, P., Boigelot, B.: Verifying systems with infinite but regular state spaces. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 88–97. Springer, Heidelberg (1998)