




# Automatic Repair and Deadlock Detection for Parameterized Systems

Swen Jacobs    
 CISPA, Saarbrücken, Germany

Mouhammad Sakr    
 SnT, University of Luxembourg

Marcus Völz    
 SnT, University of Luxembourg

**Abstract**—We present an algorithm for the repair of parameterized 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. We apply this algorithm on systems that can be represented as well-structured transition systems (WSTS), including disjunctive systems, pairwise rendezvous systems, and broadcast protocols. Moreover, we show that parameterized deadlock detection can be decided in EXPTIME for disjunctive systems, and that deadlock detection is in general undecidable for broadcast protocols.

## I. 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 the latter such that it satisfies the specification. 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. 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.

However, 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 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].

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 “repair” a system is to let it run into a deadlock as quickly as possible. Unlike non-determinism for repairing internal behavior, we are even able to introduce non-determinism for repairing communication automatically.

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.

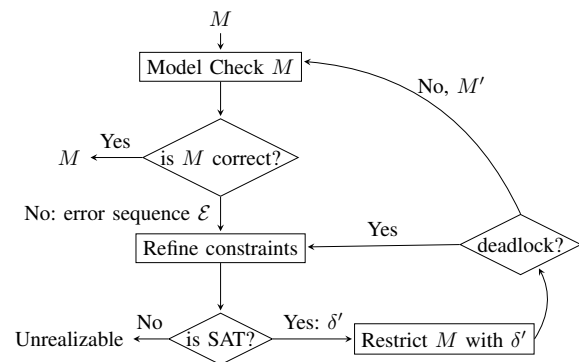


Fig. 1: Parameterized repair of concurrent systems

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 pro-

cedures exist for certain classes of systems, such as guarded protocols with disjunctive guards (or disjunctive systems) [4], pairwise systems [2] and broadcast protocols [3]. 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 allow us to avoid such error paths in future repair candidates.
- 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<sup>1</sup>, 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 actions (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 must be avoided if they use 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 consecutive *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 restrict non-determinism and communication and make the given error path impossible. To repair the system we mainly need somehow to oblige a process to wait for the action *done<sub>w</sub>!* (done writing) before entering the *writing* state. 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 IV) that enforce totality on the transition relation. Another undesirable solution would be the scheduler shown in Fig. 4, because the resulting system will deadlock immediately. This is avoided by checking reachability of deadlocks on candidate repairs. We get a solution that is safe and deadlock-free if we take Fig. 4 and flip all transitions.

**Contributions.** Our main contribution is a counterexample-guided parameterized repair approach, based on model checking of well-structured transition systems (WSTS) [15], [16]. We investigate which information a parameterized model checker needs to provide to guide the search for candidate

<sup>1</sup>For example, to allow the user to specify additional properties of the repair, such as keeping certain states reachable.

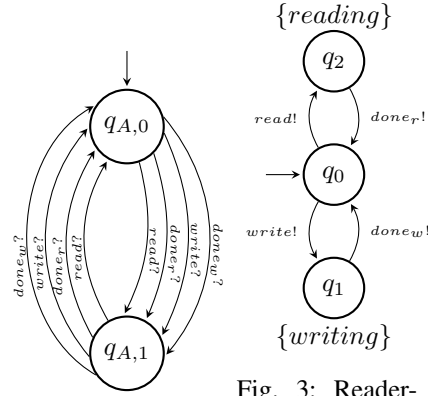


Fig. 2: Scheduler

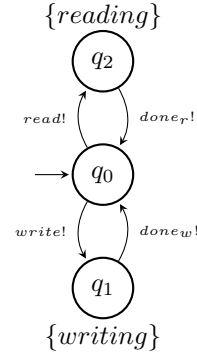


Fig. 3: Reader-Writer

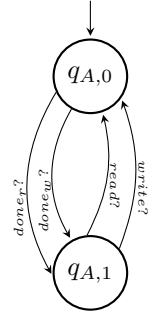


Fig. 4: Deadlocked Scheduler

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 systems and broadcast protocols.

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 deadlock detection algorithm, based on an abstract transition system, that improves on the complexity of the best known solution; (ii) for broadcast protocols 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 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.

## II. SYSTEM MODEL

For simplicity, we first restrict our attention to disjunctive systems, other systems will be considered in Sect. V-B. In the following, let  $Q$  be a finite set of states.

**Processes.** A *process template* is a transition system  $U = (Q_U, \text{init}_U, \mathcal{G}_U, \delta_U)$ , where  $Q_U \subseteq Q$  is a finite set of states including the initial state  $\text{init}_U$ ,  $\mathcal{G}_U \subseteq \mathcal{P}(Q)$  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  $\delta_U$  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.<sup>2</sup>

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. Transitions with the trivial guard  $g = Q$  are displayed without a guard. We formalize the semantics of disjunctive systems in the following.

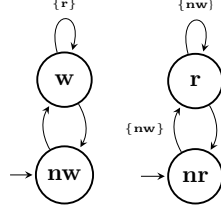


Fig. 5: Writer

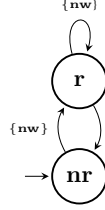


Fig. 6: Reader

**Counter System.** A *configuration* of a system  $A\|B^n$  is a tuple  $(q_A, \mathbf{c})$ , where  $q_A \in Q_A$ , and  $\mathbf{c} : Q_B \rightarrow \mathbb{N}_0$ . We identify  $\mathbf{c}$  with the vector  $(\mathbf{c}(q_0), \dots, \mathbf{c}(q_{|B|-1})) \in \mathbb{N}_0^{|B|}$ , 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 is 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 is in same state in  $g$ )

We say that the local transition  $(q_U, g, q'_U)$  is *enabled* in  $s$ .

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,
- $S_0 = \{\text{init}_A, \mathbf{c} \mid \mathbf{c}(q) = 0 \text{ if } q \neq \text{init}_B\}$  is the set of initial states,
- $\Delta$  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)  $q_A = q'_A \wedge \exists (q_i, g, q_j) \in \delta_B : \mathbf{c}(i) \geq 1 \wedge \mathbf{c}' = \mathbf{c} - \mathbf{u}_i + \mathbf{u}_j \wedge (q_A, \mathbf{c}) \models_{q_i} g$  (transition of a  $B$ -process)

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

<sup>2</sup>The form  $A\|B^n$  is only assumed for simplicity of presentation. Our results extend to systems with an arbitrary number of process templates.

Let  $s, s' \in S$  be states of  $M$ , and  $U \in \{A, B\}$ . For a transition  $(s, s') \in \Delta$  we also write  $s \rightarrow s'$ . 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  $\Delta^{\text{local}}(s) = \{t_U \mid s \xrightarrow{t_U} s'\}$ , i.e., the set of all enabled outgoing local transitions from  $s$ , and let  $\Delta(s, t_U) = s'$  if  $s \xrightarrow{t_U} s'$ . From now on we assume wlog. that each guard  $g \in \mathcal{G}$  is a singleton.<sup>3</sup>

**Runs.** A *path* of a counter system is a (finite or infinite) sequence of states  $x = s_1, s_2, \dots$  such that  $s_m \rightarrow s_{m+1}$  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  $s_1, s_2, \dots$  of the counter system 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  $E \subseteq S$ , an *error path* is a finite path that starts in an initial state and ends in  $E$ .

**The Parameterized Repair Problem.** Let  $M = (S, S_0, \Delta)$  be the counter system for process templates  $A = (Q_A, \text{init}_A, \mathcal{G}_A, \delta_A)$ ,  $B = (Q_B, \text{init}_B, \mathcal{G}_B, \delta_B)$ , 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  $A' = (Q_A, \text{init}_A, \mathcal{G}_A, \delta'_A)$  with  $\delta'_A \subseteq \delta_A$  and  $B' = (Q_B, \text{init}_B, \mathcal{G}_B, \delta'_B)$  with  $\delta'_B \subseteq \delta_B$  such that the counter system  $M'$  for  $A'$  and  $B'$  does not reach any state in  $ERR$ .

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  $\delta'$ , also denoted  $\text{Restrict}(M, \delta')$ .

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.

### III. PARAMETERIZED MODEL CHECKING OF DISJUNCTIVE SYSTEMS

In this section, we address research challenges **C1** and **C2**: after establishing that counter systems can be framed as well-structured transition systems (WSTS) (Sect. III-A), we introduce a parameterized model checking algorithm for disjunctive systems that suits our needs (Sect. III-B), and finally show how the algorithm can be modified to also check for the reachability of deadlocked states (Sect. III-C). Full proofs for the lemmas in this section can be found in the extended version [17].

#### A. Counter Systems 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  $s_0, s_1, \dots \in S^\omega$  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,  $\preceq$

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

is a wqo on  $S$  if and only if it is well-founded and has no infinite antichains.

**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 subset of minimal elements. We call this set the *minimal basis* of  $R$ , denoted  $\text{minBasis}(R)$ .

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

**WSTS [15].** 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  $\Delta$ .

*Lemma 1:* Let  $M = (S, S_0, \Delta)$  be a counter system for process templates  $A, B$ , and let  $\lesssim \subseteq S \times S$  be the binary relation defined by:

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

where  $\lesssim$  is the component-wise ordering of vectors. Then  $(M, \lesssim)$  is a WSTS.

**Predecessor, Effective pred-basis [16].** Let  $M = (S, S_0, \Delta)$  be a counter system and let  $R \subseteq S$ . Then the set of *immediate predecessors* of  $R$  is

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

A WSTS  $(M, \lesssim)$  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  $\uparrow \text{pred}(\uparrow R)$ . Note that, since  $\lesssim$  is strongly compatible with  $\Delta$ , if a set  $R \subseteq S$  is upward-closed with respect to  $\lesssim$  then  $\text{pred}(R)$  is also upward-closed.<sup>4</sup>

For backward reachability analysis, we want to compute  $\text{pred}^*(R)$  as the limit of the sequence  $R_0 \subseteq R_1 \subseteq \dots$  where  $R_0 = R$  and  $R_{i+1} = R_i \cup \text{pred}(R_i)$ . Note that if we have strong compatibility and effective pred-basis, we can compute  $\text{pred}^*(R)$  for any upward-closed set  $R$ . If we can furthermore check intersection of upward-closed sets with initial states (which is easy for counter systems), 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 2:* Let  $M = (S, S_0, \Delta)$  be a counter system for guarded process templates  $A, B$ . Then  $(M, \lesssim)$  has effective *pred-basis*.

<sup>4</sup>For a formal proof, check the extended version [17].

## B. Model Checking Algorithm

Our model checking algorithm is based on the known backwards reachability algorithm for WSTS [15]. We present it in detail to show how it stores intermediate results to return an *error sequence*, from which we derive concrete error paths.

---

### Algorithm 1 Parameterized Model Checking

---

```

1: procedure MODELCHECK(Counter System  $M, ERR$ )
2:    $tempSet \leftarrow ERR, E_0 \leftarrow ERR, i \leftarrow 1, visited \leftarrow \emptyset$ 
   // A fixed point is reached if  $visited = tempSet$ 
3:   while  $tempSet \neq visited$  do
4:      $visited \leftarrow tempSet$ 
5:      $E_i \leftarrow \text{minBasis}(\text{pred}(\uparrow E_{i-1}))$ 
6:   //  $\text{pred}$  is computed as in the proof of Lemma 2
7:     if  $E_i \cap S_0 \neq \emptyset$  then //intersect with initial states?
8:       return  $False, \{E_0, \dots, E_i \cap S_0\}$ 
9:      $tempSet \leftarrow \text{minBasis}(visited \cup E_i)$ 
10:     $i \leftarrow i + 1$ 
11:  return  $True, \emptyset$ 

```

---

Given a counter system  $M$  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*  $E_0, \dots, E_k$ , where  $E_0 = ERR$ ,  $\forall 0 < i < k : E_i = \text{minBasis}(\uparrow \text{pred}(\uparrow E_{i-1}))$ , and  $E_k = \text{minBasis}(\uparrow \text{pred}(\uparrow E_{k-1})) \cap S_0$ . 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. [15], and from Lemma 2. 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.

**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 counter system  $M$  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  $\uparrow E_0 = \{(w, (i_0, i_1)) \mid (w, (0, 1)) \lesssim (w, (i_0, i_1))\}$ , 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.

## C. 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  $\lesssim$  (defined in Sect. III-A): let  $s = (q_A, \mathbf{c}), r =$

$(q_A, \mathbf{d})$  be global states with  $s \lesssim r$ . 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.

A natural idea is to refine the wqo such that deadlocked states are upward closed. To this end, consider  $\lesssim_0 \subseteq \mathbb{N}_0^{|B|} \times \mathbb{N}_0^{|B|}$  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  $\lesssim_0 \subseteq S \times S$  where  $(q_A, \mathbf{c}) \lesssim_0 (q'_A, \mathbf{d}) \Leftrightarrow (q_A = q'_A \wedge \mathbf{c} \lesssim_0 \mathbf{d})$ .

Then, deadlocked states are upward closed with respect to  $\lesssim_0$ . However, it is not easy to adopt the WSTS approach to this case, since for our counter systems  $\text{pred}(R)$  will in general not be upward closed if  $R$  is upward closed. Instead of using  $\lesssim_0$  to define a WSTS, in the following we will use it to define a counter abstraction (similar to the approach of Pnueli et al. [18]) that can be used for deadlock detection.

The idea is that we use vectors with counter values from  $\{0, 1\}$  to represent their upward closure with respect to  $\lesssim_0$ . These upward closures will be seen as abstract states, and in the usual way define that a transition between abstract states  $\hat{s}, \hat{s}'$  exists iff there exists a transition between concrete states  $s \in \uparrow \hat{s}, s' \in \uparrow \hat{s}'$ . We formalize the abstract system in the following, assuming wlog. that  $\delta_B$  does not contain transitions of the form  $(q_i, \{q_i\}, q_j)$ , i.e., transitions from  $q_i$  that are guarded by  $q_i$ .<sup>5</sup>

**01-Counter System.** For a given counter system  $M$ , we define the 01-Counter System  $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\Delta})$ , where:

- $\hat{S} \subseteq Q_A \times \{0, 1\}^{|B|}$  is the set of states,
- $\hat{s}_0 = (\text{init}_A, \mathbf{c})$  with  $\mathbf{c}(q) = 1$  iff  $q = \text{init}_B$  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)

Define runs and deadlocks of a 01-counter system similarly as for counter systems. For a state  $s = (q_A, \mathbf{c})$  of  $M$ , define the corresponding abstract state of  $\hat{M}$  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.

*Theorem 1:* The 01-counter system  $\hat{M}$  has a deadlocked run if and only if the counter system  $M$  has a deadlocked run.

*Proof idea:* Suppose  $x = s_1, s_2, \dots, s_f$  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

<sup>5</sup>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.

easy to see that  $\hat{x} = \alpha(s_1), \alpha(s_2), \dots, \alpha(s_f)$  is a deadlocked run of  $\hat{M}$ .

Now, suppose  $\hat{x} = \hat{s}_1, \hat{s}_2, \dots, \hat{s}_f$  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  $t_1, \dots, t_{f-1}$  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  $s_1 = (\text{init}_A, \mathbf{c}_1)$  with  $\mathbf{c}_1(\text{init}_B) = 2^b$  and for every  $t_k$  in the sequence do:<sup>6</sup>

- 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  $i$ ).

By construction, after any of the transitions in  $t_1, \dots, t_{f-1}$ , 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 1:* Deadlock detection in disjunctive systems is decidable in EXPTIME (in  $|Q_B|$ ).

**An Algorithm for Deadlock Detection.** Now we can modify the model-checking algorithm to detect deadlocks in a 01-counter system  $\hat{M}$ : 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  $\text{pred}$ . Thus, an *error sequence* is of the form  $E_0, \dots, E_k$ , where  $E_0 = DEAD$ ,  $\forall 0 < i < k : E_i = \text{pred}(E_{i-1})$ , and  $E_k = E_{k-1} \cap S_0$ .

#### IV. PARAMETERIZED REPAIR ALGORITHM

Now, we can introduce a parameterized repair algorithm that interleaves the backwards model checking algorithm (Algorithm 1) with a forward reachability analysis and the computation of candidate repairs.

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

Given an error sequence  $E_0, \dots, E_k$ , let the *reachable error sequence*  $\mathcal{RE} = RE_0, \dots, RE_k$  be defined by  $RE_k = E_k$  (which by definition only contains initial states), and  $RE_{i-1} = \text{Succ}(RE_i) \cap \uparrow E_{i-1}$  for  $1 \leq i \leq k$ . That is, each  $RE_i$  contains a set of states that can reach  $\uparrow ERR$  in  $i$  steps, and are reachable from  $S_0$  in  $k - i$  steps. Thus, it 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  $\delta$  as atomic propositions, such that a satisfying assignment of the constraints corresponds to the candidate

<sup>6</sup>Note that a similar, but more involved construction is also possible with  $\mathbf{c}_1(\text{init}_B) = b$ .

repair, where only transitions that are assigned **true** remain in  $\delta'$ . 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 the extended version [17] for more examples).

**A Parameterized Repair Algorithm.** Given a counter system  $M$ , 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*  $\delta'$  or the string *Unrealizable* to denote that no repair exists.

#### Properties of Algorithm 2.

*Theorem 2 (Soundness):* For every repair  $\delta'$  returned by Algorithm 2:

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

*Proof:* The parameterized model checker guarantees that the transition relation is safe, i.e.,  $\uparrow ERR$  is not reachable. 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. ■

*Theorem 3 (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  $\delta'$ , seen as an assignment of truth values to atomic propositions  $\delta$ , was a satisfying assignment of  $accConstr \wedge initConstr$  up to this point, and is not anymore after this iteration.

If  $i = 0$ , i.e.,  $\delta'$  was never a satisfying assignment, then  $\delta'$  does not satisfy  $initConstr$  and can clearly not be a repair. If  $i > 0$ , then  $\delta'$  is a satisfying assignment for  $initConstr$  and all constraints added before round  $i$ , but not for the constraints  $\bigwedge_{s \in RE_k} BuildConstr(s, [RE_{k-1}, \dots, RE_0])$  added in this iteration of the loop, based on a reachable error sequence  $\mathcal{RE} = RE_k, \dots, RE_0$ . By construction of  $BuildConstr$ , this means we can construct out of  $\delta'$  and  $\mathcal{RE}$  a concrete error path in  $Restrict(M, \delta')$ , and  $\delta'$  can also not be a repair. ■

---

#### Algorithm 2 Parameterized Repair

---

```

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

1: procedure BUILDCONSTR(State  $s, \mathcal{RE}$ )
2:   //  $s$  is a state,  $\mathcal{RE}[1 : ]$  is a list obtained by removing
3:   // the first element from  $\mathcal{RE}$ 
4:   if  $\mathcal{RE}[1 : ]$  is empty then
5:     //if  $t_U \in \Delta^{local}(s)$  leads directly to error set, delete it ( $\neg t_U$ 
6:     // must set to true by the SAT solver)
7:     return  $\bigwedge_{t_U \in \Delta^{local}(s, \mathcal{RE}[0])} \neg t_U$ 
8:   else
9:     //else either delete  $t_U$  or delete outgoing transitions of the
10:    // target state of  $t_U$  recursively
11:    return  $\bigwedge_{t_U \in \Delta^{local}(s, \mathcal{RE}[0])} (\neg t_U \vee$ 
12:     $BuildConstr(\Delta(s, t_U), \mathcal{RE}[1 : ]))$ 

```

---

*Theorem 4 (Termination):* Algorithm 2 always terminates.

*Proof:* For a counter system 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. ■

**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  $m_e - 1$ .

## V. EXTENSIONS

### A. Beyond Reachability

Algorithm 2 can also be used for repair with respect to 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 counter system of process templates  $A$  and  $B$  that violates a safety property  $\varphi$  over the states of  $A$ , and let  $\mathcal{A} = (Q^A, q_0^A, Q_A, \delta^A, \mathcal{F})$  be the automaton that accepts all words over  $Q_A$  that violate  $\varphi$ . To repair  $M$ , the composition  $M \times \mathcal{A}$  and the set of error states  $ERR = \{((q_A, \mathbf{c}), q_{\mathcal{F}}^A) \mid (q_A, \mathbf{c}) \in S \wedge q_{\mathcal{F}}^A \in \mathcal{F}\}$  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:

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

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  $\varphi(A, B^{(k)})$  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  $M = (S, S_0, \Delta)$ ,  $B = (Q_B, \text{init}_B, \mathcal{G}_B, \delta_B)$ , and  $\mathcal{A} = (Q^A, q_0^A, Q_A \times Q_{B^k}, \delta^A, \mathcal{F})$  is the automaton that reads states of  $A \times B^k$  as actions and accepts all words that violate the property.<sup>7</sup>

**Example.** Consider again the simple reader-writer system in Figures 5 and 6 where we use the following abbreviations:  $(n)w$  for (non-)writing, and  $(n)r$  for (non-)reading. 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  $\varphi = G[(w \wedge nr_1) \implies (nr_1 Wnw)]$  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 II, 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  $\mathcal{C}$ ,  $((-, -, (*, *), q_2^A))$  (where  $(-, -)$  means any writer state and any reader state, and  $*$  means 0 or 1), and  $TRC\text{onstr}_{Disj}$ . The model checker

<sup>7</sup>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.

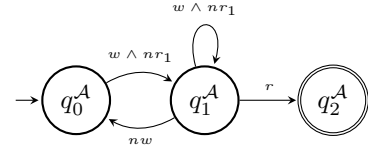


Fig. 7: Automaton for  $\neg\varphi$

in Line 4 may return the following error sequences, where we only consider states that didn't occur before:

$$\begin{aligned} E_0 &= \{((-, -, (*, *), q_2^A)\}, \\ E_1 &= \{((w, r_1, (0, 0)), q_1^A)\}, \\ E_2 &= \{((w, nr_1, (0, 0)), q_0^A), ((w, nr_1, (0, 1)), q_0^A), \\ &\quad ((w, nr_1, (1, 0)), q_0^A)\}, \\ E_3 &= \{((nw, nr_1, (0, 0)), q_0^A), ((nw, nr_1, (0, 1)), q_0^A), \\ &\quad ((w, r_1, (0, 0)), q_0^A), ((w, r_1, (0, 1)), q_0^A), ((w, r_1, (1, 0)), q_0^A)\} \end{aligned}$$

In Line 14 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.

### B. Beyond Disjunctive Systems

Furthermore, we have extended Algorithm 2 to other systems that can be framed as WSTS, in particular pairwise systems [2] and systems based on broadcasts or other global synchronizations [3], [19]. We summarize our results here, more details can be found in the extended version [17].

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  $\delta'$ , 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 TOWER-hard [20]. For broadcast protocols we can show that the situation is even worse:

*Theorem 5:* Deadlock detection in broadcast protocols is undecidable.

The main ingredient of the proof is the following lemma:

*Lemma 3:* 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 pairwise rendezvous systems.

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

- each transition only changes the vector  $\mathbf{c}$  in one of the following ways: (i) it adds to or subtracts from  $\mathbf{c}$  a unit vector, or (ii) it multiplies  $\mathbf{c}$  with a broadcast matrix  $M$  (this allows us to simulate every transition with a single transition in the broadcast system), and
- some configuration  $(q'_2, 0)$  is reachable from some configuration  $(q'_1, 0)$  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. Now, based on  $G^*$  we define process templates  $A$  and  $B$  such that  $A\|B^n$  can reach a deadlock iff  $(q'_2, 0)$  is reachable from  $(q'_1, 0)$  in  $G^*$ .

The states of  $A$  are the discrete states of  $G^*$ , plus additional states  $q', q''$ . If the state vector of  $G^*$  is  $m$ -dimensional, then  $B$  has states  $q_1, \dots, q_m$ , plus states  $\text{init}, v$ . 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  $\text{init}$  to  $q_i$  (if  $\mathbf{u}_i$  was added), or from  $q_i$  to  $\text{init}$  (if  $\mathbf{u}_i$  was subtracted). For every transition 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  $q_1, \dots, q_m$  that correspond to the effect of  $M$ .

The additional states  $q', q''$  of  $A$  are used to connect reachability of  $(q'_2, 0)$  to a deadlock in  $A\|B^n$  in the following way: (i) there are self-loops on all states of  $A$  except on  $q'$ , i.e., the system can only deadlock if  $A$  is in  $q'$ , (ii) there is a broadcast sending transition from  $q'_2$  to  $q'$  in  $A$ , which sends all  $B$ -processes that are in  $q_1, \dots, q_m$  to special state  $v$ , and (iii) from  $v$  there is a broadcast sending transition to  $\text{init}$  in  $B$ , and a corresponding receiving transition from  $q'$  to  $q''$  in  $A$ . Thus,  $A\|B^n$  can only deadlock in a configuration where  $A$  is in  $q'$  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  $q_1, \dots, q_m$ . Letting  $q_1$  be the initial state of  $A$  and  $\text{init}$  the initial state of  $B$ , such a configuration is reachable in  $A\|B^n$  if and only if  $(q'_2, 0)$  is reachable from  $(q'_1, 0)$  in  $G^*$ . ■

**Approximate Methods for Deadlock Detection.** Since solving the problem exactly is impractical or impossible in general, we propose to use approximate methods. For pairwise systems, the 01-counter system introduced as a precise abstraction for disjunctive systems in Sect. III-C 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 systems, for which the problem is decidable [21].<sup>8</sup> Another alternative is to add initial constraints that restrict the repair algorithm and imply deadlock-freedom.

<sup>8</sup>Note that in the terminology of Delzanno et al., deadlock detection is a special case of the TARGET problem.

## VI. IMPLEMENTATION & EVALUATION

We have implemented a prototype of our parameterized repair algorithm that supports the three types of systems (disjunctive, pairwise and broadcast), and safety and reachability properties. For disjunctive and pairwise systems, we have evaluated it on different variants of reader-writer-protocols, based on the ones given in Sect. I,II, 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 systems, these benchmarks are denoted “RW*i* (PR)” in Table I. A detailed treatment of one benchmark, including an explanation of the whole repair process is given in the extended version [17].

For broadcast protocols, we have evaluated our algorithm on a range of more complex benchmarks taken from the parameterized verification literature [22]: a distributed **Lock Service** (DLS) inspired by the Chubby protocol [23], a distributed **Robot Flocking** protocol (RF) [24], a distributed **Smoke Detector** (SD) [19], a sensor network implementing a **Two-Object Tracker** (TOT) [25], and the cache coherence protocol **MESI** [26] 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 I 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.

## VII. RELATED WORK

Many automatic repair approaches have been considered in the literature, most of them restricted to monolithic systems [11], [12], [27]–[30]. 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 [31], [32]. Others are based on a similar underlying counterexample-guided synthesis/repair principle, but differ in



TABLE I: Running time, number of iterations, and number of deleted transitions (#D.T.) for the different configurations. Each benchmark is listed with its number of local states, and edges. We evaluated the algorithms on different sets of errors with  $P_1 \cup P_2 = C$  where  $P_1$  and  $P_2$  are two distinct error sets that differ from one benchmark to another. Smallest number of iterations, runtime per benchmark, deleted transitions are highlighted in boldface.

Benchmark	Size		Errors	[SEP=F & EPT=F]			[SEP=T & EPT=F]			[SEP=F & EPT=T]			[SEP=T & EPT=T]		
	States	Edges		#Iter	Time	#D.T.	#Iter	Time	#D.T.	#Iter	Time	#D.T.	#Iter	Time	#D.T.
RW1 (PW)	5	12	C	3	2.5	4	3	2.9	4	<b>2</b>	<b>1.7</b>	<b>2</b>	<b>2</b>	<b>1.7</b>	<b>2</b>
RW2 (PW)	15	42	C	3	3.8	14	3	4.8	14	<b>2</b>	<b>3.2</b>	<b>7</b>	7	8.4	<b>7</b>
RW3 (PW)	35	102	C	3	820.7	34	3	<b>7.6</b>	34	<b>2</b>	552.3	<b>17</b>	17	40.3	<b>17</b>
RW4 (PW)	45	132	C	TO	TO	TO	<b>3</b>	<b>11.8</b>	44	TO	TO	TO	22	99.2	<b>22</b>
DLS	10	95	P1	<b>1</b>	<b>0.8</b>	13	<b>1</b>	<b>0.8</b>	13	3	2.4	<b>5</b>	5	5.6	<b>5</b>
DLS	10	95	P2	<b>1</b>	<b>0.8</b>	13	2	1.7	13	3	2.6	<b>9</b>	7	5.5	<b>9</b>
DLS	10	95	C	<b>2</b>	4.2	13	<b>2</b>	<b>1.5</b>	13	3	3	<b>9</b>	9	8.1	<b>9</b>
RF	10	147	P1	<b>1</b>	2.5	32	<b>1</b>	<b>1.2</b>	32	TO	TO	TO	8	12.4	<b>13</b>
RF	10	147	P2	<b>1</b>	<b>1.2</b>	32	<b>1</b>	1.3	32	TO	TO	TO	8	11.3	<b>14</b>
RF	10	147	C	<b>1</b>	7.8	32	<b>1</b>	<b>1.4</b>	32	TO	TO	TO	8	12.5	<b>12</b>
SD	6	39	C	<b>1</b>	<b>1</b>	<b>4</b>	<b>1</b>	<b>1</b>	<b>4</b>	3	2.4	<b>4</b>	3	3	<b>4</b>
2OT	12	128	P1	12	18.8	26	<b>6</b>	<b>8.3</b>	26	16	73.8	<b>17</b>	16	34	<b>17</b>
2OT	12	128	P2	<b>1</b>	<b>1.8</b>	26	<b>1</b>	<b>1.8</b>	26	4	2958	<b>11</b>	8	16.5	12
2OT	12	128	C	11	17.2	Unreal.	<b>6</b>	<b>11.7</b>	Unreal.	TO	TO	TO	11	48.6	Unreal.
MESI1	4	26	C	<b>1</b>	2.4	6	<b>1</b>	<b>0.9</b>	6	2	1.8	<b>5</b>	4	3.5	<b>5</b>
MESI2	9	71	C	<b>1</b>	<b>1.1</b>	26	<b>1</b>	<b>1.1</b>	26	3	56.4	20	6	6.8	<b>15</b>
MESI3	14	116	C	<b>1</b>	109.4	46	<b>1</b>	<b>108.1</b>	46	TO	TO	TO	6	289.9	<b>15</b>

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], [33]. *Assume-Guarantee-Repair* [34] 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* [35] 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], [33], [34], or more expressive specifications and other features like partial information [31], [32].

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 is restricted to 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 [36]–[38].

### VIII. CONCLUSION AND FUTURE WORK

We have investigated the parameterized repair problem for systems of the form  $A \parallel B^n$  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, and broadcast 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 [39], and advanced parameterized model checking techniques, e.g., *cutoff* results for disjunctive systems [6], [40], [41], or recent *pruning* results for immediate observation Petri nets, which model exactly the class of disjunctive systems [42].

### REFERENCES

- [1] I. Suzuki, "Proving properties of a ring of finite state machines," *Inf. Process. Lett.*, vol. 28, no. 4, pp. 213–214, 1988.

- [2] S. M. German and A. P. Sistla, "Reasoning about systems with many processes," *J. ACM*, vol. 39, no. 3, pp. 675–735, 1992.
- [3] J. Esparza, A. Finkel, and R. Mayr, "On the verification of broadcast protocols," in *LICS*. IEEE Computer Society, 1999, pp. 352–359.
- [4] E. A. Emerson and V. Kahlon, "Reducing model checking of the many to the few," in *CADE*, ser. LNCS, vol. 1831. Springer, 2000, pp. 236–254.
- [5] E. A. Emerson and K. S. Namjoshi, "On reasoning about rings," *Foundations of Computer Science*, vol. 14, no. 4, pp. 527–549, 2003.
- [6] E. A. Emerson and V. Kahlon, "Model checking guarded protocols," in *LICS*. IEEE Computer Society, 2003, pp. 361–370.
- [7] E. M. Clarke, M. Talupur, T. Touili, and H. Veith, "Verification by network decomposition," in *CONCUR*, ser. LNCS, vol. 3170. Springer, 2004, pp. 276–291.
- [8] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin, "Parameterized model checking of token-passing systems," in *VMCAI*, ser. LNCS, vol. 8318. Springer, 2014, pp. 262–281.
- [9] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith, "Parameterized model checking of rendezvous systems," in *CONCUR*, ser. LNCS, vol. 8704. Springer, 2014, pp. 109–124.
- [10] B. Aminof and S. Rubin, "Model checking parameterised multi-token systems via the composition method," in *IJCAR*, ser. LNCS, vol. 9706. Springer, 2016, pp. 499–515.
- [11] B. Jobstmann, A. Griesmayer, and R. Bloem, "Program repair as a game," in *17th Conference on Computer Aided Verification (CAV'05)*. Springer, 2005, pp. 226–238, LNCS 3576.
- [12] P. C. Attie, K. D. A. Bab, and M. Sakr, "Model and program repair via sat solving," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, pp. 1–25, 2017.
- [13] R. Bloem, G. Hofferek, B. Könighofer, R. Könighofer, S. Außerlechner, and R. Spörk, "Synthesis of synchronization using uninterpreted functions," in *2014 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2014, pp. 35–42.
- [14] J. McClurg, H. Hojjat, and P. Černý, "Synchronization synthesis for network programs," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 301–321.
- [15] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay, "General decidability theorems for infinite-state systems," in *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1996, pp. 313–321.
- [16] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!" *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63–92, 2001.
- [17] S. Jacobs, M. Sakr, and M. Völpl, "Parameterized repair of concurrent systems," 2021. [Online]. Available: <https://doi.org/10.48550/arXiv.2111.03322>
- [18] A. Pnueli, J. Xu, and L. D. Zuck, "Liveness with (0, 1, infity)-counter abstraction," in *CAV*, ser. Lecture Notes in Computer Science, vol. 2404. Springer, 2002, pp. 107–122.
- [19] N. Jaber, S. Jacobs, C. Wagner, M. Kulkarni, and R. Samanta, "Parameterized verification of systems with global synchronization and guards," in *CAV (1)*, ser. Lecture Notes in Computer Science, vol. 12224. Springer, 2020, pp. 299–323.
- [20] W. Czerwinski, S. Lasota, R. Lazic, J. Leroux, and F. Mazowiecki, "The reachability problem for petri nets is not elementary," *J. ACM*, vol. 68, no. 1, pp. 7:1–7:28, 2021.
- [21] G. Delzanno, A. Sangnier, and G. Zavattaro, "Parameterized verification of ad hoc networks," in *CONCUR*, ser. LNCS, vol. 6269. Springer, 2010, pp. 313–327.
- [22] N. Jaber, C. Wagner, S. Jacobs, M. Kulkarni, and R. Samanta, "Quicksilver: modeling and parameterized verification for distributed agreement-based systems," *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, pp. 1–31, 2021.
- [23] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *OSDI*. USENIX Association, 2006, pp. 335–350.
- [24] D. Canepa and M. G. Potop-Butucaru, "Stabilizing flocking via leader election in robot networks," in *SSS*, ser. Lecture Notes in Computer Science, vol. 4838. Springer, 2007, pp. 52–66.
- [25] C. Chang and J. Tsai, "Distributed collaborative surveillance system based on leader election protocols," *IET Wirel. Sens. Syst.*, vol. 6, no. 6, pp. 198–205, 2016.
- [26] E. A. Emerson and V. Kahlon, "Exact and efficient verification of parameterized cache coherence protocols," in *CHARME*, ser. LNCS, vol. 2860. Springer, 2003, pp. 247–262.
- [27] B. Demsky and M. Rinard, "Automatic detection and repair of errors in data structures," in *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03)*, 2003, pp. 78–95.
- [28] A. Griesmayer, R. Bloem, and B. Cook, "Repair of Boolean programs with an application to C," in *18th Conference on Computer Aided Verification (CAV'06)*, 2006, pp. 358–371, LNCS 4144.
- [29] S. Forrest, T. Nguyen, W. Weimer, and C. L. Goues, "A genetic programming approach to automated software repair," in *Genetic and Evolutionary Computation Conference (GECCO'09)*. ACM, 2009, pp. 947–954.
- [30] M. Monperrus, "Automatic software repair: A bibliography," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 17:1–17:24, 2018.
- [31] B. Finkbeiner and S. Schewe, "Bounded synthesis," *STTT*, vol. 15, no. 5-6, pp. 519–539, 2013.
- [32] S. Bansal, K. S. Namjoshi, and Y. Sa'ar, "Synthesis of coordination programs from linear temporal specifications," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, pp. 54:1–54:27, 2020. [Online]. Available: <https://doi.org/10.1145/3371122>
- [33] M. Vechev, E. Yahav, and G. Yorsh, "Abstraction-guided synthesis of synchronization," in *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2010, pp. 327–338.
- [34] H. Frenkel, O. Grumberg, C. Pasareanu, and S. Sheinvald, "Assume, guarantee or repair," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 211–227.
- [35] B. Finkbeiner and S. Jacobs, "Lazy synthesis," in *VMCAI*, ser. LNCS, vol. 7148. Springer, 2012, pp. 219–234.
- [36] J. Esparza, "Keeping a crowd safe: On the complexity of parameterized verification (invited talk)," in *STACS*, ser. LIPIcs, vol. 25. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 1–10.
- [37] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder, *Decidability of Parameterized Verification*, ser. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- [38] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of model checking*. Springer, 2018, vol. 10.
- [39] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening, "Symbolic counter abstraction for concurrent software," in *International Conference on Computer Aided Verification*. Springer, 2009, pp. 64–78.
- [40] S. Außerlechner, S. Jacobs, and A. Khalimov, "Tight cutoffs for guarded protocols with fairness," in *VMCAI*, ser. LNCS, vol. 9583. Springer, 2016, pp. 476–494.
- [41] S. Jacobs and M. Sakr, "Analyzing guarded protocols: Better cutoffs, more systems, more expressivity," in *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 2018, pp. 247–268.
- [42] J. Esparza, M. A. Raskin, and C. Weil-Kennedy, "Parameterized analysis of immediate observation petri nets," in *Petri Nets*, ser. Lecture Notes in Computer Science, vol. 11522. Springer, 2019, pp. 365–385.