Block Me If You Can![*]

Context-Sensitive Parameterized Verification

Parosh Aziz Abdulla[1], Frédéric Haziza[1], and Lukáš Holík[2]

[1] Uppsala University, Sweden
[2] Brno University of Technology, Czech Republic

Abstract. We present a method for automatic verification of systems with a parameterized number of communicating processes, such as mutual exclusion protocols or agreement protocols. To that end, we present a powerful abstraction framework that uses an efficient and precise symbolic encoding of (infinite) sets of configurations. In particular, it generalizes *downward-closed* sets that have successfully been used in earlier approaches to parameterized verification. We show experimentally the efficiency of the method, on various examples, including a fine-grained model of Szymanski's mutual exclusion protocol, whose correctness, to the best of our knowledge, has not been proven automatically by any other existing methods.

1 Introduction

We consider the verification of safety properties for *parameterized systems*: systems that consist of an arbitrary number of components (processes) organized according to a certain predefined topology. In this paper, we consider the case where the system has a linear topology (the processes form an array). Our method can be easily extended to other topologies such as rings, trees, or multisets (the latter are systems where the processes are entirely anonymous, e.g. Petri nets). Processes can perform two types of transitions, namely *local* and *global* transitions. In the former, the process does not need to check the states of the rest of the processes in the system. A global transition is either *universal* or *existential*. For example, in a universal transition, a process (at position i) may perform a transition only if *all* processes to its left (i.e. with index $j < i$) satisfy a property φ. In an existential transition, it is required that *some* (rather than *all*) processes satisfy φ. Parameterized systems arise naturally in the modeling of mutual exclusion algorithms, bus protocols, distributed algorithms, telecommunication protocols, and cache coherence protocols. The task is to perform *parameterized verification*, i.e. to verify correctness regardless of the number of processes. This amounts to the verification of an infinite family; namely one for each possible size of the system. We consider *safety properties*, i.e. properties that forbid reachability of bad configurations. For instance, mutual exclusion protocols must guarantee that no reachable configuration contains two processes in the critical section.

[*] supported by the Uppsala Programming for Multicore Architectures Research Center (UPMARC), the Czech Science Foundation (13-37876P, 14-11384S), Brno University of Technology (FIT-S-12-1, FIT-S-14-2486).

An important line of research in parameterized verification has been based on the observation that such systems may have invariants that are $downward\text{-}closed$ wrt. a natural ordering on the set of configurations (e.g. the subword ordering for systems with linear topologies, or the multiset ordering on Petri nets). The existence of downward-closed invariants allows the employment of well quasi-ordered transition systems [2, 1]. In particular, a downward-closed set D can be characterized by a finite set of counter-examples. This set contains the configurations that are the minimal elements of the complement of D (notice the complement of D is upward-closed). This characterization gives compact symbolic representations leading to very efficient implementations. This observation has resulted in several powerful frameworks such as the "Expand, Enlarge, and Check" method [26], monotonic abstraction [6], and small model based verification [4]. Although these frameworks are applicable to a wide range of parameterized systems, there are several classes of systems that are beyond their applicability. The reason is that such systems do not allow good downward-closed invariants, and hence over-approximating the set of reachable configurations by downward-closed sets will give false counter-examples. In this paper, we propose a method that targets a class of invariants which are needed in many practical cases and cannot be expressed as downward-closed sets, hence cannot be inferred by the above-mentioned methods. Specifically, we express invariants as quantified formulae over process indices and states within a configuration. The formulae are of the form:

$$\phi = \forall i_1, \ldots, i_n \; \exists i_{n+1}, \ldots, i_{n+m} : \psi(i_1, \ldots, i_{n+m})$$

where $i_1, \ldots, i_{n+m}$ are pairwise distinct *position variables* and $\psi(i_1, \ldots, i_{n+m})$ is a boolean formula that relates the process positions, their local states and the topological constraints at those positions. We call these properties *almost downward-closed* (henceforth $\forall_\exists$-formulae), since they are a generalization of downward-closed sets. Observe that downward-closed properties correspond to the special case where the formulae solely have universal quantification.

Let us illustrate the notion of an almost downward-closed good invariant with the example of a barrier implementation (see Fig. 1). All processes start in the state B before the barrier. The first process at the barrier moves to state P and acts as a *pivot*. All other arriving processes must wait in state W as long as there is a pivot. When all processes have arrived at the barrier, the pivot can proceed to the state A after the barrier, which in turn releases the waiting processes.

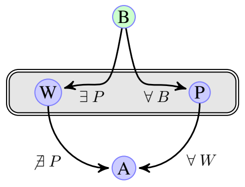


Fig. 1: Barrier.

The system is correct if there cannot be at the same time a process in the state B and a process in the state A. A waiting process W trying to move to the state A counts on the fact that *if* there is a process in B, *then* there is also a process in P. If this implication did not hold, the barrier would be incorrect, because the move from W to A could be performed under presence of B. The weakest good invariant must reflect this implication, and state that (i) A and B never coexist, and (ii) if W and B appear together then P is present. The first condition denotes a downward-closed set, any configuration that does not contain both A and B satisfies it. On the contrary, the second condition is not downward-closed. It is an implication of the form "contains W and B" ⇒ "must contain P", which can be characterized using the disjunction of a downward-closed set (the antecedent) and an upward-closed set (the consequent). (Recall $A \Rightarrow B \Leftrightarrow \neg A \vee B$ and when A is upward-closed, $\neg A$ is downward-closed). This example illustrates an almost downward-closed property, and also a situation where inferring such properties is needed. The system does not indeed have any good downward-closed invariant.

We propose a method that can fully automatically infer *almost downward-closed* invariants through the creation of *small models*. This allows to carry out parameterized verification *fully automatically* through analyzing, in an abstract domain, only a small number of processes (rather than the whole family). To define the abstraction, we will first introduce a new symbolic encoding, called *context-sensitive views*, that allows to characterize almost downward-closed sets. Context-sensitive views are generalizations of minimal elements used for characterizing downward-closed sets. They retain enough information in order to *disable* (or *block*) universal transitions, which would have been otherwise enabled without the presence of contexts. We show that our abstract predicate transformer is exact, so the method is guaranteed to find the weakest almost downward-closed good invariant (if it exists).

To simplify the presentation, we will assume in the first part of the paper that global transitions are performed atomically. However, in reality, such transitions are implemented as a for-loop ranging over process indices and do not assume atomicity. Moreover, any number of processes may be performing a for-loop simultaneously. This makes the model of fine-grained systems and the verification task significantly harder, since it requires to distinguish intermediate states of such for-loops. We show that our method retains its simplicity and efficiency when instantiated to the (more complicated) case of fine-grained parameterized systems where the atomicity assumption is dropped. To the best of our knowledge, it is the only method which combines the ability to infer almost downward-closed invariants with the support of fine-grained modeling. We have used it to fully automatically verify several systems which were not previously verified automatically. Among these, we highlight the *fully automatic* verification of the fine-grained and complete version of Szymanski's mutual exclusion protocol, which has been considered a challenge in parameterized verification.

Outline. We first consider a basic model in Section 2 which only allows atomically checked global conditions and instantiate the abstract domain for such systems in Section 4. We present our verification procedure in Section 5 and introduce in Section 6 how the settings are adapted to cope with non-atomicity. We report on our experimental results in Section 7, describe related work in Section 8 and conclude the paper in Section 9.

2 Parametrized Systems

We introduce a standard model [31, 13, 6, 30] of parameterized systems operating on a linear topology, where processes may perform local or global transitions. Formally, a *parameterized system* is a pair $\mathcal{P} = (Q, \Delta)$ where Q is a finite set of process *local states* and Δ is a set of *transition rules* over Q. A transition rule is either *local* or *global.* A local rule is of the form $s \to s'$, where the process changes its local state from s to s' independently from the local states of the other processes. A global rule is either *universal* or *existential*. It is of the form: $\mathbf{if}\, \mathbb{Q}\ j \circ i : S\ \mathbf{then}\ s \to s'$, where $\mathbb{Q} \in \{\exists, \forall\}$, $\circ \in \{<, >, \neq\}$ and $S \subseteq Q$. We call s the *source*, s' the *target*, $\mathbb{Q}$ the *quantifier* and $\circ$ the *range*. S represents a set of *witness* process states. Here, the i^{th} process checks the local states of the other processes before it makes the move. For the sake of presentation, we only consider, in this section, a version where each process checks atomically the other processes. The more realistic and more difficult case, where the atomicity assumption is dropped, will be introduced in Section 6. For instance, the condition $\forall\, j < i : S$ means that “every process j, with a lower index than i, should be in a local state that belongs to the set S”; the condition $\forall\, j {\neq} i : S$ means that “the local state of all processes, except the one at position i, should be in the set S”.

A *configuration* in $\mathcal{P}$ is a word over the alphabet Q. We use $\mathcal{C}$ to denote the set of all configurations and $c[i]$ to denote the state of the i^{th} process within the configuration c. We use $[\![a;b]\!]$ to denote the set of integers in the interval $[a;b]$ (i.e. $[\![a;b]\!] = [a;b] \cap \mathbb{N}$). For a configuration c, a position $i \leq |c|$, and a transition $\delta \in \Delta$, we define the immediate successor $\delta(c,i)$ of c under a δ-move of the i^{th} process (evaluating the condition) such that $\delta(c,i) = c'$ iff $c[i] = s$, $c'[i] = s'$, $c[j] = c'[j]$ for all $j : j \neq i$ and either (i) δ is a local rule $s \to s'$, or (ii) δ is a global rule of the form $\mathbf{if}\, \mathbb{Q}\ j \circ i : S\ \mathbf{then}\ s \to s'$, and one of the following two conditions is satisfied:
– $\mathbb{Q} = \forall$ and for all $j \in [\![1;|c|]\!]$ such that $j \circ i$, it holds that $c[j] \in S$
– $\mathbb{Q} = \exists$ and there exists some $j \in [\![1;|c|]\!]$ such that $j \circ i$ and $c[j] \in S$.

For a set of configurations $X \subseteq \mathcal{C}$, we define the *post-image* of X as the set $\mathit{post}(X) = \{\delta(c,i) \mid c \in X, i \leq |c|, \delta \in \Delta\}$.

An instance of the *reachability problem* is defined by a parameterized system $\mathcal{P} = (Q, \Delta)$, a set $I \subseteq Q^+$ of *initial configurations*, and a set $\mathcal{B} \subseteq Q^+$ of *bad configurations*. We say that $c \in \mathcal{C}$ is *reachable* iff there are $c_0, \ldots, c_l \in \mathcal{C}$ such that $c_0 \in I$, $c_l = c$, and for all $0 \leq i < l$, there are $\delta_i \in \Delta$ and $j \leq |c_i|$ such that $c_{i+1} = \delta_i(c_i, j)$. We use $\mathcal{R}$ to denote the set of all reachable configurations (from I). We say that the system $\mathcal{P}$ is *safe* with respect to I and $\mathcal{B}$ if no bad configuration is reachable, i.e. $\mathcal{R} \cap \mathcal{B} = \emptyset$.

The set I of initial configurations is usually a regular set. In order to define the set $\mathcal{B}$ of bad configurations, we use the usual *subword relation* $\sqsubseteq$, i.e., $u \sqsubseteq s_1 \ldots s_n$ iff $u = s_{i_1} \ldots s_{i_k}, 1 \leq i_1 < \ldots < i_k \leq n$. We assume that $\mathcal{B}$ is the upward-closure $\{c \in \mathcal{C} \mid \exists b \in \mathcal{B}_{min} : b \sqsubseteq c\}$ of a given *finite* set $\mathcal{B}_{min} \subseteq Q^+$ of *minimal bad configurations*. This is a common way of specifying bad configurations which often appears in practice.

3 Example: Szymanski's Protocol

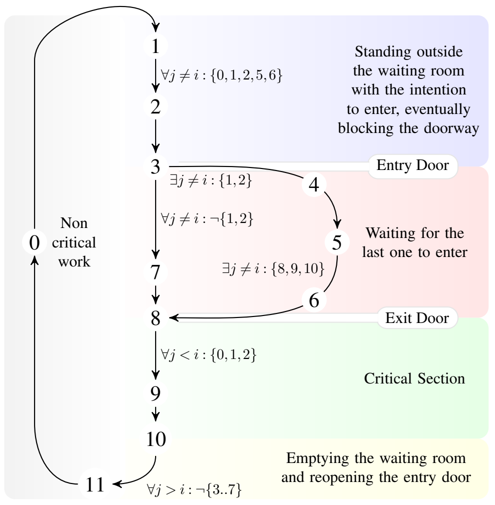
We illustrate the notion of a parameterized system with the example of Szymanski's mutual exclusion protocol [33]. The protocol ensures exclusive access to a shared resource in a system consisting of an unbounded number of processes organized in an array. The transition rules of the parameterized system are given in Fig. 3 and the source code in Fig. 2. The state of the i^{th} process is modelled with a number, which reflects the values of the program location and the local variable *flag*$[i]$. A configuration of the induced transition system is a word over the alphabet $\{⓪, \ldots, ⑪\}$ of local process states. The task is to check that the protocol guarantees exclusive access to the shared resource regardless of the number of processes. A configuration is considered to be bad if it con-

```
0  flag[i] = 1;
1  for(j=0;j<N;j++){ if(flag[j]≥3) goto 1; }
2  flag[i] = 3;
3  for(j=0;j<N;j++){
         if (flag[j] = 1) {
4              flag[i] = 2;
5              for(j=0;j<N;j++){ if(flag[j]==4) goto 7; }
6              goto 5;
         }
   }
7  flag[i] = 4;
8  for(j=0;j<i;j++){ if(flag[j]≥2) goto 8; }
9  /* Critical Section */
10 for(j=i+1;j<N;j++){ if(flag[j]==2||flag[j]==3) goto 10; }
11 flag[i] = 0; goto 0;
```

Fig. 2: Szymanski's protocol implementation (for process i)

tains two occurrences of state 9 or 10, i.e., the set of minimal bad configurations $\mathcal{B}_{min}$ is {9 9, 9 10, 10 9, 10 10}. Initially, all processes are in state 0, i.e. $I = 0^+$.

Many techniques [4, 3, 7, 13, 30, 8, 12] have been used to verify automatically the safety property of Szymanski's mutual exclusion protocol but only in restricted settings. They either assume atomicity of the global conditions and/or only consider a more compact variant of the protocol (i.e. where the invariant can be expressed solely by a downward-closed set). The full and fine-grained version has been considered a challenge in the verification community. To the best of our knowledge, this paper presents the first technique to address the challenge of verifying the protocol fully automatically without atomicity assumption.



Standing outside the waiting room with the intention to enter, eventually blocking the doorway
Entry Door
Waiting for the last one to enter
Exit Door
Critical Section
Emptying the waiting room and reopening the entry door
Non critical work
$\forall j \neq i : \{0,1,2,5,6\}$
$\exists j \neq i : \{1,2\}$
$\forall j \neq i : \neg\{1,2\}$
$\exists j \neq i : \{8,9,10\}$
$\forall j < i : \{0,1,2\}$
$\forall j > i : \neg\{3..7\}$

Fig. 3: Szymanski's protocol transition system

4 Views and $\forall_\exists$-Formulae

We introduce our symbolic encoding and show how it corresponds to $\forall_\exists$-formulae.

Context-sensitive views. A *context-sensitive view* (henceforth only called *view*) is a pair $(b_1 \dots b_k, R_0 \dots R_k)$, often written as $R_0 b_1 R_1 \dots b_k R_k$, where $b_1 \dots b_k$ is a configuration and $R_0 \dots R_k$ is a *context*, such that $R_i \subseteq Q$ for all $i \in [\![0;k]\!]$.

We call the configuration $b_1 \dots b_k$ the *base* of the view where k is its *size* and we call the set R_i the i^{th} *context*. We use $\mathcal{V}_k$ to denote the set of views of size up to k. For $k, n \in \mathbb{N}, k \leq n$, let H_n^k be the set of strictly increasing injections $h\colon [\![0;k+1]\!] \to [\![0;n+1]\!]$, i.e. $1 \leq i < j \leq k \implies 1 \leq h(i) < h(j) \leq n$. Moreover, we require that $h(0) = 0$ and $h(k+1) = n+1$.

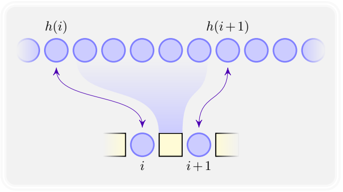


Fig. 4: Projection

Projections. We define the projection of a configuration. For $h \in H_n^k$ and a configuration $c = q_1 \dots q_n$, we use $\Pi_h(c)$ to denote the view $v = R_0 b_1 R_1 \dots b_k R_k$, obtained in the following way (see Fig. 4):
(i) $b_i = q_{h(i)}$ for $i \in [\![1;k]\!]$, (ii) $R_i = \{q_j \mid h(i) < j < h(i+1)\}$ for $i \in [\![0;k]\!]$. Intuitively, respecting the order, k elements of c are retained as the base of v, while all other elements are collected into contexts as sets in the appropriate positions.

We also define projections of views. For a view $v = R_0 b_1 R_1 \dots b_n R_n$ and $h \in H_n^k$, we overload the notation for the projection of configurations and use $\Pi_h(v)$ to denote the view $v' = R'_0 b'_1 R'_1 \dots b'_k R'_k$, such that (i) $b'_i = b_{h(i)}$ for $i \in [\![1;k]\!]$ and
(ii) $R'_i = \{b_j \mid h(i) < j < h(i+1)\} \cup (\bigcup_{h(i) \leq j < h(i+1)} R_j)$ for all $i \in [\![0;k]\!]$ (see Fig. 5).

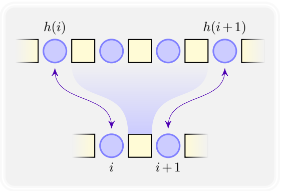


Fig. 5: View Projection

We define an *entailment relation* on views of the same size. Let $u = R_0 b_1 R_1, \dots, b_n R_n$ and $v = R'_0 b'_1 R'_1, \dots, b'_n R'_n$ be views of the same size n. We say that v *entails* u or that u is *weaker* than v, denoted $u \preccurlyeq v$, if $b_1 \cdots b_n = b'_1 \cdots b'_n$ and $R_i \subseteq R'_i$ for all $i \in [\![0;n]\!]$. Views of different sizes are not comparable. For two sets V and W of views, we write $V \preccurlyeq W$ if every $w \in W$ entails some $v \in V$. Formally, $V \preccurlyeq W \Leftrightarrow \forall w \in W, \exists v \in V, v \preccurlyeq w$. We use $\lfloor V \rfloor$ to denote the set of views in V that are *weakest*, i.e. minimal w.r.t. $\preccurlyeq$. We use $V \sqcup W$ to denote the set $\lfloor V \cup W \rfloor$.

Abstraction and Concretization. Let $k \in \mathbb{N}$. The *abstraction function* α_k maps x, a view or a configuration, into the set of its projections of the size k or smaller: $\alpha_k(x) = \{\Pi_h(x) \mid h \in H_{|x|}^{\ell}, \ell \leq \min(k, |x|)\}$. For a set X of views or of configurations, we define $\alpha_k(X)$ as the set $\lfloor \cup_{x \in X} \alpha_k(x) \rfloor$, i.e. its weakest projections. The *concretization* function γ_k maps a set of views $V \subseteq \mathcal{V}_k$ into the set of configurations $\gamma_k(V) = \{c \in \mathcal{C} \mid V \preccurlyeq \alpha_k(c)\}$.

We pinpoint the fact that views work *collectively*, rather than individually. That is, a set of configurations is characterized by a *set* of views. Consider for example that a set V of views contains the view $\text{WB}[\text{P}]$. We write contexts in square brackets and we omit empty contexts for brevity. Then, in order to characterize the configuration WBP, it must *also* contain the views $[\text{W}]\text{BP}$ and $\text{W}[\text{B}]\text{P}$ (or weaker). The three views together characterize the configuration, while the view $\text{WB}[\text{P}]$ alone cannot. Abstraction and concretization are illustrated on a larger example in Fig. 6.

Lemma 1. *For any $k \in \mathbb{N}$, $V \subseteq \mathcal{V}$ and $X \subseteq \mathcal{C}$, $X \subseteq \gamma_k(V) \iff V \preccurlyeq \alpha_k(X)$, i.e. the pair (α_k, γ_k) forms a Galois connection.*

For any set $X \subseteq \mathcal{C}$ and $k \in \mathbb{N}$, it is clear that $\gamma_k(\alpha_k(X)) \supseteq X$. In fact, we can observe that the precision of the abstraction increases with k, i.e. $\gamma_1(\alpha_1(X)) \supseteq \gamma_2(\alpha_2(X)) \supseteq$

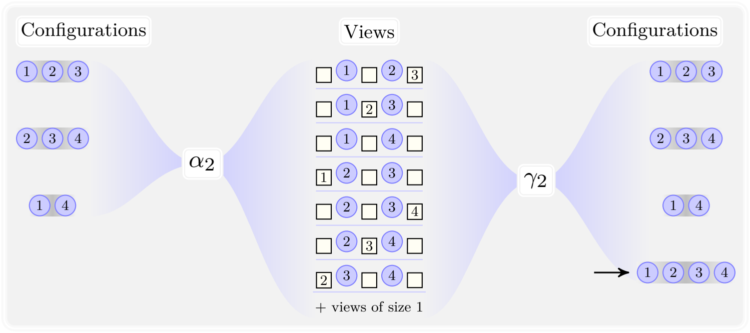


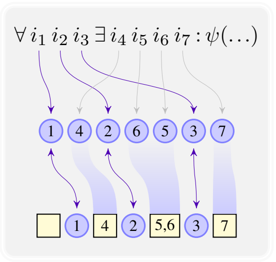
Fig. 6: Abstraction and Concretization

$\gamma_3(\alpha_3(X)) \supseteq \ldots \supseteq X$. We illustrate this property with the following example. Consider the set X of configurations of the barrier protocol from Fig. 1 described by the regular expression $\mathtt{BB}^+\mathtt{P}$. Its abstraction with $k = 1$ is the set of views $V_1 = \alpha_1(X) = \{\mathtt{B}[\mathtt{B},\mathtt{P}],[\mathtt{B}]\mathtt{B}[\mathtt{P}],[\mathtt{B}]\mathtt{P}\}$.The concretization $\gamma_1(V_1)$ is the set of configurations following the regular expression $\mathtt{B}(\mathtt{B}|\mathtt{P})^*\mathtt{B}(\mathtt{B}|\mathtt{P})^*\mathtt{P}$ (i.e. the information preserved is that configurations begin by B, end by P, and there are at least two Bs). With $k = 2$, we get $V_2 = \alpha_2(X) = \{\mathtt{BB}[\mathtt{P}],\mathtt{B}[\mathtt{B}]\mathtt{P},[\mathtt{B}]\mathtt{BP}\} \cup V_1$. Its concretization is $\gamma_2(V_2) = \mathtt{BB}^+\mathtt{P}$ which is equal to the original set X. The role of contexts may be seen already with $k = 1$: the concretization of V_1 preserves the information that there is at least one P and at least two Bs present in every configuration. This set is not downward-closed.

Views vs $\forall_\exists$-formulae. An $\forall_\exists$-*formula* is a formula of the form:

$$\phi = \forall i_1,\ldots,i_n\ \exists i_{n+1},\ldots,i_{n+m} : \psi(i_1,\ldots,i_{n+m})$$

where $i_1,\ldots,i_{n+m}$ are pairwise distinct *position variables* and $\psi(i_1,\ldots,i_{n+m})$ is a boolean combination of *basic formulae*. A basic formula is either (i) a *topological predicate* of the form $i_j < i_k$ or (ii) a *state predicate* of the form $c[i_j] = q$, where $j,k \in [\![1;n+m]\!]$ and $q \in Q$.


$\forall i_1\, i_2\, i_3\, \exists i_4\, i_5\, i_6\, i_7 : \psi(\ldots)$
1
4
2
6
5
3
7
1
4
2
5,6
3
7
Fig. 7: view $\leftrightarrow$ $\forall_\exists$-formula

The notion of satisfaction by a configuration c of a basic formula $\psi(i_1,\ldots,i_\ell)$ is defined in the natural way. More precisely, an assignment ρ is a function that maps the indices $i_1,\ldots,i_\ell$ to pairwise different positions within the configuration c (i.e. $\rho(i_j) \in [\![0;|c|]\!]$ and $\rho(i_j) \neq \rho(i_k)$ for all $j,k \in [\![1;\ell]\!]$). We write $c \models_\rho \psi$, if c satisfy the formula $\psi(i_1,\ldots,i_\ell)$ under the assignment ρ. We say that c satisfies ϕ and write $c \models \phi$, if for every assignment ρ of $i_1,\ldots,i_n$, there exists an assignment ρ' of $i_{n+1},\ldots i_{n+m}$ such that $c \models_{\rho\cup\rho'} \psi$. We use $[\![\phi]\!]$ to denote the set $\{c \in \mathcal{C} \mid c \models \phi\}$ of all configurations that satisfy ϕ.

Lemma 2. *For any set C of configurations, there exists an $\forall_\exists$-formula ϕ such that $C = [\![\phi]\!]$ iff there exists a finite set of views V and $k \in \mathbb{N}$ such that $C = \gamma_k(V)$.*

Lemma 2 shows that the $\forall_\exists$-formulae correspond to sets of views. Intuitively, the base of a view captures the predicates in an $\forall_\exists$-formula relating the position variables from the universal quantification, while the contexts capture those from the existential quantification. For example, we recall the set $V_1 = \alpha_1(X) = \{\mathtt{B}[\mathtt{B},\mathtt{P}], [\mathtt{B}]\mathtt{B}[\mathtt{P}], [\mathtt{B}]\mathtt{P}\}$, where X is the set of configurations described by the regular expression $\mathtt{B}\mathtt{B}^+\mathtt{P}$. Its concretization $\gamma_1(V_1)$ is expressed by the $\forall_\exists$-formula $\forall i\ \exists j,k : (c[i],c[j],c[k] = \mathtt{B},\mathtt{B},\mathtt{P} \wedge i < j,k) \vee (c[j],c[i],c[k] = \mathtt{B},\mathtt{B},\mathtt{P} \wedge j < i < k) \vee (c[j],c[i] = \mathtt{B},\mathtt{P} \wedge j < i)$. For $k = 2$, the concretization of $V_2 = \alpha_2(X) = \{\mathtt{B}\mathtt{B}[\mathtt{P}], \mathtt{B}[\mathtt{B}]\mathtt{P}, [\mathtt{B}]\mathtt{B}\mathtt{P}\} \cup V_1$ is expressed by the $\forall_\exists$-formula $\forall i,j\ \exists k : (c[i],c[j],c[k] = \mathtt{B},\mathtt{B},\mathtt{P} \wedge i < j < k) \vee (c[i],c[k],c[j] = \mathtt{B},\mathtt{B},\mathtt{P} \wedge i < k < j) \vee (c[k],c[i],c[j] = \mathtt{B},\mathtt{B},\mathtt{P} \wedge k < i < j)$.

5 Verification Procedure

We present our verification method for the class of parameterized systems described in Section 2. We fix a parameterized system $\mathcal{P} = (Q, \Delta)$ for the rest of the section. We use the abstract domain from Section 4. For $k \in \mathbb{N}$, the abstract post-image of a set of views V is defined, as usual, as $\alpha_k \circ \mathit{post} \circ \gamma_k(V)$. The core of our verification procedure consists in checking whether there is a $k \in \mathbb{N}$ such that the least fixpoint of $\alpha_k \circ \mathit{post} \circ \gamma_k$ is a set of views with the following properties: its concretization (i) covers the set I of initial configurations and (ii) is disjoint from the set $\mathcal{B}$ of bad configurations. More precisely, the precision of the abstraction increases with k, so we iterate the fixpoint computation $\mu X.\, \alpha_k(I) \sqcup \alpha_k \circ \mathit{post} \circ \gamma_k(X)$ for increasing values of k starting from $k = 1$, until point (ii) holds.

We present our procedure in a stepwise manner. Since $\gamma_k(V)$ is in general infinite, we need to compute the abstract post-image symbolically. First, we introduce a symbolic abstract transformer and show that it precisely corresponds to the abstract post. Although we show that is possible to compute the abstract transformer precisely (and therefore the aforementioned fixpoint), we also introduce an over-approximation for efficiency reasons. Finally, we stitch the different components together and describe the sound and complete procedure. Since the symbolic transformer is exact, if there exists an almost downward-closed invariant (i.e., good invariant expressible by an $\forall_\exists$-formula, or equivalently by a set of views), then the iteration is guaranteed to discover it and terminate for some value of k [14].

Symbolic post operator. To define the symbolic post operator, we first define a transition relation on views. For a view $v = (\mathtt{base}, \mathtt{ctx})$, $i \leq |\mathtt{base}|$, and a transition $\delta \in \Delta$, we define the symbolic immediate successor of v under a δ-move of the i^{th} process from base, denoted $\delta^{\#}(v,i)$. Informally, the moving process checks the other processes from the base. In addition, if δ is a universal transition, the moving process checks as well the processes in the contexts. If the transition is enabled, the moving process from base changes its state according to the δ-transition, otherwise it is blocked. The contexts do not change. In fact, we can here observe the role played by a context: it retains enough information in a view to *disable* (or *block*) universal transitions, which would have been otherwise enabled without the presence of contexts. This reduces the risk of running a too coarse over-approximation.

Formally, for a view $v = R_0 b_1 R_1 \ldots b_n R_n$ and $i \leq n$, $\delta^{\#}(v,i) = R_0 b'_1 R_1 \ldots b'_n R_n$ iff $b_i = s$, $b'_i = s'$, $b_j = b'_j$ for all $j : j \neq i$ and either (i) δ is a local rule $s \to s'$, or (ii) δ is a global rule of the form $\mathbf{if}\, \mathbb{Q}\ j \circ i : S\ \mathbf{then}\ s \to s'$, and one of the following two conditions is satisfied: (a) $\mathbb{Q} = \forall$ and it holds both that $b_j \in S$ for all $j \in [\![1;n]\!]$ such that $j \circ i$ and that $R_j \subseteq S$ for all $j \in [\![0;n]\!]$ such that $j \circ i$, or (b) $\mathbb{Q} = \exists$ and there exists $j \in [\![1;n]\!]$ such that $j \circ i$ and $b_j \in S$. Note that we do not need to check the contexts in the latter case. Indeed, this is supported by the fact that the views work collectively. If there is a view where a process appears in a context, then there is always another view where it appears in the base, while the others are in a context. Finally, for a set of views V, we define $\mathit{spost}(V) = \{\delta^{\#}(v,i) \mid v \in V, i \leq |v|, \delta \in \Delta\}$.

We now explain how we define the *symbolic post* operating on views. It is based on the observation that a process needs *at most one* other process as a witness in order to perform its transition (cf. existential transitions). A moving process can appear either (i) in the base of a view, or (ii) in a context. *Extending adequately* the view with one extra process is enough to determine whether the moving process, in case (i), can perform its transition. However, in case (ii), since *spost* only updates processes of the base, a first extension with one process "materializes" the moving process into the base and a second extension by one process considers its witness. Therefore, it is sufficient to extend the views with two extra processes to determine if a transition is *enabled*, whether the moving process belongs to the base or a context of a view. Formally, for a set V of views of size k and for $\ell > k$, we define the *extensions* of V of size ℓ as the set of views $\oint_k^\ell(V) = \alpha_l(\gamma_k(V))$. Finally, we define the *symbolic post* as $\alpha_k \circ \mathit{spost} \circ \oint_k^{k+2}(V)$. Lemma 3 allows us to conclude that the symbolic post is the best abstract transformer.

Lemma 3. *For any k and set of views V of size up to k,*
$$V \sqcup \alpha_k \circ \mathit{post} \circ \gamma_k(V) = V \sqcup \alpha_k \circ \mathit{spost} \circ \oint_k^{k+2}(V)$$

The definition of $\oint_k^\ell(V)$ still involves the potentially infinite set $\gamma_k(V)$, so it cannot be computed in a straightforward manner. We show how $\oint_k^\ell(V)$ can be computed via a translation to finite automata, consisting of three steps, sketched here and described in details in the technical report [5]:

1. Translate V into an $\forall_\exists$-formula ϕ such that $[\![\phi]\!] = \gamma_k(V)$ (by Lemma 2)
2. Translate ϕ into a finite automaton A_ϕ such that $L(A_\phi) = [\![\phi]\!]$
3. Compute $\alpha_\ell(L(A_\phi))$

Approximation. The described automata-theoretic procedure to compute $\oint_k^\ell(V)$ comes at some cost. Step 2 involves internally the complementation of an intermediate automaton, which is at worst exponential, both in time and space. We therefore introduce an over-approximation and compute $\sqint_k^\ell(V) = \{v \in \mathcal{V} \mid \alpha_k(v) \succsim V, |v| \leq \ell\}$, i.e. the set of views of size ℓ that can be generated from V, without inspecting its concretization first. By lemma 4 (below), it follows that the views in $\sqint_k^\ell(V)$ over-approximate the views in $\oint_k^\ell(V)$ and may enable more universal transitions than they should. Indeed, views in $\sqint_k^\ell(V)$ have (at least) the same bases as the views in $\oint_k^\ell(V)$, but they might have smaller contexts (and are therefore weaker). Consider for example the case where $k = 2$, $\ell = 3$ and the set of views $V = \{ab, bc, ac[e], ce[f], ae, be, af, bf, cf, ef\}$. The set $\sqint_2^3(V)$ contains the view $abc[e]$ but $\oint_k^\ell(V)$ contains the view $abc[e,f]$ because the smallest configuration in $\gamma_2(V)$ that has abc as a subword is $abcef$ (this is due to the view $ce[f]$

which enforces the presence of f). Another example is $V = \{ab, bc, ac[e], a[c]e, [a]ce\}$. Here, $\oint_2^3(V)$ contains $abc[e]$, however, there is no view with the base abc in $\oint_2^3(V)$ since there is no configuration with the subword abc in $\gamma_2(V)$.

Lemma 4. *For any $\ell \geq k$ and $V \subseteq \mathcal{V}$, $\oint_k^\ell(V) \preccurlyeq \oint_k^\ell(V)$*

Sound and Complete algorithm. We combine the fixpoint computation of the symbolic post with a systematic state-space exploration in order to find a bad configuration. The algorithm (described succintly in Alg. 1) proceeds by iteration over configurations and views of size up to k, starting from $k = 1$ and increasing k after each iteration. Every iteration consists in two computations in parallel: (i) Using the exact post-image, we compute the set $\mathcal{R}_k$ of configurations reachable from the initial configurations, involving only configurations of size k (line 2). Note that there are only finitely many such configurations and that we consider, in this paper, length-preserving transitions,[3] so this step terminates and (ii) the fixpoint computation of the symbolic post over views of size up to k.

Alg. 1: Verification Procedure

```
1 for $k := 1$ to $\infty$ do
2     if $\mathtt{bad}(\mathcal{R}_k)$ then return Unsafe
3     $V := \mu X \,.\, \alpha_k(I) \sqcup \alpha_k \circ spost \circ \oint_k^{k+2}(X)$
4     if $\neg\mathtt{bad}(V)$ then return Safe
```

A reachable bad configuration of some size must be reachable through a sequence of transitions involving configurations of some maximal size, so it will be eventually discovered. By lemma 4, it is sound to replace the fixpoint computation of the symbolic post with the approximated set of views $\oint_k^{k+2}$ (line 3). Finally, the termination criteria on line 2 and 4 require the use of the function `bad` which returns either if a set of configurations contains a bad configuration or whether a set of views characterizes a bad configuration. The function `bad` is implemented by checking whether any configuration from $\mathcal{B}_{min}$ appears in its input set either (i) as a subword of a configuration or (ii) within the base of a view. We do not inspect any context, because the views work collectively and there is always another view in the set which contains this context in its base.

The resulting verification algorithm is sound and terminates for some k if and only if there is a reachable bad configuration or if there is a good almost downward-closed invariant. It uses the property of small models, that is, most behaviors are captured with small instances of the systems, either in the form of configurations and views.

Acceleration. The fixpoint computation on line 3 can be accelerated by leveraging the entailment relation. It is based on the observation that $\mathcal{R}_k$ contains configurations of size up to k, which can be used as initial input for the fixpoint computation (rather than I). All views of size k in $\alpha_k(\mathcal{R}_k)$ have empty contexts (i.e. they are weakest).

[3] Although, in this paper, there is no process deletion nor creation, our method works with non length-preserving transitions. The set $\mathcal{R}_k$ is not anymore computed by simply searching through the state-space, since a sequence of transitions from a configuration of size k might lead to arbitrarily many configurations of larger sizes. The alternative definition of $\mathcal{R}_k$ is configurations that may be reached via sequences of transitions involving configurations of the size up to k. This again defines a finite search space, and it holds that every reachable configuration is within $\mathcal{R}_j$ for some j.

They avoid the computations of the symbolic post on any stronger views. A similar argument can be used to see that it is not necessary to apply *spost* on the views in $\sqint_k^{k+2}(X)$ that are stronger than the views in $\alpha_{k+2}(\mathcal{R}_{k+2})$. We therefore seed the fixpoint computation with a larger set than $\alpha_k(I)$, namely $\alpha_k(\mathcal{R}_k \cup \mathcal{R}_{k+1} \cup \mathcal{R}_{k+2})$, and cache the set of views $\alpha_{k+2}(\mathcal{R}_{k+2})$.

6 Non-atomically checked global conditions

We extend our model and method to handle parameterized systems where global conditions are *not* checked atomically. We replace both existentially and universally guarded transition rules by the following variant of a for-loop rule:

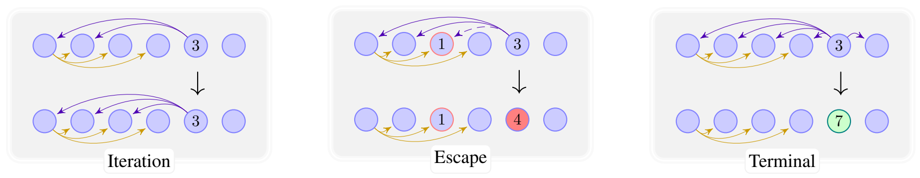
$$\textbf{if foreach } j \circ i : S \textbf{ then } s \to s' \textbf{ else } s \to e$$

where $e \in Q$ is an *escape* state and the other s, s', $\circ$ and S are named as in Section 2. For instance, line 3 of Szymanski's protocol is be replaced by $\textbf{if foreach } j \neq i : \neg\{1,2\} \textbf{ then } 3 \to 7 \textbf{ else } 3 \to 4$. Essentially, for a configuration with linear topology, a process at position i inspects the state of another process at position j, *in-order*. Without loss of generality, we will assume that the for-loops iterate through process indices in increasing order. If the state of the process at position j is not a reason for the process i to escape, process i moves on to inspect the process at position $j+1$, unless there is no more process to inspect in which case process i completes its transition.

We extend the semantics of a system with for-loop rules from transition systems of Section 2 in the following way: A configuration is now a pair $c = (q_1 \cdots q_n, \checkmark)$ where $q_1 \cdots q_n \in Q^+$ is as before and where $\checkmark : [\![1;n]\!] \to [\![0;n]\!]$ is a total map which assigns to every position i of c the last position which has been inspected by the process i. Initially, $\checkmark(i)$ is assigned 0.

We fix a rule $\delta = \textbf{if foreach } j \circ i : S \textbf{ then } s \to t \textbf{ else } s \to e$ from Δ, a configuration c with $|c| = n$, and $i \in [\![1;n]\!]$. We first define the position $next(i)$ which the process at position i is expected to inspect next. Formally, $next(i) = min\{j \in [\![1;n]\!] \mid j > \checkmark(i), j \circ i\}$ is the smallest position larger than than $\checkmark(i)$ which satisfies $next(i) \circ i$. Notice that if process i has already inspected the right-most position j which satisfies $j \circ i$, then (and only then) $next(i)$ is undefined.

We distinguish three types of δ-move on c by the process at position i: (i) $\delta_i(c,i)$ for a loop iteration, (ii) $\delta_e(c,i)$ for escaping and (iii) $\delta_t(c,i)$ for termination. Each type of move is defined only if $q_i = s$.
– $\delta_i(c,i)$ is defined if $next(i)$ is defined and $q_{next(i)} \in S$. It is obtained from c by only updating $\checkmark(i)$ to $next(i)$. Intuitively, process i is only *ticking* position $next(i)$.
– $\delta_e(c,i)$ is defined if $next(i)$ is defined and $q_{next(i)} \notin S$. It is obtained from c by changing the state of the process i to e and resetting $\checkmark(i)$ to 0. Intuitively, process i has found a reason to escape.
– $\delta_t(c,i)$ is defined if $next(i)$ is undefined, and it is obtained from c by changing the state of the process i to t and resetting $\checkmark(i)$ to 0. Intuitively, process i has reached the end of the iteration and terminates its transition (i.e. moves to its target state).



We now instantiate the abstract domain by adapting the notion of views from Section 4. A view is now of the form $(R_0 q_1 R_1 \ldots q_n R_n, \checkmark, \rho)$, where $(q_1 \cdots q_n, \checkmark)$ is a configuration called the *base*, and $(R_0, \cdots, R_n, \rho)$ is a *context*, such that $R_0, \ldots, R_n \subseteq Q$ and $\rho : [\![1;n]\!] \to 2^Q$ is a total map which assigns a subset of Q to every $i \in [\![1;n]\!]$. Intuitively, the role of $\rho(i)$ is to keep track of the processes that process i has not yet inspected in case they get mixed up in a context with other already inspected processes. This will be the case, as depicted in Fig.8, for one context only, say R_ℓ (in fact, R_ℓ is the context where $\checkmark(i)$ is projected to). It is trivial to see that contexts of higher (resp. lower) indices than ℓ contain processes that are not (resp. are) inspected by process i.

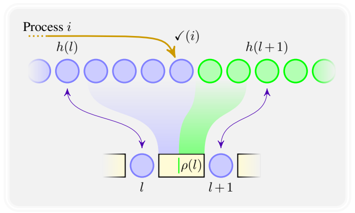
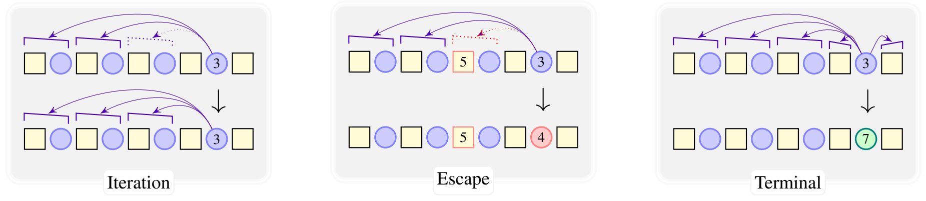


Fig. 8: Projection with non-atomicity. The blue states have been inspected by process i, the green states have not.

The projection of a configuration into a view is defined similarly as in Section 5. For $h \in H_n^k, k \leq n$, and a configuration $c = (q_1 \cdots q_n, \checkmark)$, $\Pi_h(c) = (\Pi_h(q_1 \cdots q_n), \checkmark', \rho')$ where $\checkmark'$ and ρ' are defined as follows. For all $i \in [\![1;k]\!]$, there exists ℓ such that $h(\ell) \leq \checkmark(i) < h(\ell+1)$. Then, $\checkmark'(i) = \ell$ and $\rho'(i) = \{q_j \mid \checkmark(i) < j < h(\ell+1)\}$. The projection of views is defined analogously. Note that this definition also implies that the concretization of a set of views is precise enough and reconstructs configurations with in-order ticks.

The entailment relation between the views $v = (R_0 q_1 R_1 \ldots q_n R_n, \checkmark, \rho)$ and $v' = (R'_0 q'_1 R'_1 \ldots q'_n R'_n, \checkmark', \rho')$ (of the same size) is defined such that $v \preccurlyeq v'$ iff (i) both have the same base, i.e. $(q_1 \cdots q_n, \checkmark) = (q'_1 \cdots q'_n, \checkmark')$, (ii) $R_i \subseteq R'_i$ for all $i \in [\![0;n]\!]$, and (iii) $\rho(i) \subseteq \rho'(i)$ for all $i \in [\![1;n]\!]$. This intuitively reflects that the more unticked states within a context the likelier it is for a transition to be blocked, and the larger contexts are the likelier they retain non-ticked states.

Finally, abstraction, concretization, and *spost* (and therefore symbolic post) are then adapted using the new definition of projection, entailment and *post*. This also implies that the contexts are inspected in-order and all processes in a context at once. Lemma 1, 3 and 4 hold in the same wording. The symbolic post with contexts and non-atomicity is precisely the abstract post and we use a similar over-approximation than in Section 5.



7 Experiments

We have implemented a prototype in OCaml based on our method to verify the safety property of numerous protocols.

We report the results running on a 3.1 GHz computer with 4GB memory. Table 1 displays, for various protocols with linear topology (over 2 lines), the running times (in seconds) and the final number of views generated ($|V|$). The first line is the result of the atomic version of the protocol, while the second line corresponds to the non-atomic version. The complete descriptions of the experiments can be found the technical report [5]. In most cases, the method terminates almost immediately illustrating the *small model property*: all patterns occur for small instances of the system.

Table 1: Linear topologies ± Atomicity

Protocol	Time	$\|V\|$	
$\forall_\exists$-example*	0.005	22	✓
	0.006	-	✗
Burns	0.004	34	✓
	0.011	64	✓
Dijkstra	0.027	93	✓
	0.097	222	✓
Szymanski*	0.307	168	✓
	1.982	311	✓
Szymanski (compact)*	0.006	48	✓
	0.557	194	✓
Szymanski (random)	1.156	-	✗
Bakery	0.001	7	✓
	0.006	30	✓
Gribomont-Zenner*	0.328	143	✓
	32.112	888	✓
Simple Barrier* (as array)	0.018	61	✓
	1.069	253	✓

* contexts needed ✓: Safe ✗: Unsafe

For the first example of Table 1 in the case of non-atomicity, our tool reports the protocol to be $Unsafe$ (✗). The method is sound. It is indeed a real error and not an artifact of the over-approximation. In fact, this is also the case when we intentionally tweak the implementation of Szymanski's protocol and force the for-loops to iterate randomly through the indices, in the non-atomic case. The tool reports a trace, that is, a sequence of configurations — here involving only 3 processes — as a witness of an (erroneous) scenario that leads to a violation of the mutual exclusion property.

Table 2: Petri Net with Inhibitor Arcs

Protocol	Time	$\|V\|$	
Critical Section with lock	0.001	42	✓
Priority Allocator	0.001	33	✓
Barrier with Counters	0.001	22	✓
Simple Barrier	0.001	8	✓
Light Control contexts needed	0.001	15	✓
List with Counter Automata	0.002	38	✓

The method is not limited to linear topologies. We also used the method to verify several examples with a multiset topology: Petri nets with inhibitor arcs. Inhibitor places should retain some content (therefore creating a context) in order to not fire the transition and potentially make the over-approximation too coarse. The bottom part of Table 2 lists examples where the contexts were necessary to verify the protocol, while the top part lists examples that did not require any.

Table 3: Leveraging the heuristics

Protocol		Time	$\|V\|$	it.
Agreement	insertion heuristic	8.247	199	28
	all contexts	3.950	216	1
	contexts discovery	166.893	121	4
Gribomont-Zenner	insertion heuristic	0.328	143	7
	all contexts	0.808	317	1
	contexts discovery	50.049	217	3
Szymanski, non-atomic	insertion heuristic	2.053	311	26
	all contexts	48.065	771	1
	contexts discovery	732.643	896	7

Heuristics. If $\alpha_2(\mathcal{R}_2 \cup \mathcal{R}_3) = \alpha_2(\mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4)$, it is likely the case that the computation in Alg. 1 (line 3) is already at fixpoint for $k = 2$ and discovered all the bases of the system in the sets from the above equation. It is therefore interesting to inspect whether a new base could be discovered by the symbolic post, while ignoring contexts (to consider the weakest views). If not, a fixpoint is indeed discovered and the invariant is strong enough to imply safety. If so, we can stop the computations, detect which views led to the new inserted base and remember their contexts for the next round of computations. This heuristic happen to be very successful in the case of Szymanski's protocol (in its non-atomic full version). On the other hand, this idea can be used in general: Do not remember any contexts, therefore considering the weakest views, and if the procedure discovers a counter-example, we trace the views that generated it and remember their contexts for the next round of computations, in a CEGAR-like fashion. It is however inefficient if all views most likely need a context (as shown with the ring agreement example). Table 3 presents the results of using the insertion and context discovery heuristics. The time is given in seconds and `it.` represents the number of iteration to terminate.

8 Related work

An extensive amount of work has been devoted to regular model checking, e.g. [29, 15]; and in particular augmenting regular model checking with techniques such as widening [11, 34], abstraction [12], and acceleration [8]. All these works rely on computing the transitive closure of transducers or on iterating them on regular languages. There are numerous techniques less general than regular model checking, but they are lighter and more dedicated to the problem of parameterized verification. The idea of *counter abstraction* is to keep track of the number of processes which satisfy a certain property [27, 22, 16, 17, 32]. In general, counter abstraction is designed for systems with unstructured or clique architectures, but may be used for systems with other topologies too.

Several works reduce parameterized verification to the verification of finite-state models. Among these, the *invisible invariants* method [9, 31] and the work of [30] exploit cut-off properties to check invariants for mutual exclusion protocols. In [10], 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 to compute the abstract model. *Environment abstraction* [13] combines predicate abstraction with the counter abstraction. The technique is applied to Szymanski's algorithm (with atomicity assumption).

The only work we are aware of that attempts to automatically verify systems with non-atomic global transitions is [7]. It applies the recently introduced method of *monotonic abstraction* [6], which combines regular model checking with abstraction in order to produce systems that have monotonic behaviors wrt. a well quasi-ordering on the state-space. The verification procedure in this case operates on unbounded abstract graphs, and thus is a non-trivial extension of the existing framework. The method of [26, 25] and its reformulated, generic version of [24] come with a complete method for well-quasi ordered systems which is an alternative to backward reachability analysis based on a forward exploration, similarly to our recent work [4].

Constant-size cut-offs have been defined for ring networks in [21] where communication is only allowed through token passing. More general communication mechanisms such as guards over local and shared variables are described in [20]. However, the cut-offs are linear in the number of states of the components, which makes the verification task intractable on most of our examples. The work in [28] also relies on dynamic detection of cut-off points. The class of systems considered in [28] corresponds essentially to Petri nets.

Most of the mentioned related works can verify only systems with good downward-closed invariants, up to several exceptions: Regular model checking can express even more complicated properties of states with the word topology. Our method is significantly simpler and more efficient. The data structure [23] extends the data structures discussed in [18, 19] so that they are able to express almost downward-closed sets of states with multiset topology. The work [3] allows to infer almost downward-closed invariants using an extension of backward reachability algorithm with CEGAR. Last, in [30], the need of inferring almost downward-closed invariants may be sometimes circumvent by manually introducing auxiliary variables.

The only two works we are aware of that support handling non-atomic global transitions are [4] and [3].

Our method is simpler and more efficient than most of the mentioned methods, but what distinguishes it most clearly is that it is the only one that combines handling non-atomic global transitions and automatic inference of almost downward-closed properties.

9 Conclusion and Future Work

We have presented a method for automatic verification of parameterized systems which alleviates the lack of precision from [4] that it exhibits on systems without fully downward-closed invariants. This is a unique method that combines the feature of discovering non downward-closed invariants while allowing to model systems with fine-grained transitions.

The method performs parameterized verification by only analyzing a small set of instances of the system (rather than the whole family) and captures the reachability of bad configurations to imply safety. Our algorithm relies on a very simple abstraction function, where a configuration of the system is approximated by breaking it down into smaller pieces. This gives rise to a finite representation of infinite sets of configurations while retaining enough precision. We showed that the presented algorithm is complete for systems with almost downward-closed invariants. Based on the method, we have implemented a prototype which performs efficiently on a wide range of benchmarks.

We are currently working on extending the method to the case of multi-threaded programs running on machines with different memory models. These systems have notoriously complicated behaviors. Showing that verification can be carried out through the analysis of only a small number of threads would allow more efficient algorithms for these systems.

References

1. Abdulla, P.A.: Well (and better) quasi-ordered transition systems. Bulletin of Symbolic Logic 16(4), 457–515 (2010)
2. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: LICS'96. pp. 313–321 (1996)

3. Abdulla, P.A., Delzanno, G., Rezine, A.: Approximated context-sensitive analysis for parameterized verification. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FORTE'09. LNCS, vol. 5522, pp. 41–56. Springer (2009), http://dx.doi.org/10.1007/978-3-642-02138-1_3
4. Abdulla, P.A., Haziza, F., Holík, L.: All for the price of few (parameterized verification through view abstraction). In: 14^{th} International Conference on Verification, Model Checking, and Abstract Interpretation. LNCS, vol. 7737, pp. 476–495 (2013)
5. Abdulla, P.A., Haziza, F., Holík, L.: Block me if you can! (context-sensitive parameterized verification). Technical Report FIT-TR-2014-03, Brno University of Technology (2014)
6. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: TACAS'07. LNCS, vol. 4424, pp. 721–736. Springer (2007)
7. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Handling parameterized systems with non-atomic global conditions. In: VMCAI'08. LNCS, vol. 4905, pp. 22–36. Springer (2008)
8. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Proc. CONCUR 2002, 13^{th} Int. Conf. on Concurrency Theory. LNCS, vol. 2421, pp. 116–130 (2002)
9. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: Berry, Comon, Finkel (eds.) Proc. 13^{th} Int. Conf. on Computer Aided Verification. LNCS, vol. 2102, pp. 221–234 (2001)
10. Baukus, K., Lakhnech, Y., Stahl, K.: Parameterized verification of a cache coherence protocol: Safety and liveness. In: VMCAI'02. LNCS, vol. 2294, pp. 317–330. Springer (2002)
11. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Proc. 15^{th} Int. Conf. on Computer Aided Verification. LNCS, vol. 2725, pp. 223–235 (2003)
12. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: CAV'04. LNCS, vol. 3114, pp. 372–386. Springer (2004)
13. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: VMCAI'06. LNCS, vol. 3855, pp. 126–141. Springer (2006)
14. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL. pp. 269–282. POPL '79, ACM, New York, NY, USA (1979), http://doi.acm.org/10.1145/567752.567778
15. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: CAV'01. LNCS, vol. 2102. Springer (2001)
16. Delzanno, G.: Automatic verification of cache coherence protocols. In: Emerson, Sistla (eds.) CAV'00. LNCS, vol. 1855, pp. 53–68. Springer (2000)
17. Delzanno, G.: Verification of consistency protocols via infinite-state symbolic model checking. In: FORTE'00. IFIP Conference Proceedings, vol. 183, pp. 171–186. Kluwer (2000)
18. Delzanno, G., Raskin, J.F., Begin, L.V.: Csts (covering sharing trees): Compact data structures for parameterized verification. In: Software Tools for Technology Transfer (2001)
19. Delzanno, G., Raskin, J.F.: Symbolic representation of upward-closed sets. In: TACAS'00. LNCS, vol. 1785, pp. 426–441. Springer (2000), http://dx.doi.org/10.1007/3-540-46419-0_29
20. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE'00. LNCS, vol. 1831, pp. 236–254. Springer (2000)
21. Emerson, E., Namjoshi, K.: Reasoning about rings. In: POPL'95. pp. 85–94 (1995)
22. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS'99. IEEE Computer Society (1999)
23. Ganty, P.: The Interval Sharing Tree Data Structure (1999), https://github.com/pierreganty/mist/wiki/The-Interval-Sharing-Tree-Data-Structure
24. Ganty, P., Raskin, J.F., Begin, L.V.: A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In: VMCAI'06. LNCS, vol. 3855, pp. 49–64. Springer (2006)

25. Geeraerts, G., Raskin, J.F., Begin, L.V.: Expand, enlarge and check... made efficient. In: CAV'05. LNCS, vol. 3576, pp. 394–407. Springer (2005)
26. Geeraerts, G., Raskin, J.F., Begin, L.V.: Expand, enlarge and check: New algorithms for the coverability problem of wsts. J. Comput. Syst. Sci. 72(1), 180–203 (2006)
27. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM 39(3), 675–735 (1992)
28. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV'10. LNCS, vol. 6174, pp. 645–659. Springer (2010)
29. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theoretical Computer Science 256, 93–112 (2001)
30. Namjoshi, K.S.: Symmetry and completeness in the analysis of parameterized systems. In: VMCAI'07. LNCS, vol. 4349, pp. 299–313. Springer (2007)
31. Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants. In: Proc. TACAS '01, 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. vol. 2031, pp. 82–97 (2001)
32. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infinity)-counter abstraction. In: Proc. 14^{th} Int. Conf. on Computer Aided Verification. LNCS, vol. 2404 (2002)
33. Szymanski, B.K.: A simple solution to lamport's concurrent programming problem with linear wait. In: Proceedings of the 2nd international conference on Supercomputing. pp. 621–626. ICS '88, ACM, New York, NY, USA (1988), `http://doi.acm.org/10.1145/55364.55425`
34. Touili, T.: Regular Model Checking using Widening Techniques. Electronic Notes in Theoretical Computer Science 50(4) (2001), proc. of VEPAS'01