Int J Softw Tools Technol Transfer (2016) 18:495–516
DOI 10.1007/s10009-015-0406-x

PV 2014

Parameterized verification through view abstraction

Parosh Abdulla[1] · **Frédéric Haziza**[1] · **Lukáš Holík**[2]

Published online: 23 November 2015

Abstract We present a simple and efficient framework for automatic verification of systems with a parametric 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. In particular, the method handles the fine-grained and full version of Szymanski's mutual exclusion protocol, whose correctness, to the best of our knowledge, has not been proven automatically by any other existing methods.

Keywords Parameterized systems · Safety · Small model properties · View abstraction

P. Abdulla and F. Haziza: supported in part by the Uppsala Programming for Multicore Architectures Research Center (UPMARC). L. Holík: supported by the Czech Science Foundation (project 13-37876P).

✉ Frédéric Haziza
frederic.haziza@it.uu.se

[1] Uppsala University, Uppsala, Sweden

[2] Brno University of Technology, Brno, Czech Republic

1 Parameterized verification

Many systems can be characterized as a family of finite-state systems with one (or more) parameter ranging over an unbounded domain. For each fixed value of the parameter, the system is finite-state. For example, in the case of a program manipulating a list, the parameter could be the length of the list. The system as a whole is a set of finite-state systems, one for each length of the list. For a network protocol, the parameter could be the topology of the network and the system is to be proven safe regardless of how the network nodes are arranged.

The *parameterized verification* problem is to prove correctness of the system, against some specification, for all values of the parameter. A system that contains an a priori unknown parameter must behave properly regardless of the value of the parameter. It is therefore considered infinite-state.

We concentrate in this paper on a specific class of *parameterized systems*, namely, systems consisting of an arbitrary number of processes. The size of the system is the parameter of the verification problem. For each value n of the parameter, the system S_n is the parallel composition of n processes, which can interact with each other at any time. Given a specification, we are interested in proving safety for the family $(S_n)_{n\geq 0}$—as a whole—by showing the non-reachability of some (potentially infinite) set $\mathcal{B}$ of *bad configurations*, starting from some set $\mathcal{I}$ of *initial configurations*. Processes are usually but not necessarily copies of each other.

$$S_n = \underbrace{P \parallel P \parallel \cdots \parallel P}_{n \text{ times}} \quad \text{Can } (S_n)_{n\geq 0} \text{ reach } \mathcal{B} \text{ from } \mathcal{I}?$$

Parameterized systems arise naturally in the modeling of mutual exclusion algorithms, distributed protocols, or cache

coherence protocols. For instance, sensor networks typically consist of thousands of identical nodes, web services must handle millions of requests of the same type. Mutual exclusion must be guaranteed regardless of the number of processes that participate in a given session of the protocol. Cache coherence must be ensured regardless of the number of cache lines or the number of physical processors. Those systems can be handled easily for small values of the parameter, that is, if the system involves a few components. It is interesting to extend the verification to the parameterized case. The total state-space for the whole family is albeit infinite. We use the following strategy:

① We extract a model from each process in the system. This defines the operational semantics with states and transitions for the entire family.
② We use an over-approximation to derive an abstract model from the original one. This defines a new state-space and set of transitions.
③ We determine two sets in the abstract model: (i) the initial states, usually mirroring the initial settings of the program, and (ii) the bad states, usually along the lines of the targeted property.
④ We finally check whether the bad states are reachable from the initial states in the abstract model.

The main focus of this paper lies in that last step. By construction of the over-approximation, showing that the abstract model is safe, will imply that the original system is also correct with respect to its specification, regardless of the number of processes in the system.

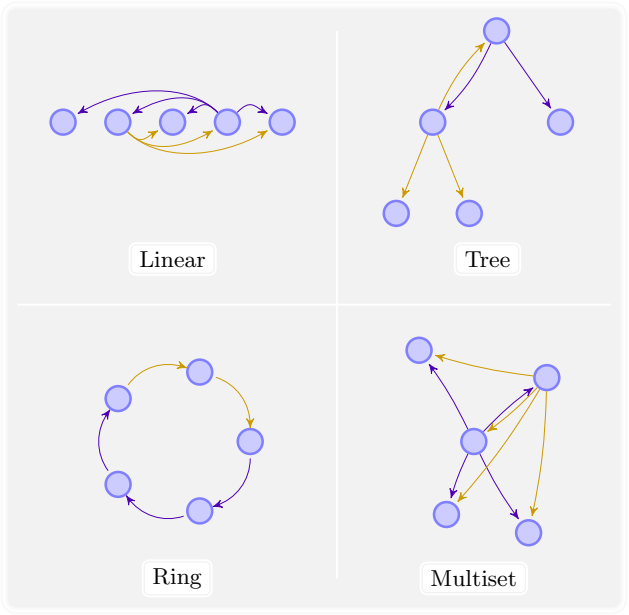


Fig. 1 Topologies

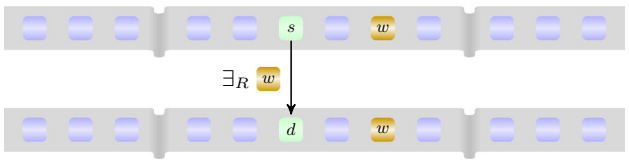


Fig. 2 Example of global transition for a linear topology: a process in state s may change to state d provided that there exists another process in state w on its right

1.1 Features

The parallel composition of several processes can be further characterized with the following features:

- the topology,
- how processes communicate with each other and
- the nature of each process itself (finite-state or not).

The topology describes the way the processes are arranged and implicitly how they can refer to each other, without necessarily revealing their identity. For the *linear* topology, processes are organized in an array and can distinguish between their left and right neighbors. For a *ring*, they can inspect their immediate neighbor, while for a *tree*, they inspect their parent and/or children processes. Finally, the case where there is no particular structure and where a process can refer to any other processes is called a *multiset* topology.

Processes can interact with each other and perform actions potentially in any order, or simultaneously. Those actions are conditioned on the status of the other processes: before it performs its action, a process can inspect, according to the topology (see Fig. 1), other processes whose status can allow or prevent the action from the process. We refer to these transitions as being *guarded* by *global conditions*, or just *global transitions*. For example, in a linear topology, a process (at position i) may be able to perform a transition only if all processes to its left (i.e., with index $j < i$) satisfy a particular property. Sometimes, it is only required for *some* rather than *all* as depicted in Fig. 2. On the other hand, it is occasionally not necessary to refer to other processes at all before performing an action. These actions are by opposition called *local transitions*.

A major issue is that global conditions are not necessarily checked atomically. In case other processes can perform transitions, while a global condition check is carried out, the system must in fact distinguish intermediate states. These interleavings make the state-space grow even further. The salient aspect of the method presented in this paper is to handle elegantly even this complex setting.

Besides local and global transitions, we complete the list with other transitions, which might depend on the topology. For a *broadcast transition*, a process (called the *initiator*) is forced to change state synchronously with an arbitrary number of processes. The initiator might be the only process to move. On the other hand, for a *rendez-vous* transition, two processes are required to change state simultaneously. For *shared variable* update, a process communicates the updated value to the other processes. Finally, process *creation* and *deletion* make the topology dynamic, but it is often not a major issue, as we will show in later sections.

1.2 Formal definition

To simplify the presentation, we will only focus, in this section, on the case where processes are governed by a finite-state automaton and organized in a linear topology. Other topologies are presented in later sections.

A parameterized system is a pair $\mathcal{P} = (Q, \Delta)$ where Q is a finite set of *local states* of a process 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 $\texttt{src} \rightarrow \texttt{dst}$, where the process changes its local state from `src` to `dst` independently from the local states of the other processes. A global rule is either *universal* or *existential*. Recall that a global rule depends on the topology, so it is here of the form:

if $\mathbb{Q}\ j \sim i\ :\ S$ **then** $\texttt{src} \rightarrow \texttt{dst}$

where $\mathbb{Q}$ is either $\exists$ or $\forall$, for existential and universal conditions, respectively, where $\sim$ is either $<$, $>$ or $\neq$, to indicate which processes are concerned, and where S is a subset of Q, describing the state of the other *witness* processes. We call i the current process, `src` the *source*, `dst` the *destination*, $\mathbb{Q}$ the *quantifier* and $\sim$ the *range*. Informally, the ith process checks the local states (among S) of the other processes when it makes the move. For instance, the condition $\forall\ j < i\ :\ S$ means that every process j, with a lower index than the current process i, should be in a local state that belongs to the set S. The condition $\exists\ j > i\ :\ S$ means that there should be a process j with higher index than i with a local state listed in the set S, etc. We consider, in this section only, 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 the later sections.

A *configuration* of $\mathcal{P}$ is a word over the alphabet Q, i.e., an array of process states. We use $\mathcal{C}$ to denote the set of all configurations. For a configuration $c \in \mathcal{C}$, we use $c[i]$ to denote the state at position i in the array and $|c|$ for its size. Let $\mathbb{N}$ be the set of natural numbers. 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 $c \in \mathcal{C}$, a position $i \leq |c|$, and a rule $\delta \in \Delta$, we define how to transform the configuration c into another configuration by allowing the ith process in the array to perform the transition δ. Formally, we define the immediate successor of c under a δ-move of the ith process, such that $\delta(c, i) = c'$ if and only if $c[i] = \texttt{src}$, $c'[i] = \texttt{dst}$, $c[j] = c'[j]$ for all $j : j \neq i$ and either (i) δ is a local rule $\texttt{src} \rightarrow \texttt{dst}$, or (ii) δ is a global rule **if** $\mathbb{Q}\ j \sim i\ :\ S$ **then** $\texttt{src} \rightarrow \texttt{dst}$, and one of the following two conditions is satisfied: (a) $\mathbb{Q} = \forall$ and for all $j \in [\![1,|c|]\!]$ such that $j \sim i$, it holds that $c[j] \in S$, or (b) $\mathbb{Q} = \exists$ and there exists some $j \in [\![1,|c|]\!]$ such that $j \sim i$ and $c[j] \in S$. Note that $\delta(c, i)$ is not defined if $c[i] \neq \texttt{src}$. We use $c \xrightarrow{\delta} c'$ when $c' = \delta(c, i)$ for some $i \leq |c|$, and $c \rightarrow c'$ if $c \xrightarrow{\delta} c'$ for some $\delta \in \Delta$. We define $\xrightarrow{*}$ as the reflexive transitive closure of $\rightarrow$ (i.e., the result of repeatedly applying $\rightarrow$).

1.3 The reachability problem

An instance of the *reachability problem* is defined by

- a parameterized system $\mathcal{P} = (Q, \Delta)$,
- a set $\mathcal{I} \subseteq Q^+$ of *initial configurations*, and
- a set $\mathcal{B} \subseteq Q^+$ of *bad configurations*.

We say that $c \in \mathcal{C}$ is *reachable* if there are $c_0, \ldots, c_m \in \mathcal{C}$ such that $c_0 \in \mathcal{I}$, $c_m = c$, and for all $0 \leq n < m$, there are $\delta_n \in \Delta$ and $j \leq |c_n|$ such that $c_{n+1} = \delta_n(c_n, j)$. In other words,

$$c_0 \in \mathcal{I} \rightarrow c_1 \rightarrow \cdots \rightarrow c_{m-1} \rightarrow c_m = c \text{ i.e. } c_0 \in \mathcal{I} \xrightarrow{*} c.$$

We use $\mathcal{R}$ to denote the set of all reachable configurations (from $\mathcal{I}$), and $\mathcal{R}_k$ for the configurations of size k. We say that the system $\mathcal{P}$ is *safe* with respect to $\mathcal{I}$ and $\mathcal{B}$ if no bad configuration is reachable, i.e., the sets $\mathcal{R}$ and $\mathcal{B}$ do not intersect ($\mathcal{R} \cap \mathcal{B} = \emptyset$). The set $\mathcal{I}$ of initial configurations is usually a regular set.

In order to define the set $\mathcal{B}$ of bad configurations, we use an *entailment relation* $\sqsubseteq$ and 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 the set of bad configurations which often appears in practice. For example, in the (atomic) linear case, the entailment is the usual *subword relation*. For tree topologies, the entailment is the *tree embedding relation*. These relations will be defined in the relevant sections.

1.4 A challenging example

We illustrate the notion of a parameterized system with Szymanski's mutual exclusion protocol [49,50]. Among other

```
0  flag[i] := 1
1  wait until ∀j≠i : flag[j] ∈ {0,1,2}
2  flag[i] := 3
3  if ∃j≠i : flag[j] = 1 then
4        flag[i] := 2
5        wait until ∃j≠i : flag[j] = 4
6  end
7  flag[i] := 4
8  wait until ∀j < i : flag[j] ∈ {0,1}
9  /* Critical Section */
10 wait until ∀j > i : flag[j] ∈ {0,1,4}
11 flag[i] := 0; goto 0;
```

Fig. 3 Szymanski's protocol (for process i)

properties, this protocol ensures exclusive access to a shared resource in a system consisting of an unbounded number of processes organized in an array. The *critical section* is the portion of code in the program which threads are allowed to execute only one at a time. The source code for each process and for the *atomic version* of the protocol is presented in Fig. 3. The full version of Szymanski's mutual exclusion protocol will be handled when we extend the current model in order to drop the atomicity assumption. The critical section is here composed of the statements on line 9 and 10.

Each process participating in a session of the protocol is represented at a fixed position in the array. The state of the ith process reflects the values of the local variable $flag[i]$ and how far it has proceeded in its execution (i.e., its program location). It can therefore be encoded using a unique number for each combination of the values.

A configuration of the induced transition system is a word over the alphabet {0, 1, …, 11} of local process states. The size of a configuration is the parameter of the system. The initial configurations characterize the program before any execution, e.g., using the initial values of the program variables. Here, all processes are initially in state 0, i.e., $\mathcal{I} = 0^+$. The bad configurations are derived from the targeted property. For a mutual exclusion protocol, a configuration is considered to be bad if it contains two occurrences of state 9 or 10, i.e., at least two processes are in their critical section simultaneously. In other words, the bad configurations belong to an infinite set characterized by the following patterns: $\cdots 9 \cdots 9 \cdots$, $\cdots 9 \cdots 10 \cdots$, $\cdots 10 \cdots 9 \cdots$ and $\cdots 10 \cdots 10 \cdots$.

A global transition moves a process from a state to another, provided that the other processes respect the global condition. Here, for example, processes move from state 1 to 2, if the other processes are all in state 0, 1, 2, 5, or 6. If not, the process does not perform the transition and remains in state 1. The intuition, that Szymanski gives in [49], is the presence of a "waiting room", with an entering and exiting door, before processes move into their critical section. The

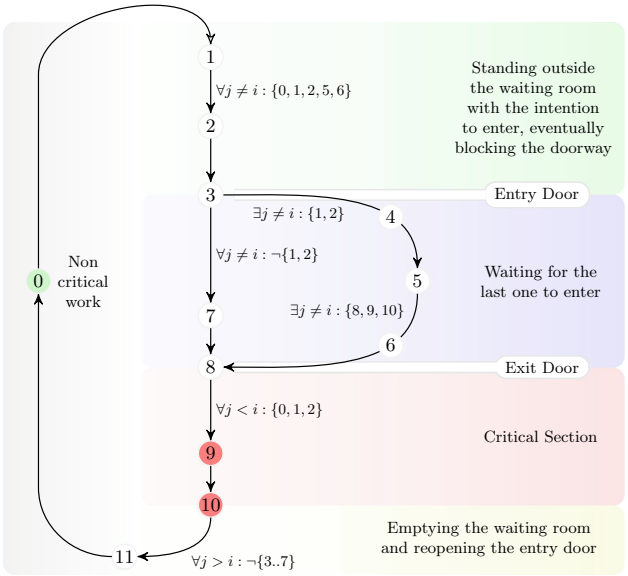


Fig. 4 Transition diagram for Szymanski's protocol: the label on an edge represent the global conditions that other processes must respect in order to perform the transition. There is no label when the transition is local

transition rules are depicted in Fig. 4 using a simple graphical representation, often called a transition diagram.

The task is to check that the protocol guarantees exclusive access to the shared resource regardless of the number of processes, i.e., to show that the bad configurations are not reachable from the initial ones. Many techniques [3,6,8,10,18,19,45] have been used to verify automatically this 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. 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.

2 Verification method for the linear case

We present a forward reachability analysis to solve the reachability problem introduced in previous sections. We focus, here again, on parameterized systems with a linear topology. To further simplify the presentation, the set of states Q is finite, the transitions from Δ are all size-preserving and the global conditions are checked atomically. Some of those restrictions will be dropped in later sections.

The key insight of the method is to take advantage of the fact that the small instances of the system (i.e., for configurations of small sizes) give enough information to derive the behavior of the system in general. We can see the small

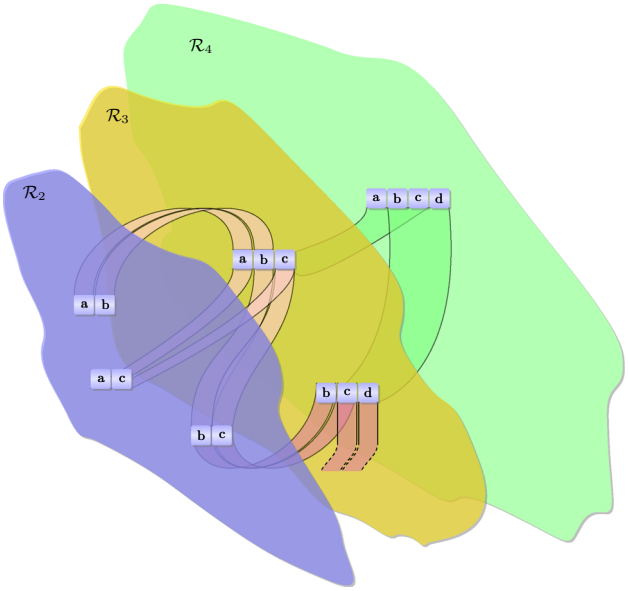


Fig. 5 Repeated patterns

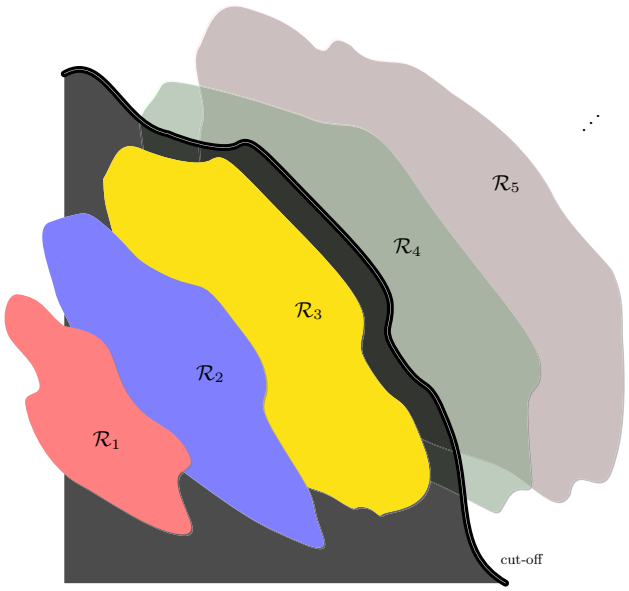


Fig. 6 Small Model Property: the method detects a cut-off point beyond which the verification procedure need not continue

configurations as *patterns* that will only be repeated and intertwined in configurations of larger sizes, as illustrated in Fig. 5. Consider, for example, the configurations with six processes, where one process is passive in, say, its initial state. The remaining five processes of such configurations often (but not necessarily) "cover" the configurations based only on five processes. Moreover, bad patterns, if existing, are often detected already when only a few processes are involved.

The main idea of our method is to exploit this *small model property* and perform parameterized verification by only exploring a small number of *fixed* instances to prove the system safe—for all sizes of configurations. In practice, it is often the case that we only need to compute the finite sets $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ to determine whether the whole set $\mathcal{R}$ contains a bad configuration.

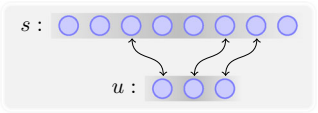
The method *automatically* detects a *cut-off* point beyond which the verification procedure need not continue. Intuitively, it means that the information already collected during the exploration of the state-space until the cut-off point allows us to conclude that no bad configurations will occur in the larger instances. The cut-off detection is performed on-the-fly during the verification procedure itself (and illustrated in Fig. 6).

The configurations from the first instances of the system are abstracted but retain enough information to "reconstruct" the sets of reachable configurations of larger sizes, as we create larger configurations by combining small patterns. In fact, the collected patterns allow to characterize an *over-approximation* of the reachable configurations. It is possible that a recombination of patterns creates a configuration that the original system would not have computed. The key is to abstract the small configurations while retaining enough information in order to not over-approximate the set of reachable configurations too much. Nevertheless, inspecting only small (and finite) instances of the system allows for an efficient method. As usual, if the over-approximation does not contain any bad configuration, we can safely conclude that the system is safe.

2.1 Abstract domain

We first introduce the abstraction at the heart of the method, focusing solely on *atomically* checked global conditions. We call it *view abstraction* since it considers the configurations of the system from the perspective of a few *fixed* number of processes. The abstraction is parameterized by a constant k, and any configuration is "broken down into pieces" of size (at most) k, called *views*. A view retains the information about k processes from a configuration and abstracts away the other processes. There is a certain freedom in what information to retain and what to abstract away. For the k remaining processes, the information could be partial or intact, while the information about the abstracted processes could be fully or partially discarded. This choice defines the level of abstraction that transforms configurations into views.

We first define a simple abstraction: for every $k \in \mathbb{N}$, the k chosen processes are retained intact, while the other abstracted processes are ignored. A view is then a subword of a configuration, using the usual *subword relation*, i.e., $u \sqsubseteq s_1 \dots s_n$ iff $u = s_{i_1} \dots s_{i_k}$, $1 \leq i_1 < \dots < i_k \leq n$.



We use $\mathcal{V}$ to denote the set of all views (and $\mathcal{V}_k$ for the set of views of size up to k). Observe that views here resemble configurations of smaller sizes, so we have $\mathcal{V}_k = \{v \in \mathcal{V} \mid |v| \leq k\} \subseteq \{c \in \mathcal{C} \mid |c| \leq k\}$. This does not need, however, to be the case (c.f. views when the atomicity assumption is dropped in Sect. 5).

The *abstraction function* $\alpha_k : \mathcal{C} \mapsto 2^{\mathcal{V}_k}$ maps a configuration c into the set of all its views (subwords) of size up to k:

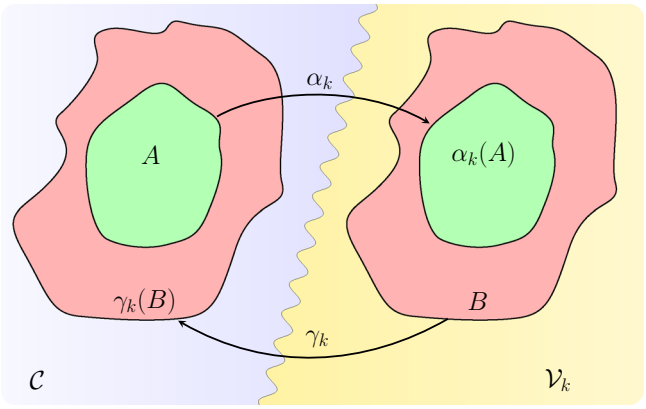
$$\alpha_k(c) = \{v \in \mathcal{V}_k \mid v \sqsubseteq c\}.$$

We lift α_k to sets of configurations as usual. The *concretization function* $\gamma_k : 2^{\mathcal{V}_k} \mapsto 2^{\mathcal{C}}$ takes as input a set of views $V \subseteq \mathcal{V}_k$, and returns the set of configurations that can be reconstructed from the views in V. In other words,

$$\gamma_k(V) = \{c \in \mathcal{C} \mid \alpha_k(c) \subseteq V\}$$

Lemma 1 *(α_k, γ_k) forms a Galois connection.*

Proof We want to prove that, for any $k \in \mathbb{N}$ and for any set $A \subseteq \mathcal{C}$ and $B \subseteq \mathcal{V}_k$, $\alpha_k(A) \subseteq B \Leftrightarrow A \subseteq \gamma_k(B)$.



We first need the following trivial properties: for any $k \in \mathbb{N}$ and for any sets $X, Y \subseteq \mathcal{C}$ and $V, W \subseteq \mathcal{V}_k$,

(i) $X \subseteq Y \Rightarrow \alpha_k(X) \subseteq \alpha_k(Y)$
(ii) $V \subseteq W \Rightarrow \gamma_k(V) \subseteq \gamma_k(W)$
(iii) $\alpha_k(\gamma_k(V)) \subseteq V$
(iv) $X \subseteq \gamma_k(\alpha_k(X))$

Going back to the lemma, $\Rightarrow$ is proven using (ii) and (iv). $\Leftarrow$ is proven using (i) and (iii). □

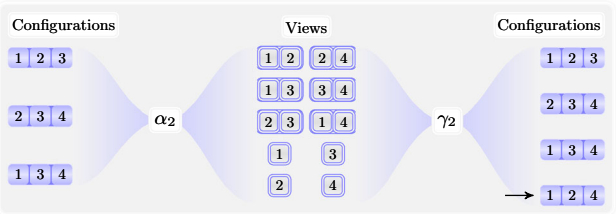


Fig. 7 Consider the configurations on the *left*, their views in the *middle* and their concretization on the *right*. The concretization contains the original set of configurations, but also the extra configuration 1 2 4, i.e., this abstraction is an over-approximation

In general, the set of views *collectively* represent a set of reachable configurations. This is an important subtlety to understand: in this abstraction, the configurations are merely scattered across views. If the information about a given process is ignored in one view, it will necessarily appear in another view.

This abstraction is an over-approximation because the views can reconstruct the configurations they emerge from, but they could also be recombined to form new configurations that were not part of the original system, for any size, as illustrated in the example from Fig. 7.

It is important to observe the precision of the set $\gamma_k(V)$. As we mentioned earlier, the views work collectively to represent the set of configurations. If a view, say 1 2, is recombined into the configuration 1 2 3 in $\gamma_2(V)$, it must be the case that the views 1 3 and 2 3 were *also* present in V. The correlation between the k processes appearing in a view increases the precision of the abstraction. In other words, we get more precise recombinations with larger k.

Lemma 2 *For a set of configurations $X \in \mathcal{C}$,*

$$\gamma_1(\alpha_1(X)) \supseteq \gamma_2(\alpha_2(X)) \supseteq \gamma_3(\alpha_3(X)) \supseteq \cdots \supseteq X$$

Proof We fix $k \in \mathbb{N}$ and a set $X \subseteq \mathcal{C}$. We first introduce an auxiliary function f_{k+1}, similar to α_{k+1}: for a configuration $c \in \mathcal{C}$, we define the function retaining from c only the views of exactly size $k+1$, i.e., $f_{k+1}(c) = \alpha_{k+1}(c) \setminus \alpha_k(c) = \{v \in \mathcal{V} \mid v \sqsubseteq c, |v| = k+1\}$. We lift f_{k+1} to sets of configurations as usual. In particular, $f_{k+1}(X)$ and $\alpha_k(X)$ are disjoint. Therefore,

$$\begin{aligned}
\gamma_{k+1}(\alpha_{k+1}(X)) &= \{c \in \mathcal{C} \mid \alpha_{k+1}(c) \subseteq \alpha_{k+1}(X)\} \\
&= \{c \in \mathcal{C} \mid (\alpha_k(c) \cup f_{k+1}(c)) \subseteq (\alpha_k(X) \cup f_{k+1}(X))\} \\
&= \{c \in \mathcal{C} \mid \alpha_k(c) \subseteq \alpha_k(X) \wedge f_{k+1}(c) \subseteq f_{k+1}(X)\} \\
&\subseteq \{c \in \mathcal{C} \mid \alpha_k(c) \subseteq \alpha_k(X)\} = \gamma_k(\alpha_k(X))
\end{aligned}$$

□

In other words, views of larger sizes impose more constraints on the set of configurations it represents. For example, con-

sider the set $X = \{$ 1 2 3 3 $\}$ with only one configuration. We compute that

$$\begin{aligned}
\alpha_1(X) &= \{\boxed{1}, \boxed{2}, \boxed{3}\} \\
\gamma_1(\alpha_1(X)) &= \{(1|2|3)^+\} \\
\alpha_2(X) &= \{\boxed{1}\boxed{2}, \boxed{1}\boxed{3}, \boxed{2}\boxed{3}, \boxed{3}\boxed{3}\} \cup \alpha_1(X) \\
\gamma_2(\alpha_2(X)) &= \{1\ 2\ 3^*, 1\ 3^*, 2\ 3^*, 3^+\} \\
\alpha_3(X) &= \{\boxed{1}\boxed{2}\boxed{3}, \boxed{1}\boxed{3}\boxed{3}, \boxed{2}\boxed{3}\boxed{3}\} \cup \alpha_2(X) \\
\gamma_3(\alpha_3(X)) &= \{1\ 2\ 3\ 3, 1\ 2\ 3, 1\ 3\ 3, 2\ 3\ 3, \\
&\quad 1\ 2, 1\ 3, 2\ 3, 3\ 3, 1, 2, 3\}.
\end{aligned}$$

In particular, the configuration 1 2 3 3 3 does belong to $\gamma_2(\alpha_2(X))$ but not to $\gamma_3(\alpha_3(X))$.

Finally, for a set of configurations $X \subseteq \mathcal{C}$, we define the *post-image* of X as the set $post(X) = \{\delta(c, i) \mid c \in X, i \leq |c|, \delta \in \Delta\} = \{c' \mid c \in X, c \to c'\}$. The *abstract post-image* of a set of views $V \subseteq \mathcal{V}_k$ is defined, as usual, as the composition $Apost_k(V) = \alpha_k(post(\gamma_k(V)))$.

2.2 Procedure

Now that we have defined the abstraction function and how to jump between sets of configurations and sets of views, we are ready to describe the procedure that manipulates views. The procedure is a forward analysis, composed of two nested loops (on line 1 and 3 in Algorithm 1). The inner-loop explores if the initial configurations can reach the set $\mathcal{B}$ of bad configurations, in the abstract domain using views of size up to k. The outer-loop determines a *cut-off* point K such that the views $V \subseteq \mathcal{V}_K$ of size K, computed by the inner-loop, satisfy the following properties:

(i) V is an invariant for all instances, i.e., $\mathcal{R} \subseteq \gamma_K(V)$ and $Apost_K(V) \subseteq V$
(ii) V is sufficient to prove safety, i.e., $\gamma_K(V) \cap \mathcal{B} = \emptyset$.

Algorithm 1: Verification Scheme

```
1 for k := 1 to ∞ do
2     if R_k ∩ B ≠ ∅ then return Unsafe
3     V := μX . α_k(I) ∪ Apost_k(X)
4     if γ_k(V) ∩ B = ∅ then return Safe
```

Point (i) expresses that we have reached a set of views that collectively over-approximates the set of all reachable configurations $\mathcal{R}$, and that we cannot get new views by applying the abstract post-image. So the set V is "stable" and point (ii) states that none of the configurations represented by V are bad, so the system is safe.

The outer-loop on line 1 searches for a suitable k. The procedure starts by computing (on line 2) the reachable configurations of size k, concretely (i.e., without abstraction), denoted $\mathcal{R}_k$. Formally, $\mathcal{R}_k = post^*(\mathcal{I}_k)$ where $\mathcal{I}_k$ is the set of initial configurations of size k and $post^*$ is the repeated application of the (concrete) post-image.[1] If a bad configuration is discovered during that step, the system is surely not safe, and we can even pinpoint a counter-example. Otherwise, the procedure continues to the inner-loop, a fixpoint computation (line 3) of a set of views that at least covers the views of size k generated from the initial configurations. The inner-loop consists of computing the recombinations from the so-far collected set of views, and detecting if the abstract post-image generates new views. In such a case, the procedure loops and starts again from the new set of views. If not, the fixpoint is reached and no new views will be detected.

Assume that the inner-loop on line 3 reaches a fixpoint. The set V, by construction at the end of the inner-loop, contains the views of $\mathcal{I}$ and is stable under the abstract post-image, i.e., for some k, we have both $\alpha_k(\mathcal{I}) \subseteq V$ and $Apost_k(V) \subseteq V$. It is not hard to derive that this set covers in fact all the views of $\mathcal{R}$, that is $\alpha_k(\mathcal{R}) \subseteq V$ and the set V fulfills point (i).

Lemma 3 *For any $k \in \mathbb{N}$ and any fixpoint V of $Apost_k$ that covers $\alpha_k(\mathcal{I})$, it holds that $\mathcal{R} \subseteq \gamma_k(V)$.*

Proof We fix $k \in \mathbb{N}$ and a set $V \subseteq \mathcal{V}_k$ such that both $\alpha_k(\mathcal{I}) \subseteq V$ and $Apost_k(V) \subseteq V$ hold. The latter expands to $\alpha_k(post(\gamma_k(V))) \subseteq V$. Therefore, by Lemma 1, it holds that $\mathcal{I} \subseteq \gamma_k(V)$ and $post(\gamma_k(V))) \subseteq \gamma_k(V)$. $\gamma_k(V)$ is thus a fixpoint of *post* greater than $\mathcal{I}$. We then derive that $post(\mathcal{I}) \subseteq post(\gamma_k(V)) \subseteq \gamma_k(V)$ and therefore, by iteration, $\mathcal{R} = post^*(\mathcal{I}) \subseteq \gamma_k(V)$. □

The cut-off condition is tested on line 4. There are two outcomes: (a) if the test fails, we do not know whether the system is indeed unsafe or whether the abstraction introduced a spurious behavior. The procedure increases k, hence the precision of the abstraction, and reiterates the outer-loop (line 1), or (b) the test succeeds so the computed set V fulfills point (ii) (and point (i) by the fixpoint computation) so the system is then safe.

2.3 Implementation

Algorithm 1 is only a scheme and computing $Apost_k(V)$ is a central component of the verification procedure. It cannot be computed in a straightforward manner because the set $\gamma_k(V)$ is in general infinite. However, we can easily see that applying the abstraction function on the post-image of

[1] Recall that we consider here transitions that do not change the size of a configuration.

the configurations, which are the (potentially infinite) reconstructions $\gamma_k(V)$ from some set of views V, will mostly return the same views that were already in V, and generate a few new ones. Indeed, for each configuration, only a "small" part changes, the other parts will be abstracted into the same views. This is why it is interesting to enable all transitions on views directly, as if they were performed on configurations, effectively removing the need to re-construct the full configurations.

Going in that direction, we notice that the global condition of a transition can mention some process states that have been abstracted away, making the transition disabled. In order to enable those transitions, we try to *extend* the views such that they would encompass the (missing) witness processes, necessary to perform the transitions. In fact, we show that it is sufficient to consider only the configurations in $\gamma_k(V)$ with size up to $k+1$ (that is, extensions with only *one* extra witness). There are finitely many such configurations, and their post-image can be computed. Formally, for $\ell \geq 0$ and $V \subseteq \mathcal{V}_k$, we define

$$\oint_k^{\ell}(V) := \{c \in \mathcal{C} \mid \alpha_k(c) \subseteq V,\ |c| \leq \ell\}$$

and we now prove that the set of views $\alpha_k(post(\gamma_k(V))) \cup V$ can be instead computed using the following (finite) set $\oint_k^{k+1}(V)$.

Lemma 4 *For any $k \in \mathbb{N}$ and $V \subseteq \mathcal{V}_k$,*

$$\alpha_k(post(\gamma_k(V))) \cup V = \alpha_k\left(post\left(\oint_k^{k+1}(V)\right)\right) \cup V$$

Using this lemma, we can alleviate the problem of reconstructing the (potentially infinite) set of configurations $\gamma_k(V)$ in the abstract post computations. Instead, it is enough to extend the views V of size k in a consistent way, yielding a finite set, and apply the transitions on those extended views. The union on both side of the equation handles the case where a transition is potentially disabled.

Moreover, it is the case that the transitions have *small preconditions*, that is, the views do not need to be extended by much in order for transitions to be enabled. Depending on the transition, we can hence get away with a configuration that is only a slightly extended version of some view (here, extended with one extra *witness* process, see Fig. 9).

Proof of Lemma 4 We will need the following definition in the proof. Given a configuration $c = q_1 \dots q_n$, $k \leq n$, and a subset of its positions $\mathbb{P} = \{i_1, \dots, i_l\} \subseteq [\![1,n]\!]$ with $l \leq k$, and $i_1 < \dots < i_l$, we use $view(c, \mathbb{P})$ to denote the subword $q_{i_1} \dots q_{i_l}$ of c defined over $\mathbb{P}$. For two configurations c, c' such that $c \to c'$ is a transition of the system (i.e., that there exists $i \leq n$ and $\delta \in \Delta$ such that $c' = \delta(c, i)$), we can make use of the following observations (using q_i as the source and q_i' as destination of δ).

1. When δ is a local or universally quantified rule, then for any subwords $c_L \sqsubseteq q_1 \dots q_{i-1}$ and $c_R \sqsubseteq q_{i+1} \dots q_n$, the transition $c_L\, q_i\, c_R \to c_L\, q_i'\, c_R$ is also a transition of the system, induced by δ. Intuitively, such transitions are still enabled when processes on the left or right of the mover are ignored.
2. When δ is an existentially quantified rule, it is of the form **if** $\exists\ j \circ i\ :\ S$ **then** $q_i \to q_i'$. If a needed witness from S is at position j in c, then every transition $view(c, \mathbb{P}) \to view(c', \mathbb{P})$, where both $i, j \in \mathbb{P}$, is also a transition induced by δ. In other words, the transition is still enabled if the views encompass both the mover and the witness.

We prove the lemma by showing that for any configuration $c \in \gamma_k(V)$ of size $m > k+1$ such that there is a transition $c \to c'$ 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 $k+1$ and a transition $d \to d'$ with $v' \in \alpha_k(d')$.

By definition, $v' = view(c', \mathbb{P})$ for some $\mathbb{P} \subseteq [\![1,m]\!]$ (of size k). We use v to denote the view $view(c, \mathbb{P})$. Notice that since $c \in \gamma_k(V)$, any view of c of size at least k belongs to $\gamma_k(V)$, therefore we also have $v \in \gamma_k(V)$. The transition $c \to c'$ is induced by some rule $r \in \Delta$. For local and global rules, the induced transitions change only the state on one position of a configuration. Let $i \in [\![1,m]\!]$ be the index of the position in which c' differs from c. If $i \notin \mathbb{P}$, then $v = view(c, \mathbb{P}) = view(c', \mathbb{P}) = v'$. In this case, we trivially have $v' \in V$. Assume now that $i \in \mathbb{P}$. If r is a local or universally quantified rule, then $v \to v'$ is a transition of the system by observation 1. We can hence take $d = v$ and $d' = v'$.

The interesting case is when r is existentially guarded, i.e., of the form **if** $\exists\ j \circ i\ :\ S$ **then** $q \to q'$. There are two sub-cases: (1) there is a witness from S at some position $j \in \mathbb{P}$ enabling the transition $c \to c'$. Then v still contains the witness on an appropriate position needed to fire r. Therefore $v \to v'$ is a transition of the system induced by r and we can conclude as in the case when r is a local or universally quantified rule. (2) no witness enabling the transition $c \to c'$ is at a position $j \in \mathbb{P}$. Then there is no guarantee that $v \to v'$ is a transition of the system. However, the witness enabling the transition $c \to c'$ must be at some position $j \in [\![1,m]\!]$. We create a configuration of size at most $k+1$ by including this position j to v. Let $\mathbb{P}' = \mathbb{P} \cup \{j\}$. $view(c, \mathbb{P}') \to view(c', \mathbb{P}')$ is a transition of the system induced by r by observation 2. We clearly have that $v' \in \alpha_k(view(c', \mathbb{P}'))$. Since $view(c, \mathbb{P}') \sqsubseteq c$ and $c \in \gamma_k(V)$, we also have that $view(c, \mathbb{P}') \in \gamma_k(V)$. We may therefore take $d = view(c, \mathbb{P}')$ and $d' = view(c', \mathbb{P}')$. The lemma is proven. □

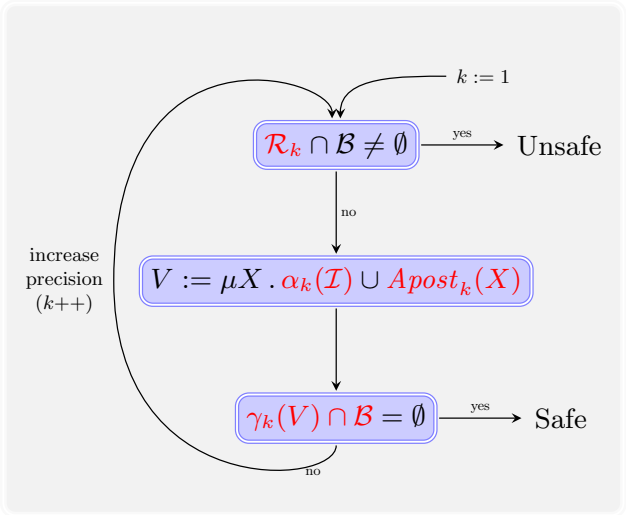


Fig. 8 Algorithm 1 as a diagram

Fig. 9 Consider the configuration on the top and a view v (marked with a black border). The transition **if** $\exists\, j > i \,:\, \{w\}$ **then** $s \rightarrow t$ is disabled on the view as-is, but it is enabled in case the view is extended with the state w and gives then the view v'

To implement an effective procedure, the following steps are required (marked in red in Fig. 8):

① *Computing the abstraction* $\alpha_k(\mathcal{I})$ *of initial configurations* $\mathcal{I}$ is usually a (very simple) regular set, and $\alpha_k(\mathcal{I})$ is easily computed using a straightforward automata construction. For instance, in the case of Szymanski's protocol from Sect. 1.4, all processes are initially in state 0, hence $\alpha_k(\mathcal{I})$ contains only the words of size $l \leq k$ consisting of only the letter 0.

② *Computing the abstract post-image* We circumvent the potential infinite set $\gamma_k(V)$ and only consider the abstract post-image of the configurations from the finite set $\oint_k^{k+1}(V)$ (Lemma 4).

③ *Evaluating the test* $\gamma_k(V) \cap \mathcal{B} = \emptyset$. Since $\mathcal{B}$ is usually the upward-closure of a finite set $\mathcal{B}_{min}$, the test can be carried out by testing whether there is $b \in \mathcal{B}_{min}$ such that $\alpha_k(b) \subseteq V$. For instance, for Szymanski's protocol, all the bad configurations follow the same patterns: 9 9, 9 10, 10 9 or 10 10. This means that, for $k = 2$, they must at least contain one of the views 9 9, 10 9, 9 10 or 10 10. We check whether any latter view is included in the fixpoint.

④ *Exact reachability analysis* Line 2 requires the computation of $\mathcal{R}_k$. Since $\mathcal{R}_k$ is finite, it can be computed with any (fast) procedure for exact exploration of finite-state systems.

The implementation presented in Algorithm 2 fulfills the above requirements.

Algorithm 2: Verification procedure

```
1 for k := 1 to ∞ do
2     if α_k(R_k) ∩ B_min ≠ ∅ then return Unsafe
3     V := μX . α_k(R_k) ∪ α_k ∘ post ∘ ∮_k^{k+1}(X)
4     if V ∩ B_min = ∅ then return Safe
```

2.3.1 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 $\mathcal{I}$), We can namely use the set $\alpha_k(\mathcal{R}_k)$. Furthermore, a similar argument can be used to see that it is not necessary to apply *post* on the views in $\oint_k^{k+1}(X)$ that are already in $\alpha_{k+1}(\mathcal{R}_{k+1})$. We therefore seed the fixpoint computation with a larger set than $\alpha_k(\mathcal{I})$, namely $\alpha_k(\mathcal{R}_k \cup \mathcal{R}_{k+1})$, and cache the set of views $\alpha_{k+1}(\mathcal{R}_{k+1})$ (note that the latter requires to also compute $\mathcal{R}_{k+1}$ on line 2).

The effect of these accelerations is that most experiments turn out to be (almost already) at fixpoint, when we use $\alpha_k(\mathcal{R}_k \cup \mathcal{R}_{k+1})$ as initial input. This demonstrates the efficiency of the method and that most behaviors are captured by small instances of the system.

2.4 Termination and completeness

We have shown so far that the method is sound, i.e., when it returns that the system is safe, it is indeed the case. Let us denote V_k the least fixpoint computed on line 3 at iteration k of the outer-loop. This section is technical but Fig. 10 illustrates the intuition. We present two interesting results. The first result is that the precision of the fixpoint V_k is increasing with k.

Lemma 5 $\forall k \in \mathbb{N}, \gamma_{k+1}(V_{k+1}) \subseteq \gamma_k(V_k)$

Proof We first need an auxiliary result. We fix a $k \in \mathbb{N}$ and recursively define V_k^i such that, $V_k^0 = \alpha_k(\mathcal{I})$ and $\forall i \in \mathbb{N}, V_k^{i+1} = \alpha_k \circ post \circ \gamma_k(V_k^i)$. The following statement holds:

$\forall i \in \mathbb{N}, \gamma_{k+1}(V_{k+1}^i) \subseteq \gamma_k(V_k^i)$.

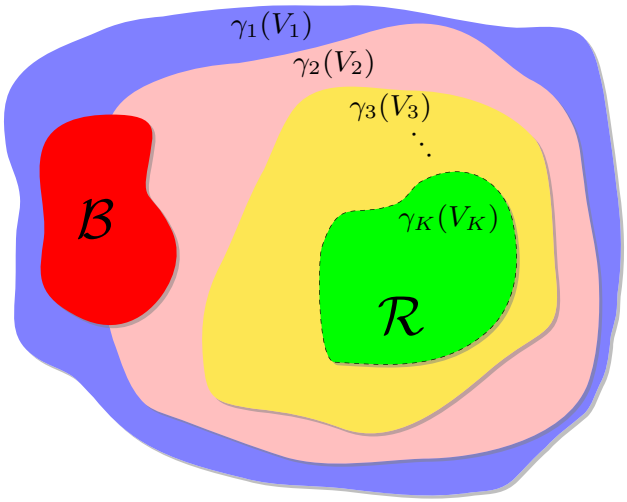


Fig. 10 Increasing precision of the fixpoint computation

We prove it by induction. We recall Lemma 2 about the increasing precision for larger k: for $k \in \mathbb{N}$ and $X \in \mathcal{C}$, $\gamma_{k+1}(\alpha_{k+1}(X)) \subseteq \gamma_k(\alpha_k(X))$. For the base case, $\gamma_{k+1}(V^0_{k+1}) = \gamma_{k+1}(\alpha_{k+1}(\mathcal{I})) \subseteq \gamma_k(\alpha_k(\mathcal{I})) = \gamma_k(V^0_k)$ (by Lemma 2). Assume now that the statement holds for some i.

$$\begin{aligned}\gamma_{k+1}(V^{i+1}_{k+1}) &= \gamma_{k+1}(\alpha_{k+1}(post(\gamma_{k+1}(V^i_{k+1}))))\\ &\subseteq \gamma_{k+1}(\alpha_{k+1}(post(\gamma_k(V^i_k)))) \text{ (by induction)}\\ &\subseteq \gamma_k(\alpha_k(post(\gamma_k(V^i_k)))) \text{ (by Lemma 2)}\end{aligned}$$

So $\gamma_{k+1}(V^{i+1}_{k+1}) \subseteq \gamma_k(V^{i+1}_k)$ and the statement is proven for $i+1$. We can now prove the lemma (using Lemma 2).

$$\begin{aligned}\gamma_{k+1}(V_{k+1}) = \gamma_{k+1}(\cup_{i\in\mathbb{N}} V^i_{k+1}) &= \gamma_{k+1}(V^0_{k+1} \cup \bigcup_{i\in\mathbb{N}} V^{i+1}_{k+1})\\ &= \gamma_{k+1}(\alpha_{k+1}(\mathcal{I}) \cup \bigcup_{i\in\mathbb{N}} \alpha_{k+1} \circ post \circ \gamma_{k+1}(V^i_{k+1})))\\ &= \underbrace{\gamma_{k+1}(\alpha_{k+1}}(\mathcal{I} \cup \bigcup_{i\in\mathbb{N}} post(\underbrace{\gamma_{k+1}(V^i_{k+1})}))\\ &\subseteq \gamma_k(\alpha_k(\mathcal{I} \cup \bigcup_{i\in\mathbb{N}} post(\gamma_k(V^i_k))))\\ &= \gamma_k(\alpha_k(\mathcal{I}) \cup \bigcup_{i\in\mathbb{N}} \alpha_k \circ post \circ \gamma_k(V^i_k))\\ &= \gamma_k(V^0_k \cup \bigcup_{i\in\mathbb{N}} V^{i+1}_k) = \gamma_k(V_k)\end{aligned}$$

□

If the system is not safe, there surely is a k such that $\mathcal{R}_k$ exhibits the error. Conversely, the existence of k for which the test on line 4 succeeds for a safe system cannot be guaranteed—since the problem is generally undecidable—and the algorithm may not terminate. As a second result, we show that such a guarantee can be given under the additional requirement of monotonicity of the transition relation with respect to a well quasi-ordering (WQO for short). We first introduce some definitions from the WQO framework [1,2], in general, and we describe the second result in the context of linear topology using the subword relation $\sqsubseteq$ (see Sect. 3.6 for the general case).

A preorder $\preccurlyeq$ over a set S is said to be a *well quasi-order* (WQO) if for any infinite sequence $s_0, s_1, s_2, \ldots$ of elements of S, there exists i and j such that $i < j$ and $s_i \preccurlyeq s_j$. We define the *upward-closure* of a set $A \subseteq \mathcal{C}$ as the set $\uparrow A = \{c \in \mathcal{C} \mid \exists a \in A : a \preccurlyeq c\}$ and its *downward-closure* as the set $\downarrow A = \{c \in \mathcal{C} \mid \exists a \in A : c \preccurlyeq a\}$. We say that a set $A \subseteq \mathcal{C}$ is *upward-closed* if $A = \uparrow A$, and *downward-closed* if $A = \downarrow A$. Moreover, for an upward-closed set U, we define the *minimal elements* of U, as the set M such that

(i) *[Closure]* $\uparrow M = U$, i.e., U can be generated by taking the upward-closure of M,
(ii) *[Minimality]* $\forall a, b \in M, a \preccurlyeq b \implies a = b$, i.e., all the elements of M are incomparable (w.r.t. $\preccurlyeq$).

The set M is therefore uniquely defined, called the *generator* of the upward-closed set U and denoted $gen(U)$. In particular, if $\preccurlyeq$ is a WQO, the definition implies that there is no infinite sequence of incomparable elements, so every generator is finite.

A relation $\mathcal{R} \subseteq S \times S$ is *monotonic* w.r.t. $\preccurlyeq$ if whenever $(s_1, s_2) \in \mathcal{R}$ and $s_1 \preccurlyeq s'_1$, then there is s'_2 with $(s'_1, s'_2) \in \mathcal{R}$ and $s_2 \preccurlyeq s'_2$. Given a monotonic relation $f \subseteq S \times S$ (w.r.t. $\preccurlyeq$) and a set $A \subseteq S$, it holds that if $f(A) \subseteq A$, then $f(\downarrow A) \subseteq \downarrow A$, where $f(A)$ is the image of A defined as $\{a' \mid \exists a \in A : (a, a') \in f\}$. We say that a transition system $(\mathcal{C}, \rightarrow)$ is *well quasi-ordered* by a WQO $\preccurlyeq$, if $\rightarrow$ is monotonic w.r.t. $\preccurlyeq \subseteq \mathcal{C} \times \mathcal{C}$.

We now state the completeness result in the context of linear topologies and the abstraction from Sect. 2.1. The subword relation $\sqsubseteq$ is a well quasi-ordering.

Lemma 6 *If $\mathcal{R}$ is downward-closed (w.r.t. $\sqsubseteq$), there exists K such that $\mathcal{R} = \gamma_K(V_K)$.*

Proof By Lemma 3, we have already established that $\forall k \in \mathbb{N}$, $\mathcal{R} \subseteq \gamma_k(V_k)$, when V_k is a fixpoint of $Apost_k$. A set $X \in \mathcal{C}$ is inductive if it is stable under *post* and covers $\mathcal{I}$, i.e., $X \supseteq \mathcal{I}$ and $post(X) \subseteq X$. In particular, $\mathcal{R}$ is the smallest inductive set.

We now show that there exists a K such that, for any set X of configurations that is inductive and downward-closed (w.r.t. $\sqsubseteq$), it holds that $\gamma_K(V_K) \subseteq X$. We will therefore conclude by applying the statement to $X = \mathcal{R}$.

We use $\overline{X}$ to denote the complement of X (i.e., $\overline{X} = \mathcal{C} \setminus X$). Since X is downward-closed, its complement $\overline{X}$ is upward-closed and is generated by a set of minimal elements $M =$

$gen(\overline{X})$. The subword relation $\sqsubseteq$ is a WQO and hence the set M is finite. Let $M = \{m_1, \ldots, m_\ell\}$ and $K = \max_{1 \le i \le \ell} |m_i|$.

We show first that $\gamma_K(\alpha_K(X)) = X$. We already know that $X \subseteq \gamma_K(\alpha_K(X))$ by Lemma 1. Take an element $s \notin X$. It follows that $s \in \uparrow M$ and there exists then $t \in M$ such that $t \sqsubseteq s$ with $|t| \le K$. This implies that $t \notin \alpha_K(X)$ and $s \notin \gamma_K(\alpha_K(X))$. We have then proved that $\gamma_K(\alpha_K(X)) \subseteq X$, if X is downward-closed. If X is inductive, we also have that $post(X) \subseteq X$ (and $X \supseteq \mathcal{I}$), so we conclude that

$$post(X) = post(\gamma_K(\alpha_K(X))) \subseteq X \text{ and hence}$$
$$\alpha_K(post(\gamma_K(\alpha_K(X)))) \subseteq \alpha_K(X).$$

The set of views $\alpha_K(X)$ is then a fixpoint of the abstract post $Apost_K$ (which covers $\alpha_K(\mathcal{I})$). Since V_K is defined as the least fixpoint of $Apost_K$ covering $\alpha_K(\mathcal{I})$, we have $V_K \subseteq \alpha_K(X)$ and therefore $\gamma_K(V_K) \subseteq X$ (by Lemma 1). □

Corollary 1 *If $\mathcal{R}$ is downward-closed and $\mathcal{R} \cap \mathcal{B} = \emptyset$, Algorithm 1 is guaranteed to terminate.*

Corollary 2 *If $\mathcal{B}$ is upward-closed and the transition relation $\rightarrow$ is monotonic, then*

(a) $\mathcal{R} \cap \mathcal{B} = \emptyset \Leftrightarrow \downarrow\mathcal{R} \cap \mathcal{B} = \emptyset$
(b) *Algorithm 1 is guaranteed to terminate.*

This implies that our method is complete for parameterized systems with linear topology, local transitions and existential transitions. The induced transition relation is then monotonic w.r.t. the subword relation $\sqsubseteq$. Systems with linear topology and universal transitions do not satisfy the assumptions: the induced transition relation is not monotonic.

3 Extensions

In this section, we describe how to extend the class of parameterized systems that we presented in Sect. 1.2. The extensions are obtained

1. by extending the types of allowed transitions and
2. by considering topologies other than the linear one.
3. by replacing the atomically checked global transitions with more realistic for-loops (covered in Sect. 5).

As we shall see, the extensions can be handled by merely instantiating the method of Sect. 2 with straightforward extensions of the abstraction, concretization and post operator functions.

3.1 More communication mechanisms

Broadcast In a broadcast transition, an arbitrary number of processes change states simultaneously. A broadcast rule is a pair $(q \rightarrow q', \{r_1 \rightarrow r_1', \ldots, r_m \rightarrow r_m'\})$. 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 $q \rightarrow q'$. 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 $q, r_1, \ldots r_m$ remain passive during the broadcast. Formally, the broadcast rule induces transitions $c \rightarrow c'$ of $\mathcal{P}$ where for some $i : 1 \le i \le |c|$, $c[i] = q$, $c'[i] = q'$, and for each $j : 1 \le j \ne i \le |c|$, if $c[j] = r_k$ (for some k) then $c'[j] = r_k'$, otherwise $c[j] = c'[j]$.

In a similar manner to global 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 4 still holds. Therefore, the verification method from Sect. 2.2 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 do not imply non-determinism, but that the rendez-vous requires multiple occurrences of r in the configuration to be triggered. Formally, the rule induces transitions $c \rightarrow c'$ of $\mathcal{P}$ such that there are $i_1, \ldots, i_m$ with $i_j \neq i_k$ for all $j \neq k$, such that $c[i_1] \ldots c[i_m] = r_1 \ldots r_m$, $c'[i_1] \ldots c'[i_m] = r_1' \ldots r_m'$, and $c'[\ell] = c[\ell]$ if $\ell \notin \{i_1, \ldots, i_m\}$.

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 δ 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 $\bullet \rightarrow r, \bullet \notin Q$ (acting as a placeholder), which are used to model creation of processes, and of the form $r \rightarrow \bullet$, 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 $\bullet \rightarrow r$, 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 $r \rightarrow \bullet$). Finally, all occurrences of $\bullet$ are removed. Formally, a generalized rendez-vous rule induces a transition $c \rightarrow c'$ if there is $c_\bullet \in \{\bullet\}^* c[1]\{\bullet\}^* \cdots \{\bullet\}^* c[|c|]\{\bullet\}^*$ such that $c_\bullet \rightarrow c'_\bullet$ is a transition of the system with states $Q \cup \{\bullet\}$

induced by δ (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 processes (but still a fixed number) 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 4 holds here only in a weaker variant:

Lemma 7 *Let Δ contain local, global, broadcast, and rendez-vous rules, and let $m+1$ is the largest arity of a rendez-vous rule in Δ. Then, for any $k \in \mathbb{N}$ and set of views $V \subseteq \mathcal{V}_k$,*

$$\alpha_k(post(\gamma_k(V))) \cup V = \alpha_k\left(post\left(\oint_k^{k+m}(V)\right)\right) \cup V$$

Proof For a broadcast transition, in a similar manner to the proof of Lemma 4, the initiator is treated analogously to a witness of an existentially quantified transition. If the position i of the initiating process is in $\mathbb{P}$, then $v \to v'$ is a transition of the system. Otherwise we include i, i.e., we define $\mathbb{P}' = \mathbb{P} \cup \{i\}$, in which case $view(c, \mathbb{P}') \to view(c', \mathbb{P}')$ is a transition of the system and we take $d = view(c, \mathbb{P}')$ and $d' = view(c', \mathbb{P}')$.

For rendez-vous transitions, the proof is again similar to the case of existentially quantified transitions. A difference is that, in order to create a transition of the system, it is necessary to put into $\mathbb{P}'$ all witness positions needed to fire the transition and that are missing in $\mathbb{P}$. It results that $view(c, \mathbb{P}')$ may be of size at most $k + m$, as reflected in the statement of the lemma. □

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, α_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 α_k, Lemma 4 and Lemma 3 still hold. The method of Sect. 2.2 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 [37]. 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.

3.2 Transitions that do not preserve size

We now discuss the case when transitions do not preserve the size of configurations, which happens in the case of generalized 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 [34], we only consider runs of the system visiting configurations of the size up to k. That is, on line 2 of Algorithm 2, instead of computing $\mathcal{R}_k = \mu X.\, \mathcal{I}_k \cup post(X)$, we compute its under-approximation $\mu X.\, (\mathcal{I} \cup post(X)) \cap Q^k$, i.e., the set of configurations that can be reached traversing configurations of size at most k. The computation terminates provided that Q^k is finite. The algorithm is still guaranteed to return `Unsafe` if a configuration of $\mathcal{B}$ is reachable, because there is a maximal size $k \in \mathbb{N}$ for which such a bad configuration is reachable by a finite path involving configurations of the size at most k.

3.3 Tree topology

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

Trees A (binary) *tree* over a finite set Q is a (partial) mapping $t : \{0,1\}^* \to Q$ such that the set of its *nodes* $N_t = \{n \in \{0,1\}^* \mid t(n) \text{ is defined}\}$ is prefix closed. The node ϵ is called the *root* and the 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' is a *subtree* of t, denoted $t' \preceq t$, iff there exists a injective map $e : N_{t'} \to N_t$ 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 in Sect. 1.2. A parameterized system $\mathcal{P} =$

(Q, Δ) induces a transition system $(\mathcal{C}, \rightarrow)$ where $\mathcal{C}$ is the set of complete binary trees over Q. The set Δ of transition rules is a set of *local* and *tree* transition rules. The transitions of $\rightarrow$ are obtained from rules of Δ as follows. A *local* rule is of the form $q \rightarrow q'$ and it locally changes the label of a node from q to q'. A *tree* rule is a triple $q(q_0, q_1) \rightarrow q'(q'_0, q'_1)$. The rule can be applied to a node v and its left and right children v_0, v_1 with labels q, q_0, and q_1, respectively, and it changes their labels to q', q'_0, and q'_1, respectively.

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

Verification The text of Sect. 2 can be taken almost verbatim and extended in the following way. Words are replaced by complete binary trees, the subword relation is replaced by the subtree relation, and k now refers to the depth of trees rather than the length of words. 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 Sect. 2.1, based on α_k. The set $\mathcal{I}$ may be given in the form of a tree automaton. The computation of $\alpha_k(\mathcal{I})$ may be then done over the structure of the tree automaton. We can compute the abstract post-image since Lemma 4 holds here in the same wording as in Section 2.2.[2] The test $\gamma_k(V) \cap \mathcal{B} = \emptyset$ is carried out in the same way as in Sect. 2.2 since $\mathcal{B}$ is an upward-closure of a set $\mathcal{B}_{min}$ w.r.t. $\preceq$. The points ①-④ of Sect. 2.3 (on page 8) are thus satisfied and Algorithm 2 can be used as a verification procedure for systems with tree topology.

3.4 Ring topology

We extended the method to systems with 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 such systems similarly to systems with linear topology, and a configuration $c \in Q^+$ is now interpreted as a circular word. The set Δ may contain local transitions and *near-neighbor* transitions. A near-neighbor transition is a pair $(q_1 \rightarrow q'_1, q_2 \rightarrow q'_2)$. It induces the transition $c \rightarrow c'$ if either $c = c_L\, q_1 q_2\, c_R$ and $c' = c_L\, q'_1 q'_2\, c_R$ (i.e., the 2 processes are adjacent in the configuration c) or $c = q_2\, \bar{c}\, q_1$ and $c' = q'_2\, \bar{c}\, q'_1$ (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.

[2] In fact, the post-image is rather implemented in a similar manner than the symbolic post-image described in Sect. 4.2.1.

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 linear topology is that the standard subword relation is in all definitions replaced by the circular subword relation $\trianglelefteq$. An equivalent of Lemma 4 holds here in unchanged wording, points ①-④ are satisfied, and Algorithm 2 is thus a verification procedure for systems with ring topology.

3.5 Multiset topology

Systems with multiset topology are a special case of the systems with linear topology from Sect. 1.2. A typical example of such systems is a Petri net, which precisely corresponds to parameterized systems with only (generalized) rendez-vous transitions. 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. Therefore, 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).

3.6 Completeness for WQO systems

The reasoning in Sect. 2.4 is based on the natural notion of the size of a configuration. Its generalization is the notion of a *discrete measure* over a set S, a function $\|.\| : S \rightarrow \mathbb{N}$ which fulfills the property that for every $k \in \mathbb{N}$, the set $\{s \in S \mid \|s\| = k\}$ is finite. A discrete measure is necessary to obtain the completeness result as it allows enumerating elements of S of the same size. We note that the existence of a discrete measure is implied by a stronger restriction of [16] to the so-called discrete transition systems.

Given a well quasi-ordered system and a measure $\|.\| : \mathcal{C} \rightarrow \mathbb{N}$, we define an abstraction function $\alpha_k, k \in \mathbb{N}$ such that $\alpha_k(c) = \{c' \in \mathcal{C} \mid c' \preccurlyeq c, \|c'\| \leq k\}$. The corresponding concretization γ_k and abstract post-image $Apost_k$ are then defined based on α_k and $\|.\|$ as in Sect. 2.1. Lemma 3 holds here in the same wording as in Sect. 2.2, which implies that if the test on line 4 succeeds, the system is indeed safe.

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

Theorem 1 is the main component of the completeness result and guarantees that for a safe well quasi-ordered sys-

tem, there exists k for which the test on line 4 of Algorithm 1 succeeds. This implies that our method is complete for, e.g., lossy channel systems [9], for systems that are generated by monotonic abstraction [7], and in particular for Petri nets. The latter correspond indeed to parameterized systems with multiset topology and generalized rendez-vous transitions. Notice that, as evident from our experiments (in Sect. 6), the method can in practice handle even systems that are outside the class of WQO systems, because it is sufficient to have a good downward-closed invariant (i.e., one that does not intersect with $\mathcal{B}$). The only example not having that is Szymanski's mutual exclusion protocol (where every $\gamma_k(V_k)$ intersects with $\mathcal{B}$). The method must be extended to cope with the full and non-atomic version of the protocol (see Sects. 4 and 5).

4 Contexts

The simple abstraction presented in Sect. 2.1 allows us to compute invariants that are downward-closed w.r.t. $\sqsubseteq$, but 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.

Let us briefly explain where the problem comes from. Consider for example the view a b. It emerges from some configurations that contain the former view as a subword. It might be the case that *all* those configurations do also contain d on the right of b, that is, all the configurations contain the subword a b. In such a case, the transition **if** $\forall\, j > i : \neg\{d\}$ **then** $a \to c$ will be enabled on the view a b d, but it would not have been enabled on any of the configurations that contain that view as a subword. The problem is now that the new view c b is potentially harmful in the sense that it can lead to a counter-example. Notice moreover that no extension to views of larger size will always disable the transition on those views (i.e., increasing the precision will not eliminate the problem).

In this section, we introduce a new type of views and adapt the method from Sect. 2 to target a class of invariants which are needed in many practical cases and cannot be expressed as downward-closed sets. This is in fact the case of Szymanski's mutual exclusion protocol where the (simple) abstraction from Sect. 2.1 always triggered false-positives, regardless of the size of the parameter.

4.1 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 ith *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., $1 \leq i < j \leq k \Rightarrow 1 \leq h(i) < h(j) \leq n$. Moreover, we require that $h(0) = 0$ and $h(k+1) = n+1$.

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. 11):

(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 from 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: (see Fig. 12)

(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 $i \in [\![0,k]\!]$.

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

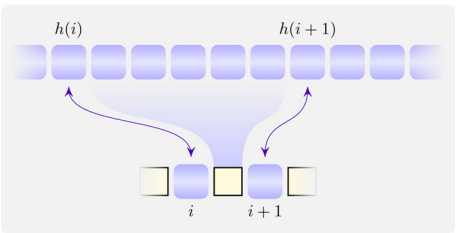


Fig. 11 Projection

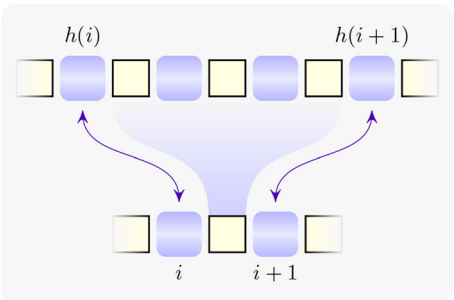


Fig. 12 View projection

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 $\lfloor V \cup W \rfloor$.

We are now ready to define the abstraction and concretization functions using context-sensitive views as a symbolic encoding. For $k \in \mathbb{N}$, the *abstraction function* α_k maps x, a view or a configuration, into the set of its projections of size k or smaller:

$$\alpha_k(x) = \{\Pi_h(x) \mid h \in H^{\ell}_{|x|}, \ell \leq \min(k, |x|)\}$$

For a set X of views or of configurations, we define $\alpha_k(X)$ as the set of its weakest projections $\lfloor \cup_{x \in X} \alpha_k(x) \rfloor$. The *concretization* function γ_k maps a set of views $V \subseteq \mathcal{V}_k$ into the set of configurations

$$\gamma_k(V) = \{c \in \mathcal{C} \mid V \preccurlyeq \alpha_k(c)\}$$

4.2 Verification procedure and approximation

The verification procedure from Algorithm 2 must be adapted to cope with contexts. The procedure will still take advantage of extensions, but the latter are no longer configurations of smaller size. They are now equipped with a context, and the concrete post from Sect. 2.1 cannot directly be used on them. Moreover, since $\gamma_k(V)$ is in general infinite, we need to compute the abstract post-image symbolically.

Although it is possible to compute the abstract transformer precisely [4], we introduce an over-approximation for efficiency reasons and show that the resulting procedure is sound and complete.

4.2.1 Symbolic post operator

To define the symbolic post operator, we first define a transition relation on views. For a view $v = (\texttt{base}, \texttt{ctx})$, $i \leq |\texttt{base}|$, and a transition $\delta \in \Delta$, we define the symbolic immediate successor of v under a δ-move of the ith 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 $i \leq n$, we have that $\delta^{\#}(v, i) = R_0 b'_1 R_1 \ldots b'_n R_n$ iff $b_i = \texttt{src}$, $b'_i = \texttt{dst}$, $b_j = b'_j$ for all $j : j \neq i$ and either

(i) δ is a local rule $\texttt{src} \to \texttt{dst}$, or
(ii) δ is a global rule of the form **if** $\mathbb{Q}\, j \sim i : S$ **then** $\texttt{src} \to \texttt{dst}$, and one of the following two conditions is satisfied: ($\sim$ is as in Sect. 1.2)
   (a) $\mathbb{Q} = \forall$ and it holds both that $b_j \in S$ for all $j \in [\![1,n]\!]$ such that $j \sim i$ and that $R_j \subseteq S$ for all $j \in [\![0,n]\!]$ such that $j \sim i$, or
   (b) $\mathbb{Q} = \exists$ and there exists $j \in [\![1,n]\!]$ such that $j \sim i$ 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 $spost(V) = \{\delta^{\#}(v, i) \mid v \in V, i \leq |v|, \delta \in \Delta\}$.

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. an existential transition as illustrated in Fig. 9). 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 "materializes" a moving process into the base and a second extension 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 $\alpha_k \circ spost \circ \oint_k^{k+2}(V)$. In a similar manner to Lemma 4, we show that the symbolic post is the best abstract transformer.

Lemma 8 *For any k and set of views V of size up to k, it holds that*

$$V \sqcup \alpha_k \circ post \circ \gamma_k(V) = V \sqcup \alpha_k \circ spost \circ \oint_k^{k+2}(V)$$

4.2.2 Approximation

The computation of the above exact symbolic post comes at some cost, mostly due to the computation of $\oint_k^\ell(V)$. We therefore introduce an over-approximation and compute the set

$$\boxed{\int}_k^\ell(V) = \{v \in \mathcal{V} \mid \alpha_k(v) \succcurlyeq V, |v| \leq \ell\}$$

i.e., the set of views of size ℓ that can be generated from V, without inspecting its concretization first. Views in $\boxed{\int}_k^\ell(V)$ have (at least) the same bases as the views in $\oint_k^\ell(V)$, but they might have smaller contexts (and are therefore weaker). As a consequence, they might enable more (universal) transitions than their counterparts in $\oint_k^\ell(V)$. Consider for example the case where $k = 2$, $\ell = 3$ and the set of views $V = \{ab, bc, ac[e], ce[f], ae, be, af, bf, cf, ef\}$—we write contexts in brackets, and ignore the empty contexts for brevity. The set $\boxed{\int}_2^3(V)$ contains the view $abc[e]$ but $\oint_2^3(V)$ 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, $\boxed{\int}_2^3(V)$ 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 $\gamma_2(V)$.

In fact, $\boxed{\int}_k^\ell(V)$ over-approximates $\oint_k^\ell(V)$, i.e., for any $\ell \geq k$ and $V \subseteq \mathcal{V}$, it holds that

$$\boxed{\int}_k^\ell(V) \preccurlyeq \oint_k^\ell(V)$$

Algorithm 3: Verification Procedure

```
1 for k := 1 to ∞ do
2     if bad(R_k) then return Unsafe
3     V := μX . α_k(I) ⊔ α_k ∘ spost ∘ ⊡_k^{k+2}(X)
4     if ¬bad(V) then return Safe
```

We now adapt Algorithm 2 to use the symbolic post and the approximation above-mentioned. In the resulting algorithm, Algorithm 3, it is sound to replace $\oint_k^{k+2}$ with the set $\boxed{\int}_k^{k+2}$, in the fixpoint computation of the symbolic post (line 3).

The termination criteria on line 2 and 4 require the use of the function `bad`, which returns whether a set of configurations contains a bad configuration or whether a set of views characterizes a bad configuration, depending on the type of its input parameter. If its input parameter is a set X of configurations, the function `bad` is implemented by checking whether any configuration from $\mathcal{B}_{min}$ can be a subword of some configuration in X. If its input parameter is a set V of views, the function `bad` is implemented by checking whether an element of $\mathcal{B}_{min}$ can appear within the base of a view from V. We do not inspect any context, because the views work collectively and there is always another view in the set which contains this context element in its base.

Acceleration In a similar manner to Sect. 2.3.1, the fixpoint computation on line 3 can be accelerated by leveraging the entailment relation. The configurations in $\mathcal{R}_k$ can be used as initial input for the fixpoint computation (rather than $\mathcal{I}$). All views of size k in $\alpha_k(\mathcal{R}_k)$ have empty contexts (i.e., they are weakest). 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 $\boxed{\int}_k^{k+2}(X)$ 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 $\alpha_k(\mathcal{I})$, namely $\alpha_k(\mathcal{R}_k \cup \mathcal{R}_{k+1} \cup \mathcal{R}_{k+2})$, and cache the set of views $\alpha_{k+2}(\mathcal{R}_{k+2})$.

5 Dropping the atomicity assumption

We now extend the model from the previous section to handle parameterized systems where global conditions are *not* checked atomically. It is necessary for the model to keep track of intermediate configurations when a non-atomic global condition is evaluated at the same time as another transition, potentially also guarded by a global condition. We use the model presented in Sect. 1.2. However, both existentially and universally guarded transitions are replaced with a variant of a for-loop rule of the form:

if foreach $j \sim i : S$ **then** `src` $\rightarrow$ `dst` **else** `src` $\rightarrow e$

where $e \in Q$ is the *escape* state. We recall that, for a configuration with linear topology, in that model, a process inspects the states of the other processes *in-order*. Therefore, it is sufficient to only keep track of the last position that each process has inspected (using the total map $\checkmark : [\![1,n]\!] \rightarrow [\![0,n]\!]$, initially assigned the value 0). It follows that, for every process i, lower positions than $\checkmark(i)$ (in the available range $\sim$) are then also inspected, while higher positions are not yet inspected.[3]

In order to instantiate an abstract domain, we need to handle the fact that a context is a set and does not reflect which processes have been inspected by another given process. We extend the (context-sensitive) views with some extra information. A view is now of the form $(R_0 q_1 R_1 \ldots q_n R_n, \checkmark, \rho)$,

[3] In the case processes do not loop in-order, $\checkmark$ is replaced with a binary relation $R \subseteq [\![1,n]\!] \times [\![1,n]\!]$ on positions, initially empty. When process i inspects process j, the pair (i, j) is added to the relation. We say that i *ticks* j. This can be implemented with a matrix of size $n \times n$ of boolean values and allows us to cover the case where processes inspect each other in a random order.

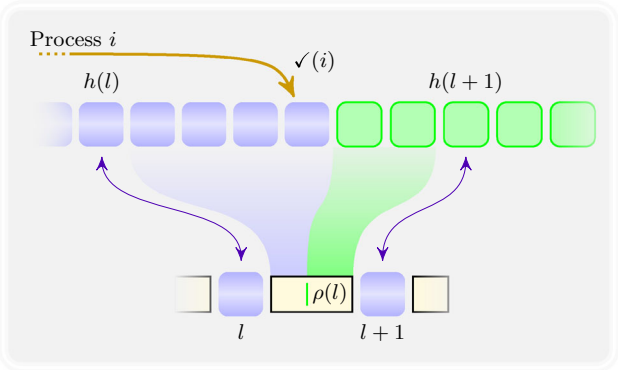


Fig. 13 Projection with non-atomicity. The *blue* states have been inspected by process i, the *green* states have not. The abstraction needs to distinguish for that context which states have been inspected from those that have not

where $(q_1 \ldots q_n, \checkmark)$ is a configuration called the *base*, and $(R_0, \ldots, R_n, \rho)$ 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. 13, for one context only, say R_ℓ (in fact, R_ℓ is the context where $\checkmark(i)$ is projected to). It is obvious that contexts of higher (resp. lower) indices than ℓ contain processes that are not (resp. are) inspected by process i.

In order to use Algorithm 3 with the new abstract domain (i.e., abstraction and concretization), we need in fact only to adjust the notion of projections, the entailment relation and the symbolic post on views. The procedure will then run as in the previous section.

The projection of a configuration into a view is defined in a similar manner as in Sect. 4.1. For $h \in H_n^k, k \leq n$, and a configuration $c = (q_1 \ldots q_n, \checkmark)$, $\Pi_h(c) = (\Pi_h(q_1 \ldots q_n), \checkmark', \rho')$ 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., $(q_1 \ldots q_n, \checkmark) = (q'_1 \ldots q'_n, \checkmark')$,
(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.

Since we do not maintain any order in the contexts, we cannot make a particular view reflect an intermediate state of an in-order for-loop rule. However, recall that views work collectively, so there will be another view distinguishing that intermediate state. Therefore, the symbolic post *spost* is adapted from the previous post operators, with the particularity that it "ticks" each context at once as a block and handles the extra ρ information adequately. Intuitively, process i inspects its "next" context by "ticking" the elements of $\rho(i)$ unless it has to escape. If it then cannot escape, it moves on to the following position and marks the following context as unticked (the new value of $\rho(i)$). Notice that the inspected context might contain process states that would make the transition escape (but not $\rho(i)$). It means that those processes potentially changed state *after* they got ticked. Finally, process i terminates if there is no more context or base element to inspect.

Formally, we fix a view $v = (R_0 b_1 R_1 \ldots b_n R_n, \checkmark, \rho)$ of size n, a position $i \in [\![1,n]\!]$ and a global transition $\delta =$ **if foreach** $j \sim i : S$ **then** $\texttt{src} \to \texttt{dst}$ **else** $\texttt{src} \to e$ from Δ (since the case of a local transition is trivial). We distinguish three types of symbolic $\delta^\#$-move on v by the process in the base at position i: (a) $\delta^\#_{it}(v,i)$ for a loop iteration, (b) $\delta^\#_{esc}(v,i)$ for escaping and (c) $\delta^\#_{term}(v,i)$ for termination. Each type of move is defined only if $b_i = \texttt{src}$ (see example in Fig. 14).

Recall that $\checkmark(i)$ represents the position that process i has last inspected, which is either 0 at the start or always points to an element of the base.

(a) Carefully looking at the indices in the view, process i now inspects the processes of the *first* context that it has not inspected, i.e., the context $R_{\checkmark(i)}$. $\delta^\#_{it}(v,i)$ is defined if both the following two properties are satisfied: (i) $\rho(i) \subseteq S$ and (ii) $\checkmark(i)+1 \sim i, \checkmark(i)+1 \leq n$ and $b_{\checkmark(i)+1} \in S$. It is then obtained from v by only updating $\checkmark(i)$ to $\checkmark(i)+1$ and resetting $\rho(i)$ to $R_{\checkmark(i)+1}$.
(b) $\delta^\#_{esc}(v,i)$ is defined if one of the following two properties is satisfied: (i) $\rho(i) \not\subseteq S$ or (ii) $\checkmark(i)+1 \sim i, \checkmark(i)+1 \leq n$ and $b_{\checkmark(i)+1} \notin S$. It is obtained from v by changing the state of the process i to e and resetting $\checkmark(i)$ to 0 and $\rho(i)$ to $\emptyset$. Intuitively, process i has found a reason to escape.
(c) $\delta^\#_{term}(v,i)$ is defined if $\checkmark(i)+1 \nsim i$ or $\checkmark(i)+1 > n$. It is obtained from v by changing the state of the process i to $\texttt{dst}$ and resetting $\checkmark(i)$ to 0 and $\rho(i)$ to $\emptyset$. Intuitively, process i has reached the end of the iteration and terminates its transition (i.e., moves to its target state).

This concludes how we adapted Algorithm 3 to cope with non-atomically checked global conditions in the presence of contexts.

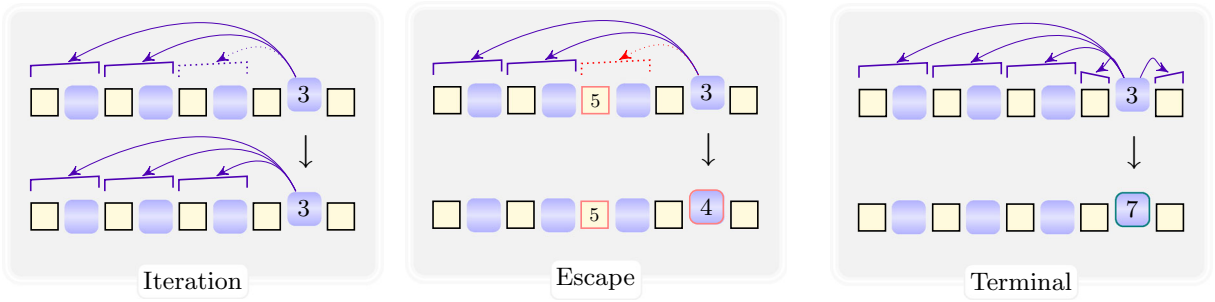


Fig. 14 Non-atomic "ticking" for the global transition **if foreach** $j \neq i : \neg\{1, 2\}$ **then** $3 \rightarrow 7$ **else** $3 \rightarrow 4$. Note that contexts are ticked at once

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, in different settings. The examples cover cache coherence protocols, communication protocols through trees and rings, petri nets, and mutual exclusion protocols.

We report the results in Table 1 running on a 3.1 GHz computer with 4 GB 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 ($|V|$). For the protocols with linear topology, we display the results over two lines: 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 at [37]. In most cases, the method terminates almost immediately demonstrating the efficiency of exploiting 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.

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.

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.

The bulk of the running time of the algorithm lies in the computation of the set $\phi_k^{k+\ell}(V)$ and also the set $\mathcal{R}_k$. An example on which the algorithm fails is the *Kanban* system from [40]. This is a typical case where the cut-off condition is satisfied at high values of k and [40] 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.

Heuristics Recording contexts in views makes the number of generated views much larger. Contexts introduce a necessary precision at the cost of extra computational complexity. It is therefore interesting to use contexts only when necessary.

As explained earlier, to accelerate Algorithm 3, we seed it with the initial set of views $\alpha_k(\mathcal{R}_k \cup \mathcal{R}_{k+1} \cup \mathcal{R}_{k+2})$. According to our experimentation, when adding an extra process to the picture does not lead to the creation of new bases (of size 2), adding yet another one is not likely to change that fact either. In other words, when we observe that the set of views $\alpha_2(\mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4)$ and $\alpha_2(\mathcal{R}_2 \cup \mathcal{R}_3)$ give us the same set of bases, it is then likely the case that all bases of size 2 of the abstract system have been already discovered. Algorithm 3 is then probably close to its fixpoint for $k = 2$ (on line 3). This is a good guess for k.

We devise a heuristic that takes advantage of this observation. In a first phase, we ignore the contexts as in the abstraction from Sect. 2.1, using the above initial set. If the algorithm encounters a base that was not in this initial set, we stop and mark each view that contributed to generate this new member. In a second phase, we run the algorithm anew and also remember the contexts for all the marked views. That way, contexts are inserted only into views that need them.

We have now two alternatives: we can either be proactive and stop the algorithm as soon as a new view is inserted (compared to the initial set), or we can let the inserted views be, in the hope that they would not lead to an erroneous behavior.

In Table 3, we call the first alternative the insertion heuristic: we ignore the contexts and insert the necessary ones as soon as they are needed. This heuristic happens to be very successful in the case of Szymanski's protocol (in its

Table 1 Experimental results

	Protocol	Time	k	$\|V\|$	
Array	Demo (toy example)*	0.005	2	22	✓
		0.006		-	✗
	Burns	0.004	2	34	✓
		0.011		64	✓
	Dijkstra	0.027	2	93	✓
		0.097		222	✓
	Szymanski*	0.307	2	168	✓
		1.982		311	✓
	Szymanski (compact)*	0.006	2	48	✓
		0.557		194	✓
	Szymanski (random)	1.156	2	-	✗
	Bakery (simplified)	0.001	2	7	✓
		0.006		30	✓
	Gribomont-Zenner*	0.328	2	143	✓
		32.112		888	✓
	Simple Barrier*	0.018	2	61	✓
		1.069		253	✓
Multiset	MOSI Coherency	0.01	1	10	✓
	German's Coherency	15.3	6	1890	✓
Petri Net	German (simplified)	0.03	2	43	✓
	BH250	2.85	2	503	✓
	MOESI Coherency	0.01	1	13	✓
	Critical Section	0.01	5	27	✓
	Kanban	?	≥ 20	?	✓
Tree	Percolate	0.05	2	34	✓
	Tree Arbiter	0.7	2	88	✓
	Leader Election	0.1	2	74	✓
Ring	Token Passing	0.01	2	2	✓

✓`Safe`, ✗`Unsafe`
* contexts needed

Table 2 Petri net with inhibitor arcs ($k = 2$)

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	0.001	15	✓
List with Counter Automata	0.002	38	✓

contexts needed

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

non-atomic full version, see Fig. 15). We call the second alternative the context discovery heuristic: 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. The latter 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, and additionally the alternative without heuristics, where contexts are used for all views from the beginning. The time is given in seconds and **it.** represents the number of iteration to terminate.

7 Related work

An extensive amount of work has been devoted to regular model checking, e.g., [22,41]; and in particular augmenting regular model checking with techniques such as widening [17,51], abstraction [18], and acceleration [10]. 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 [23,24,29,35,46]. 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 [13,47] and the work of [45] exploit cut-off properties to check invariants for mutual exclusion protocols. In [15], 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

```
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. 15 Szymanski's protocol (for process i), in its non-atomic version. To the best of our knowledge, our method is the first to verify fully automatically the safety property of this protocol, without the atomicity assumption (using contexts)

model. *Environment abstraction* [19] 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 [8]. It applies the recently introduced method of *monotonic abstraction* [7], which combines regular model checking with abstraction in order to produce systems that have monotonic behaviors with respect to 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. Another method inspired by [7] is [12,36]. It is a generic framework blending [7] with SMT solving, and it is in principe applicable to systems with non-atomic guards.

The method of [33,34] and its reformulated, generic version of [32] 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 [6].

Constant-size cut-offs have been defined for ring networks in [28] where communication is only allowed through token passing. More general communication mechanisms such as guards over local and shared variables are described in [27]. 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 [40] also relies on dynamic detection of cut-off points. The class of systems considered in [40] 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 [31] extends the data structures discussed in [25,26] 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 [45], 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 [6] 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.

8 Conclusion and future work

We have presented a special class of infinite-state systems, namely parameterized systems, and an efficient method to automatically verify their safety property. These systems are organized with topologies such as words, trees, rings, or multisets.

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 have proved that the presented algorithm is complete for a wide class of well quasi-ordered systems and it is precise enough to verify systems with fine-grained transitions (i.e., the non-atomic case).

Based on the method, we have implemented a prototype which performs efficiently on a wide range of benchmarks. Obviously, the bottleneck of the method is when the computation of $\mathcal{R}_k$ is necessary for high values of k or if the set of bad configurations is complex (in terms of size). We plan therefore to integrate the method with tools that can perform efficient forward reachability analysis, like SPIN [38], and to use efficient symbolic encodings for compact representations for the set of views.

Moreover, 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 exist-

ing ones, e.g., thread-modular abstraction [30] and Cartesian abstraction [43]. We have already successfully extended the framework to the case of multi-threaded programs operating on dynamic heap structures [5]. These systems have notoriously complicated behaviors, showing that verification can be carried out through the analysis of only a small number of threads allowed for more efficient algorithms for these systems. We are currently working on extending the models to infinite-state systems and multi-threaded programs running on machines with different memory models. This requires to modify the abstraction, the entailment, and how views are interpreted.

References

1. Abdulla, P.A.: Well (and better) quasi-ordered transition systems. Bull. Symb. 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, vol. 5522 of *LNCS*, pp. 41–56. Springer (2009)
4. Abdulla, P.A., Haziza, F., Holík, L.: Block me if you can!—context-sensitive parameterized verification. In: SAS14, pp. 1–17 (2014)
5. Abdulla, P.A., Haziza, F., Holík, L., Jonsson, B., Rezine, A.: An integrated specification and verification technique for highly concurrent data structures. In: TACAS13, pp. 324–338 (2013)
6. Abdulla, P.A., Haziza, F., Holík, L.: All for the price of few (parameterized verification through view abstraction). In: Proceedings of VMCAI '13, 14th International Conference on Verification, Model Checking, and Abstract Interpretation, vol. 7737 of LNCS, pp. 476–495 (2013)
7. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: TACAS'07, vol. 4424 of LNCS, pp. 721–736. Springer (2007)
8. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Handling parameterized systems with non-atomic global conditions. In: VMCAI08, vol. 4905 of LNCS, pp. 22–36. Springer (2008)
9. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. In: Proc. LICS '93, 8th IEEE Int. Symp. on Logic in Computer Science, pp. 160–170 (1993)
10. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Proc. CONCUR '02, 13th International Conference on Concurrency Theory, vol. 2421 of LNCS, pp. 116–130. Springer (2002)
11. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.-K.: Algorithmic analysis of programs with well quasi-ordered domains. Inf. Comput. **160**(1–2), 109–127 (2000)
12. Alberti, F., Ghilardi, S., Sharygina, N.: A framework for the verification of parameterized infinite-state systems. In: Proceedings of the 29th Italian Conference on Computational Logic, vol. 1195 of CEUR Workshop Proceedings, pp. 303–308. CEUR-WS.org (2014)
13. Arons, T., Pnueli, A., Ruah, S. Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: CAV'01, vol. 2102 of LNCS, pp. 221–234. Springer (2001)
14. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and cartesian abstraction for model checking C programs. STTT **5**(1), 49–58 (2003)
15. Baukus, K., Lakhnech, Y., Stahl, K.: Parameterized verification of a cache coherence protocol: Safety and liveness. In: VMCAI02, vol. 2294 of LNCS, pp 317–330. Springer (2002)
16. Bingham, J.D., Hu, A.J.: Empirically efficient verification for a class of infinite-state systems. In: TACAS'05, vol. 3440 of LNCS, pp. 77–92. Springer (2005)
17. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: CAV'03, vol. 2725 of LNCS, pp. 223–235. Springer (2003)
18. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: CAV'04, vol. 3114 of LNCS, pp. 372–386. Springer (2004)
19. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: VMCAI'06, vol. 3855 of LNCS, pp. 126–141. Springer (2006)
20. Clarke, E.M., Emerson, A.E.: Design and synthesis of synchronization skeletons using branching-time temporal logic. Logic of Programs. Workshop, pp. 52–71. UK, UK, Springer-Verlag, London (1982)
21. Clarke, E.M., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: VMCAI06, vol. 3855 of LNCS, pp. 126–141. Springer (2006)
22. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: CAV'01, vol. 2102 of LNCS. Springer (2001)
23. Delzanno, G.: Automatic verification of cache coherence protocols. In: Emerson, Sistla (eds.) CAV'00, vol. 1855 of LNCS, pp. 53–68. Springer (2000)
24. Delzanno, G.: Verification of consistency protocols via infinite-state symbolic model checking. In: FORTE'00, vol. 183 of IFIP Conference Proceedings, pp. 171–186. Kluwer (2000)
25. Delzanno, G., Raskin, J.-F.: Symbolic representation of upward-closed sets. In: TACAS'00, vol. 1785 of LNCS, pp. 426–441. Springer (2000)
26. Delzanno, G., Raskin, J.-F., Van Begin, L.: Csts (covering sharing trees): Compact data structures for parameterized verification. In: Software Tools for Technology Transfer (2001)
27. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE'00, vol. 1831 of LNCS, pp. 236–254. Springer (2000)
28. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: POPL'95, pp. 85–94 (1995)
29. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS'99. IEEE Computer Society (1999)
30. Flanagan, C., Qadeer, S.: Thread-modular model checking. In: SPIN'03, vol. 2648 of LNCS, pp. 213–224. Springer (2003)
31. Ganty, P., Meuter, C., Delzanno, G., Kalyon, G., Raskin, J.-F., Van Begin, L.: Symbolic data structure for sets of k-uples. Technical Report 570, Université Libre de Bruxelles, Belgium (2007)
32. Ganty, P., Raskin, J.-F., Van Begin, L.: A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In: VMCAI06, vol. 3855 of LNCS, pp. 49–64. Springer (2006)
33. Geeraerts, G., Raskin, J.-F., Van Begin, L.: Expand, enlarge and check... made efficient. In: CAV'05, vol. 3576 of LNCS, pp. 394–407. Springer (2005)
34. Geeraerts, G., Raskin, J.-F., Van Begin, L.: Expand, enlarge and check: new algorithms for the coverability problem of WSTS. J. Comput. Syst. Sci. **72**(1), 180–203 (2006)
35. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J ACM **39**(3), 675–735 (1992)
36. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Towards smt model checking of array-based systems. In: Automated Reasoning, vol. 5195 of LNCS, pp. 67–82. Springer (2008)
37. Haziza, F.: Experiments | parameterized verification through view abstraction. http://www.it.uu.se/research/docs/fm/apv/parametrized/experiments/ (2013)

38. Holzmann, G.J.: The model checker spin. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
39. IEEE Computer Society. IEEE standard for a high performance serial bus. Std 1394–1995 (1996)
40. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV'10, vol. 6174 of LNCS, pp. 645–659. Springer (2010)
41. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theor. Comput. Sci. **256**, 93–112 (2001)
42. Lynch, N.A., Shamir, B.-P.: Distributed algorithms, lecture notes for 6.852, fall 1992. Technical Report MIT/LCS/RSS-20, MIT (1993)
43. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-modular verification is cartesian abstract interpretation. In: ICTAC'06, vol. 4281 of LNCS, pp. 183–197. Springer (2006)
44. Malkis, A., Podelski, A., Rybalchenko, A.: Precise thread-modular verification. In: SAS'07, vol. 4634 of LNCS, pp. 218–232. Springer (2007)
45. Namjoshi, K.S.: Symmetry and completeness in the analysis of parameterized systems. In: VMCAI07, vol. 4349 of LNCS, pp. 299–313. Springer (2007)
46. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infinity)-counter abstraction. In: CAV'02, vol. 2404 of LNCS. Springer (2002)
47. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: TACAS'01, vol. 2031 of LNCS, pp. 82–97. Springer (2001)
48. Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: Proceedings of the 5th Colloquium on International Symposium on Programming, pp. 337–351, London, UK, UK, Springer-Verlag (1982)
49. Szymanski, B.K.: A simple solution to lamport's concurrent programming problem with linear wait. Proceedings of the 2nd International Conference on Supercomputing. ICS '88, pp. 621–626. NY, USA, ACM, New York (1988)
50. Szymanski, B.K.: Mutual exclusion revisited. In: Proc. Fifth Jerusalem Conference on Information Technology, IEEE Computer Society Press, Los Alamitos, CA, pp. 110–117. IEEE Computer Society Press (1990)
51. Touili, T.: Regular Model Checking using Widening Techniques. Electronic Notes in Theoretical Computer Science, 50(4), (2001) Proc. of VEPAS'01
52. Vojnar, T.: Private communication, June (1993)