# All for the Price of Few

**(Parameterized Verification through View Abstraction)**

Parosh Aziz Abdulla[1], Frédéric Haziza[1], and Lukáš Holík[12]

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

**Abstract.** We present a simple and efficient framework for automatic verification of systems with a parameteric number of communicating processes. The processes may be organized in various topologies such as words, multisets, rings, or trees. Our method needs to inspect only a small number of processes in order to show correctness of the whole system. It relies on an abstraction function that views the system from the perspective of a fixed number of processes. The abstraction is used during the verification procedure in order to dynamically detect cut-off points beyond which the search of the state space need not continue. We show that the method is complete for a large class of well quasi-ordered systems including Petri nets. Our experimentation on a variety of benchmarks demonstrate that the method is highly efficient and that it works well even for classes of systems with undecidable verification problems.

## 1 Introduction

We address verification of safety properties for *parameterized systems* that consist of arbitrary numbers of components (processes) organized according to a regular pattern. 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. The term *parameterized* refers to the fact that the size of the system is (implicitly) a parameter of the verification problem. Parameterized systems arise naturally in the modeling of mutual exclusion algorithms, bus protocols, distributed algorithms, telecommunication protocols, and cache coherence protocols. For instance, the specification of a mutual exclusion protocol may be parameterized by the number of processes that participate in a given session of the protocol. In such a case, it is interesting to verify correctness regardless of the number of participants in a particular session. As usual, the verification of safety properties can be reduced to the problem of checking the reachability of a set of *bad configurations* (the set of configurations that violate the safety property).

*Existing approaches.* An important approach to parameterized verification has been *regular model checking* [25, 5, 9] in which regular languages are used as symbolic representations of infinite sets of system configurations, and automata-based techniques are employed to implement the verification procedure. The main problem with such techniques is that they are heavy since they usually rely on several layers of computationally expensive automata-theoretic constructions, in many cases leading to a state space explosion that severely limits their applicability. Another class of methods analyze *approximated* system behavior through the use of abstraction techniques. Such methods include *counter abstraction* [22, 30], *invisible invariant* generation [6, 31], *environment abstraction* [11], and *monotonic abstraction* [3] (see Section 7).

In a similar manner to [24], this work is inspired by a strong empirical evidence that parameterized systems often enjoy a *small model property*. More precisely, analyzing only a small number of processes (rather than the whole family) is sufficient to capture the reachability of bad configurations. On the one hand, bad configurations can often be characterized by minimal conditions that are possible to specify through a fixed number of *witness* processes. For instance, in a mutual exclusion protocol, a bad configuration contains *two* processes in their critical sections; and in a cache coherence protocol, a bad configuration contains *two* cache lines in their *exclusive* states. In both cases, having the two witnesses is sufficient to make the configuration bad (regardless of the actual size of the configuration). On the other hand, it is usually the case that such bad patterns (if existing) appear already in small instances of the system, as observed in our experimental section.

*Our approach.* We introduce a method that exploits the small model property, and performs parameterized verification by only inspecting a small set of *fixed* instances of the system. Furthermore, the instances that need to be considered are often small in size (typically three or four processes) which allows for a very efficient verification procedure. The framework can be applied uniformly to generate *fully automatic* verification algorithms for wide classes of parameterized systems including ones that operate on linear, ring, or tree-like topologies, or systems that contain unbounded collections of anonymous processes (the latter class is henceforth referred to as having a *multiset* topology).

At the heart of the method is an operation that allows to detect *cut-off* points beyond which the verification procedure need not continue. Intuitively, reaching a cut-off point means that we need not inspect larger instances of the system: the information collected so far during the exploration of the state space allows us to conclude safely that no bad configurations will occur in the larger instances. The cut-off analysis is executed *dynamically* in the sense that it is performed on-the-fly during the verification procedure itself. It is based on an abstraction function, called *view abstraction*, parameterized by a constant $k$, and it approximates a configuration by the set of all its projections containing at most $k$ processes. We call the sub-configurations *views*. For instance, when a configuration is a word of process states (represented as an array of processes), its abstraction is the set of all its subwords of length at most $k$. Furthermore, for a given set of views $X$, its concretization, denoted as $\gamma_k(X)$, is the set of configurations (of any size) for which *all* their views belong to $X$.

The verification method performs two search procedures in parallel. The first performs a standard (explicit-state) forward reachability analysis trying to find a bad configuration among system configurations of size $k$ (for some natural number $k$). If a bad configuration is encountered then the system is not safe. The second procedure performs a *symbolic* forward reachability analysis in the abstract domain of sets of views of size at most $k$. When the computation terminates, it will have collected an over-approximation of all views of size up to $k$ of all reachable configurations (of all sizes).

If there is no bad configuration in the concretization of this set, then a cut-off point has been found and the system can be claimed safe. If neither of the parallel procedures reaches a conclusion during iteration $k$, the value of $k$ is increased by one (thus increasing the precision of the abstraction). Notice that the abstract search requires computing the *abstract post-image* of a set $X$ of views of size at most $k$, which is the set $X'$ of views (of size at most $k$) of successors of $\gamma_k(X)$. Obviously, this cannot be performed straightforwardly since the set of configurations $\gamma_k(X)$ is infinite. A crucial contribution of the paper is to show that, for all the classes of parameterized systems that we consider, it is sufficient to only compute successors of configurations from $\gamma_k(X)$ that are of the size at most $k+\ell$, where $\ell$ is a small constant, typically 1. Intuitively, the reason is that the precondition for firing a transition is the presence of a *bounded* number of processes in certain states. The views need only to encompass these processes in order to determine the successor view. This property is satisfied by a wide class of concurrent systems including the ones we consider in this paper. For instance, in rendez-vous communication between a pair of processes, the transition is conditioned by the states of *two* processes; in broadcast communication, *one* process initiates the transition (while the other processes may be in any state); in existential global transitions (see below), we need *two* processes, namely the witness and the process performing the transition; in Petri nets, the number of required processes is bounded by the in-degree of the transitions (which is fixed for a given Petri net), etc. We will show formally that this property is satisfied by all the types of transitions we consider.

*Applications.* We have instantiated the method to obtain automatic verification procedures for four classes of parameterized systems, namely systems where the processes are organized as arrays, rings, trees, or multisets. Each instantiation is straightforward and is achieved by defining the manner in which we define the views of a configuration. More precisely, these views are (naturally) defined as subwords, cyclic subwords, subtrees, resp. subsets for the above four classes. Once the views are fixed we obtain a fully automatic procedure for all parameterized systems in the class. In the systems we consider, we allow a rich set of features, in which processes may perform local transitions, rendez-vous, broadcasts, and universally or existentially guarded transitions. In a universally guarded transition, the process checks whether the states of *all* other processes inside the system satisfy a given constraint before it performs the transition. In an existentially quantified transition, the processes checks that there is *at least one* other process satisfying the condition. Furthermore, we allow dynamic behaviors such as the creation and deletion of processes during the execution of the system.

In the basic variant of our method, we assume that existential and universal global conditions of transitions are checked atomically. The same assumption is made in many landmark works on parameterized systems (e.g. [11, 31, 10, 5, 6, 29, 3]). However, actual implementations of global checks are usually not atomic. They are typically implemented as for-loops ranging over indices of processes. Iterations of such a loop may be interleaved with transitions of other processes, therefore modeling the loop as an atomic transition means under-approximating the behavior of the system. Verification of systems with non-atomic global checks is significantly harder. It requires to distinguish intermediate states of a for-loop performed by a process. Their number is proportional to the number of processes in the system. Moreover, any number of processes may be performing a for-loop at the same time. As we will show, our method can be easily adapted to this setting, while retaining its simplicity and efficiency.

*Implementation.* We have implemented a prototype based on the method and run it on a wide class of benchmarks, including mutual exclusion protocols on arrays (e.g., Burns', Szymanski's, and Dijkstra's protocols), cache coherent protocols (e.g., MOSI and German's protocol), different protocols on tree-like architectures (e.g. percolate, arbiter, and leader election), ring protocols (token passing), and different Petri nets.

The class of systems we consider have undecidable reachability properties, and hence our method is necessarily incomplete (the verification procedure is not guaranteed to terminate in case the safety property is satisfied). However, as shown by our experimentation, the tool terminates efficiently on all the tested benchmarks.

*Completeness.* Although the method is not complete in general, we show that is complete for a large class of systems, namely those that induce *well quasi-ordered* transition systems [2, 1] and satisfy certain additional technical requirements. This implies that our method is complete for e.g., Petri nets. Notice that, as evident from our experiments, the method can in practice handle even systems that are outside the class.

*Outline.* To simplify the presentation, we instantiate our framework in a step-wise manner. In Section 2, we introduce our model for parameterized systems operating on linear topologies and describe our verification method in Section 3. In Section 4, we describe how the framework can be extended to incorporate other kinds of transitions such as broadcast, rendez-vous, dynamic process deletion/creation, and non-atomic checks of global conditions; and to cover other classes of topologies such as ring, multiset, and tree-like structures. The completeness of our method for well quasi-ordered systems is shown in Section 5. We report on our experimental results in Section 6, and describe related work in Section 7. Finally, we give some conclusions and directions for future research in Section 8.

## 2 Parameterized Systems

We introduce a standard notion of a parameterized system operating on a linear topology, where processes may perform local transitions or universally/existentially guarded transitions (this is the standard model used e.g. in [31, 11, 3, 29]).

A parameterized system is a pair $\mathcal{P} = (Q, \Delta)$ where $Q$ is a finite set of *local states* of a process and $\Delta$ 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 of the local states of the other processes. A global rule 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 $S \subseteq Q$. Here, the $i$th process checks also the local states of the other processes when it makes the move. For instance, the condition $\forall j < i : S$ means that "for every $j$ such that $j < i$, the $j$th process should be in a local state that belongs to the set $S$"; the condition $\forall j \neq i : S$ means that "all processes except the $i$th one should be in local states that belong to the set $S$"; etc.

A parameterized system $\mathcal{P} = (Q, \Delta)$ induces a *transition system* (TS) $\mathcal{T} = (C, \to)$ where $C = Q^*$ is the set of its *configurations* and $\to \subseteq C \times C$ is the *transition relation*.

We use $c[i]$ to denote the state of the $i$th process within the configuration $c$. The transition relation $\rightarrow$ contains a transition $c \rightarrow c'$ with $c[i] = s$, $c'[i] = s'$, $c[j] = c'[j]$ for all $j : j \neq i$ iff either (i) $\Delta$ contains a local rule $s \rightarrow s'$, or (ii) $\Delta$ contains a global rule $\textbf{if}\, \mathbb{Q}\, j \circ i : S$ **then** $s \rightarrow s'$, and one of the following conditions is satisfied:

- $\mathbb{Q} = \forall$ and for all $j : 1 \leq j \leq |c|$ such that $j \circ i$, it holds that $c[j] \in S$.
- $\mathbb{Q} = \exists$ and there exists $j : 1 \leq j \leq |c|$ such that $j \circ i$ and $c[j] \in S$.

An instance of the *reachability problem* is defined by a parameterized system $\mathcal{P} = (Q, \Delta)$, a regular set $I \subseteq Q^+$ of *initial configurations*, and a set $Bad \subseteq Q^+$ of *bad configurations*. Let $\sqsubseteq$ be the usual *subword relation*, i.e., $u \sqsubseteq s_1 \dots s_n$ iff $u = s_{i_1} \dots s_{i_k}, 1 \leq i_1 \dots i_k \leq n$ and $i_j < i_{j+1}$ for all $j : 1 \leq j < k$. We assume that $Bad$ is the upward closure $\{c \mid \exists b \in B : b \sqsubseteq c\}$ of a given *finite* set $B \subseteq Q^+$ of *minimal bad configurations*. This is a common way of specifying bad configurations which often appears in practice, see e.g. the running example of Burn's mutual exclusion protocol below. We say that $c \in C$ is *reachable* iff there are $c_0, \dots, c_l \in C$ such that $c_0 \in I$, $c_l = c$, and $c_i \rightarrow c_{i+1}$ for all $0 \leq i < l$. We use $\mathcal{R}$ to denote the set of all reachable configurations. We say that the system $\mathcal{P}$ is *safe* w.r.t. $I$ and $Bad$ if no bad configuration is reachable, i.e. $\mathcal{R} \cap Bad = \emptyset$.

We define the *post-image* of a set $X \subseteq C$ to be the set $post(X) := \{c' \mid c \rightarrow c' \wedge c \in X\}$. For $n \in \mathbb{N}$ and a set of configurations $S \subseteq C$, we use $S_n$ to denote its subset $\{c \in S \mid |c| \leq n\}$ of configurations of size up to $n$.

*Running example.* We illustrate the notion of a parameterized systems with the example of Burns' mutual exclusion protocol [26]. The protocol ensures exclusive access to a shared resource in a system consisting of an unbounded number of processes organized in an array. The pseudocode of the process at the $i$th position of the array and the transition rules of the parameterized system are given in Figure 1. A state of the $i$th process consists of a program location and a value of the local variable $flag[i]$. Since the value of $flag[i]$ is invariant at each location, states correspond to locations.

A configuration of the induced transition system is a word over the alphabet $\{1, \dots, 6\}$ of local process states. The task is to check that the protocol guarantees exclusive access to the shared resource (line 6) regardless of the number of processes. A configuration is considered to be bad if it contains two occurrences of state 6, i.e., the set of minimal bad configurations $B$ is $\{$66$\}$. Initially, all processes are in state 1, i.e. $I = 1^+$.

```
Burns($i$)
1 $flag[i] := 0$;
2 if $\exists j < i :\ flag[i] = 1$ then goto 1;
3 $flag[i] := 1$;
4 if $\exists j < i :\ flag[i] = 1$ then goto 1;
5 await $\forall j > i : flag[j] \neq 1$;
6 $flag[i] := 0$; goto 1                      CS
```

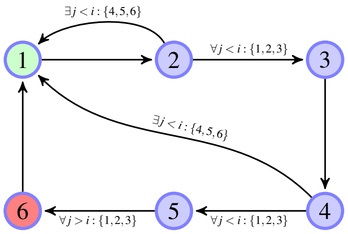

$\exists j < i : \{4,5,6\}$
1
2
$\forall j < i : \{1,2,3\}$
3
$\exists j < i : \{4,5,6\}$
6
$\forall j > i : \{1,2,3\}$
5
$\forall j < i : \{1,2,3\}$
4

**Fig. 1.** Pseudocode and transition rules of Burns' protocol.

## 3 Verification Method

In this section, we describe our verification method instantiated to the case of parameterized systems described in Section 2. First, we describe the abstraction we use, then we present the procedure.

### 3.1 View Abstraction

We abstract a configuration $c$ by a set of *views* each of which is a subword of $c$. The *abstraction function* $\alpha_k : C \to 2^{C_k}$ maps a configuration $c$ into the set $\alpha_k(c) = \{v \in C_k \mid v \sqsubseteq c\}$ of all its views (subwords) of size up to $k$. We lift $\alpha_k$ to sets of configurations as usual. For every $k \in \mathbb{N}$, the *concretization* function $\gamma_k : 2^{C_k} \to 2^C$ inputs a set of views $V \subseteq C_k$, and returns the set of configurations that can be reconstructed from the views in $V$. In other words, $\gamma_k(V) = \{c \in C \mid \alpha_k(c) \subseteq V\}$.

*Abstract post-image.* As usual, the *abstract post-image* of a set of views $V \subseteq C_k$ is defined as $Apost_k(V) = \alpha_k(post(\gamma_k(V)))$. Computing $Apost_k(V)$ is a central component of our verification procedure. It cannot be computed straightforwardly since the set $\gamma_k(V)$ is typically infinite. As a main contribution of the paper, we show that it is sufficient to consider only those configurations in $\gamma_k(V)$ whose sizes are up to $k+1$. There are finitely many such configurations, and hence their post-image can be computed. Formally, for $\ell \geq 0$, we define $\gamma_k^\ell(V) := \gamma_k(V) \cap C_\ell$ and show the following *small model* lemma for the class of systems of Section 2. We will show similar lemmas for the other classes of systems that we present in the later sections.

**Lemma 1.** *For any $k \in \mathbb{N}$ and $X \subseteq C_k$, $\alpha_k(post(\gamma_k(X))) \cup X \;=\; \alpha_k(post(\gamma_k^{k+1}(X))) \cup X$.*

The property of the transition relation which allows us to prove the lemma is that, loosely speaking, the transitions have *small preconditions*. That is, there is a transition that can be fired from a configuration $c$ and generate a view $v \in C_k$ only if $c$ contains a certain view $v'$ of some limited size, here up to $k+1$.

*Running Example.* Consider for instance the set $V = \{1,2,3,4,6,12,16,32,34,42\} \subseteq C_2$ of views of Burns' protocol. We will illustrate that we need to reason only about configurations of $\gamma_2(V)$, which are of size at most 3, to decide which views belong to $Apost_2(V)$.

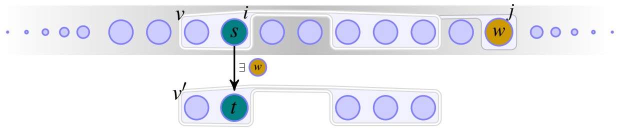
Take the existentially guarded transition $2 \to 1$. It can be fired only from configurations that contain 2 together with a witness from $\{4,5,6\}$ on the left. $Apost_2(V)$ contains the view 31 since $\gamma_2^{2+1}(V)$ contains 342 from where the existential transition $2 \to 1$ can be fired. (342 belongs to $\gamma_2(V)$ because all its views 2, 3, 4, 32, 34, and 42 are in $V$). It does not contain the view 22 since 12 cannot be completed by the needed witness (12 cannot be extended by, e.g., 6 since $V$ does not contain 26 and 62).

Consider now the universally guarded transition $2 \to 3$. The transition can be fired only from configurations that contain 2. Since $2 \to 3$ can be fired on $32 \in \gamma_2(V)$, $Apost_2(V)$ contains 33. But it does not contain the view 43 since the universal guard prevents firing $2 \to 3$ on configurations containing 42.

*Proof.* We present the part of the proof of Lemma 1 which deals with existentially guarded transitions. The parts dealing with local and universally guarded transitions are simpler and are moved to the appendix. We will show that for any configuration $c \in \gamma_k(V)$ of size $m > k+1$ such that there is a transition $c \rightarrow c'$ induced by an existentially guarded rule $r \in \Delta$ with $v' \in \alpha_k(c')$, the following holds: Either $v' \in V$ or there is a configuration $d \in \gamma_k(V)$ of size at most $k+1$ with a transition $d \rightarrow d'$ induced by $r$ with $v' \in \alpha_k(d')$.

A subset of positions $p = \{i_1, \ldots, i_l\} \subseteq \{1, \ldots, n\}, l \leq k$, with $i_1 < \ldots < i_l$ of a configuration $c = s_1 \ldots s_n$ defines the view $view(c,p) = s_{i_1} \ldots s_{i_l}$ of $c$. By definition, $v'$ equals $view(c',p)$ for some $p \subseteq \{1, \ldots, m\}$. Let $v$ be $view(c,p)$. Notice that since $c \in \gamma_k(V)$, any view of $c$ of size at least $k$ belongs to $\gamma_k(V)$. Therefore also $v \in \gamma_k(V)$. Let $1 \leq i \leq m$ be the index of the position in which $c'$ differs from $c$. If $i \notin p$, then $v = view(c,p) = view(c',p) = v'$. In this case, we trivially have $v' \in V$. We can take $d = v$ and $d' = v'$.

Assume now that $i \in p$. Let $r$ be the rule **if** $\exists j \circ i : S$ **then** $s \rightarrow t$ where $\circ \in \{<, >, \neq\}$. There are two cases: 1) there is a witness $w$ from $S$ at some position $j \in p$ enabling the transition $c \rightarrow c'$. Then $v$ still contains the witness on an appropriate position needed to fire $r$. Therefore $v \rightarrow v'$ is a transition of the system induced by $r$, and we can take $d = v$ and $d' = v'$. 2) no witness enabling the transition $c \rightarrow c'$ is at a position $j \in p$. Then there is no guarantee that $v \rightarrow v'$ is a transition of the system. However, the witness enabling the transition $c \rightarrow c'$ is at some position $j \in \{1, \ldots, m\}$. We will create a configuration of size at most $k+1$ by including this position $j$ to $v$, as illustrated in the figure. Let $p' = p \cup \{j\}$. Then $view(c,p') \rightarrow view(c',p')$ is a transition of the system induced by $r$ since $view(c,p')$ contains both $s$ and a witness from $S$ at an appropriate position. We clearly have that $v' \in \alpha_k(view(c',p'))$. We also have that $view(c,p') \in \gamma_k(V)$ since $view(c,p') \sqsubseteq c$ and $c \in \gamma_k(V)$. We may therefore take $d = view(c,p')$ and $d' = view(c',p')$. □



### 3.2 Procedure

Our verification procedure for solving an instance of the verification problem defined in Section 2 is described in Algorithm 1. It performs two search procedures in parallel. Specifically, it searches for a bad configuration reachable from initial configurations of size $k$; and it searches for a cut-off point $k$ where it derives a set of views $V \subseteq C_k$ such that

(i) $V$ is an invariant for the instances of the system (that is, $\mathcal{R} \subseteq \gamma_k(V)$ and $Apost_k(V) \subseteq V$), and

(ii) which is sufficient to prove $\mathcal{R}$ safe (that is, $\gamma_k(V) \cap Bad = \emptyset$).

**Algorithm 1:** Verification Procedure

```
1 for $k := 1$ to $\infty$ do
2     if $\mathcal{R}_k \cap Bad \neq \emptyset$ then return Unsafe
3     $V := \mu X.\alpha_k(I) \cup Apost_k(X)$
4     if $\gamma_k(V) \cap Bad = \emptyset$ then return Safe
```

For a given $k$, an invariant $V$ is computed on line 3. Notice that $V$ is well-defined since $\gamma_k, post, \alpha_k$ and hence also $Apost_k$ are monotonic functions for all $k \in \mathbb{N}$ (w.r.t. $\subseteq$). Lemma 2 guarantees that $V$ is indeed an invariant:

**Lemma 2.** *For any $i \in \mathbb{N}$ and $X \subseteq C_i$, $\alpha_i(I) \subseteq X \wedge Apost_i(X) \subseteq X \implies \alpha_i(\mathcal{R}) \subseteq X$.*

If the system is unsafe, the search on line 2 will eventually discovers a bad configuration. The cut-off condition is tested on line 4. If the test does not pass, then we do not know whether the system is indeed unsafe or whether the analysis has hit a spurious counterexample (due to a too liberal abstraction). Therefore, the algorithm increases precision of the abstraction by increasing $k$ and reiterating the loop. An effective implementation of the procedure requires carrying out the following steps:

1. *Computing the abstraction $\alpha_k(I)$ of initial configurations.* This step is usually easy. For instance, in the case of Burns' protocol, all processes are initially in state 1, hence $\alpha_k(I)$ contains only the words $1^l, l \leq k$. Generally, $I$ is a (very simple) regular set, and $\alpha_k(I)$ is computed using a straightforward automata construction.
2. *Computing the abstract post-image.* Thanks to Lemma 1, the abstract post-image can be computed by applying $\gamma_k^{k+1}$ (which yields a finite set), *post*, and $\alpha_k$ (in that order).
3. *Evaluating the test $\gamma_k(V) \cap Bad = \emptyset$.* Since *Bad* is the upward closure of a finite set $B$, the test can be carried out by testing whether there is $b \in B$ such that $\alpha_k(b) \subseteq V$.
4. *Exact reachability analysis.* Line 2 requires the computation of $\mathcal{R}_k$. Since $\mathcal{R}_k$ is finite, this can be done using any procedure for exact state space exploration.

Since the problem is generally undecidable, existence of $k$ for which the test on line 4 succeeds for a safe system cannot be guaranteed and the algorithm may not terminate. However, as discussed in Section 5, such a guarantee can be given under the additional requirement of monotonicity of transition relation w.r.t. a well-quasi ordering. The method terminates otherwise for all our examples discussed in Section 6, many of which are *not* well quasi-ordered.

*Running example.* When run on Burns' protocol, Algorithm 1 starts by computing $\mathcal{R}_1 = \{1, \ldots, 6\}$. Because $\mathcal{R}_1$ does not contain any bad configurations, the algorithm moves onto computing the fixpoint $V_1$ of line 3. The iteration starts with $X = \alpha_1(I) = \{1\}$ and continues until $X = V_1 = \{1, \ldots, 6\}$. The test on line 4 subsequently fails since $\gamma_1(V_1)$ contains 66. Since both tests fail, the first iteration does not allow us to conclude whether the protocol is safe or not, so the algorithm increases the precision of the abstraction by increasing $k$.

In the second iteration with $k = 2$, $\mathcal{R}_2$ is still safe. The fixpoint computation starts with $X = \alpha_2(I) = \{1, 11\}$. When $Apost_2$ is applied on $\{1, 11\}$, we first construct the set $\gamma_2^{2+1}(\{1, 11\})$ which contains the extension 111 of 11, 11 and 1. Their successors are $2, 12, 21$, and $112, 121, 211$, which are abstracted into $\{1, 2, 11, 12, 21\}$. The fixpoint computation continues with $X = \{1, 2, 11, 12, 21\}$ and constructs the concretization $\gamma_2^3(X) = X \cup \{112, 121, 211\}$. Their successors are $2, 3, 12, 21, 22, 31, 13$, and 122, 212, 221, 113, 131, 311 which are abstracted into the views $1, 2, 3, 11, 12, 21, 22, 31, 13$. The next iteration will start with $X = \{1, 2, 3, 11, 12, 21, 22, 13, 31\}$. The computation reaches, after 8 further iterations, the fixpoint $X = V_2$ which contains all words from $\{1, \ldots, 6\} \cup \{1, \ldots, 6\}^2$ except 65 and 66. This set satisfies the assumptions of Lemma 2, and hence it is guaranteed to contain all views (of size at most 2) of all reachable configurations of the system. Since the view 66 is not present (recall $\alpha_2(Bad) = \{6, 66\}$), no reachable configuration of the system is bad. The algorithm reached the cut-off point $k = 2$ of Burns' protocol, and the system is proved safe.

# 4 Extensions

In this section, we describe how to extend the class of parameterized systems that we presented in Section 2. The extensions are obtained 1) by extending the types of transition rules that we allow, 2) by replacing transitions with atomically checked global conditions by more realistic for-loops, and 3) by considering topologies other than the linear ones. As we shall see below, the extensions can be handled by our method with straightforward extensions of the method of Section 3.

## 4.1 More Communication Mechanisms

*Broadcast.* In a broadcast transition, an arbitrary number of processes change states simultaneously. A broadcast rule is a pair $(s \rightarrow s', \{r_1 \rightarrow r'_1, \ldots, r_m \rightarrow r'_m\})$. It is deterministic in the sense that $r_i \neq r_j$ for $i \neq j$. The broadcast is initiated by a process, called the *initiator*, which triggers the transition rule $s \rightarrow s'$. Together with the initiator, an arbitrary number of processes, called the *receptors*, change state simultaneously. More precisely, if the local state of a process is $r_i$, then the process changes its local state to $r'_i$. Processes whose local states are different from $s, r_1, \ldots r_m$ remain passive during the broadcast. Formally, the broadcast rule induces transitions $c \longrightarrow c'$ of $\mathcal{T}$ where for some $i : 1 \leq i \leq |c|$, $c[i] = s$, $c'[i] = s'$, and for each $j : 1 \leq j \neq i \leq |c|$, if $c[j] = r_k$ (for some $k$) then $c'[j] = r'_k$, otherwise $c[j] = c'[j]$.

In a similar manner to globally guarded transitions, broadcast transitions have small preconditions. Namely, to fire a transition, it is enough that an initiator is present in the transition. More precisely, for parameterized systems with local, global, and broadcast transitions, Lemma 1 still holds (in the proof of Lemma 1, the initiator is treated analogously to a witness of an existential transition). Therefore, the verification method from Section 3 can be used without any change.

*Rendez-vous.* In rendez-vous, multiple processes change their states simultaneously. A *simple rendez-vous* transition rule is a tuple of local rules $\delta = (r_1 \rightarrow r'_1, \ldots, r_m \rightarrow$

$r'_m), m > 1$. Multiple occurrences of local rules with the same source state $r$ in the tuple does not mean non-determinism, but that the rendez-vous requires multiple occurrences of $r$ in the configuration to be triggered. Formally, the rule induces transitions $c \to c'$ of $\mathcal{T}$ such that there are $i_1, \dots, i_m$ with $i_j \neq i_k$ for all $j \neq k$, such that $c[i_1]\cdots c[i_m] = r_1 \cdots r_m$, $c'[i_1]\cdots c'[i_m] = r'_1 \cdots r'_m$, and $c'[\ell] = c[\ell]$ if $\ell \notin \{i_1, \dots, i_m\}$.

Additionally, we define a *generalized rendez-vous (or just rendez-vous) transition rules* in order to model *creation and deletion* of processes and also Petri net transitions that change the number of tokens in the net. A generalized rendez-vous rule $\delta$ is as a simple rendez-vous rule, but it can in addition to the local rules contain two types of special rules: of the form $\bullet \to r, \bullet \notin Q$ (acting as a placeholder), which are used to model creation of processes, and of the form $r \to \bullet$, which are used to model deletion of processes. When a generalized rendez-vous rule is fired, the starting configuration is first enriched with $\bullet$ symbols in order to facilitate creation of processes by the rule $\bullet \to r$, then the rule is applied as if it was a simple rendez-vous rule, treating $\bullet$ as a normal state (states of the processes that are to be deleted are rewritten to $\bullet$ by the rules $r \to \bullet$). Finally, all occurrences of $\bullet$ are removed. Formally, a generalized rendez-vous rule induces a transition $c \to c'$ if there is $c_\bullet \in \{\bullet\}^* c[1] \{\bullet\}^* \cdots \{\bullet\}^* c[|c|] \{\bullet\}^*$ such that $c_\bullet \to c'_\bullet$ is a transition of the system with states $Q \cup \{\bullet\}$ induced by $\delta$ (treated as a simple rendez-vous rule), and $c'$ arises from $c'_\bullet$ by erasing all occurrences of $\bullet$.

Rendez-vous transitions have small preconditions, but unlike existentially quantified transitions, firing a transition may require presence of more than two (but still a fixed number) processes in certain states (the number is the arity of the transition). It essentially corresponds to requiring the presence of more than one witness. This is why Lemma 1 holds here only in the weaker variant:

**Lemma 3.** *Let $\Delta$ contain rules of any previously described type (i.e., local, global, broadcast, rendez-vous), and let $m+1$ is the largest arity of a rendez-vous rule in $\Delta$. Then, for any $k$ and $V \subseteq C_k$, $\alpha_k(post(\gamma_k(V))) \cup V \;=\; \alpha_k(post(\gamma_k^{k+m}(V))) \cup V$.*

*Global variables.* Communication via shared variables is modeled using a special process, called *controller*. Its local state records the state of all shared variables in the system. A configuration of a system with global variables is then a word $s_1 \dots s_n c$ where $s_1, \dots, s_n$ are the states of individual processes and $c$ is the state of the controller. An individual process can read and update a shared variable. A read is modeled by a rendez-vous rule of the form $(s \to s', c \to c)$ where $c$ is a state of the controller and $s, s'$ are states of the process. An update is modeled using a rendez-vous rule $(s \to s', c \to c')$.

To verify systems with shared variables of finite domains, we use a variant of the abstraction function which always keeps the state of the controller in the view. Formally, for a configuration $wc$ where $w \subseteq Q^+$ and $c$ is the state of the controller, $\alpha_k$ returns the set of words $vc$ where $v$ is a subword of $w$ of length at most $k$. The concretization and abstract-post image are then defined analogously as before, based on $\alpha_k$, Lemma 1 and Lemma 2 still hold. The method of Section 3 can be thus used in the same way as before.

Another type of global variable is a *process pointer*, i.e., a variable ranging over process indices. This is used, e.g., in Dijkstra's mutual exclusion protocol. A process pointer is modeled by a local Boolean flag $p$ for each process state. The value of $p$ is *true* iff the pointer points to the process (it is true for precisely one process in every configuration). An update of the pointer is modeled by a rendez-vous transition rule which sets to *false* the flag of the process currently pointed to by the pointer and sets to *true* the flag of the process which is to become the target of the pointer.

### 4.2 Transitions that do not Preserve Size

We now discuss the case when the transition relation does not preserve size of configurations, which happens in the case of generalised rendez-vous. $\mathcal{R}_k$ then cannot be computed straightforwardly since computations reaching configurations of the size up to $k$ may traverse configurations of larger sizes. Therefore, similarly as in [21], we only consider runs of the system visiting configurations of the size up to $k$. That is, on line 2 of Algorithm 1, instead of computing $\mathcal{R}_k = \mu X . I_k \cup post(X)$, we compute its under-approximation $\mu X . (I \cup post(X)) \cap C_k$. The computation terminates provided that $C_k$ is finite. The algorithm is still guaranteed to return Unsafe if a configuration in *Bad* is reachable, since then there is $k \in \mathbb{N}$ such that the bad configuration is reachable by a finite path traversing configurations of the size at most $k$.

### 4.3 Non-atomic Global Conditions

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

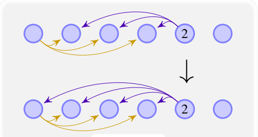
$$\textbf{if foreach } j \circ i : S \textbf{ then } q \to r \textbf{ else } q \to s$$

where $q, r, s \in Q$ is resp. a source state, a target state, and an escape state, $\circ \in \{<, >, \neq\}$, and $S \subseteq Q$ is a *condition*. For instance, line 2 of Burns' protocol would be replaced by **if foreach** $j < i : \{1,2,3\}$ **then** $2 \to 3$ **else** $2 \to 1$.

The semantics of a system with for-loop rules is defined as an extension of the transition system from Section 2. Configurations are extended with a binary relation over their positions, that is, a configuration is now a pair $(c, \checkmark)$ where $c$ is a word over $Q$ and $\checkmark$ is a binary relation over its positions $\{1, \ldots, |c|\}$. The relation $\checkmark$ is used to encode intermediate states of for-loops. Intuitively, a process at position $i$ performing a for-loop puts $(i, j)$ into $\checkmark$ to mark that it has processed the position $j$.

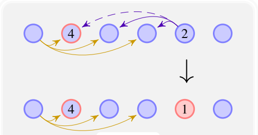
Formally, a parameterized system $\mathcal{P} = (Q, \Delta)$ which includes for-loop rules induces a transition system $\mathcal{T} = (C, \to)$ where $C \subseteq Q^+ \times (\mathbb{N} \times \mathbb{N})$. For technical convenience, we assume that a source of a for-loop rule in $\Delta$ is not a source of any other rule in $\Delta$.[3]Then every for-loop rule **if foreach** $j \circ i : S$ **then** $q \to r$ **else** $q \to s$ induces transitions $t = (w, \checkmark) \to (w', \checkmark')$ with $w[i] = q$ for some $i : 1 \leq i \leq |w|$ which may be of the following three forms: (illustrated using the aforementioned example rule from Burn's protocol).

[3] Without this restriction, the state of a process would have to contain additional information recording which for-loop is the process currently performing. Note that the restriction does not limit the modeling power of the formalism. Any potential branching may be moved to predecessors of the sources of the for-loop.



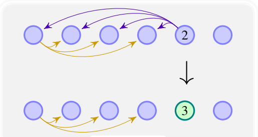
Iteration

**Iteration:** The $i$th process checks that the state of a next unchecked process in the range is in $S$ and marks it. That is, there is $j : 1 \leq j \leq |w|$ with $j \circ i$, $(i, j) \notin \checkmark$, $w[j] \in S$, and the resulting configuration has $w' = w$ and $\checkmark' = \checkmark \cup \{(i, j)\}$.



Escape

**Escape:** If the state of some process in the range which is still to be checked violates the loop condition, then the $i$th process may escape to the state $s$. That is, there is $j : 1 \leq j \leq |w|$ with $j \circ i$, $(i, j) \notin \checkmark$, and $w[j] \notin S$. The resulting configuration has $w'[k] = w[k]$ for all $k \neq i$ and $w[i] = s$. The execution of the for-loop ends and the marks of process $i$ are reset, i.e., $\checkmark' = \checkmark \setminus \{(i,k) \mid k \in \mathbb{N}\}$.



Terminal

**Terminal:** When the states of all processes from the range have been successfully checked, the for-loop ends and the $i$th process moves to the terminal state $r$. That is, if there is no $j : 1 \leq j \leq |w|$ with $j \circ i$ and $(i, j) \notin \checkmark$, then $w'[k] = w[k]$ for all $k \neq i$, $w'[i] = r$, and $\checkmark' = \checkmark \setminus \{(i,k) \mid k \in \mathbb{N}\}$.

Other rules behave as before on the $w$ part of configurations and they do not influence the $\checkmark$ part. That is, a local, broadcast, or rendez-vous rule induces transitions $(w, \checkmark) \rightarrow (w', \checkmark)$ where $w \rightarrow w'$ is a transition induced by the rule as described in Section 2.

*Verification.* To verify systems with for-loop rules using our method, we define an abstraction $\alpha_k$. Intuitively, we view a configuration $c = (w, \checkmark)$ as a graph with vertices being the positions of $w$ and edges being defined by (i) the ordering of the positions and (ii) the relation $\checkmark$. The vertices are labeled by the states of processes at the positions. $\alpha_k(c)$ then returns the set of subgraphs of $c$ where every subgraph contains a subset of at most $k$ vertices of $c$ (positions of $w$) and the maximal subset of edges of $c$ adjacent with the chosen vertices.

Formally, given a configuration $c = (w, \checkmark)$, $\alpha_k(c)$ is the set of views $v = (w', \checkmark') \in C$ of size at most $k$ (i.e., $|w'| = l \leq k$) such that there exists an injection $\rho : \{1, \ldots l\} \rightarrow \{1, \ldots, |w|\}, l \leq k$ where for all $i, j : 1 \leq i, j \leq l$:

- $i < j$ iff $\rho(i) < \rho(j)$,
- $w'[i] = w[\rho(i)]$ (i.e., $w' \sqsubseteq w$), and
- $(i, j) \in \checkmark'$ iff $(\rho(i), \rho(j)) \in \checkmark$.

The notions of concretization and abstract post-image are defined in the same manner as in Section 3 based on based on $\alpha$. Lemma 1 holds here in the same wording (as shown in the appendix). Thus the verification method for systems with for-loops is analogous to the method of Section 3.

### 4.4 Tree Topology

We extend our method to systems where configurations are trees. For simplicity, we restrict ourselves to complete binary trees.

*Trees.* Let $N$ be a prefix closed set of words over the alphabet $\{0,1\}$ called *nodes* and let $Q$ be a finite set. A (binary) *tree* over $Q$ is a mapping $t : N \to Q$. The node $\varepsilon$ is called the *root*, nodes that are not prefixes of other nodes are called *leaves*. For a node $v = v'i, i \in \{0,1\}$, $v'$ is the *parent* of $v$, the node $v0$ is the *left child* of $v$ and $v1$ is its *right child*. Every node $v' = vw, w \in \{0,1\}^+$ is a *descendant* of $v$. The *depth* of the tree is the length of the longest leaf. A tree is *complete* if all its leaves have the same length and every non-leaf node has both children. A tree $t' : N' \to Q$ is a *subtree* of $t$, denoted $t' \preceq t$, iff there exists a injective map $e : N' \to N$ which respects the descendant relation and labeling. That is, $t'(v) = t(e(v))$ and $v$ is a descendant of $v'$ iff $e(v)$ is a descendant of $e(v')$.

*Parameterized systems with tree topology.* The definitions for parameterized systems with a tree topology are analogous to the definitions for systems with a linear topology (Section 2). A parameterized system $\mathcal{P} = (Q, \Delta)$ induces a transition system $\mathcal{T} = (C, \to)$ where $C$ is the set of complete trees over $Q$. The set $\Delta$ of transition rules is a set of *local* and *tree* transition rules. The transitions of $\to$ are obtained from rules of $\Delta$ as follows. A *local* rule is of the form $s \to s'$ and it locally changes the label of a node from $s$ to $s'$. A *tree* rule is a triple $s(s_0, s_1) \to s'(s'_0, s'_1)$. The rule can be applied to a node $v$ and its left and right children $v_0$, $v_1$ with labels $s$, $s_0$, and $s_1$, respectively, and it changes their labels to $s'$, $s'_0$, and $s'_1$, respectively.

The reachability problem is defined in a similar manner to the case of linear systems. The set $B$ of minimal bad configurations is a finite set of trees over $Q$, $I$ is a regular tree-language, and *Bad* is the upward closure of $B$ w.r.t. the subtree relation $\preceq$. In the notation $C_n$ and $\mathcal{R}_n$, $n$ refers to the depth of trees rather than to the length of words.

*Verification.* The verification method of Section 3 is easily extended to the tree topology. The text of Section 3 can be taken almost verbatim with the difference that instead of words, we manipulate complete trees, subword relation is replaced by subtree relation, and $k$ now refers to the depth of trees rather than the length of words. That is, a view of size $k$ is a tree of depth $k$ and the abstraction $\alpha_k(t)$ returns all complete subtrees of depth at most $k$ of the tree $t$. Concretization and abstract post-image are defined analogously as in Section 3, based on $\alpha_k$. The set $I$ may be given in the form of a tree automaton. The computation of $\alpha_k(I)$ may be then done over the structure of the tree automaton. We can compute the abstract post-image since Lemma 1 holds here in the same wording as in Section 3. The test $\gamma_k(V) \cap Bad = \emptyset$ is carried out in the same way as in Section 3 since *Bad* is an upward closure of a set $B$ w.r.t. $\preceq$. The points 1-4 of Section 3 are thus satisfied and Algorithm 1 can be used as a verification procedure for systems with tree topology.

### 4.5 Ring Topology

The method can be extended also to systems with a ring topology. In a parameterized system with ring topology, processes are organized in a circular array and they synchronize by near-neighbor communication. We model system with a ring topology as systems with linear topology of Section 2, where a configuration $c \in Q^+$ is interpreted as a circular word. The set $\Delta$ may contain local and *near-neighbor* transition rules. A near-neighbor rule is a pair $(s_1 \to s'_1, s_2 \to s'_2)$. It induces the transition $c \to c'$ of $\to$ if either $c = c_L s_1 s_2 c_R$ and $c' = c_L s'_1 s'_2 c_R$ (i.e. the 2 processes are adjacent in the configuration $c$) or $c = s_2 \bar{c} s_1$ and $c' = s'_2 \bar{c} s'_1$ (i.e. the 2 processes are positioned at the end of the configuration $c$). The latter case covers the communication between the extremities since configurations encode circular words.

*Verification.* A word $u$ is a *circular subword* of a word $v$, denoted $u \trianglelefteq v$, iff there are $v_1, v_2$ such that $v = v_1 v_2$ and $u \sqsubseteq v_2 v_1$. The only difference compared to the method for the systems with a linear topology is that the standard subword relation is in all definitions replaced by the circular subword relation $\trianglelefteq$. An equivalent of Lemma 1 holds here in unchanged wording, points 1-4 are satisfied, and Algorithm 1 is thus a verification procedure for systems with ring topology.

### 4.6 Multiset Topology

Systems which we refer to as systems with multiset topology are a special case of the systems with a linear topology of Section 2. Typical representatives of these systems are Petri nets, which correspond precisely to systems of Section 4 with only (generalized) rendez-vous transitions. Systems with multiset topology may contain all types of transitions including local, global, broadcast, and rendez-vous, with the exception of global transitions with the scope of indices $j > i$ and $j < i$ (i.e., only $j \neq i$ is permitted). Since the processes have no way of distinguishing their respective positions within a configuration, the notion of ordering of positions within a configuration is not meaningful and configurations can be represented as multisets.

## 5 Completeness for Well Quasi-Ordered Systems

In this section, will show that the scheme desribed by Algorithm 1 is complete for a wide class of well-quasi ordered systems. To state the result in general terms, we will first give some definitions from the theory of well quasi-ordered systems (c.f. [1]).

A *well quasi-ordering* (WQO) is a preorder $\preceq$ over a set $S$ such that for every infinite sequence $s_1, s_2, \ldots$ of elements of $S$, there exists $i$ and $j$ such that $i < j$ and $s_i \preceq s_j$. The *upward-closure* $\uparrow T$ of a set $T \subseteq S$ w.r.t. $\preceq$ is the set $\{s \in S \mid \exists t \in T : t \preceq s\}$ and its *downward-closure* is the set $\downarrow T = \{s \in S \mid \exists t \in T : s \preceq t\}$. A set is *upward-closed* if it equals its upward-closure and it is *downward-closed* if it equals its downward-closure. If $T$ is upward closed, its complement $S \setminus T$ is downward closed and, conversely, if $T$ is downward closed, its complement is upward closed. For every upward closed set $T$, there exists a minimal (w.r.t $\subseteq$) set $Gen$ such that $\uparrow Gen = T$, called *generator* of $T$, which is finite. If moreover $\preceq$ is a partial order, then $Gen$ is unique.

A relation $R \subseteq S \times S$ is *monotonic* w.r.t. $\preceq$ if whenever $(s_1, s_2) \in R$ and $s_1 \preceq s'_1$, then there is $s'_2$ with $(s'_1, s'_2) \in R$ and $s_2 \preceq s'_2$. Given a relation $f \subseteq S \times S$ monotonic w.r.t. $\preceq$ and a set $T \subseteq S$, it holds that if $f(T) \subseteq T$, then $f(\downarrow T) \subseteq \downarrow T$, where $f(T)$ is the image of $T$ defined as $\{t' \mid \exists t \in T : (t, t') \in f\}$.

The reasoning in Section 3 is based on the natural notion of a size of a configuration. Its generalization is the notion of a *discrete measure* over a set $S$, a function $|.| : S \to \mathbb{N}$

which fulfills the property that for every $k \in \mathbb{N}$, $\{s \in S \mid |s| = k\}$ is finite. A discrete measure is necessary to obtain the completeness result as it allows enumerating elements of $S$ of the same size. In particular, this property guarantees termination of the fixpoint computation on Line 3 of Algorithm 1. We note that the existence of a discrete measure is implied by a stronger restriction of [8] to the so called discrete transition systems.

We say that a transition system $\mathcal{T} = (C, \rightarrow)$ is *well-quasi ordered* by a WQO $\preceq \subseteq C \times C$ if $\rightarrow$ is monotonic w.r.t. $\preceq$. Given a well-quasi ordered transition system and a measure $|.| : C \rightarrow \mathbb{N}$, we define an abstraction function $\alpha_k, k \in \mathbb{N}$ such that $\alpha_k(c) = \{c' \in C \mid c' \preceq c\}$. The corresponding concretization $\gamma_k$ and abstract post-image $Apost_k$ are then defined based on $\alpha_k$ and $|.|$ as in Section 3.1.

Lemma 2 holds here in the same wording as in Section 3. The main component of the completeness result is the following theorem.

**Theorem 1.** *Let $\mathcal{T} = (C, \rightarrow)$ be a well-quasi ordered transition system with a measure $|.|$. Let $I$ be any subset of $C$ and let $Bad$ be upward-closed w.r.t. $\preceq$. Then, if $\mathcal{T}$ is safe w.r.t. $I$ and $Bad$, then there is $k \in \mathbb{N}$ such that for $V = \mu X.\alpha_k(I) \cup Apost_k(X)$, $Bad \cap \gamma_k(V) = \emptyset$.*

*Proof.* Recall first that $\gamma_k, post, Apost_k, \alpha_k$ are monotonic functions w.r.t. $\subseteq$ for all $k \in \mathbb{N}$. Let $Gen$ be the minimal generator of the upward closed set $C \setminus \downarrow\mathcal{R}$. We will prove that $k$ can be chosen as $k = \max\{|c| \mid c \in Gen\}$. Such $k$ exists because $Gen$ is finite.

We first show an auxiliary claim that $\gamma_k(\alpha_k(\downarrow\mathcal{R})) \subseteq \downarrow\mathcal{R}$. Let $s \in \gamma_k(\alpha_k(\downarrow\mathcal{R}))$. For the sake of contradiction, suppose that $s \notin \downarrow\mathcal{R}$. We have that $s \in C \setminus \downarrow\mathcal{R} = \uparrow Gen$ and there is a generator $t \in Gen$ with $t \preceq s$. By the definition of $k$, $|t| \leq k$. Since $t \in Gen$, $t \notin \downarrow\mathcal{R}$ and hence $t \notin \alpha_k(\downarrow\mathcal{R})$. But due to this and since $t \preceq s$, we have that $s \notin \gamma_k(\alpha_k(\downarrow\mathcal{R}))$ (by the definition of $\gamma_k$) which contradicts the initial assumption and the claim is proven.

Next, we argue that $\alpha_k(\downarrow\mathcal{R})$ is stable under abstract post, that is, $Apost_k(\alpha_k(\downarrow\mathcal{R})) \subseteq \alpha_k(\downarrow\mathcal{R})$. Since $\mathcal{R}$ is stable under $post$ and $post$ is monotonic w.r.t. $\preceq$, we know that $\downarrow\mathcal{R}$ is stable under $post$ (that is, $post(\downarrow\mathcal{R}) \subseteq \downarrow\mathcal{R}$). Then, by the definition of $Apost_k$, and by monotonicity of $\alpha_k$ w.r.t. $\subseteq$, we have $Apost_k(\alpha_k(\downarrow\mathcal{R})) = \alpha_k(post(\gamma_k(\alpha_k(\downarrow\mathcal{R})))) \subseteq \alpha_k(post(\downarrow\mathcal{R})) \subseteq \alpha_k(\downarrow\mathcal{R})$.

Since $\downarrow\mathcal{R}$ contains $I$, $\alpha_k(I) \subseteq \alpha_k(\downarrow\mathcal{R})$. $\alpha_k(\downarrow\mathcal{R})$ is thus a fixpoint of $\lambda X.\alpha_k(I) \cup Apost_k(X)$. Because $V$ is the least fixpoint of $\lambda X.\alpha_k(I) \cup Apost_k(X)$, $V \subseteq \alpha_k(\downarrow\mathcal{R})$. From, $\mathcal{R} \cap Bad = \emptyset$ and since $Bad$ is upward closed, we know that $\downarrow\mathcal{R} \cap Bad = \emptyset$. Because $\gamma_k(V) \subseteq \gamma_k(\alpha_k(\downarrow\mathcal{R})) \subseteq \downarrow\mathcal{R}$ and $\downarrow\mathcal{R} \cap Bad = \emptyset$, $\gamma_k(V) \cap Bad = \emptyset$. □

Theorem 1 guarantees that for a safe well quasi-ordered system, there exists $k$ for which the test on line 4 of Algorithm 1 succeeds. Conversely, Lemma 2, which, as mentioned above, still holds for the general class of well-quasi ordered systems, then assures than if the test on line 2 succeeds, the system is indeed safe.

*Complete algorithm.* The schema described by Algorithm 1 (or its variant from Section 4.2 if the transition relation is not size-preserving) gives a complete verification procedure for a well quasi-ordered system provided that all the four steps of its for-loop can be effectively evaluated. This is guaranteed by the following requirements:

i. $\alpha_k(I)$ can be computed,
ii. the measure $|.|$ is discrete,

iii. for a configuration $c$, $post(c)$ and $\alpha_k(c)$ can be computed,
iv. for a finite set of views $V$, $\gamma_k^{k+1}(V)$ can be computed, and
v. a variant of Lemma 1 holds.

Point (i) is point 1 of Section 3. Points (ii)-(v) guarantee that we can compute abstract post-image (point 2 of Section 3). We can test $\gamma_k(V) \cap Bad = \emptyset$ (point 3 of Section 3) since due to (ii), $V$ is always finite. Exact reachability analysis of configurations of a bounded size (point 4 of Section 3) can be carried out since we can iterate *post* due to (iii) and the iteration terminates after a finite number of steps due to (ii). Point (ii) also assures termination of the computation of the fixpoint on line 3 ($V$ is always finite).

Overall, Algorithm 1 is a complete verification procedure for parameterized systems of Section 2 with local and existential transitions rules, broadcast and rendez-vous. The induced transition relation is indeed monotonic w.r.t. the preorder $\sqsubseteq$ which is a WQO and the length of a configuration is a discrete measure. An important subclass of such systems are Petri nets, which, as mentioned in Section 4, correspond to systems with multiset topology and generalized rendez-vous transition rules. Systems of Section 2 with universally guarded transition rules do not satisfy the assumptions: the induced transition relation is not monotonic.

## 6 Experimental results

Based on our method, we have implemented a prototype in OCaml to check safety properties for a number of parameterized systems with different topologies. The examples cover cache coherence protocols, communication protocols through trees and rings and mutual exclusion protocols.

**Table 1.** Experimental Results

| | **Protocol** | **Time** | $k$ | $\lvert V\rvert$ | $\gamma_k^{k+\ell}(V)$ |
|---|---|---|---|---|---|
| Array | Demo (toy example) | 0.01s | 2 | 17 | 53 |
| | Burns | 0.01s | 2 | 34 | 186 |
| | Dijkstra | 0.07s | 2 | 93 | 695 |
| | Szymanski | 0.02s | 2 | 48 | 264 |
| Multiset | MOSI Coherency | 0.01s | 1 | 10 | 23 |
| | German's Coherency | 15.3s | 6 | 1890 | 15567 |
| Petri Net | German (simplified) | 0.03s | 2 | 43 | 96 |
| | BH250 | 2.85s | 2 | 503 | 503 |
| | MOESI Coherency | 0.01s | 1 | 13 | 20 |
| | Critical Section | 0.01s | 5 | 27 | 46 |
| | Kanban | ? | $\geq 20$ | ? | ? |
| Tree | Percolate | 0.05s | 2 | 34 | 933 |
| | Tree Arbiter | 0.7s | 2 | 88 | 7680 |
| | Leader Election | 0.1s | 2 | 74 | 362 |
| Ring | Token Passing | 0.01s | 2 | 2 | 2 |

We report the results in Table 1, running on a 2.4 GHz laptop with 4GB memory. We have categorized the experiments per topology. We display the running times (in seconds), the value of $k$ and the final number of views generated ($|V|$). In most cases, the method terminates almost immediately illustrating the *small model property*: all patterns occur for small instances of the system. Observe that the sizes of the views are small as well, confirming the intuition that interactions between processes are of limited scope.

The bulk of the algorithm lies in the computation of the set $\gamma_k^{k+\ell}(V)$ and also the set $\mathcal{R}_k$. An example on which the algorithm fails is the *Kanban* system from [24]. This is a typical case where the cut-off condition is satisfied at high values of $k$. [24] refers to the computation of, at least, the set $\mathcal{R}_{20}$. $\mathcal{R}_{20}$ is large and so is the concretization of its views.

## 7 Related Work

An extensive amount of work has been devoted to regular model checking, e.g. [25, 12]; and in particular augmenting regular model checking with techniques such as widening [9, 32], abstraction [10], and acceleration [5]. All these works rely on computing the transitive closure of transducers or on iterating them on regular languages. Our method is significantly simpler and more efficient.

A technique of particular interest for parameterized systems is that of *counter abstraction*. The idea is to keep track of the number of processes which satisfy a certain property [22, 17, 13, 14, 30]. In general, counter abstraction is designed for systems with unstructured or clique architectures. As mentioned, our method can cope with these kinds of systems but also with more general classes of topologies. Several works reduce parameterized verification to the verification of finite-state models. Among these, the *invisible invariants* method [6, 31] and the work of [29] exploit cut-off properties to check invariants for mutual exclusion protocols. The success of the method depends on the heuristic used in the generation of the candidate invariant. This method sometimes (e.g. for German's protocol) requires insertion of auxiliary program variables for completing the proof. The nature of invariants generated by our method is similar to that of the aforementioned works, since our invariant sets of views of size at most $k$ can be seen as universally quantified assertions over reachable $k$-tuples of processes.

In [7], finite-state abstractions for verification of systems specified in WS1S are computed on-the-fly by using the weakest precondition operator. The method requires the user to provide a set of predicates on which to compute the abstract model.

The idea of refining the view abstraction by increasing $k$ is similar in spirit to the work of [28] which discusses increasing precision of thread modular verification (Cartesian abstraction) by remembering some relationships between states of processes. Their refinement mechanism is more local, targeting the source of undesirable imprecision; however, it is not directly applicable to parameterized verification.

*Environment abstraction* [11] combines predicate abstraction with the counter abstraction. The technique is applied to Szymanski's algorithm. The model of [11] contains a more restricted form of global conditions than ours, and also does not include features such as broadcast communication, rendez-vous communication, and dynamic creation and deletion of processes.

Recently, we have introduced the method of *monotonic abstraction* [3] that combines regular model checking with abstraction in order to produce systems that have monotonic behaviors w.r.t. a well quasi-ordering on the state space. In contrast to the method of this paper, the abstract system still needs to be analyzed using full symbolic reachability analysis on an infinite-state system. The only work we are aware of which attempts to automatically verify systems with non-atomic global transitions is [4] which applies monotonic abstraction. The abstraction in this case amounts to a verification procedure that operates on unbounded graphs, and thus is a non-trivial extension of the existing framework. As we saw, our method is easily extended to the case of non-atomic transitions.

The method of [21, 20] and its reformulated, generic version of [19] are in principle similar to ours. They come with a complete method for well-quasi ordered systems which is an alternative to backward reachability analysis based on a forward exploration. Unlike our method, they target well-quasi ordered systems only and have not been instantiated for topologies other than multisets and lossy channel systems.

Constant-size cut-offs have been defined for ring networks in [16] where communication is only allowed through token passing. More general communication mechanisms such as guards over local and shared variables are described in [15]. 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 closest work to ours is the one in [24] that also relies on dynamic detection of cut-off points. The class of systems considered in [24] corresponds essentially to Petri nets. In particular, it cannot deal with systems with linear or tree-like topologies. The method relies on the ability to perform backward reachability analysis on the underlying transition system. This means that the algorithm of [24] cannot be applied on systems with undecidable reachability problems (such as the ones we consider in this paper). The method of [24] is yet complete.

## 8 Conclusion and Future Work

We have presented a uniform framework for automatic verification of different classes of parameterized systems with topologies such as words, trees, rings, or multisets, with an extension to handle non-atomic global conditions. The framework allows to perform parameterized verification by only considering a small set of instances of the system. We have proved that the presented algorithm is complete for a wide class of well quasi-ordered systems. 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 framework to the case of multi-threaded programs operating on dynamic heap structures. 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 for more efficient algorithms for these systems. Furthermore, our algorithm relies on a very simple abstraction function, where a configuration of the system is approximated by its sub-structures (subwords, subtrees, etc.). We believe that our approach can be lifted to more general classes of abstractions. This would allow for abstraction schemes that are more precise than existing ones, e.g., thread-modular abstraction [18] and Cartesian abstraction [27].

Obviously, the bottleneck in the application of the method is when the cut-off condition is only satisfied at high values of $k$ (see e.g., the *Kanban* example in Section 6). We plan therefore to integrate the method with advanced tools that can perform efficient forward reachability analysis, like SPIN [23], and to use efficient symbolic encodings for compact representations for the set of views.

## 9 Acknowledgements

This work was supported by the Uppsala Programming for Multicore Architectures Research Center (UpMarc) and the Czech Science Foundation (project P103/10/0306).

## 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., 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)
4. 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)
5. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: CONCUR'02. LNCS, vol. 2421, pp. 116–130. Springer (2002)
6. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: CAV'01. LNCS, vol. 2102, pp. 221–234. Springer (2001)
7. 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)
8. Bingham, J.D., Hu, A.J.: Empirically efficient verification for a class of infinite-state systems. In: TACAS'05. LNCS, vol. 3440, pp. 77–92. Springer (2005)
9. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: CAV'03. LNCS, vol. 2725, pp. 223–235. Springer (2003)
10. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: CAV'04. LNCS, vol. 3114, pp. 372–386. Springer (2004)
11. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: VMCAI'06. LNCS, vol. 3855, pp. 126–141. Springer (2006)
12. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: CAV'01. LNCS, vol. 2102. Springer (2001)

13. Delzanno, G.: Automatic verification of cache coherence protocols. In: Emerson, Sistla (eds.) CAV'00. LNCS, vol. 1855, pp. 53–68. Springer (2000)
14. 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)
15. 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)
16. Emerson, E., Namjoshi, K.: Reasoning about rings. In: POPL'95. pp. 85–94 (1995)
17. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS'99. IEEE Computer Society (1999)
18. Flanagan, C., Qadeer, S.: Thread-modular model checking. In: SPIN'03. LNCS, vol. 2648, pp. 213–224. Springer (2003)
19. 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)
20. 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)
21. 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)
22. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM 39(3), 675–735 (1992)
23. Holzmann, G.J.: The model checker spin. IEEE Trans. Software Eng. 23(5), 279–295 (1997)
24. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV'10. LNCS, vol. 6174, pp. 645–659. Springer (2010)
25. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theor. Comput. Sci. 256, 93–112 (2001)
26. Lynch, N.A., Shamir, B.P.: Distributed algorithms, lecture notes for 6.852, fall 1992. Tech. Rep. MIT/LCS/RSS-20, MIT (1993)
27. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-modular verification is cartesian abstract interpretation. In: ICTAC'06. LNCS, vol. 4281, pp. 183–197. Springer (2006)
28. Malkis, A., Podelski, A., Rybalchenko, A.: Precise thread-modular verification. In: SAS'07. LNCS, vol. 4634, pp. 218–232. Springer (2007)
29. Namjoshi, K.S.: Symmetry and completeness in the analysis of parameterized systems. In: VMCAI'07. LNCS, vol. 4349, pp. 299–313. Springer (2007)
30. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infinity)-counter abstraction. In: CAV'02. LNCS, vol. 2404. Springer (2002)
31. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: TACAS'01. LNCS, vol. 2031, pp. 82–97. Springer (2001)
32. Touili, T.: Regular Model Checking using Widening Techniques. Electronic Notes in Theoretical Computer Science 50(4) (2001), proc. of VEPAS'01