Automatic WSTS-based Repair and Deadlock Detection of Parameterized Systems

Tom Baumeister
Helmholtz Center for Information Security
Swen Jacobs
jacobs@cispa.de

Helmholtz Center for Information Security
Mouhammad Sakr
University of Luxembourg
Marcus Völp
University of Luxembourg

Research Article

Keywords: Parameterized Model Checking, Parameterized Repair, Well-structured Transition Systems, Deadlock Detection

Posted Date: July 22nd, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4635496/v1

Additional Declarations: No competing interests reported.

Automatic WSTS-based Repair and Deadlock Detection of Parameterized Systems

Tom Baumeister[1], Swen Jacobs[1*], Mouhammad Sakr[2*] and Marcus Völp[2]

[1]CISPA Helmholtz Center for Information Security, Saarbrücken, Germany.
[2]SnT, Luxembourg University, Luxembourg.

*Corresponding author(s). E-mail(s): jacobs@cispa.de; mouhammad.sakr@uni.lu;
Contributing authors: tom.baumeister@cispa.de; marcus.voelp@uni.lu;

Abstract

We present an algorithm for the repair of parameterized systems that can be represented as well-structured transition systems. The repair problem is, for a given process implementation, to find a refinement such that a given safety property is satisfied by the resulting parameterized system, and deadlocks are avoided. Our algorithm uses a parameterized model checker to determine the correctness of candidate solutions and employs a constraint system to rule out candidates. Parameterized systems that fall into our class include disjunctive systems, pairwise rendezvous systems, broadcast protocols, and certain global synchronization protocols. Moreover, we show that parameterized deadlock detection and similar global properties can be decided in EXPTIME for disjunctive systems, and that deadlock detection is in general undecidable for broadcast protocols.

Keywords: Parameterized Model Checking, Parameterized Repair, Well-structured Transition Systems, Deadlock Detection

Acknowledgments: This work was supported by the German Research Foundation (DFG) grant GSP & Co (497132954). Tom Baumeister carried out this work as a member of the Saarbrücken Graduate School of Computer Science. This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference C22/IS/17432184. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0

Author Contribution Statement: All authors made substantial contributions to the conception and design of the work, and to drafting and revising the work. M.S. and S.J. wrote the main manuscript text, except for Section 7, which was written by T.B. T.B., S.J. and M.S. contributed to proofs and proof ideas in the manuscript. Implementation and experimentation was mainly done by M.S., with feedback and minor contributions by the other authors.

1 Introduction

Concurrent systems are hard to get correct, and are therefore a promising application area for formal methods. For systems that are composed of an *arbitrary* number of processes n, methods such as *parameterized* model checking can provide correctness guarantees that hold regardless of n. While the parameterized model checking problem (PMCP) is undecidable even if we restrict systems to uniform finite-state processes [1], there exist several approaches that decide the problem for specific classes of systems and properties [2–10].

However, if parameterized model checking detects a fault in a given system, it does not tell us how to repair it such that the specification is satisfied. To repair the system, the user has to find out which behavior of the system causes the fault, and how it can be corrected. Both tasks may be nontrivial.

For faults in the internal behavior of a process, the approach we propose is based on a similar idea as existing repair approaches [11, 12]: we start with a *non-deterministic* implementation, and restrict non-determinism to obtain a correct implementation. In practice, this non-determinism may have been added by a designer to "propose" possible repairs for a system that is known or suspected to be faulty.

Moreover, only repairing a process internally will not be enough in the presence of concurrency. We need to go beyond existing repair approaches, and also repair the *communication* between processes to ensure that the large number of possible interactions between processes is correct as well. We do so by choosing the right options out of a set of possible interactions, combining the idea above with that of synchronization synthesis [13, 14]. Unlike non-determinism for repairing internal behavior, we are even able to introduce non-determinism for repairing communication automatically.

In addition to guaranteeing safety properties, we aim for an approach that avoids introducing *deadlocks*, which is particularly important for a repair algorithm, since often the easiest way to avoid unsafe states (if these do not include deadlocked states) is to let the system run into a deadlock as quickly as possible.

Regardless of whether faults are fixed in the internal behavior or in the communication of processes, we aim for a parameterized correctness guarantee, i.e., the repaired implementation should be correct in a system with any number of processes. We show how to achieve this by integrating techniques from parameterized model checking into our repair approach.

High-Level Parameterized Repair Algorithm. Figure 1 sketches the basic idea of our parameterized repair algorithm.

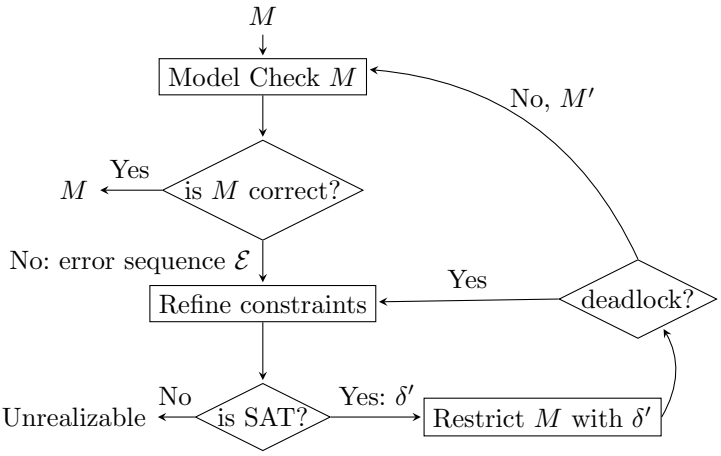

Fig. 1: Parameterized repair of concurrent systems

The algorithm starts with a representation M of the parameterized system, based on non-deterministic models of the components, and checks if error states are reachable for any size of M. If not, the components are already correct. Otherwise, the parameterized model checker returns an error sequence $\mathcal{E}$, i.e., one or more concrete error paths. $\mathcal{E}$ is then encoded into constraints that ensure that any component that satisfies them will avoid the error paths detected so far. A SAT solver is used to find out if any solution still exists, and if so we restrict M to components that avoid previously found errors. To guarantee that this restriction does not introduce deadlocks, the next step is a parameterized deadlock detection. This provides similar information as the model checker, and is used to refine the constraints if deadlocks are reachable. Otherwise, M' is sent to the parameterized model checker for the next iteration.

Research Challenges. Parameterized model checking in general is known to be undecidable, but different decision procedures exist for certain classes of systems, such as guarded protocols with disjunctive guards (also called disjunctive systems) [4], pairwise rendezvous systems [2], broadcast protocols [3], and global synchronization protocols (GSPs) [15]. However, these theoretical solutions are not uniform and do not provide practical algorithms that allow us to extract the information needed for our repair approach. Therefore, the

following challenges need to be overcome to obtain an effective parameterized repair algorithm for a broad class of systems:

C1 The parameterized model checking algorithm should be uniform, and needs to provide information about error paths in the current candidate model that allows us to avoid these paths in future candidate solutions.

C2 We need an effective approach for parameterized deadlock detection, preferably supplying similar information as the model checker.

C3 We need to identify an encoding of the discovered information into constraints such that the repair process is sufficiently flexible[1], and sufficiently efficient to handle examples of interesting size.

Parameterized Repair: an Example. Consider a system with one scheduler (Fig. 2) and an arbitrary number of reader-writer processes (Fig. 3), running concurrently and communicating via pairwise rendezvous, i.e., every send action (e.g. *write*!) needs to synchronize with a receive action (e.g. *write*?) by another process. In this system, multiple processes can be in the *writing* state at the same time, which is an error if they write to a shared resource. We want to repair the system by restricting communication of the scheduler.

According to the idea in Fig. 1, the parameterized model checker searches for reachable errors, and it may find that after two sequential *write*! transitions by different reader-writer processes, they both occupy the *writing* state at the same time. This information is then encoded into constraints on the behavior of processes, which are used to restrict non-determinism and communication, making the given error path impossible. However, in our example all errors could be avoided by simply removing all outgoing transitions of state $q_{A,0}$ of the scheduler. To avoid such repairs, our algorithm uses *initial constraints* (see Section 4) that enforce totality on the transition relation. Another undesirable solution would be the scheduler shown in Fig. 4, because, even though there are outgoing transitions from state $q_{A,0}$, the resulting system will deadlock immediately. This is avoided by checking reachability of deadlocks on candidate solutions. We get a solution that is safe and deadlock-free if we take Fig. 4 and flip pre- and post-state for all transitions.

Contributions. Our main contribution is a counterexample-guided parameterized repair approach, based on model checking of well-structured transition systems (WSTS) [16, 17]. We investigate which information a parameterized model checker needs to provide to guide the search for candidate repairs, and how this information can be encoded into propositional constraints. Our repair algorithm supports internal repairs and repairs of the communication behavior, while systematically avoiding deadlocks in many classes of systems, including disjunctive systems, pairwise rendezvous systems and broadcast protocols. For GSPs we show that naive repair could destroy the well-behavedness property that ensures that they are WSTSs, and identify additional conditions under which well-behavedness is preserved, and therefore parameterized repair is possible.

[1] For example, to allow the user to specify additional properties of the repair, such as keeping certain states reachable.

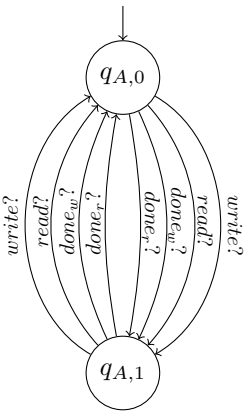

Fig. 2: Scheduler

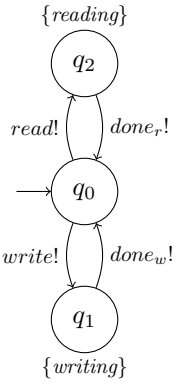

Fig. 3: Reader-Writer

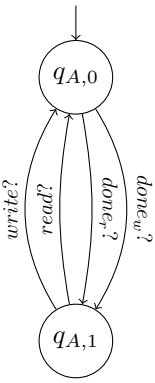

Fig. 4: deadlocked Scheduler

Since existing model checking algorithms for WSTS do not support deadlock detection, our approach has a subprocedure for this problem, which relies on *new theoretical results*: (i) for disjunctive systems, we provide a novel abstract transition system that can be used to check certain properties more efficiently, and in particular improves on the complexity of the best known solution for deadlock detection; (ii) for broadcast protocols and GSPs we prove that deadlock detection is in general undecidable, so approximate methods have to be used. We also discuss approximate methods to detect deadlocks in pairwise rendezvous systems, which can be used as an alternative to the existing approach that has a prohibitive complexity.

Finally, we evaluate an implementation of our algorithm on benchmarks from different application domains, including a distributed lock service and a robot-flocking protocol.

Compared to the short version of this paper [18], we have added full proofs or expanded proof ideas for all theorems and lemmas, have significantly extended our treatment of cases that go beyond reachability properties (Section 5) or beyond disjunctive systems (Section 6), and have added GSPs as a new class of systems that can be repaired (Section 7).

2 System Model

For simplicity, we first restrict our attention to disjunctive systems, other systems will be considered in Section 6. In the following, let Q be a finite set of states.

Processes. A *process template* is a transition system $U = (Q_U, \mathsf{init}_U, \mathcal{G}_U, \delta_U)$, where $Q_U \subseteq Q$ is a finite set of states including the initial state init_U, $\mathcal{G}_U \subseteq \mathcal{P}(Q)$ is a set of guards, and $\delta_U : Q_U \times \mathcal{G}_U \times Q_U$ is a guarded transition relation.

We denote by t_U a transition of U, i.e., $t_U \in \delta_U$, and by $\delta_U(q_U)$ the set of all outgoing transitions of $q_U \in Q_U$. We assume that δ_U is *total*, i.e., for

every $q_U \in Q_U$, $\delta_U(q_U) \neq \emptyset$. Define the *size* of U as $|U| = |Q_U|$. An instance of template U will be called a *U-process*.

Disjunctive Systems. Fix process templates A and B with $Q = Q_A \dot\cup Q_B$, and let $\mathcal{G} = \mathcal{G}_A \cup \mathcal{G}_B$ and $\delta = \delta_A \cup \delta_B$. We consider systems $A\|B^n$, consisting of one A-process and n B-processes in an interleaving parallel composition.[2]

The systems we consider are called "disjunctive" since guards are interpreted disjunctively, i.e., a transition with a guard g is enabled if there *exists* another process that is currently in one of the states in g. Figures 5 and 6 give examples of process templates. An example disjunctive system is $A\|B^n$, where A is the writer and B the reader, and the guards determine which transition can be taken by a process, depending on its own state and the state of other processes in the system. For instance, a reader process in state nr can move to state r if and only if there is a writer process in state nw. Transitions with the trivial guard $g = Q$ are displayed without a guard. We formalize the semantics of disjunctive systems in the following.

Fig. 5: Writer

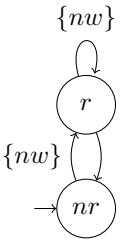

Fig. 6: Reader

Counter System. A *configuration* of a system $A\|B^n$ is a tuple $(q_A, \mathbf{c})$, where $q_A \in Q_A$, and $\mathbf{c} : Q_B \to \mathbb{N}_0$. We identify $\mathbf{c}$ with the vector $(\mathbf{c}(q_0), \ldots, \mathbf{c}(q_{|B|-1})) \in \mathbb{N}_0^{|B|}$, and also use $\mathbf{c}(i)$ to refer to $\mathbf{c}(q_i)$. Intuitively, $\mathbf{c}(i)$ indicates how many processes are in state q_i. We denote by $\mathbf{u}_i$ the unit vector with $\mathbf{u}_i(i) = 1$ and $\mathbf{u}_i(j) = 0$ for $j \neq i$.

Given a configuration $s = (q_A, \mathbf{c})$, we say that the guard g of a local transition $(q_U, g, q'_U) \in \delta_U$ is *satisfied in* s, denoted $s \models_{q_U} g$, if one of the following conditions holds:

(a) $q_U = q_A$, and $\exists q_i \in Q_B$ with $q_i \in g$ and $\mathbf{c}(i) \geq 1$
(A takes the transition, a B-process is in g)

(b) $q_U \neq q_A$, $\mathbf{c}(q_U) \geq 1$, and $q_A \in g$
(B-process takes the transition, A is in g)

(c) $q_U \neq q_A$, $\mathbf{c}(q_U) \geq 1$, and $\exists q_i \in Q_B$ with $q_i \in g$, $q_i \neq q_U$ and $\mathbf{c}(i) \geq 1$
(B-process takes the transition, another B-process in different state in g)

(d) $q_U \neq q_A$, $q_U \in g$, and $\mathbf{c}(q_U) \geq 2$
(B-process takes the transition, another B-process in same state in g)

We say that the local transition (q_U, g, q'_U) is *enabled* in s, and in the following will assume wlog. that each guard $g \in \mathcal{G}$ is a singleton.[3].

[2]The form $A\|B^n$ is only assumed for simplicity of presentation. Our results naturally extend to systems $A_1 \| \ldots \| A_k \| B_1^{n_1} \| \ldots \| B_l^{n_l}$ with an arbitrary number of process templates, each of which can appear either a fixed or a parametric number of times. To see this, first note that any process templates that appear a fixed number of times can be represented as a single big process A. Moreover, we can extend counter systems to count the number of processes in each local state for multiple templates, and the resulting systems will still be WSTSs.

[3]This is not a restriction as any local transition (q_U, g, q'_U) with a guard $g \in \mathcal{G}$ and $\mathbf{g} > 1$ can be split into $\mathbf{g}$ transitions $(q_U, g_1, q'_U), \ldots, (q_U, g_\mathbf{g}, q'_U)$ where for all $i \leq \mathbf{g} : g_i \in g$ is a singleton guard.

Then the *configuration space* of all systems $A\|B^n$, for fixed A, B but arbitrary $n \in \mathbb{N}$, is the transition system $M = (S, S_0, \Delta)$ where:

- $S \subseteq Q_A \times \mathbb{N}_0^{|B|}$ is the set of states,
- $S_0 = \{(init_A, \mathbf{c}) \mid \mathbf{c}(q) = 0 \text{ if } q \neq init_B)\}$ is the set of initial states,
- Δ is the set of transitions $((q_A, \mathbf{c}), (q'_A, \mathbf{c}'))$ s.t. one of the following holds:
  1. $\mathbf{c} = \mathbf{c}' \wedge \exists (q_A, g, q'_A) \in \delta_A : (q_A, \mathbf{c}) \models_{q_A} g$ (transition of A)
  2. $q_A = q'_A \wedge \exists (q_i, g, q_j) \in \delta_B : \mathbf{c}(i) \geq 1 \wedge \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j \wedge (q_A, \mathbf{c}) \models_{q_i} g$ (transition of a B-process)

We will also call M the *counter system* (CS) of A and B, and will call configurations *states* of M, or *global states*.

Let $s, s' \in S$ be states of M, and $U \in \{A, B\}$. For a transition $(s, s') \in \Delta$ we also write $s \to s'$. If the transition is based on the local transition $t_U = (q_U, g, q'_U) \in \delta_U$, we also write $s \xrightarrow{t_U} s'$ or $s \xrightarrow{g} s'$. Let $\Delta^{local}(s) = \{t_U \mid s \xrightarrow{t_U} s'\}$, i.e., the set of all enabled outgoing local transitions from s, and let $\Delta(s, t_U) = s'$ if $s \xrightarrow{t_U} s'$.

Runs. A *path* of a CS is a (finite or infinite) sequence of states $x = s_1, s_2, \ldots$ such that $s_m \to s_{m+1}$ for all $m \in \mathbb{N}$ with $m < |x|$ if the path is finite. A *maximal path* is a path that cannot be extended, and a *run* is a maximal path starting in an initial state. We say that a run is *deadlocked* if it is finite. Note that every run $s_1, s_2, \ldots$ of the CS corresponds to a run of a fixed system $A\|B^n$, i.e., the number of processes does not change during a run. Given a set of error states $ERR \subseteq S$, an *error path* is a finite path that starts in an initial state and ends in ERR.

The Parameterized Repair Problem. Let $M = (S, S_0, \Delta)$ be the CS for process templates $A = (Q_A, \mathsf{init}_A, \mathcal{G}_A, \delta_A)$, $B = (Q_B, \mathsf{init}_B, \mathcal{G}_B, \delta_B)$, and $ERR \subseteq Q_A \times \mathbb{N}_0^{|B|}$ a set of error states. The *parameterized repair problem* is to decide if there exist process templates $A' = (Q_A, \mathsf{init}_A, \mathcal{G}_A, \delta'_A)$ with $\delta'_A \subseteq \delta_A$ and $B' = (Q_B, \mathsf{init}_B, \mathcal{G}_B, \delta'_B)$ with $\delta'_B \subseteq \delta_B$ such that the CS M' for A' and B' does not reach any state in ERR and does not have deadlocked runs.

If they exist, we call $\delta' = \delta'_A \cup \delta'_B$ a *repair* for A and B. We call M' the *restriction* of M to δ', also denoted $Restrict(M, \delta')$. Note that by our assumption that the local transition relations are total, a trivial repair that disables all transitions from some state is not allowed.

In the context of our repair algorithm, a set of transitions $\delta' \subseteq \delta_A \cup \delta_B$ that may or may not be a repair is sometimes called a *candidate repair*, and the restriction of M to δ' a *candidate model*.

Moreover, note that the complexity of the repair problem follows from the complexity of the model checking and deadlock detection problems. That is, if C is the asymptotic worst-case complexity of both model checking and deadlock detection, we get an NC (non-deterministic C) bound for repair, as we can guess the right repair and then verify it in time bounded by C. For disjunctive systems, the coverability problem is PSPACE-hard (which follows from the complexity of the coverability problem for IO-Petri nets [19]), and

we show in Section 3.3 that deadlock detection is in EXPTIME. Therefore C=EXPTIME in this case, and the complexity of repair is NEXPTIME.

3 Parameterized Model Checking of Disjunctive Systems

In this section, we address research challenges **C1** and **C2**: after establishing that CSs can be framed as well-structured transition systems (WSTS) (Section 3.1), we introduce a parameterized model checking algorithm for disjunctive systems that suits our needs (Section 3.2), and finally show how the algorithm can be modified to also check for the reachability of deadlocked states (Section 3.4).

3.1 CSs as WSTS

Well-quasi-order. Given a set of states S, a binary relation $\preceq \subseteq S \times S$ is a *well-quasi-order* (wqo) if $\preceq$ is reflexive, transitive, and if any infinite sequence $s_0, s_1, \ldots \in S^\omega$ contains a pair $s_i \preceq s_j$ with $i < j$.

A subset $R \subseteq S$ is an *antichain* if any two distinct elements of R are incomparable wrt. $\preceq$. Therefore, $\preceq$ is a wqo on S if and only if it is well-founded and has no infinite antichains.

For example, given a CS $M = (S, S_0, \Delta)$ for process templates A, B, the binary relation $\lessapprox \subseteq S \times S$ defined by

$$(q_A, \mathbf{c}) \lessapprox (q'_A, \mathbf{d}) \;\Leftrightarrow\; (q_A = q'_A \wedge \mathbf{c} \lesssim \mathbf{d}),$$

where $\lesssim$ is the component-wise ordering of vectors, is a wqo. The set $\{(q_1, (1,0,0)), (q_1, (0,2,0)), (q_1, (0,1,1)), (q_2, (1,0,0))\}$ is an antichain wrt. $\lessapprox$.

Upward-closed Sets. Let $\preceq$ be a wqo on S. The *upward closure* of a set $R \subseteq S$, denoted $\uparrow R$, is the set $\{s \in S \mid \exists s' \in R : s' \preceq s\}$. We say that R is *upward-closed* if $\uparrow R = R$. If R is upward-closed, then we call $B \subseteq S$ a *basis* of R if $\uparrow B = R$. If $\preceq$ is also antisymmetric, then any basis of R has a unique finite subset of minimal elements. We call this set the *minimal basis* of R, denoted $minBasis(R)$.

Compatibility. Given a CS $M = (S, S_0, \Delta)$, we say that a wqo $\preceq \subseteq S \times S$ is *compatible* with Δ if the following holds: $\forall s, s', r \in S :$ if $s \to s'$ and $s \preceq r$ then $\exists r'$ with $s' \preceq r'$ and $r \to^* r'$. We say $\preceq$ is *strongly compatible* with Δ if the above holds with $r \to r'$ instead of $r \to^* r'$.

WSTS [16]. We say that $(M, \preceq)$ with $M = (S, S_0, \Delta)$ is a *well-structured transition system* if $\preceq$ is a wqo on S that is compatible with Δ. Then the following lemma can be concluded from existing results [6, 17], but we can also give a simple direct proof.

Lemma 1. *Let $M = (S, S_0, \Delta)$ be a CS for process templates A, B, and $\lessapprox$ as defined above. Then $(M, \lessapprox)$ is a WSTS.*

Proof To show that $\lessapprox$ is strongly compatible with Δ, let $s = (q_A, \mathbf{c}), s' = (q'_A, \mathbf{c}'), r = (q_A, \mathbf{d}) \in S$ such that $s \xrightarrow{t_U} s' \in \Delta$ and $s \lessapprox r$. Since the transition t_U is enabled in s, it is also enabled in r and $\exists r' = (q'_A, \mathbf{d}') \in S$ with $r \xrightarrow{t_U} r' \in \Delta$. Then it is easy to see that $s' \lessapprox r'$: either t_U is a transition of A, then we have $\mathbf{c} = \mathbf{c}'$ and $\mathbf{d} = \mathbf{d}'$, or t_U is a transition of B with $t_U = (q_i, g, q_j)$, then $q_A = q'_A$ and $\mathbf{c}' = \mathbf{c} - \mathbf{c}_i + \mathbf{c}_j \lesssim \mathbf{d} - \mathbf{c}_i + \mathbf{c}_j = \mathbf{d}'$. □

Predecessor, Effective *pred*-basis [17]. Let $M = (S, S_0, \Delta)$ be a CS and let $R \subseteq S$. Then the set of *immediate predecessors* of R is

$$pred(R) = \{s \in S \mid \exists r \in R : s \to r\}.$$

A WSTS $(M, \lessapprox)$ *has effective pred-basis* if there exists an algorithm that takes as input any finite set $R \subseteq S$ and returns a finite basis of $\uparrow pred(\uparrow R)$. Note that, since $\lessapprox$ is strongly compatible with Δ, if a set $R \subseteq S$ is upward-closed with respect to $\lessapprox$ then $pred(R)$ is also upward-closed.

Lemma 2. *The wqo $\lessapprox$ is strongly compatible with Δ. Hence, if $R \subseteq S$ is upward-closed with respect to $\lessapprox$ then $pred(R)$ is also upward-closed.*

Proof Suppose $pred(R)$ is not upward-closed, then $\exists s_1, s_2$ with $s_1 \in pred(R), s_2 \notin pred(R)$ and $s_1 \lessapprox s_2$. We have $s_1 \in pred(R)$ then there exists a local transition $t_U \in \delta_U$, $s'_1 \in R$ with $s_1 \xrightarrow{t_U} s'_1 \in \Delta$. However, by definition of the strongly compatible wqo $\lessapprox$, we have t_U is enabled in s_2, $s_2 \xrightarrow{t_U} s'_2 \in \Delta$, and $s'_1 \lessapprox s'_2$. Hence $s'_2 \in R$ (R is upward-closed) and $s_2 \in pred(R)$. Contradiction. □

For backward reachability analysis, we want to compute $pred^*(R)$ as the limit of the sequence $R_0 \subseteq R_1 \subseteq \ldots$ where $R_0 = R$ and $R_{i+1} = R_i \cup pred(R_i)$. Note that if we have strong compatibility and effective pred-basis, we can compute $pred^*(R)$ for any upward-closed set R. If we can furthermore check intersection of upward-closed sets with initial states (which is easy for CSs), then reachability of arbitrary upward-closed sets is decidable.

The following lemma, like Lemma 1, can be considered folklore. We present it here mainly to show *how* we can effectively compute the predecessors, which is an important ingredient of our model checking algorithm.

Lemma 3. *Let $M = (S, S_0, \Delta)$ be a CS for guarded process templates A, B. Then $(M, \lessapprox)$ has effective pred-basis.*

Proof Let $R \subseteq S$ be finite. Since $pred(\uparrow R)$ will be upward-closed with respect to $\lessapprox$, it is sufficient to prove that a *basis* of $pred(\uparrow R)$ can be computed from R. To simplify the following presentation, let $g = \{q_t\}$, $f = ((t = j \wedge \mathbf{c}'(j) = 1) \vee (\mathbf{c}'(t) \geq 1 \wedge \mathbf{c}'(j) = 0))$.

Consider the following set of states:

$$
\begin{aligned}
CBasis = \{(q_A, \mathbf{c}) \in S \mid \exists (q'_A, \mathbf{c}') \in R : \\
[\,(q_A, g, q'_A) \in \delta_A \wedge (\,(q_A, \mathbf{c}) \models_{q_A} g \wedge (\mathbf{c} = \mathbf{c}')\,) \vee \\
(\mathbf{c}'(t) = 0 \wedge \mathbf{c} = \mathbf{c}' + \mathbf{u}_t)\,] \\
\vee [\,(q_i, g, q_j) \in \delta_B \wedge q_A = q'_A \wedge \\
[\,(\,(q_A, \mathbf{c}) \models_{q_i} g \wedge\ (\mathbf{c} = \mathbf{c}' + \mathbf{u}_i - \mathbf{u}_j)\,) \\
\vee (\mathbf{c}'(t) = 0 \wedge \mathbf{c}'(j) \geq 1 \wedge \mathbf{c} = \mathbf{c}' + \mathbf{u}_i - \mathbf{u}_j + \mathbf{u}_t) \\
\vee (f \wedge \mathbf{c} = \mathbf{c}' + \mathbf{u}_i) \\
\vee (\,\mathbf{c}'(t) = 0 \wedge \mathbf{c}'(j) = 0 \wedge \mathbf{c} = \mathbf{c}' + \mathbf{u}_i + \mathbf{u}_t)\ ]\,]\ \}.
\end{aligned}
$$

Clearly, $CBasis \subseteq pred(\uparrow R)$, and $CBasis$ is finite. We claim that also $CBasis \supseteq minBasis(pred(\uparrow R))$. For the purpose of reaching a contradiction, assume $CBasis \not\supseteq minBasis(pred(\uparrow R))$, which implies that there exists a $(q_A, \mathbf{c}) \in (minBasis(pred(\uparrow R)) \cap \neg CBasis)$. Since $(q_A, \mathbf{c}) \notin CBasis$, there exists $(q'_A, \mathbf{c}') \notin R$ with $(q_A, \mathbf{c}) \rightarrow (q'_A, \mathbf{c}')$ and since $(q_A, \mathbf{c}) \in minBasis(pred(\uparrow R))$, there is a $(q'_A, \mathbf{d}') \in R$ with $(q'_A, \mathbf{d}') \lessapprox (q'_A, \mathbf{c}')$. We differentiate between two cases:

- Case 1: Suppose $(q_A, \mathbf{c}) \xrightarrow{t_A} (q'_A, \mathbf{c}')$ with $t_A = (q_A, g, q'_A) \in \delta_A$ and $(q_A, \mathbf{c}) \models_{q_A} g$. Then $\mathbf{c} = \mathbf{c}'$, and by definition of CBasis there exists $(q_A, \mathbf{d}) \in CBasis$ with $[(q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}') \wedge \mathbf{d} = \mathbf{d}' \wedge \mathbf{d}'(t) \geq 1]$ or $[(q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}' + u_t) \wedge \mathbf{d} = \mathbf{d}' + u_t \wedge \mathbf{d}'(t) = 0]$. Furthermore, we have $\mathbf{d}' \lesssim \mathbf{c}'$, which implies $(q_A, \mathbf{d}) \lessapprox (q_A, \mathbf{c})$ with $(q'_A, \mathbf{d}') \in R$. Contradiction.
- Case 2: Suppose $(q_A, \mathbf{c}) \xrightarrow{t_B} (q'_A, \mathbf{c}')$ with $t_B = (q_i, g, q_j) \in \delta_B$ and $(q_A, \mathbf{c}) \models_{q_i} g$. Then $q_A = q'_A \wedge \mathbf{c} = \mathbf{c}' + \mathbf{u}_i - \mathbf{u}_j$. By definition of CBasis there exists $(q_A, \mathbf{d}) \in CBasis$ such that one of the following holds:
  - $(q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}') \ \wedge\ \mathbf{d}' = \mathbf{d} - \mathbf{u}_i + \mathbf{u}_j$
  - $\mathbf{d}'(t) = 0 \wedge \mathbf{d}'(j) \geq 1 \wedge (q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}' + \mathbf{u}_t) \ \wedge\ \mathbf{d}' + \mathbf{u}_t = \mathbf{d} - \mathbf{u}_i + \mathbf{u}_j$
  - $f \wedge (q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}' + \mathbf{u}_j) \ \wedge\ \mathbf{d}' + \mathbf{u}_j = \mathbf{d} - \mathbf{u}_i + \mathbf{u}_j$
  - $\mathbf{d}'(t) = 0 \wedge \mathbf{d}'(j) = 0 \wedge (q_A, \mathbf{d}) \rightarrow (q'_A, \mathbf{d}' + \mathbf{u}_t + \mathbf{u}_j) \wedge \mathbf{d}' + \mathbf{u}_t + \mathbf{u}_j = \mathbf{d} - \mathbf{u}_i + \mathbf{u}_j$

  Furthermore, we have $\mathbf{d}' \lesssim \mathbf{c}'$, which implies that $(q_A, \mathbf{d}) \lessapprox (q_A, \mathbf{c})$ with $(q_A, \mathbf{d}) \in minBasis(pred(\uparrow R))$. Contradiction.

□

3.2 Model Checking Algorithm

Our model checking algorithm is based on the known backwards reachability algorithm for WSTS [16]. We present it in detail to show how it stores intermediate results to return an *error sequence*, from which we derive concrete error paths.

Given a CS M and a finite basis ERR of the set of error states, Algorithm 1 iteratively computes the set of predecessors until it reaches an initial state, or a fixed point. The procedure returns either $True$, i.e. the system is safe, or an *error sequence* $E_0, \ldots, E_k$, where $E_0 = ERR$, $\forall 0 < i < k : E_i = minBasis(\uparrow pred(\uparrow E_{i-1}))$, and $E_k = minBasis(\uparrow pred(\uparrow E_{k-1})) \cap S_0$. That is, every E_i is the minimal basis of the states that can reach ERR in i steps.

Properties of Algorithm 1. Correctness of the algorithm follows from the correctness of the algorithm by Abdulla et al. [16], and from Lemma 3. Termination follows from the fact that a non-terminating run would produce an infinite minimal basis, which is impossible since a minimal basis is an antichain.

Algorithm 1 Parameterized Model Checking

```
1: procedure MC(M,ERR)
2:     tempSet ← ERR, E_0 ← ERR, i ← 1, visited ← ∅
       // A fixed point is reached if visited = tempSet
3:     while tempSet ≠ visited do
4:         visited ← tempSet
5:         E_i ← minBasis(pred(↑E_{i−1}))
6:       //pred is computed as in the proof of Lemma 3
7:         if E_i ∩ S_0 ≠ ∅ then   //intersects with initial states?
8:             return False, {E_0, ..., E_i ∩ S_0}
9:         end if
10:        tempSet ← minBasis(visited ∪ E_i)
11:        i ← i + 1
12:    end while
13:    return True, ∅
14: end procedure
```

While the coverability problem is PSPACE-hard for disjunctive systems (follows from [19]), the exact complexity of the backwards algorithm has not been analyzed to the best of our knowledge. Since disjunctive systems can be expressed as VASSs, we know that the complexity is at most 2EXPTIME [20].

Example. Consider the reader-writer system in Figures 5 and 6. Suppose the error states are all states where the writer is in w while a reader is in r. In other words, the error set of the corresponding CS M is $\uparrow E_0$ where $E_0 = \{(w,(0,1))\}$ and $(0,1)$ means zero reader-processes are in nr and one in r. Note that $\uparrow E_0 = \{(w,(i_0,i_1)) \mid (w,(0,1)) \lessapprox (w,(i_0,i_1))\}$, i.e. all elements with the same w, $i_0 \geq 0$ and $i_1 \geq 1$. If we run Algorithm 1 with the parameters $M, \{(w,(0,1))\}$, we get the following error sequence: $E_0 = \{(w,(0,1))\}$, $E_1 = \{(nw,(0,1))\}$, $E_2 = \{(nw,(1,0))\}$, with $E_2 \cap S_0 \neq \emptyset$, i.e., the error is reachable.

3.3 01-Counter Abstraction

In this section, we present the *01-counter system* (similar to [21]), a finite-state system that represents an abstraction of the unbounded size CS. We then show how it can be used to model check disjunctive systems.

The idea is that we use vectors with counter values from $\{0,1\}$ as abstract states, intuitively distinguishing for every local state only whether it is occupied by at least 1 process, or not. That is, for a state $s = (q_A, \mathbf{c})$ of M, we define the corresponding abstract state as $\alpha(s) = (q_A, \hat{\mathbf{c}})$ with $\hat{\mathbf{c}}(i) = 0$ if $\mathbf{c}(i) = 0$, and $\hat{\mathbf{c}} = 1$ otherwise. Then, a transition between abstract states $\hat{s}, \hat{s}'$ exists iff there exists a transition between concrete states s, s' with $\alpha(s) = \hat{s}$ and $\alpha(s') = \hat{s}'$. We formalize the abstract system in the following, assuming wlog. that δ_B does not contain transitions of the form $(q_i, \{q_i\}, q_j)$, i.e., transitions from q_i that are guarded by q_i.[4]

[4] A system that does not satisfy this assumption can easily be transformed into one that does, with a linear blowup in the number of states, and preserving reachability properties including reachability of deadlocks.

01-**Counter System.** For a given CS M, we define the 01-*Counter System* (01-CS) $\hat{M} = (\hat{S}, \hat{s_0}, \hat{\Delta})$, where:

- $\hat{S} \subseteq Q_A \times \{0,1\}^{|B|}$ is the set of states,
- $\hat{s}_0 = (\mathsf{init}_A, \mathbf{c})$ with $\mathbf{c}(q) = 1$ iff $q = \mathsf{init}_B$ is the initial state,
- $\hat{\Delta}$ is the set of transitions $((q_A, \mathbf{c}), (q'_A, \mathbf{c}'))$ s.t. one of the following holds:
  1. $\mathbf{c} = \mathbf{c}' \wedge \exists (q_A, g, q'_A) \in \delta_A : (q_A, \mathbf{c}) \models_{q_A} g$ (transition of A)
  2. $q_A = q'_A \wedge \exists (q_i, g, q_j) \in \delta_B : (q_A, \mathbf{c}) \models_{q_i} g \wedge \mathbf{c}(i) = 1 \wedge$
     $[(\mathbf{c}(j) = 0 \wedge (\mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j \vee \mathbf{c}' = \mathbf{c} + \mathbf{u}_j)) \vee$
     $(\mathbf{c}(j) = 1 \wedge (\mathbf{c}' = \mathbf{c} - \mathbf{u}_i \vee \mathbf{c}' = \mathbf{c}))]$ (transition of a B-process)

Runs and deadlocks of a 01-CS are defined similarly as for CSs.

Theorem 1. *Given a minimal basis E of error states and let $\hat{E} = \{\alpha(q_a, \mathbf{c}) \mid (q_a, \mathbf{c}) \in E\}$. $\uparrow\hat{E}$ is reachable in the 01-CS $\hat{M}$ iff $\uparrow E$ is reachable in the CS M.*

Proof " $\Rightarrow$ " Suppose $x = s_1, s_2, \ldots, s_e$ is a run of M with $s_e \in \uparrow E$. Note that for any $s \in S$, a transition based on local transition $t_U \in \delta_U$ is enabled iff a transition based on t_U is enabled in the abstract state $\alpha(s)$ of $\hat{M}$. Then it is easy to see that $\hat{x} = \alpha(s_1), \alpha(s_2), \ldots, \alpha(s_e)$ is a run of $\hat{M}$ with $\alpha(s_e) \in \uparrow\hat{E}$.

" $\Leftarrow$ " Now, suppose $\hat{x} = \hat{s}_1, \hat{s}_2, \ldots, \hat{s}_e$ is a run of $\hat{M}$ with $\hat{s}_e = (q_a, \mathbf{c}) \in \uparrow\hat{E}$. Let b be the number of transitions $(\hat{s}_k, \hat{s}_{k+1})$ based on some $t_B = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 1$, i.e., the transitions where we keep a 1 in position i. Furthermore, let $t_1, \ldots, t_{e-1}$ be the sequence of local transitions that $\hat{x}$ is based on. Moreover, we pick a state $(q_a, \mathbf{c}_e)$ from $\uparrow E \cap \{(q_a, \mathbf{c}') \mid \alpha(q_a, \mathbf{c}') = (q_a, \mathbf{c})\}$. Then, we can construct a run of M that reaches a state in $\uparrow E$ as follows: We start in $s_1 = (\mathsf{init}_A, \mathbf{c}_1)$ with $\mathbf{c}_1(\mathsf{init}_B) = 2^{b+m}$ where $m = \Sigma_{0<i<|Q_B|}\mathbf{c}_e(i)$ (Intuitively, we add m-many B-processes to ensure that we can always keep $\mathbf{c}_e(i)$-many processes in each local position of the error state). Then, for every t_k in the sequence do:

- if $t_k \in \delta_A$, we take the same transition once,
- if $t_k = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 0$, we take the same local transition until position i becomes empty, and
- if $t_k = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 1$, we take the same transition local transition $\frac{c - \mathbf{c}_e(i)}{2}$ times, where c is the number of processes that are in position i before. Intuitively, we keep at least $\mathbf{c}_e(i)$-many processes in the given position i, and we move half of the remaining processes to j and keep the other half in i.

By construction, after any of the transitions in $t_1, \ldots, t_{e-1}$, the same positions as in $\hat{x}$ will be occupied in our constructed run $x = s_1, \ldots, s_n$. Additionally, for $s_n = (q_{a_n}, \mathbf{c}_n)$ it holds in every position i that $\mathbf{c}_n(i) \geq \mathbf{c}_e(i)$. Since by construction $q_{a_n} = q_a$, the constructed run ends in an error state $s_n \in \uparrow E$. □

Since the state space of $\hat{M}$ is exponential in $|Q_B|$ and checking reachability in finite-state systems is quadratic in the state space, we obtain the following.

Corollary 1. *Checking reachability of upward-closed sets in disjunctive systems is decidable in EXPTIME (in $|Q_B|$).*

3.4 Deadlock Detection in Disjunctive Systems

The repair of concurrent systems is much harder than fixing monolithic systems. One of the sources of complexity is that a repair might introduce a deadlock, which is usually an unwanted behavior. In this section we show how we can detect deadlocks in disjunctive systems.

Note that a set of deadlocked states is in general not upward-closed under $\lessapprox$ (defined in Section 3.1): let $s = (q_A, \mathbf{c}), r = (q_A, \mathbf{d})$ be global states with $s \lessapprox r$. If s is deadlocked, then $\mathbf{c}(i) = 0$ for every q_i that appears in a guard of an outgoing local transition from s. Now if $\mathbf{d}(i) > 0$ for one of these q_i, then some transition is enabled in r, which is therefore not deadlocked.

Therefore, we introduce the order $\lesssim_0 \subseteq \mathbb{N}_0^{|B|} \times \mathbb{N}_0^{|B|}$ where

$$\mathbf{c} \lesssim_0 \mathbf{d} \;\Leftrightarrow\; (\mathbf{c} \lesssim \mathbf{d} \wedge \forall i \leq |B| : (\mathbf{c}(i) = 0 \Leftrightarrow \mathbf{d}(i) = 0)),$$

and $\lessapprox_0 \subseteq S \times S$ where $(q_A, \mathbf{c}) \lessapprox_0 (q'_A, \mathbf{d}) \;\Leftrightarrow\; (q_A = q'_A \wedge \mathbf{c} \lesssim_0 \mathbf{d})$.

Note that the set of all deadlocked states of a disjunctive system is the upward closure wrt. $\lessapprox_0$ of a finite set S of configurations, also denoted $\uparrow_0 S$. Thus, we could check the reachability of deadlocked states if we could adapt the standard WSTS approach to $\lessapprox_0$.

However, this is not easy, since for our CSs $pred(R)$ will in general not be upward-closed if R is upward-closed. Instead of using $\lessapprox_0$ to define a WSTS, in the following we will directly use the 01-CS to check reachability of upward-closed sets wrt. $\uparrow_0$.

Theorem 2. *The* 01*-CS* $\hat{M}$ *has a deadlocked run if and only if the CS* M *has a deadlocked run.*

Proof Suppose $x = s_1, s_2, \ldots, s_f$ is a deadlocked run of M. Note that for any $s \in S$, a transition based on local transition $t_U \in \delta_U$ is enabled if and only if a transition based on t_U is enabled in the abstract state $\alpha(s)$ of $\hat{M}$. Then it is easy to see that $\hat{x} = \alpha(s_1), \alpha(s_2), \ldots, \alpha(s_f)$ is a deadlocked run of $\hat{M}$.

Now, suppose $\hat{x} = \hat{s}_1, \hat{s}_2, \ldots, \hat{s}_f$ is a deadlocked run of $\hat{M}$. Let b be the number of transitions $(\hat{s}_k, \hat{s}_{k+1})$ based on some $t_B = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 1$, i.e., the transitions where we keep a 1 in position i. Furthermore, let $t_1, \ldots, t_{f-1}$ be the sequence of local transitions that $\hat{x}$ is based on. Then we can construct a deadlocked run of M in the following way: We start in $s_1 = (\mathsf{init}_A, \mathbf{c}_1)$ with $\mathbf{c}_1(\mathsf{init}_B) = 2^b$ and for every t_k in the sequence do:[5]

- if $t_k \in \delta_A$, we take the same transition once,
- if $t_k = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 0$, we take the same local transition until position i becomes empty, and
- if $t_k = (q_i, g, q_j) \in \delta_B$ with $\hat{s}_{k+1}(i) = 1$, we take the same local transition $\frac{c}{2}$ times, where c is the number of processes that are in position i before (i.e., we move half of the processes to j, and keep the other half in i).

[5] Note that a similar, but more involved construction is also possible with $\mathbf{c}_1(\mathsf{init}_B) = b$.

By construction, after any of the transitions in $t_1, \ldots, t_{f-1}$, the same positions as in $\hat{x}$ will be occupied in our constructed run, thus the same transitions are enabled. Therefore, the constructed run ends in a deadlocked state. □

Corollary 2. *Deadlock detection in disjunctive systems is decidable in EXPTIME (in $|Q_B|$).*

Note that we can generalize the result above based on the proof of Thm. 1 and the discussion about $\lessapprox_0$ above. To this end, first note that for any upward-closed set of error states $\uparrow_0 E$, $\hat{E} = \{\alpha(q_a, \mathbf{c}) \mid (q_a, \mathbf{c}) \in E\}$ is its minimal basis wrt. $\lessapprox_0$. Then we get the following.

Corollary 3. *Let $\uparrow_0 E$ be a set of error states and $\hat{E} = \{\alpha(q_a, \boldsymbol{c}) \mid (q_a, \boldsymbol{c}) \in E\}$. Then the 01-CS $\hat{M}$ can reach a state in $\uparrow_0 E$ iff the CS M can reach a state in $\hat{E}$.*

An Algorithm for Deadlock Detection. Now we can modify the model-checking algorithm to detect deadlocks in a 01-CS $\hat{M}$: instead of passing a basis of the set of errors in the parameter ERR, we pass a finite set of deadlocked states $DEAD \subseteq \hat{S}$, and predecessors can directly be computed by *pred*. Thus, an *error sequence* is of the form $E_0, \ldots, E_k$, where $E_0 = DEAD$, $\forall 0 < i < k : E_i = pred(E_{i-1})$, and $E_k = E_{k-1} \cap S_0$.

4 Parameterized Repair Algorithm

Now we can introduce a parameterized repair algorithm according to Figure 1, i.e., that interleaves the backwards model checking algorithm (Algorithm 1) with a forward reachability analysis for the computation of candidate repairs, and uses deadlock detection to detect undesirable repairs.

Forward Reachability Analysis. In the following, for a set $R \subseteq S$, let $Succ(R) = \{s' \in S \mid \exists s \in R : s \to s'\}$. Furthermore, for $s \in S$, let $\Delta^{local}(s, R) = \{t_U \in \delta \mid t_U \in \Delta^{local}(s) \wedge \Delta(s, t_U) \in R\}$.

Given an error sequence $E_0, \ldots, E_k$, let the *reachable error sequence* $\mathcal{RE} = RE_0, \ldots, RE_k$ be defined by $RE_k = E_k$ (which by definition only contains initial states), and $RE_{i-1} = Succ(RE_i) \cap \uparrow E_{i-1}$ for $1 \leq i \leq k$. That is, each RE_i is a set of states that can reach $\uparrow ERR$ in i steps, and are reachable from S_0 in $k - i$ steps. Thus, $\mathcal{RE}$ represents a set of concrete error paths of length k.

Constraint Solving for Candidate Repairs. The generation of candidate repairs is guided by constraints over the local transitions δ as atomic propositions, such that a satisfying assignment of the constraints corresponds to the candidate repair, where only transitions that are assigned `true` remain in δ'. During an execution of the algorithm, these constraints ensure that all error paths discovered so far will be avoided, and include a set of fixed constraints that express additional desired properties of the system, as explained in the following.

Initial Constraints. To avoid the construction of repairs that violate the totality assumption on the transition relations of the process templates, every repair for disjunctive systems has to satisfy the following constraint:

$$TRConstr_{Disj} = \bigwedge_{q_A \in Q_A} \bigvee_{t_A \in \delta_A(q_A)} t_A \wedge \bigwedge_{q_B \in Q_B} \bigvee_{t_B \in \delta_B(q_B)} t_B$$

Informally, $TRConstr_{Disj}$ guarantees that a candidate repair returned by the SAT solver never removes all local transitions of a local state in $Q_A \cup Q_B$. Furthermore a designer can add constraints that are needed to obtain a repair that conforms with their requirements, for example to ensure that certain states remain reachable in the repair (see Section 6 for more examples).

4.1 A Parameterized Repair Algorithm.

Given a CS M, a basis ERR of the error states, and initial Boolean constraints $initConstr$ on the transition relation (including at least $TRConstr_{Disj}$), Algorithm 2 returns either a *repair* δ' or the string *Unrealizable* to denote that no repair exists.

To this end, it first uses the parameterized model checker (Algorithm 1) on M' (initially, $M' = M$) and ERR (Line 4). If error states are reachable, it conducts a forward reachability analysis (Lines 6-7), and uses the reachable error sequence to construct constraints that will avoid all error paths that it represents (Line 8), which are added to any constraints that have been computed before (Line 9). Based on the accumulated constraints, a candidate repair is computed (Line 11), and the resulting candidate model is then checked for deadlocks (Lines 15-16). If deadlocks are detected, the current candidate repair is excluded by adding its negation as a constraint (Line 17), and the SAT solver is asked for a new candidate repair (Lines 18 and 11). If model checking does not reach any error states, then the current candidate repair is returned as a correct repair (Line 20), and if at any point the constraints become unsatisfiable, the algorithm returns "Unrealizable" (Line 13).

4.2 Properties of Algorithm 2.

Theorem 3 (Soundness)**.** *For every repair δ' returned by Algorithm 2:*

- *$Restrict(M, \delta')$ is safe, i.e., $\uparrow ERR$ is not reachable,*
- *the transition relation of $Restrict(M, \delta')$ is total in the first two arguments, and*
- *if $\delta' \neq \delta$ then $Restrict(M, \delta')$ is deadlock-free.*

Proof The parameterized model checker guarantees that the transition relation is safe, i.e., $\uparrow ERR$ is not reachable, and the deadlock checker guarantees that deadlocks are not reachable for any restriction of M. Moreover, the transition relation constraint $TRConstr$ is part of $initConstr$ and guarantees that, for any candidate repair returned by the SAT solver, the transition relation is total. □

Algorithm 2 Parameterized Repair

1: **procedure** ParamRepair(M, ERR, $InitConstr$)
2: $accConstr \leftarrow InitConstr$, $isCorrect \leftarrow False$, $\delta' \leftarrow \delta$, $M' \leftarrow M$
3: **while** $isCorrect = False$ **do**
4: $isCorrect, [E_0, \ldots, E_k] \leftarrow MC(M', ERR)$
5: **if** $isCorrect = False$ **then**
6: $RE_k \leftarrow E_k$ //E_k contains only initial states
7: $RE_{k-1} \leftarrow Succ(RE_k) \cap \uparrow E_{k-1}, \ldots, RE_0 \leftarrow Succ(RE_1) \cap \uparrow E_0$
//for every initial state in RE_k, compute its constraints:
8: $newConstr \leftarrow \bigwedge_{s \in RE_k} BuildConstr(s, [RE_{k-1}, \ldots, RE_0])$
//accumulate iterations' constraints:
9: $accConstr \leftarrow newConstr \wedge accConstr$
//reset deadlock constraints:
10: $ddlockConstr \leftarrow True$
11: $\delta', isSAT \leftarrow SAT(accConstr \wedge ddlockConstr)$
12: **if** $isSAT = False$ **then**
13: **return** $Unrealizable$
14: **end if**
//compute a new candidate using the repair δ':
15: $M' = Restrict(M, \delta')$
//if M reaches a deadlock get a new repair:
16: **if** $HasDeadlock(M')$ **then**
17: $ddlockConstr \leftarrow \neg\delta' \wedge ddlockConstr$
18: jump to line 11
19: **end if**
20: **else return** δ' //a repair is found!
21: **end if**
22: **end while**
23: **end procedure**

1: **procedure** BuildConstr(State s, $\mathcal{RE}$) //$\mathcal{RE} = [RE_{k-1}, \ldots, RE_0]$
//s is a state, $\mathcal{RE}[1:]$ is a list obtained by removing the first element from $\mathcal{RE}$:
2: **if** $\mathcal{RE}[1:]$ is empty **then**
//if $t_U \in \Delta^{local}(s)$ leads directly to error set, delete it:
3: **return** $\bigwedge_{t_U \in \Delta^{local}(s, \mathcal{RE}[0])} \neg t_U$
4: **else**
//else either delete t_U or all outgoing transitions of the target state of t_U recursively:
5: **return** $\bigwedge_{t_U \in \Delta^{local}(s, \mathcal{RE}[0])} (\neg t_U \vee$
$BuildConstr(\Delta(s, t_U), \mathcal{RE}[1:]))$
6: **end if**
7: **end procedure**

Theorem 4 (Completeness)**.** *If Algorithm 2 returns "Unrealizable", then the parameterized system has no repair.*

Proof Algorithm 2 returns "Unrealizable" if $accConstr \wedge initConstr$ has become unsatisfiable. We consider an arbitrary $\delta' \subseteq \delta$ and show that it cannot be a repair. Note that for the given run of the algorithm, there is an iteration i of the loop such that δ', seen as an assignment of truth values to atomic propositions δ, was a satisfying assignment of $accConstr \wedge initConstr$ up to this point, and is not anymore after this iteration.

If $i = 0$, i.e., δ' was never a satisfying assignment, then δ' does not satisfy $initConstr$ and can clearly not be a repair. If $i > 0$, then δ' is a satisfying assignment

for *initConstr* and all constraints added before round i, but not for the constraints $\bigwedge_{s \in RE_k} BuildConstr(s, [RE_{k-1}, \ldots, RE_0]) \wedge ddlockConstr$ added in this iteration of the loop, based on a reachable error sequence $\mathcal{RE} = RE_k, \ldots, RE_0$ and any reachable deadlocks in the resulting restriction(s). Then either δ' is explicitly excluded due to deadlocks in Line 17, or by construction of *BuildConstr* we know that we can construct out of δ' and $\mathcal{RE}$ a concrete error path in $Restrict(M, \delta')$. Therefore, δ' can not be a repair of M. □

Theorem 5 (Termination)**.** *Algorithm 2 always terminates.*

Proof For a CS based on A and B, the number of possible repairs is bounded by $2^{|\delta|}$. In every iteration of the algorithm, either the algorithm terminates, or it adds constraints that exclude at least the repair that is currently under consideration. Therefore, the algorithm will always terminate. □

Local Witnesses are Upward-closed. We show another property of our algorithm: even though for the reachable error sequence $\mathcal{RE}$ we do not consider the upward closure, the error paths we discover are in a sense upward-closed. This implies that an $\mathcal{RE}$ of length k represents *all possible error paths* of length k. We formalize this in the following.

Given a reachable error sequence $\mathcal{RE} = RE_k, \ldots, RE_0$, we denote by $\mathcal{UE}$ the sequence $\uparrow RE_k, \ldots, \uparrow RE_0$. Furthermore, let a *local witness* of $\mathcal{RE}$ be a sequence $\mathcal{T}_{\mathcal{RE}} = t_{U_k} \ldots t_{U_1}$ where for all $i \in \{1, \ldots, k\}$ there exist $s \in RE_i, s' \in RE_{i-1}$ with $s \xrightarrow{t_{U_i}} s'$. We define similarly the local witness $\mathcal{T}_{\mathcal{UE}}$ of $\mathcal{UE}$.

Lemma 4. *Let $\mathcal{RE}$ be a reachable error sequence. Then every local witness $\mathcal{T}_{\mathcal{UE}}$ of $\mathcal{UE}$ is also a local witness of $\mathcal{RE}$.*

Proof Let $\mathcal{T}_{\mathcal{UE}} = t_{U_k} \ldots t_{U_1}$. Then there exist $s_k \in \uparrow E_k = \uparrow RE_k$, $s_{k-1} \in \uparrow RE_{k-1}, \ldots$, $s_0 \in \uparrow RE_0$ such that $s_k \xrightarrow{t_{U_k}} s_{k-1} \xrightarrow{t_{U_{k-1}}} \ldots\ldots \xrightarrow{t_{U_2}} s_1 \xrightarrow{t_{U_1}} s_0$. Let $s_0 = (q_A^0, \mathbf{d}^0)$, and let $t_{U_1} = (q_{U_{i_1}}, \{q_{t_1}\}, q_{U_{j_1}})$. Then, by construction of $\mathcal{E}$, there exists $(q_A^0, \mathbf{c}^0) \in E_0, (q_A^1, \mathbf{c}^1) \in E_1$ with $(q_A^0, \mathbf{c}^0) \lessapprox (q_A^0, \mathbf{d}^0)$ and $(q_A^1, \mathbf{c}^1) \xrightarrow{t_{U_1}} (q_A^0, \mathbf{c}^0)$ or $(q_A^1, \mathbf{c}^1) \xrightarrow{t_{U_1}} (q_A^0, \mathbf{c}^0 + \mathbf{u}_{t_1})$, hence t_{U_1} is enabled in $(q_A^1, \mathbf{c}^1)$. Using the same argument we can compute $(q_A^2, \mathbf{c}^2) \in E_2, (q_A^3, \mathbf{c}^3) \in E_3, \ldots$ until we reach the state $(q_A^k, \mathbf{c}^k) \in E_k$ where t_{U_k} is enabled. Therefore we have the sequence $s_k^R \xrightarrow{t_{U_k}} s_{k-1}^R \xrightarrow{t_{U_{k-1}}} \ldots\ldots \xrightarrow{t_{U_2}} s_1^R \xrightarrow{t_{U_1}} s_0^R$ with $s_k^R = (q_A^k, \mathbf{c}^k) \in RE_k = E_k$ and for all $i < k$ we have $s_i^R \in RE_i$, as they are reachable from $s_k^R \in RE_k$ and $(q_A^i, \mathbf{c}^i) \lessapprox s_i^R$ which guarantees that t_{U_i} is enabled in s_i^R. □

Complexity of the Algorithm. As noted in Section 2, the complexity of the repair problem is NC if the combined complexity of coverability checking and deadlock detection is C. For the worst-case complexity of Algorithm 2, this non-determinism may be replaced by an additional exponential, as the number of candidate repairs is at most exponential in the input.

In practice, the hope is that the algorithm will quickly converge to a correct solution, as its search for candidate repairs is guided by the constraints that encode actual errors observed in the candidate repairs. This is also what we observe in our experiments in Section 8, where the number of iterations seems to grow linearly in the number of local states of the system..

What can be done if a repair doesn't exist? If Algorithm 2 returns "Unrealizable", then there is no repair for the given input. To still obtain a repair, a designer can add more non-determinism and/or allow for more communication between processes, and then run the algorithm again on the new instance of the system. Moreover, unlike in monolithic systems, even if the result is "Unrealizable", it may still be possible to obtain a solution that is good enough in practice. For instance, we can change our algorithm slightly as follows: When the SAT solver returns "UNSAT" after adding the constraints for an error sequence, instead of terminating we can continue computing the error sequence until a fixed point is reached. Then, we can determine the minimal number of processes m_e that is needed for the last candidate repair to reach an error, and conclude that this candidate is safe for any system up to size $m_e - 1$.

5 Beyond Reachability

Algorithm 2 can also be used for repair with respect to more general safety properties, based on the automata-theoretic approach to model checking. We assume that the reader is familiar with finite-state automaton and with the automata-theoretic approach to model checking.

Checking Safety Properties. Let $M = (S, S_0, \Delta)$ be a CS of process templates A and B, let φ be a safety property defined as a regular set of words over the states of A, and let $\mathcal{A} = (Q^{\mathcal{A}}, q_0^{\mathcal{A}}, Q_A, \delta^{\mathcal{A}}, \mathcal{F})$ be the automaton that accepts all words over Q_A that violate φ. To repair M, the composition $M \times \mathcal{A}$ and the set of error states $ERR = \{((q_A, \mathbf{c}), q_{\mathcal{F}}^{\mathcal{A}}) \mid (q_A, \mathbf{c}) \in S \wedge q_{\mathcal{F}}^{\mathcal{A}} \in \mathcal{F}\}$ can be given as inputs to the procedure $ParamRepair$.

Corollary 1. *Let $\lesssim_{\mathcal{A}} \subseteq (M \times \mathcal{A}) \times (M \times \mathcal{A})$ be a binary relation defined by:*

$$((q_A, \mathbf{c}), q^{\mathcal{A}}) \lesssim_{\mathcal{A}} ((q'_A, \mathbf{c}'), q'^{\mathcal{A}}) \Leftrightarrow \mathbf{c} \lesssim \mathbf{c}' \wedge q_A = q'_A \wedge q^{\mathcal{A}} = q'^{\mathcal{A}}$$

then $((M \times \mathcal{A}), \lesssim_{\mathcal{A}})$ is a WSTS with effective pred-basis.

Similarly, the algorithm can be used for any safety property $\varphi(A, B^{(k)})$ defined as a regular set of words over the states of A, and of k B-processes. To this end, we consider the composition $M \times B^k \times \mathcal{A}$ with $M = (S, S_0, \Delta)$, $B = (Q_B, \mathsf{init}_B, \mathcal{G}_B, \delta_B)$, and $\mathcal{A} = (Q^{\mathcal{A}}, q_0^{\mathcal{A}}, Q_A \times Q_B^k, \delta^{\mathcal{A}}, \mathcal{F})$ is the automaton

that reads states of $A \times B^k$ as actions and accepts all words that violate the property.[6]

Example. Consider again the simple reader-writer system in Figures 5 and 6, and assume that instead of local transition $(nr, \{nw\}, r)$ we have an unguarded transition (nr, Q, r). We want to repair the system with respect to the safety property $\varphi = G[(w \wedge nr_1) \implies (nr_1 W nw)]$ where G, W are the temporal operators *always* and *weak until*, respectively. Figure 7 depicts the automaton equivalent to $\neg\varphi$. To repair the system we first need to split the guards as mentioned in Section 2, i.e., (nr, Q, r) is split into $(nr, \{nr\}, r), (nr, \{r\}, r), (nr, \{nw\}, r)$, and $(nr, \{w\}, r)$. Then we consider the composition $\mathcal{C} = M \times B \times \mathcal{A}$ and we run Algorithm 2 on the parameters $\mathcal{C}$, $((-, -, (*, *), q_2^{\mathcal{A}}))$ (where $(-, -)$ means any writer state and any reader state, and $*$ means 0 or 1), and $TRConstr_{Disj}$. The model checker in Line 4 may

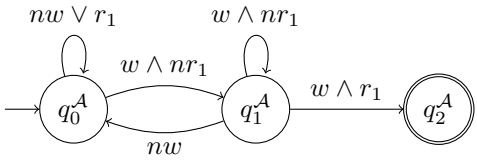

Fig. 7: Automaton for $\neg\varphi$

return the following error sequences, where we only consider states that didn't occur before:
$E_0 = \{((-, -, (*, *)), q_2^{\mathcal{A}})\}$,
$E_1 = \{((w, r_1, (0, 0)), q_1^{\mathcal{A}})\}$,
$E_2 = \{((w, nr_1, (0, 0)), q_0^{\mathcal{A}}), ((w, nr_1, (0, 1)), q_0^{\mathcal{A}}), ((w, nr_1, (1, 0)), q_0^{\mathcal{A}})\}$,
$E_3 = \{((nw, nr_1, (0, 0)), q_0^{\mathcal{A}}), ((nw, nr_1, (0, 1)), q_0^{\mathcal{A}}),$
$((w, r_1, (0, 0)), q_0^{\mathcal{A}}), ((w, r_1, (0, 1)), q_0^{\mathcal{A}}), ((w, r_1, (1, 0)), q_0^{\mathcal{A}})\}$

In Line 11 we find out that the error sequence can be avoided if we remove the transitions $\{(nr, \{nr\}, r), (nr, \{r\}, r), (nr, \{w\}, r)\}$. Another call to the model checker in Line 4 finally assures that the new system is safe. Note that some states were omitted from error sequences in order to keep the presentation simple.

A Note on Deadlock Detection. We want to point out that the absence of deadlocks, even though it is a safety property, does not fall into the class of properties considered in this section. The reason is that a deadlock is a state of the system where *all* processes together satisfy a property (not having any enabled transitions), whereas the safety properties considered here are such that they can be evaluated by projecting the runs of the parameterized system to a fixed number of processes.

[6] By symmetry, property $\varphi(A, B^{(k)})$ can be violated by these k explicitly modeled processes iff it can be violated by any combination of k processes in the system.

In the literature, properties like deadlock detection have also been called *target reachability*, whereas the properties we consider in this section are variants of *coverability* or *control state reachability* [22]. For some system models, it is known that target reachability or deadlock detection are more difficult than coverability, e.g., reconfigurable broadcast networks [23]. We will see in the next section that this also holds for PR systems and broadcast protocols.

6 Repair of Synchronizing Systems

Algorithm 2 is not restricted to disjunctive systems. In principle, it can be used for any system that can be modeled as a WSTS with effective *pred*-basis, as long as we can construct the transition relation constraint ($TRConstr$) for the corresponding system. We have extended Algorithm 2 to other systems that can be framed as WSTS, in particular pairwise rendezvous systems (PR) [2] and systems based on broadcasts (BC) [3]. In these systems, a transition is taken *synchronously* by two processes or by all processes in the system, respectively.

Both types of systems are known to be WSTS, and there are two remaining challenges:

1. how to find suitable constraints to determine a restriction δ', and
2. how to exclude deadlocks.

The first is relatively easy, but the constraints become more complicated because we now have synchronous transitions of multiple processes. Deadlock detection is decidable for pairwise systems, but the best known method is by reduction to reachability in VASS [2], which has recently been shown to be Ackermann-complete [24]. For broadcast protocols we can show that the situation is even worse, i.e., the deadlock detection problem is undecidable.

In the following, we formalize these two classes of systems with their respective transition relation constraints. Then, we discuss deadlock detection in these systems, including new results for deadlock detection in broadcast protocols. Finally, we show how to encode error sequences in the procedure BSC (short-hand for *BuildSynchronousConstraint*) that is used for transition relations with synchronous actions, thus extending our repair algorithm to these systems.

6.1 Synchronizing Process Template

Since these two classes of systems require processes to synchronize on certain actions, we first introduce a different notion of process templates.

Processes. A *synchronizing process template* is a transition system $U = (Q_U, \mathsf{init}_U, \Sigma, \delta_U)$ with

- $Q_U \subseteq Q$ is a finite set of states including the initial state init_U,
- $\Sigma = \Sigma_{sync} \times \{?, !, ??, !!\} \cup \{\tau\}$ where Σ_{sync} is a set of synchronizing actions, and τ is an internal action,
- $\delta_U : Q_U \times \Sigma \times Q_U$ is a transition relation.

Synchronizing actions like $(a, !)$ or $(b, ?)$ are shortened to $a!$ and $b?$. Actions of the form $a!, a!!$ are called *sending actions*, and actions of the form $a?, a??$ *receiving actions*. A transition (q, σ, q') is a *sending transition* if σ is a sending action, and a *receiving transition* otherwise.

All processes mentioned in the following are based on a synchronizing process template. We will define global systems based on either PR or BC synchronization in the following subsections.

6.2 Pairwise Rendezvous (PR) Systems

A PR system [2] consists of a finite number of processes running concurrently. As before, we consider systems of the form $A \| B^n$. The semantics is interleaving, except for actions where two processes synchronize. That is, at every time step, either exactly one process makes an internal transition τ, or exactly two processes synchronize on a single action $a \in \Sigma_{sync}$. For a synchronizing action $a \in \Sigma_{sync}$, the initiator process locally executes the $a!$ action and the recipient process locally executes the $a?$ action.

Similar to disjunctive systems, the configuration space of all systems $A \| B^n$, for fixed A, B but arbitrary $n \in \mathbb{N}$, is the CS $M^{PR} = (S, S_0, \Delta)$, where:

- $S \subseteq Q_A \times \mathbb{N}_0^{|B|}$ is the set of states,
- $S_0 = \{(init_A, \mathbf{c}) \mid \forall q_B \in Q_B : \mathbf{c}(q_B) = 0 \text{ if } q_B \neq init_B)\}$ is the set of initial states,
- Δ is the set of transitions $((q_A, \mathbf{c}), (q'_A, \mathbf{c}'))$ such that one of the following holds:
  1. $(q_A, \tau, q'_A) \in \delta_A \wedge \mathbf{c} = \mathbf{c}'$ (internal transition A)
  2. $\exists q_i, q_j : (q_i, \tau, q_j) \in \delta_B \wedge c(i) \geq 1 \wedge \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j \wedge q_A = q'_A$ (internal transition B)
  3. $a \in \Sigma_{sync} \wedge (q_A, a!, q'_A) \in \delta_A \wedge \exists q_i, q_j : (q_i, a?, q_j) \in \delta_B \wedge c(i) \geq 1, \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j$ (synchronizing transition A, B)
  4. $a \in \Sigma_{sync} \wedge (q_A, a?, q'_A) \in \delta_A \wedge \exists q_i, q_j : (q_i, a!, q_j) \in \delta_B \wedge c(i) \geq 1, \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j$ (synchronizing transition B, A)
  5. $\exists q_i, q_j : (q_i, a!, q_j) \in \delta_B \wedge \exists q_l, q_m : (q_l, a?, q_m) \in \delta_B \wedge c(i) \geq 1 \wedge c(l) \geq 1 \wedge \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j - \mathbf{u}_l + \mathbf{u}_m$ (synchronizing transition B, B)

The following result can be considered folklore, a proof can be found in the survey by Bloem et al. [25].

Lemma 5. *Let $M^{PR} = (S, S_0, \Delta)$ be a CS for process templates A, B with PR synchronization. Then $(M^{PR}, \lesssim)$ is a WSTS with effective pred-basis.*

In particular, for PR systems the worst-case complexity of Algorithm 1 is known to be doubly exponential [20].

Example (Reader-Writer). Consider again the PR system that consists of one scheduler (Figure 2) and a parameterized number of instances of the reader-writer process template (Figure 3). Since the scheduler has transitions for all receive actions from every state, we cannot guarantee that, at any

moment, there is at most one reader-writer process in the *writing* state q_1. We will in Section 6.5 how our algorithm can automatically repair this protocol.

Initial Constraints. The constraint $TRConstr_{PR}$, ensuring that not all local transitions from any given local state are removed, is constructed in a similar way as $TRConstr_{Disj}$.

Furthermore, the user may want to ensure that in the returned repair, either (a) for all $a \in \Sigma_{sync}$, $t_{a!}$ is deleted if and only if all $t_{a?}$ are deleted, or (b) that synchronized actions are *deterministic*, i.e., for every state q_U and every synchronized action a, there is exactly one transition on $a?$ from q_U. We give *user constraints* that ensure such behavior.

Denote by $t_{a?}, t_{a!}$ synchronous local transitions based on an action a. Then, the constraint ensuring property (a) is

$$\bigwedge_{a \in \Sigma_{sync}} [(t_{a!} \wedge (\bigvee_{t_{a?} \in \delta} t_{a?})) \vee (\neg t_{a!} \wedge (\bigwedge_{t_{a?} \in \delta} \neg t_{a?}))]$$

To encode property (b), for $U \in \{A, B\}$ and $a \in \Sigma_{sync}$, let $\{t_{q_U}^{a_?^1}, \ldots, t_{q_U}^{a_?^m}\}$ be the set of all $a?$ transitions from state $q_U \in Q_U$. Additionally, let $one(t_{q_U}^{a_?}) = \bigvee_{j \in \{1,\ldots,m\}} [t_{q_U}^{a_?^j} \bigwedge_{l \neq j} \neg t_{q_U}^{a_?^l}]$. Then, (b) is ensured by

$$\bigwedge_{a \in \Sigma_{sync}} \bigwedge_{q_U \in Q} one(t_{q_U}^{a_?})$$

6.3 Broadcast Protocols

In broadcast protocols, the semantics is interleaving, except for actions where all processes synchronize, with one process "broadcasting" a message to all other processes. Such a broadcast synchronization can be used to select a special process while the system is running, so we can restrict our model to systems of the form B^n, i.e., without a distinguished process A.

Formally, at every time step either exactly one process makes an internal transition τ, or all processes synchronize on a single action $a \in \Sigma_{sync}$. For a synchronized action $a \in \Sigma_{sync}$, we say that the initiator process executes the $a!!$ action and all recipient processes execute the $a??$ action. For every action $a \in \Sigma_{sync}$ and every state $q_B \in Q_B$, there exists a state $q'_B \in Q_B$ such that $(q_B, a??, q'_B) \in \delta_B$. Like Esparza et al. [3], we assume w.l.o.g. that the transitions of recipients are deterministic for any given action, which implies that the effect of a broadcast message on the recipients can be modeled by multiplication of a *broadcast matrix*[7], i.e., a matrix where every column contains one 1 and all other entries are 0. We denote by $\mathbf{M}_a$ the broadcast matrix for action a.

Then, the configuration space of all systems B^n, for fixed broadcast protocol B but arbitrary $n \in \mathbb{N}$, is the CS $M^{BC} = (S, S_0, \Delta)$ where:

[7] Also known as *transfer matrix* in the literature.

- $S \subseteq \mathbb{N}_0^{|B|}$ is the set of states,
- $S_0 = \{\mathbf{c} \mid \forall q_B \in Q_B : \mathbf{c}(q_B) = 0 \text{ iff } q_B \neq init_B)\}$ is the set of initial states,
- Δ is the set of transitions $(\mathbf{c}, \mathbf{c}')$ such that one of the following holds:
  1. $\exists q_i, q_j \in Q_B : (q_i, \tau, q_j) \in \delta_B \wedge \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j$ (internal transition)
  2. $\exists a \in \Sigma_{sync} : \mathbf{c}' = \mathbf{M}_a \cdot (\mathbf{c} - \mathbf{u}_i) + \mathbf{u}_j$ (broadcast)

Like for disjunctive and PR systems, the configuration space of a broadcast protocol is well-structured:

Lemma 6. *[3] Let $M^{BC} = (S, S_0, \Delta)$ be a CS for process template B with BC synchronization. Then $(M^{BC}, \lesssim)$ is a WSTS with effective pred-basis.*

In particular, the worst-case complexity of Algorithm 1 is Ackermannian for broadcast protocols [26].

Example (MESI Protocol). Consider the cache coherence protocol MESI in Figure 8, where:

- M stands for *modified* and indicates that the cache has been changed.
- E stands for *exclusive* and indicates that no other process seizes this cache line.
- S stands for *shared* and indicates that more than one process hold this cache line.
- I stands for *invalid* and indicates that the cache's content is not guaranteed to be valid as it might have been changed by some process.

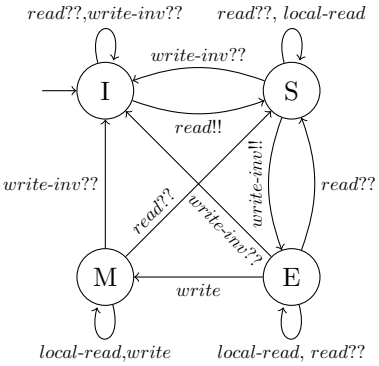

Fig. 8: MESI protocol

Initially all processes are in I and let a state vector be as follows: (M, E, S, I). An important property for MESI protocol is that a cache line should not be modified by one process (in state M) and in shared state for another process (in state S), but the given protocol does not guarantee this. We will see later how our algorithm can automatically repair the protocol.

Initial Constraints. $TRConstr_{BC}$ is defined similarly to $TRConstr_{PR}$, except that we do not have process A and can omit transitions of A. We denote by $t_{a??}, t_{a!!}$ synchronous transitions based on an action a. To ensure that in any repair and for all $a \in \Sigma_{sync}$, $t_{a!!}$ is deleted if and only if all $t_{a??}$ are deleted, the designer can use the following constraint:

$$\bigwedge_{a \in \Sigma_{sync}} [(t_{a!!} \wedge (\bigwedge_{q_B \in Q_B} \bigvee_{t_{a??} \in \delta_B(q_B)} t_{a??})) \vee (\neg t_{a!!} \wedge (\bigwedge_{t_{a??} \in \delta_B} \neg t_{a??}))]$$

It should be noted that our previous paper [18] had slightly different initial constraints. This was because we made an assumption that if from a given local state there was no receiving transition on a given global action, then there must be a (hidden) receiving transition that is a self-loop on that state. The initial constraints given here work without this assumption.

6.4 Deadlock Detection

Deadlock Detection for PR Systems. German and Sistla [2] have shown that deadlock detection in PR systems can be reduced to reachability in VASS, and vice versa. Thus, at least a rudimentary version of repair including deadlock detection is possible, where the deadlock detection only excludes exactly the current candidate repair, but may not be able to provide constraints on candidates that may be considered in the future.

However, the reachability problem in VASS has recently been shown to be Ackermann-complete [24], so a practical solution is unlikely to be based on an exact approach. Therefore, we will discuss approximate methods for deadlock detection after looking at the problem for broadcast protocols.

Deadlock Detection for Broadcast Protocols. Somewhat surprisingly, it was thus far unknown whether deadlock detection is decidable for broadcast protocols. We can show that the answer is negative.

Theorem 6. *Deadlock detection in broadcast protocols is undecidable.*

The main ingredient of the proof is a reduction from the reachability problem of affine VASS to the deadlock detection problem in broadcast protocols. To formally state it, let an *affine VASS* be a tuple $\mathcal{V} = (d, Q, T)$, where $d \geq 1$ is the dimension of $\mathcal{V}$, Q a finite set of control states, and $T \subseteq Q \times \mathbb{N}^{d \times d} \times \mathbb{Z}^d \times Q$ is a finite set of transitions. For every transition $t = (q, A, \mathbf{v}_t, q')$, let $src(t) := q, M(t) := A, \Delta(t) := \mathbf{v}_t$ and $tgt(t) := q'$. A *configuration* of $\mathcal{V}$ is a pair $(q, \mathbf{v}) \in Q \times \mathbb{N}^d$. For all $t \in T$, we write $(q, \mathbf{v}) \xrightarrow{t} (q', \mathbf{v}')$ if $src(t) = q, tgt(t) = q'$, $\mathbf{v}' = M(t) \cdot \mathbf{v} + \Delta(t)$, and $\mathbf{v}' \in \mathbb{N}^d$. It can easily be seen that, for any given broadcast protocol, its CS can be expressed as an affine VASS.

The *reachability problem* in affine VASS takes as input an affine VASS $\mathcal{V}$ and two configurations $(q_1, \mathbf{v}_1)$ and $(q_2, \mathbf{v}_2)$, and asks to decide whether $(q_1, \mathbf{v}_1) \rightarrow^*$

$(q_2, \mathbf{v}_2)$ in $\mathcal{V}$. It has recently be shown that this problem is undecidable, even if matrices $M(t)$ are restricted to broadcast matrices [27].

Lemma 7. *There is a polynomial-time reduction from the reachability problem of affine VASS with broadcast matrices to the deadlock detection problem in broadcast protocols.*

Proof We modify the construction from the proofs of Theorems 3.17 and 3.18 from German and Sistla [2], using affine VASS instead of VASS and broadcast protocols instead of PR systems.

Starting from an arbitrary affine VASS G that only uses broadcast matrices and where we want to check if configuration $(q_2, \mathbf{v}_2)$ is reachable from $(q_1, \mathbf{v}_1)$, we first transform it to an affine VASS G^* with the following properties

- each transition only changes the vector $\mathbf{v}$ in one of the following ways: (i) it adds to or subtracts from $\mathbf{v}$ a unit vector, or (ii) it multiplies $\mathbf{v}$ with a broadcast matrix M (this allows us to simulate every transition of the affine VASS with a single transition in the broadcast protocol), and
- some configuration $(q_2', \mathbf{0})$ is reachable from some configuration $(q_1', \mathbf{0})$ in G^* if and only if $(q_2, \mathbf{c}_2)$ is reachable from $(q_1, \mathbf{c}_1)$ in G.

The transformation is straightforward by splitting more complex transitions and adding auxiliary states to get from $(q_1', \mathbf{0})$ to $(q_1, \mathbf{v}_1)$ and from $(q_2, \mathbf{v}_2)$ to $(q_2', \mathbf{0})$. Now, based on G^* we define process templates A and B such that $A \| B^n$ can reach a deadlock iff $(q_2', \mathbf{0})$ is reachable from $(q_1', \mathbf{0})$ in G^*.

The states of A are the discrete states of G^*, plus additional states q_A', q_A''. If the dimension of G^* is d, then B has states $q_1, \ldots, q_d$, plus states init, v. Then, corresponding to every transition in G^* that changes the state from q to q' and either adds or subtracts unit vector $\mathbf{u}_i$, we have a rendezvous sending transition from q to q' in A, and a corresponding receiving transition in B from init to q_i (if $\mathbf{u}_i$ was added), or from q_i to init (if $\mathbf{u}_i$ was subtracted). For every transition of G^* that changes the state from q to q' and multiplies $\mathbf{c}$ with a matrix M, A has a broadcast sending transition from q to q', and receiving transitions between the states $q_1, \ldots, q_m$ that correspond to the effect of M.

The additional states q_A', q_A'' of A are used to connect reachability of $(q_2', \mathbf{0})$ to a deadlock in $A \| B^n$ in the following way: (i) there are self-loops on all states of A except on q_A', i.e., the system can only deadlock if A is in q_A', (ii) there is a broadcast sending transition from q_2' to q_A' in A, which sends all B-processes that are in $q_1, \ldots, q_m$ to special state v, and (iii) from v there is a broadcast sending transition to init in B, and a corresponding receiving transition from q_A' to q_A'' in A. Thus, $A \| B^n$ can only deadlock in a configuration where A is in q_A' and there are no B-processes in v, which is only reachable through a transition from a configuration where A is in q_2' and no B-processes are in $q_1, \ldots, q_m$. Letting q_1' be the initial state of A and init the initial state of B, such a configuration is reachable in $A \| B^n$ if and only if $(q_2', \mathbf{0})$ is reachable from $(q_1', \mathbf{0})$ in G^*. □

Approximate Methods for Deadlock Detection. Since solving the problem exactly is impractical or even undecidable in general, we propose to use approximate methods such as the following:

- For PR systems, the 01-CS introduced as a precise abstraction for disjunctive systems in Section 3.4 can also be used, but in this case it is not precise, i.e., it may produce spurious deadlocked runs. Another possible overapproximation is a system that simulates pairwise transitions by a pair of disjunctive transitions.
- For broadcast protocols we can use lossy broadcast protocols as an approximation, for which the problem is decidable [22].[8]
- Another option for both types of systems is to add initial constraints that restrict the repair algorithm and are sufficient to imply deadlock-freedom.

6.5 Repair of PR Systems and Broadcast Protocols

We introduce the last missing ingredient to extend Algorithm 2 to PR systems and broadcast protocols, and then revisit the examples from Sections 6.2 and 6.3 to show how they can be automatically repaired.

Synchronous Systems Constraints. The procedure BuildConstr in Algorithm 2 does not take into consideration synchronous actions. Hence, we need a new procedure that offers special treatment for synchronization.

To simplify presentation we assume w.l.o.g. that each $a+$, with $+ \in \{!, !!\}$, appears on exactly one local transition. We denote by $\Delta_{sync}(s, a)$ the state obtained by executing action a in state s. Additionally, let $\Delta^{local}_{sync}(s, a) = \{(q_U, a_*, q'_U) \in \delta \mid * \in \{?, !, ??, !!\}$, and a is enabled in $s\}$, and let $T(s, a) = \bigvee_{t_a \in \Delta^{local}_{sync}(s,a)} \neg t_a$. In a broadcast protocol we say that an action a is enabled in a global state $\mathbf{c}$ if $\exists i, j < |B|$ s.t. $\mathbf{c}(i) > 0$ and $(q_{B_i}, a!!, q_{B_j}) \in \delta_B$. In a PR system we say that an action a is enabled in a global state $(\mathbf{c})$ if $\exists i, j, k, l < |B|$ s.t. $\mathbf{c}(i) > 0, \mathbf{c}(j) > 0)$ and $(q_{B_i}, a!, q_{B_k}), (q_{B_j}, a?, q_{B_l}), \in \delta_B$.

Given a synchronous system $M^X = (S, S_0, \Sigma, \Delta)$ with $X \in \{BR, PR\}$, a state s, and a reachable error sequence $\mathcal{RE}$, Algorithm 3 computes a propositional formula over the set of local transitions that encodes all possible ways for a state s to avoid reaching an error.

Algorithm 3 Synchronous Constraint Computation

```
1: procedure BSC(State s, RE)
2:     if RE[1 :] is empty then
3:         return ⋀_{t_U ∈ Δ^local(s,RE[0])} ¬t_U
                  ⋀_{a ∈ Σ_sync ∧ Δ(s,a) ∈ RE[0]} T(s, a)
4:     else
5:         return ⋀_{t_U ∈ Δ^local(s,RE[0])} (¬t_U ∨
                                  BSC(Δ(s, t_U), RE[1 :]))
                  ⋀_{a ∈ Σ_sync ∧ Δ(s,a) ∈ RE[0]} [T(s, a) ∨
                                  BSC(Δ(s, t_a), RE[1 :])]}
6:     end if
7: end procedure
```

[8] Note that in the terminology of Delzanno et al., deadlock detection is a special case of the Target problem.

Example (Repair of Reader-Writer). Consider again the reader-writer example as a PR system, given in Figures 2 and 3. Now we can show how our algorithm repairs this protocol.[9]

First, we give names to all transitions in both process templates. Let

$t_1 = (q_0, (write!), q_1)$, $t_2 = (q_{A,0}, (write?), q_{A,1})$,
$t_3 = (q_{A,1}, (write?), q_{A,0})$, $t_4 = (q_0, (read!), q_2)$,
$t_5 = (q_{A,0}, (read?), q_{A,1})$, $t_6 = (q_{A,1}, (read?), q_{A,0})$,
$t_7 = (q_1, (done_w!), q_0)$, $t_8 = (q_{A,1}, (done_w?), q_{A,0})$,
$t_9 = (q_{A,0}, (done_w?), q_{A,1})$, $t_{10} = (q_2, (done_r!), q_0)$,
$t_{11} = (q_{A,1}, (done_r?), q_{A,0})$, $t_{12} = (q_{A,0}, (done_r?), q_{A,1})$.

Let the set of error states be $ERR = \uparrow\{(q_{A,0}, (0,2,0))(q_{A,1}, (0,2,0))\}$, and let the initial constraint be $TRConstr_{PR} \wedge UserConstr_{PR}$, where $UserConstr_{PR}$ is

$$(t_1 \wedge (t_2 \vee t_3)) \wedge (t_4 \wedge (t_5 \vee t_6)) \wedge (t_7 \wedge (t_8 \vee t_9)) \wedge (t_{10} \wedge (t_{11} \vee t_{12})),$$

encoding property (a) of the initial constraints described in Sect. 6.2.

Then running our repair algorithm will produce the following results (nonessential states are omitted):

- The first call to the model checker and executing Lines 4-7 yields:
  $RE_0 = \{(q_{A,0}, (0,2,0))\}$,
  $RE_1 = \{(q_{A,1}, (1,1,0))\}$,
  $RE_2 = \{(q_{A,0}, (2,0,0))\}$.
- Executing Lines 8-9 yields the following constraint: $accConstr_1 = TRConstr_{PR} \wedge UserConstr_{PR} \wedge (\neg t_1 \vee \neg t_2 \vee \neg t_3)$.
- As a solution for $accConstr_1$, the SAT solver may provide the following: $\neg t_2 \wedge \neg t_6 \wedge \neg t_9 \wedge \neg t_{12}$.
- Using this solution to restrict δ_A and δ_B, the second call to the model checker and executing Lines 4-7 yields:
  $RE_0 = \{(q_{A,0}, (0,2,0))\}$, $RE_1 = \{(q_{A,1}, (1,1,0))\}$,
  $RE_2 = \{(q_{A,0}, (2,1,0))\}$, $RE_3 = \{(q_{A,1}, (3,0,0))\}$,
  $RE_4 = \{(q_{A,0}, (4,0,0))\}$.
- Executing Lines 8-9 yields the following constraints:
  $accConstr_2 = accConstr_1 \wedge (\neg t_1 \vee \neg t_3 \vee \neg t_4 \vee \neg t_5)$.
- As a solution for $accConstr_2$, the SAT solver may provide the following: $\neg t_3 \wedge \neg t_5 \wedge \neg t_9 \wedge \neg t_{12}$.
- Using this solution to restrict δ_A and δ_B, the third call to the model checker and executing Lines 4-7 yields:
  $RE_0 = \{(q_{A,0}, (0,2,0))\}$, $RE_1 = \{(q_{A,1}, (1,1,0))\}$,
  $RE_2 = \{(q_{A,0}, (2,1,0))\}$, $RE_3 = \{(q_{A,1}, (3,0,0))\}$,
  $RE_4 = \{(q_{A,0}, (3,0,0))\}$.

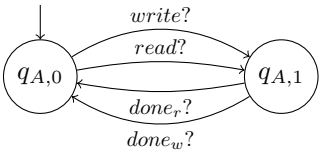

Fig. 9: Safe Scheduler

[9] We omit a detailed treatment of deadlock detection in this and the following example.

- Executing Lines 8-9 yields the following constraints: $accConstr_3 = accConstr_2 \wedge (\neg t_1 \vee \neg t_2 \vee \neg t_4 \vee \neg t_6)$.
- As a solution for $accConstr_3$, the SAT solver may provide the following: $\neg t_3 \wedge \neg t_6 \wedge \neg t_9 \wedge \neg t_{12}$.
- Using this solution to restrict δ_A and δ_B, the fourth call of the model checker returns true and we obtain the correct scheduler in Figure 9.

Example (Repair of MESI Protocol). Now consider again the MESI example as a broadcast protocol from Section 6.3. We show how Algorithm 2 repairs the protocol.

The property we consider is that there should not be processes in state M and state S at the same time, i.e., the set of error states is: $\uparrow\{(1,0,1,0)\}$. We can run Algorithm 2 on system M, error states $\uparrow\{(1,0,1,0)\}$, and initial constraints $TRConstr_{BC} \wedge \bigwedge_{a\in\Sigma_{sync}} \bigwedge_{q_B\in Q_B} one(t_{q_B}^{a??})$. The model checker will return the following error sequence (nonessential states are omitted):
$E_0 = \{(1,0,1,0)\}, E_1 = \{(0,1,1,0)\}, E_2 = \{(0,1,0,1)\}, E_3 = \{(0,0,1,1)\}$, $E_4 = \{(0,0,0,2)\}$.
Invoking procedure BSC (Algorithm 3) instead of procedure BuildConstr on Line 8 in Algorithm 2 will return the following Boolean formula:
$newConstr = \neg(\mathrm{I}, read!!, \mathrm{S}) \vee \neg(\mathrm{I}, read??, \mathrm{I}) \vee \neg(\mathrm{S}, write\text{-}inv!!, \mathrm{E}) \vee \neg(\mathrm{I}, write\text{-}inv??, \mathrm{I}) \vee \neg(\mathrm{E}, read??, \mathrm{E}) \vee \neg(\mathrm{I}, read!!, \mathrm{S}) \vee \neg(\mathrm{E}, write, \mathrm{S})$.
Running the SAT solver in Line 11 on

$$newConstr \wedge TRConstr'_{BC} \wedge \bigwedge_{a\in\Sigma_{sync}} \bigwedge_{q_B\in Q_B} one(t_{q_B}^{a??}) \bigwedge_{t_U\in\{\delta_B^\tau\}} t_U$$

will return the only solution $\neg(\mathrm{E}, read??, \mathrm{E})$ which clearly fixes the system.

7 Repair of Global Synchronization Protocols

Global synchronization protocols (GSPs) [15] generalize many existing computational models that have been considered in the parameterized verification literature, including pairwise rendezvous, asynchronous rendezvous [28] and broadcast protocols.

In this section, we show how to extend our algorithm to the repair of GSPs. While the basic ideas remain the same as before, the model is more complex than the previous models, and the repair process becomes more intricate.

Informal Description of the Model. In GSPs, each global transition synchronizes all processes, where an arbitrary number of processes act as senders while the remaining ones react uniformly as receivers. Thus, like in broadcast protocols we can restrict our model to systems of the form B^n, for a given process template B.

GSPs support two basic types of transitions: *k-sender transitions* and *k-maximal transitions*. A k-sender transition can only fire if at least k processes can act as senders and is fired with exactly k senders. Further, a k-maximal

transition can only be fired if the number m of processes that can act as senders is at least 1, and is fired with $min(m, k)$ senders. Additionally, each transition can contain a *global guard* such that a transition is enabled if it can be fired and if all processes are in the subset of states identified by the global guard.

Structure of the Section. In Sections 7.1 to 7.3, we recap the definitions from [15] for global synchronization with and without guards and how they can be framed as WSTSs. In Section 7.4, we show how Algorithm 2 can be extended to repair systems modeled as GSPs.

7.1 Guarded Synchronizing Process Template

A *guarded synchronizing process template* is a labeled transition system $B = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta_B)$,

- Q_B is a finite set of states including the initial state init_B,
- Σ is a set of local actions,
- $\mathcal{G}_B \subseteq \mathcal{P}(Q_B)$ is a set of guards,
- $\delta_B \subseteq Q_B \times \Sigma \times \mathcal{G}_B \times Q_B$ is a guarded transition relation.

Σ is based on a set $\mathcal{A}$ of *global* actions, where each $a \in \mathcal{A}$ has an *arity* $k \geq 1$ and is either a *k-sender action* or a *k-maximal action*. For every global action $a \in \mathcal{A}$ with arity k, Σ contains *local* actions $a_1!!, \ldots, a_k!!, a??$. Actions $a_1!!, \ldots, a_k!!$ are called *sending* actions and $a??$ is called a *receiving* action. A (local) transition $(q, \sigma, g, q') \in \delta_B$ is *unguarded* if $g = Q_B$. A process B is called *unguarded* if every transition is unguarded.

Like Jaber et al. [15], we assume that receives are deterministic, and that sends are *unique*, i.e., for every local action $a_i!!$ there is exactly one transition $(q, a_i!!, g, q') \in \delta_B$.

7.2 Global Synchronization Without Guards

For an unguarded process $B = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta_B)$, the configuration space for all systems B^n, for a fixed unguarded process B but arbitrary $n \in \mathbb{N}$ is the CS $M^{GSP} = (S, S_0, \Delta)$, where

- $S \subseteq \mathbb{N}_0^{|B|}$ is the set of states,
- $S_0 = \{\mathbf{c} \mid \forall q_B \in Q_B : \mathbf{c}(q_B) = 0 \text{ iff } q_B \neq \mathsf{init}_B\}$ is the set of initial states,
- $\Delta \subseteq S \times S$ is the global transition relation, separated into k-sender and k-maximal actions:

k-sender Action. A k-sender action $a \in \mathcal{A}$ with local sending transition $q_i \xrightarrow{a_i!!} q_i'$ for $i \in \{1, \ldots, k\}$ can be fired from a global state s if there are k processes that can take these local transitions. Upon firing the action, each of the local transitions on actions $a_i!!$ is taken by exactly one process, and all other processes take a transition on action $a??$ to arrive in state s'. Formally, we assign to each k-sender action $a \in \mathcal{A}$ (i) a vector $\mathbf{v}_a \in S$ containing the number of expected senders for each state $r \in Q_B : \mathbf{v}_a(r) = |\{q_B \xrightarrow{a_i!!} q_B' \mid q_B = r\}|$, (ii) a vector $\mathbf{v}_a'$ containing the number of senders that will be in each state

$r \in Q_B$ after the transition: $\mathbf{v}'_a(r) = |\{q_B \xrightarrow{a_i!!} q'_B \mid q'_B = r\}|$, and (iii) a broadcast matrix M_a.

Then, a transition from global state s is possible (enabled) if there exists an action a with $s(q_i) \geq \mathbf{v}_a(q_i)$ for all $i \in \{1, \ldots, k\}$, and the resulting global state can be computed as $s' = M_a \cdot (s - \mathbf{v}_a) + \mathbf{v}'_a$, and we write $s \to s'$.

Internal transitions are a special case of k-sender transitions, where $k = 1$ and M_a is the identity matrix.

k-maximal Action. A k-maximal action $a \in \mathcal{A}$ with local sending transitions $q_i \xrightarrow{a_i!!} q'_i$ for $i \in \{1, \ldots, k\}$ can be fired from a global state s if there is at least one process that can take one of these local transitions. Upon firing the action, for each state q_i with at least one local transition $q_i \xrightarrow{a_i!!} q'_i$, (i) if $s(q_i) \geq \mathbf{v}_a(q_i)$ then each of the local transitions $q_i \xrightarrow{a_i!!} q'_i$ is taken by exactly one process, or, (ii) if $s(q_i) < \mathbf{v}_a(q_i)$ then a total of $s(q_i)$ of the local transitions $q_i \xrightarrow{a_i!!} q'_i$ is taken, each by exactly one process. All other processes take a transition on the receiving action $a??$ to arrive in the new global state s'. Formally, we again assign to each action a vectors $\mathbf{v}_a, \mathbf{v}'_a$ and a broadcast matrix M_a. If $s(q_i) \geq \mathbf{v}_a(q_i)$ for all $i \in \{1, \ldots, k\}$, then s' is computed as above. For cases where this does not hold, we assign to the action an additional set of vector pairs $\mathbf{u}_a, \mathbf{u}'_a$ with different numbers of senders that actually participate, and s' is computed based on a vector-pair with the maximal number of senders that is supported by s.

For GSPs without guards, the configuration space is always well-structured.

Theorem 7 (Theorem 2 from [15]). *Let $M^{GSP} = (S, S_0, \Delta)$ be a CS for an unguarded process B. Then $(M^{GSP}, \lessapprox)$ is a WSTS with effective pred-basis.*

Deadlock Detection for GSPs. Since GSPs without guards extend broadcast protocols, deadlock detection for these systems is undecidable by Theorem 6, and the same holds for GSPs with guards.

7.3 Global Synchronization With Guards

Guarded System. Let the *support* of a global state s be $\mathsf{supp}(s) = \{q_B \in Q_B \mid s(q_B) > 0\}$, i.e., the set of local states that appear at least once in s. Then, the semantics of a global transition on action a with guard g, denoted $q_B \xrightarrow{a,g} q'_B$, is as defined before, except that the transition is enabled only if $\mathsf{supp}(s) \subseteq g$.

Example (Smoke Detector). Consider the smoke detector process depicted in Figure 10, taken from [15]. When detecting smoke, up to two processes should report the smoke to the fire department. The GSP uses different types of transitions: an internal transition from ENV to Ask guarded by g_1, a

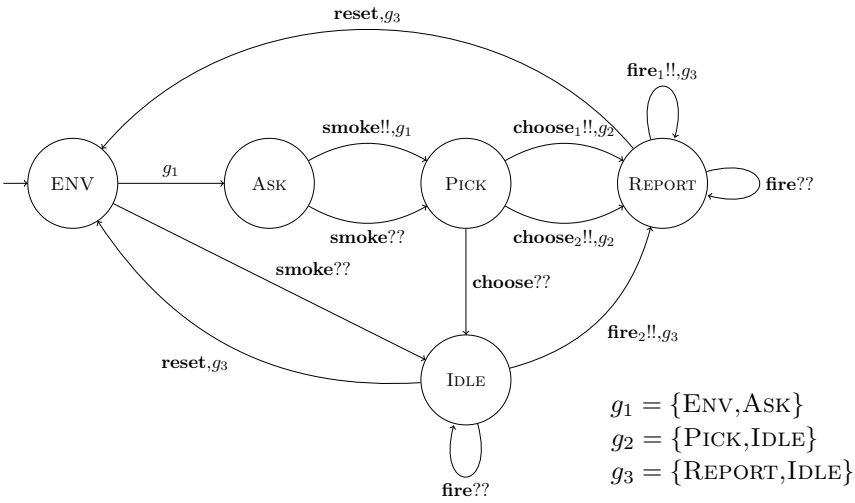

Fig. 10: Smoke Detector

broadcast (1-sender action) on action **smoke** guarded by g_1 and a negotiation (set of 1-sender actions) on action **reset** guarded by g_3, i.e., a synchronous transition with no dedicated sender. Furthermore, the 2-maximal actions **choose** guarded by g_2 and **fire** guarded by g_3 can report the smoke.

Since compatibility under $\preceq$ does not hold in general for GSPs with guards, we introduce a refined wqo, denoted $\trianglelefteq$, that is based on the semantics of guards. Furthermore, compatibility with respect to $\trianglelefteq$ only holds under the following conditions.

Refined wqo. Let $\mathcal{G}_B$ be the set of guards that appear on transitions in B. Then the wqo $\trianglelefteq$ is defined as

$$s \trianglelefteq t \text{ } \textit{iff} \text{ } (s \preceq t \wedge \forall g \in \mathcal{G}_B : (\mathsf{supp}(s) \subseteq g \Leftrightarrow \mathsf{supp}(t) \subseteq g)).$$

Strong Guard-Compatibility for k-Sender Actions. For a k-sender action a with local sending transitions $q_i \xrightarrow{a_i!!,g} q_i'$ for $\{1,\ldots,k\}$, let $\hat{q}_B$ be the set of all states $q_i, \hat{q}_B'$ the set of states q_i', and M_a the broadcast matrix. We say that action a is *strongly guard compatible* if the following holds for all $g' \in \mathcal{G}_B$:

$$\hat{q}_B' \subseteq g' \Rightarrow \forall q_B \in g : M_a(q_B) \in g' \qquad \text{(C1)}$$

Strong Guard-Compatibility for k-Maximal Actions. Consider a k-maximal action a with local transitions $q_i \xrightarrow{a_i!!,g} q_i'$ for $i \in \{1,\ldots,k\}$ and broadcast matrix M_a. Then the action a is *strongly guard-compatible* if the

following holds for all $g' \in \mathcal{G}_B$:

$$(\hat{q}'_B \cap g' \neq \emptyset) \Rightarrow (\hat{q}'_B \subseteq g' \wedge \forall q_B \in g : M_a(q_B) \in g') \tag{C2}$$

To support a larger class of systems, the previous conditions can be relaxed at the cost of making them more complex.

Refinement: Weak Guard-Compatibility. We write $r_B \triangleleft q_B$ if, for all guards $g \in \mathcal{G}_B, q_B \in g \Rightarrow r_B \in g$. Similarly, we write $r_B \triangleleft h$ for a set of states h if, for all guards $g \in \mathcal{G}_B, h \subseteq g \Rightarrow r_B \in g$. If there exists a path of unguarded internal transitions from q_B to q'_B, we write $q_B \rightsquigarrow q'_B$. Then, condition (C1) can be relaxed to

$$\hat{q}'_B \subseteq g' \Rightarrow \forall q_B \in g : (M_a(q_B) \in g' \vee \varphi_{\rightsquigarrow}), \tag{C1w}$$

where $\varphi_{\rightsquigarrow} = \exists q'_B \in Q_B : (q'_B \triangleleft \hat{q}'_B \wedge M_a(q_B) \rightsquigarrow q'_B)$.

In a similar way, condition (C2) can be relaxed to obtain condition (C2w) by replacing $M_a(q_B) \in g'$ with $M_a(q_B) \in g' \vee \exists q'_B \in Q_B : (\forall q'_i : (q'_B \triangleleft q'_i) \wedge M_a(q_B) \rightsquigarrow q'_B)$. Actions that satisfy condition (C1w) or (C2w) are called *weakly guard-compatible*.

Refinement: Internal Transitions. We write $q_B \rightsquigarrow_g q'_B$ if there exists a path of internal transitions from q_B to q'_B such that each transition is guarded by some $g' \supseteq g$. Then, we say that an internal action a is *weakly guard-compatible* if the following holds for every $g' \in \mathcal{G}_B$ (note that $\hat{q}_B$ and $\hat{q}'_B$ are both singleton sets, denoted q_B and q'_B in the following) and every $r_B \in g$:

$$q_B \notin g' \wedge q'_B \in g' \Rightarrow \exists r'_B : (r'_B \triangleleft q'_B \wedge r_B \rightsquigarrow_{g''} r'_B), \tag{C3w}$$

where $g'' = g \cup \{r_B \mid r_B \triangleleft q'_B\}$.

Well-Behavedness. A process B is *well-behaved* if every action is (weakly) guard compatible. For well-behaved GSPs, the configuration space is well-structured.

Theorem 8 (Theorem 3 from [15]). *Let $M^{GSP} = (S, S_0, \Delta)$ be a CS for a (guarded) process B that is well-behaved. Then $(M^{GSP}, \trianglelefteq)$ is a WSTS with effective pred-basis.*

7.4 Repair of Guarded GSPs

In the following, we present how to extend Algorithm 2 to repair GSPs by building constraints for k-sender and k-maximal actions. Furthermore, we revisit the smoke detector from Section 7.3 to automatically repair it.

Initial Constraints. To avoid repairs that have receiving transitions without a matching sending transition, we define constraints $TRConstr_{GSP}$ in a similar way as $TRConstr_{BC}$. That is, in any repair for an action $a \in \mathcal{A}$ all k-maximal

(k-sender) transitions $t_{a??}$ are deleted iff all $t_{a_j!!}$ for some $j \leq k$ are deleted. For k-maximal (k-sender) transitions, the designer can use the following constraint:

$$\bigwedge_{a \in \mathcal{A}} [((\bigwedge_{i \leq k} \bigvee_{t_{a_i!!} \in \delta_B} t_{a_i!!}) \wedge (\bigvee_{t_{a??} \in \delta_B} t_{a??})) \vee ((\bigvee_{i \leq k} \bigwedge_{t_{a_i!!} \in \delta_B} \neg t_{a_i!!}) \wedge (\bigwedge_{t_{a??} \in \delta_B} \neg t_{a??}))]$$

Guarded Synchronous Systems Constraints. As described in Section 6.5, Algorithm 3 computes a propositional formula over the set of local transitions for synchronous constraints. The same approach also works for guarded synchronization protocols. Let $\Delta^{local}_{GSP}(s,a) = \{(q_B, a_*, q'_B) \in \delta_B \mid * \in \{??, !!\}$, and a is enabled in $s\}$, and let $T_{GSP}(s,a) = \bigvee_{t_a \in \Delta^{local}_{GSP}(s,a)} \neg t_a$. Then, by replacing $T(s,a)$ with $T_{GSP}(s,a)$ in Algorithm 3, it automatically builds synchronous system constraints for global synchronization protocols.

Well-behavedness of a Repair. Recall that a repair of a process $B = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta_B)$ is a subset $\delta'_B \subseteq \delta_B$ such that the CS M^{GSP} based on process $B' = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta'_B)$ is safe. Furthermore, model checking of a candidate repair, such as Algorithm 1, requires M^{GSP} to be a WSTS with effective pred-basis. For GSPs with guards, this holds if B is well-behaved.

However, note that even if B is well-behaved, a restriction B' may not be well-behaved. To see this, consider the well-behaved GSP shown in Figure 11, with 1-sender actions a and b two internal transitions, denoted by τ_1, τ_2. Intuitively, every action is weakly guard-compatible because by taking an internal transition the receivers can reach a state that satisfies the guard conditions. For the set of error states $ERR = \uparrow\{(0,2)\}$, the GSP is not safe. A candidate repair could delete the internal transition τ_1 from q_0 to q_1. However then the resulting GSP is not well-behaved, since action a is not weakly guard-compatible anymore. Note that this also holds by interpreting actions a and b as 1-maximal actions.

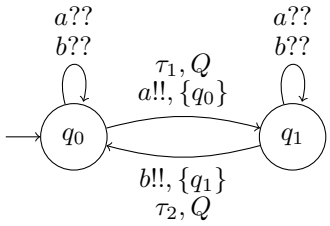

Fig. 11: A well-behaved guarded GSP

The following lemma shows that every restriction of a well-behaved GSP, is well-behaved if every action is strongly guard-compatible. Note that the proof makes use of our assumption that receives are deterministic.

Lemma 8. *Given a well-behaved guarded process $B = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta_B)$ where each $a \in \mathcal{A}$ is strongly guard-compatible, let $B' = (Q_B, \mathsf{init}_B, \Sigma, \mathcal{G}_B, \delta'_B)$ with $\delta'_B \subseteq \delta_B$. Then B' is well-behaved.*

Proof Assume that B' is not well-behaved. Then there exists an action $a \in \mathcal{A}$ that is not (weakly) guard-compatible.

- Assume that a is a k-sender action with local sending transitions $q_i \xrightarrow{a_i!!,g} q_i'$ for $i \in \{1,\ldots,k\}$. Let $\hat{q}_B$ be the set of all states q_i, $\hat{q}_B'$ the set of states q_i' and M_a the broadcast matrix. Since a is not guard-compatible, by condition (C1w), there exists $g' \in \mathcal{G}_B$ with $\hat{q}_B' \subseteq g'$ such that $\exists q_B \in g : M_a(q_B) \notin g'$ and $\forall q_B' \in Q_B : (q_B' \not\triangleleft \hat{q}_B' \vee M_a(q_B) \not\rightsquigarrow q_B')$. Since by assumption receives are deterministic and $M_a(q_B) \notin g'$, by condition (C1) a is not strongly guard-compatible for B which is clearly a contradiction.

- Assume that a is a k-maximal action with local sending transitions $q_i \xrightarrow{a_i!!,g} q_i'$ for $i \in \{1,\ldots,k\}$. Let $\hat{q}_B$ be the set of all states q_i, $\hat{q}_B'$ the set of states q_i' and M_a the broadcast matrix. Since a is not guard-compatible, by condition (C2w), there exists $g' \in \mathcal{G}_B$ with $\hat{q}_B' \cap g' \neq \emptyset$ such that $\hat{q}_B' \not\subseteq g'$ (1) or $\exists q_B \in g : M_a(q_B) \notin g'$ (2) and $\forall q_B' \in Q_B : (q_B' \not\triangleleft \hat{q}_B'$ or $M_a(q_B) \not\rightsquigarrow q_B')$. If $\hat{q}_B' \not\subseteq g'$ (1) then by condition (C2) a is not strongly guard-compatible for B which is clearly a contradiction. Otherwise, if $\exists q_B \in g : M_a(q_B) \notin g'$ (2), then under the assumption that receives are deterministic and by condition (C2) a is not strongly guard-compatible for B which is clearly a contradiction.

- Assume that a is an internal transition with local sending transition $q_B \xrightarrow{a!!,g} q_B'$ where the broadcast matrix M_a is the identity matrix. Since we assume that a is not guard-compatible by condition (C3w), there exists $r_B \in g$ and $g' \in \mathcal{G}_B$ with $q_B \notin g'$ and $q_B' \in g'$ such that $\forall r_B' : r_B' \not\lhd q_B' \vee r_B \not\rightsquigarrow_{g''} r_B'$ where $g'' = g \cup \{r_B \mid r_B \lhd q_B'\}$. In particular, for $r_B' = r_B$, it holds that $r_B' \not\triangleleft q_B'$. Thus, there exists $g''' \in \mathcal{G}_B$ with $q_B' \in g'''$ and $r_B = M_a(r_B) \notin g'''$. Thus, by condition (C1), a is not strongly guard-compatible for B which is clearly a contradiction.

□

Thus, the repair approach works for all GSPs where each action is strongly guard-compatible. For GSPs that include actions that are not strongly guard-compatible, e.g., internal transitions, we can only offer partial solutions:

- if the input is weakly guard-compatible and the candidate repair does not remove internal transitions, then the repair will also be weakly guard-compatible
- a more expensive approach involves the preprocessing for constructing constraints that encode for each action the internal transitions that preserve weak guard-compatibility. That is, for every action a that only satisfies weak guard-compatibility, we identify the set of internal transitions $\delta_a \subset \delta_B$ such that removal of a transition in δ_a would destroy weak guard-compatibility. Then, we add constraints that forbid removal of any transition in δ_a unless all transitions on a are removed.

Example (Smoke Detector). Consider the GSP that models a smoke detector in Figure 10. The considered property specifies that at most two processes are allowed to report the smoke, i.e., the set of error states is: $\uparrow\{(0,0,0,0,3)\}$.

We run Algorithm 2 on M^{GSP}, $\uparrow\{(0,0,0,0,3)\}$, $TRConstr_{GSP}$. The model checker returns the following error sequence where nonessential states are omitted:

$E_0 = \{(0,0,0,0,3)\}, E_1 = \{(0,0,0,1,2)\}, E_2 = \{(0,0,3,0,0)\}, E_3 =$

$\{(0,3,0,0,0)\}, E_4 = \{(1,2,0,0,0)\}, E_5 = \{(2,1,0,0,0)\}, E_6 = \{(3,0,0,0,0)\}$.

Running the procedure BSC (Algorithm 3) in Line 8 will return the following Boolean formula:

$$\begin{aligned} newConstr = &\neg(\textsc{Report}, \mathbf{fire}_1!!, g_3, \textsc{Report}) \vee \neg(\textsc{Idle}, \mathbf{fire}_2!!, g_3, \textsc{Report}) \\ &\vee \neg(\textsc{Report}, \mathbf{fire}??, \textsc{Report}) \vee \neg(\textsc{pick}, \mathbf{choose}_1!!, g_2, \textsc{Report}) \\ &\vee \neg(\textsc{Pick}, \mathbf{choose}_2!!, g_2, \textsc{Report}) \vee \neg(\textsc{Pick}, \mathbf{choose}??, \textsc{Idle}) \\ &\vee \neg(\textsc{Ask}, \mathbf{smoke}!!, g_1, \textsc{Pick}) \vee \neg(\textsc{Ask}, \mathbf{smoke}??, \textsc{Pick}) \\ &\vee \neg(\textsc{Env}, g_1, \textsc{Ask}) \end{aligned}$$

Running the SAT solver in Line 11 on $newConstr \wedge TRConstr_{GSP}$ will return the following deadlock-free solution which fixed the system:

$$\begin{aligned} &\neg(\textsc{Report}, \mathbf{fire}_1!!, g_3, \textsc{Report}) \wedge \neg(\textsc{Idle}, \mathbf{fire}_2!!, g_3, \textsc{Report}) \\ &\wedge \neg(\textsc{Report}, \mathbf{fire}??, \textsc{Report}) \wedge \neg(\textsc{Idle}, \mathbf{fire}??, \textsc{Idle}) \end{aligned}$$

8 Implementation & Evaluation

We have implemented a prototype of our parameterized repair algorithm that supports the three basic types of systems (disjunctive, pairwise rendezvous and broadcast), and safety and reachability properties. GSPs or more complex properties are currently not supported.

For disjunctive and pairwise rendezvous systems, we have evaluated it on different variants of reader-writer-protocols, based on the ones given in Sections 1 and 2, where we replicated some of the states and transitions to test the performance of our algorithm on bigger benchmarks. For disjunctive systems, all variants have been repaired successfully in less than 2s. For pairwise rendezvous systems, these benchmarks are denoted "RWi (PR)" in Table 1.

For broadcast protocols, we have evaluated our algorithm on a range of more complex benchmarks taken from the parameterized verification literature [29]: a distributed **Lock Service** (DLS) inspired by the Chubby protocol [30], a distributed **Robot Flocking** protocol (RF) [31], a distributed **Smoke Detector** (SD) [15], a sensor network implementing a **Two-Object Tracker** (2OT) [32], and the cache coherence protocol **MESI** [33] in different variants constructed similar as for RW.

Typical desired safety properties are mutual exclusion and similar properties. Since deadlock detection is undecidable for broadcast protocols, the absence of deadlocks needs to be ensured with additional initial constraints.

On all benchmarks, we compare the performance of our algorithm based on the valuations of two flags: SEP and EPT. The SEP ("single error path") flag indicates that, instead of encoding all the model checker's computed error paths, only one path is picked and encoded for SAT solving. When the EPT ("error path transitions") flag is raised the SAT formula is constructed so that

only transitions on the extracted error paths may be suggested for removal. Note that in the default case, even transitions that are unrelated to the error may be removed. Table 1 summarizes the experimental results we obtained.

We note that the algorithm deletes fewer transitions when the EPT flag is raised (EPT=T). This is because we tell the SAT solver explicitly not to delete transitions that are not on the error paths. Removing fewer transitions might be desirable in some applications. We observe the best performance when the SEP flag is set to true (SEP=T) and the EPT flag is false. This is because the constructed SAT formulas are much simpler and the SAT solver has more freedom in deleting transitions, resulting in a small number of iterations.

9 Related Work

Many automatic repair approaches have been considered in the literature, most of them restricted to monolithic systems [11, 12, 34–37]. Additionally, there are several approaches for synchronization synthesis and repair of *concurrent systems*. Some of them differ from ours in the underlying approach, e.g., being based on automata-theoretic synthesis [38, 39]. Others are based on a similar underlying counterexample-guided synthesis/repair principle, but differ in other aspects from ours. For instance, there are approaches that repair the program by adding atomic sections, which forbid the interruption of a sequence of program statements by other processes [13, 40]. *Assume-Guarantee-Repair* [41] combines verification and repair, and uses a learning-based algorithm to find counterexamples and restrict transition guards to avoid errors. In contrast to ours, this algorithm is not guaranteed to terminate. From *lazy synthesis* [42] we borrow the idea to construct the set of *all* error paths of a given length instead of a single concrete error path, but this approach only supports systems with a fixed number of components. Some of these existing approaches are more general than ours in that they support certain infinite-state processes [13, 40, 41], or more expressive specifications and other features like partial information [38, 39].

The most important difference between our approach and all of the existing repair approaches is that, to the best of our knowledge, none of them provide correctness guarantees for systems with a parametric number of components. This includes also the approach of McClurg et al. [14] for the synthesis of synchronizations in a software-defined network. Although they use a variant of Petri nets as a system model, which would be suitable to express parameterized systems, their restrictions are such that the approach only considers a fixed number of components. In contrast, we include a parameterized model checker in our repair algorithm, and can therefore provide parameterized correctness guarantees. There exists a wealth of results on parameterized model checking, collected in several good surveys recently [25, 43, 44].

Table 1: Running time, number of iterations, and number of deleted transitions (#DT) for the different configurations. Each benchmark is listed with its number of local states and edges. For the set of errors, P_1 and P_2 are two distinct error sets that differ from one benchmark to another, and $C = P_1 \cup P_2$. Smallest number (over all settings) of iterations, runtime per benchmark, deleted transitions are highlighted in boldface.

(a) Results without EPT (error path transitions) setting.

Benchmark	**Size**		**Errors**	**[SEP=F & EPT=F]**			**[SEP=T & EPT=F]**		
	States	Edges		#Iter	Time	#DT	#Iter	Time	#DT
RW1 (PW)	5	12	C	3	2.5	4	3	2.9	4
RW2 (PW)	15	42	C	3	3.8	14	3	4.8	14
RW3 (PW)	35	102	C	3	820.7	34	3	**7.6**	34
RW4 (PW)	45	132	C	TO	TO	TO	**3**	**11.8**	44
DLS	10	95	P1	**1**	**0.8**	13	**1**	**0.8**	13
DLS	10	95	P2	**1**	**0.8**	13	2	1.7	13
DLS	10	95	C	**2**	4.2	13	**2**	**1.5**	13
RF	10	147	P1	**1**	2.5	32	**1**	**1.2**	32
RF	10	147	P2	**1**	**1.2**	32	**1**	1.3	32
RF	10	147	C	**1**	7.8	32	**1**	**1.4**	32
SD	6	39	C	**1**	**1**	**4**	**1**	**1**	**4**
2OT	12	128	P1	12	18.8	26	**6**	**8.3**	26
2OT	12	128	P2	**1**	**1.8**	26	**1**	**1.8**	26
2OT	12	128	C	11	17.2	Unreal.	**6**	**11.7**	Unreal.
MESI1	4	26	C	**1**	2.4	6	**1**	**0.9**	6
MESI2	9	71	C	**1**	**1.1**	26	**1**	**1.1**	26
MESI3	14	116	C	**1**	109.4	46	**1**	**108.1**	46

(b) Results with EPT (error path transitions) setting.

Benchmark	**Size**		**Errors**	**[SEP=F & EPT=T]**			**[SEP=T & EPT=T]**		
	States	Edges		#Iter	Time	#DT	#Iter	Time	#DT
RW1 (PW)	5	12	C	**2**	**1.7**	**2**	**2**	**1.7**	**2**
RW2 (PW)	15	42	C	**2**	**3.2**	**7**	7	8.4	**7**
RW3 (PW)	35	102	C	**2**	552.3	**17**	17	40.3	**17**
RW4 (PW)	45	132	C	TO	TO	TO	22	99.2	**22**
DLS	10	95	P1	3	2.4	**5**	5	5.6	**5**
DLS	10	95	P2	3	2.6	**9**	7	5.5	**9**
DLS	10	95	C	3	3	**9**	9	8.1	**9**
RF	10	147	P1	TO	TO	TO	8	12.4	**13**
RF	10	147	P2	TO	TO	TO	8	11.3	**14**
RF	10	147	C	TO	TO	TO	8	12.5	**12**
SD	6	39	C	3	2.4	**4**	3	3	**4**
2OT	12	128	P1	16	73.8	**17**	16	34	**17**
2OT	12	128	P2	4	2958	**11**	8	16.5	12
2OT	12	128	C	TO	TO	TO	11	48.6	Unreal.
MESI1	4	26	C	2	1.8	**5**	4	3.5	**5**
MESI2	9	71	C	3	56.4	20	6	6.8	**15**
MESI3	14	116	C	TO	TO	TO	6	289.9	**15**

10 Conclusion and Future Work

We have investigated the parameterized repair problem for systems of the form $A \| B^n$ with an arbitrary $n \in \mathbb{N}$. We introduced a general parameterized repair algorithm, based on interleaving the generation of candidate repairs with parameterized model checking and deadlock detection, and instantiated this approach to different classes of systems that can be modeled as WSTS:

disjunctive systems, pairwise rendezvous systems, broadcast protocols, and global synchronization protocols.

Since deadlock detection is an important part of our method, we investigated this problem in detail for these classes of systems, and found that the problem can be decided in EXPTIME for disjunctive systems, and is undecidable for broadcast protocols.

Besides reachability properties and the absence of deadlocks, our algorithm can guarantee general safety properties, based on the automata-theoretic approach to model checking. On a prototype implementation of our algorithm, we have shown that it can effectively repair non-deterministic overapproximations of many examples from the literature. Moreover, we have evaluated the impact of different heuristics or design choices on the performance of our algorithm in terms of repair time, number of iterations, and number of deleted transitions.

A limitation of the current algorithm is that it cannot guarantee any *liveness properties*, like termination or the absence of undesired loops. Also, it cannot automatically *add behavior* (states, transitions, or synchronization options) to the system, in case the repair for the given input is unrealizable. We consider these as important avenues for future work. Moreover, in order to improve the practicality of our approach we want to examine the inclusion of symbolic techniques for counter abstraction [45], and advanced parameterized model checking techniques, e.g., *cutoff* results for disjunctive systems [6, 46, 47], or recent *pruning* results for immediate observation Petri nets, which model exactly the class of disjunctive systems [19].

References

[1] Suzuki, I.: Proving properties of a ring of finite state machines. Inf. Process. Lett. **28**(4), 213–214 (1988)

[2] German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992)

[3] Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS, pp. 352–359. IEEE Computer Society (1999)

[4] Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE. LNCS, vol. 1831, pp. 236–254. Springer (2000)

[5] Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Foundations of Computer Science **14**(4), 527–549 (2003)

[6] Emerson, E.A., Kahlon, V.: Model checking guarded protocols. In: LICS, pp. 361–370. IEEE Computer Society (2003)

[7] Clarke, E.M., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: CONCUR. LNCS, vol. 3170, pp. 276–291. Springer

(2004)

[8] Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized model checking of token-passing systems. In: VMCAI. LNCS, vol. 8318, pp. 262–281. Springer (2014)

[9] Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. In: CONCUR. LNCS, vol. 8704, pp. 109–124. Springer (2014)

[10] Aminof, B., Rubin, S.: Model checking parameterised multi-token systems via the composition method. In: IJCAR. LNCS, vol. 9706, pp. 499–515. Springer (2016). https://doi.org/10.1007/978-3-319-40229-1_34

[11] Jobstmann, B., Griesmayer, A., Bloem, R.: Program repair as a game. In: 17th Conference on Computer Aided Verification (CAV'05), pp. 226–238. Springer (2005). LNCS 3576

[12] Attie, P.C., Bab, K.D.A., Sakr, M.: Model and program repair via sat solving. ACM Transactions on Embedded Computing Systems (TECS) **17**(2), 1–25 (2017)

[13] Bloem, R., Hofferek, G., Könighofer, B., Könighofer, R., Außerlechner, S., Spörk, R.: Synthesis of synchronization using uninterpreted functions. In: 2014 Formal Methods in Computer-Aided Design (FMCAD), pp. 35–42 (2014). IEEE

[14] McClurg, J., Hojjat, H., Černỳ, P.: Synchronization synthesis for network programs. In: International Conference on Computer Aided Verification, pp. 301–321 (2017). Springer

[15] Jaber, N., Jacobs, S., Wagner, C., Kulkarni, M., Samanta, R.: Parameterized verification of systems with global synchronization and guards. In: CAV (1). Lecture Notes in Computer Science, vol. 12224, pp. 299–323. Springer (2020)

[16] Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, pp. 313–321 (1996). IEEE

[17] Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science **256**(1-2), 63–92 (2001)

[18] Jacobs, S., Sakr, M., Völp, M.: Automatic repair and deadlock detection for parameterized systems. In: FMCAD (2022)

[19] Esparza, J., Raskin, M.A., Weil-Kennedy, C.: Parameterized analysis

of immediate observation petri nets. In: Petri Nets. Lecture Notes in Computer Science, vol. 11522, pp. 365–385. Springer (2019)

[20] Bozzelli, L., Ganty, P.: Complexity analysis of the backward coverability algorithm for VASS. In: RP. Lecture Notes in Computer Science, vol. 6945, pp. 96–109. Springer (2011)

[21] Pnueli, A., Xu, J., Zuck, L.D.: Liveness with (0, 1, infty)-counter abstraction. In: CAV. Lecture Notes in Computer Science, vol. 2404, pp. 107–122. Springer (2002)

[22] Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR. LNCS, vol. 6269, pp. 313–327. Springer (2010). https://doi.org/10.1007/978-3-642-15375-4_22

[23] Delzanno, G., Sangnier, A., Zavattaro, G.: Verification of ad hoc networks with node and communication failures. In: FMOODS/FORTE. Lecture Notes in Computer Science, vol. 7273, pp. 235–250. Springer (2012)

[24] Leroux, J.: The reachability problem for petri nets is not primitive recursive. In: FOCS, pp. 1241–1252. IEEE (2021)

[25] Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015). https://doi.org/10.2200/S00658ED1V01Y201508DCT013

[26] Schmitz, S., Schnoebelen, P.: The power of well-structured systems. In: CONCUR. Lecture Notes in Computer Science, vol. 8052, pp. 5–24. Springer (2013)

[27] Blondin, M., Raskin, M.: The complexity of reachability in affine vector addition systems with states. In: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 224–236 (2020)

[28] Delzanno, G., Raskin, J., Begin, L.V.: Towards the automated verification of multithreaded java programs. In: TACAS. Lecture Notes in Computer Science, vol. 2280, pp. 173–187. Springer (2002)

[29] Jaber, N., Wagner, C., Jacobs, S., Kulkarni, M., Samanta, R.: Quicksilver: modeling and parameterized verification for distributed agreement-based systems. Proc. ACM Program. Lang. **5**(OOPSLA), 1–31 (2021)

[30] Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: OSDI, pp. 335–350. USENIX Association (2006)

[31] Canepa, D., Potop-Butucaru, M.G.: Stabilizing flocking via leader election in robot networks. In: SSS. Lecture Notes in Computer Science, vol. 4838, pp. 52–66. Springer (2007)

[32] Chang, C., Tsai, J.: Distributed collaborative surveillance system based on leader election protocols. IET Wirel. Sens. Syst. **6**(6), 198–205 (2016)

[33] Emerson, E.A., Kahlon, V.: Exact and efficient verification of parameterized cache coherence protocols. In: CHARME. LNCS, vol. 2860, pp. 247–262. Springer (2003). https://doi.org/10.1007/978-3-540-39724-3_22

[34] Demsky, B., Rinard, M.: Automatic detection and repair of errors in data structures. In: Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03), pp. 78–95 (2003)

[35] Griesmayer, A., Bloem, R., Cook, B.: Repair of Boolean programs with an application to C. In: 18th Conference on Computer Aided Verification (CAV'06), pp. 358–371 (2006). LNCS 4144

[36] Forrest, S., Nguyen, T., Weimer, W., Goues, C.L.: A genetic programming approach to automated software repair. In: Genetic and Evolutionary Computation Conference (GECCO'09), pp. 947–954. ACM (2009)

[37] Monperrus, M.: Automatic software repair: A bibliography. ACM Comput. Surv. **51**(1), 17–11724 (2018)

[38] Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5-6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z

[39] Bansal, S., Namjoshi, K.S., Sa'ar, Y.: Synthesis of coordination programs from linear temporal specifications. Proc. ACM Program. Lang. **4**(POPL), 54–15427 (2020). https://doi.org/10.1145/3371122

[40] Vechev, M., Yahav, E., Yorsh, G.: Abstraction-guided synthesis of synchronization. In: Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 327–338 (2010)

[41] Frenkel, H., Grumberg, O., Pasareanu, C., Sheinvald, S.: Assume, guarantee or repair. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 211–227 (2020). Springer

[42] Finkbeiner, B., Jacobs, S.: Lazy synthesis. In: VMCAI. LNCS, vol. 7148, pp. 219–234. Springer (2012)

[43] Esparza, J.: Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In: STACS. LIPIcs, vol. 25, pp. 1–10. Schloss

Dagstuhl - Leibniz-Zentrum fuer Informatik (2014). https://doi.org/10.4230/LIPIcs.STACS.2014.1

[44] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking vol. 10. Springer (2018)

[45] Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Symbolic counter abstraction for concurrent software. In: International Conference on Computer Aided Verification, pp. 64–78 (2009). Springer

[46] Außerlechner, S., Jacobs, S., Khalimov, A.: Tight cutoffs for guarded protocols with fairness. In: VMCAI. LNCS, vol. 9583, pp. 476–494. Springer (2016). https://doi.org/10.1007/978-3-662-49122-5_23

[47] Jacobs, S., Sakr, M.: Analyzing guarded protocols: Better cutoffs, more systems, more expressivity. In: International Conference on Verification, Model Checking, and Abstract Interpretation, pp. 247–268 (2018). Springer