Tight Cutoffs for Guarded Protocols with Fairness

Simon Außerlechner[1], Swen Jacobs[2], Ayrat Khalimov[1]

[1] IAIK, Graz University of Technology, Austria
[2] Reactive Systems Group, Saarland University, Germany

Abstract. Guarded protocols were introduced in a seminal paper by Emerson and Kahlon (2000), and describe systems of processes whose transitions are enabled or disabled depending on the existence of other processes in certain local states. We study parameterized model checking and synthesis of guarded protocols, both aiming at formal correctness arguments for systems with any number of processes. Cutoff results reduce reasoning about systems with an arbitrary number of processes to systems of a determined, fixed size. Our work stems from the observation that existing cutoff results for guarded protocols i) are restricted to closed systems, and ii) are of limited use for liveness properties because reductions do not preserve fairness. We close these gaps and obtain new cutoff results for open systems with liveness properties under fairness assumptions. Furthermore, we obtain cutoffs for the detection of global and local deadlocks, which are of paramount importance in synthesis. Finally, we prove tightness or asymptotic tightness for the new cutoffs.

1 Introduction

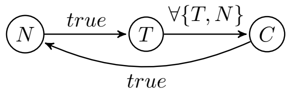
Concurrent hardware and software systems are notoriously hard to get correct. Formal methods like model checking or synthesis can be used to guarantee correctness, but the state explosion problem prevents us from using such methods for systems with a large number of components. Furthermore, correctness properties are often expected to hold for an *arbitrary* number of components. Both problems can be solved by *parameterized* model checking and synthesis approaches, which give correctness guarantees for systems with any number of components without considering every possible system instance explicitly.

While parameterized model checking (PMC) is undecidable in general [25], there exist a number of methods that decide the problem for specific classes of systems [12, 14, 16], as well as semi-decision procedures that are successful in many interesting cases [9, 18, 21]. In this paper, we consider the *cutoff* method that can guarantee properties of systems of arbitrary size by considering only systems of up to a certain fixed size, thus providing a decision procedure for PMC if components are finite-state.

We consider systems that are composed of an arbitrary number of processes, each an instance of a process template from a given, finite set. Process templates can be viewed as synchronization skeletons [11], i.e., program abstractions that suppress information not necessary for synchronization. In our system model, processes communicate by guarded updates, where guards are statements about other processes that are interpreted either conjunctively ("every other process satisfies the guard") or disjunctively ("there exists a process that satisfies the guard"). Conjunctive guards can model atomic sections or locks, disjunctive guards can model token-passing or to some extent pairwise rendezvous (cf. [13]).

This class of systems has been studied by Emerson and Kahlon [12], and cutoffs that depend on the size of process templates are known for specifications of the form $\forall \bar{p}.\ \Phi(\bar{p})$, where $\Phi(\bar{p})$ is an $\mathsf{LTL}\backslash\mathsf{X}$ property over the local states of one or more processes $\bar{p}$. Note that this does not allow us to specify fairness assumptions, for two reasons: i) to specify fairness, additional atomic propositions for enabledness and scheduling of processes are needed, and ii) specifications with global fairness assumptions are of the form $(\forall \bar{p}.\ \mathit{fair}(\bar{p})) \rightarrow (\forall \bar{p}.\ \Phi(\bar{p}))$. Because neither is supported by [12], the existing cutoffs are of limited use for reasoning about liveness properties.

Emerson and Kahlon [12] mentioned this limitation and illustrated it using the process template on the figure on the right. Transitions from the initial state N to the "trying" state T, and from the critical state C to N are always possible, while the transition from T to C is only possible if no other process is in C. The existing cutoff results can be used to prove safety properties like mutual exclusion for systems composed of arbitrarily many copies of this template. However, they cannot be used to prove starvation-freedom properties like $\forall p.\ \mathsf{A}\,\mathsf{G}(T_p \rightarrow \mathsf{F}\,C_p)$, stating that every process p that enters its local state T_p will eventually enter state C_p, because without fairness of scheduling the property does not hold.



Also, Emerson and Kahlon [12] consider only closed systems. Therefore, in this example, processes always try to enter C. In contrast, in open systems the transition to T might be a reaction to a corresponding input from the environment that makes entering C necessary. While it is possible to convert an open system to a closed system that is equivalent under LTL properties, this comes at the cost of a blow-up.

Motivation. Our work is inspired by applications in parameterized synthesis [17], where the goal is to automatically construct process templates such that a given specification is satisfied in systems with an arbitrary number of components. In this setting, one generally considers *open systems* that interact with an uncontrollable environment, and most specifications contain liveness properties that cannot be guaranteed without fairness assumptions. Also, one is in general interested in synthesizing deadlock-free systems. *Cutoffs* are essential for parameterized synthesis, and we will show in Sect. 4 how size-dependent cutoffs can be integrated into the parameterized synthesis approach.

Contributions.

– We show that existing cutoffs for model checking of $\mathsf{LTL}\backslash\mathsf{X}$ properties are in general not sufficient for systems with *fairness assumptions*, and provide new cutoffs for this case.

– We improve some of the existing cutoff results, and give separate cutoffs for the problem of *deadlock detection*, which is closely related to fairness.
– We prove *tightness* or asymptotical tightness for all of our cutoffs, showing that smaller cutoffs cannot exist with respect to the parameters we consider.

Moreover, all of our cutoffs directly support *open systems*, where each process may communicate with an adversarial environment. This makes the blow-up incurred by translation to an equivalent closed system unnecessary. The results presented here are based on a more detailed preliminary version of this paper [4].

2 Related Work

As mentioned, we extend the results of Emerson and Kahlon [12] who study PMC of guarded protocols, but do not support fairness assumptions, nor provide cutoffs for deadlock detection. In [13] they extended their work to systems with limited forms of guards and broadcasts, and also proved undecidability of PMC of conjunctive guarded protocols wrt. LTL (including X), and undecidability wrt. $\mathsf{LTL}\backslash\mathsf{X}$ for systems with both conjunctive and disjunctive guards.

Bouajjani et al. [7] study parameterized model checking of resource allocation systems (RASs). Such systems have a bounded number of resources, each owned by at most one process at any time. Processes are pushdown automata, and can request resources with high or normal priority. RASs are similar to conjunctive guarded protocols in that certain transitions are disabled unless a processes has a certain resource. RASs without priorities and with processes being finite state automata can be converted to conjunctive guarded protocols (at the price of blow up), but not vice versa. The authors study parameterized model checking wrt. $\mathsf{LTL}\backslash\mathsf{X}$ properties on arbitrary or on strong-fair runs, and (local or global) deadlock detection. The proof structure resembles that of [12], as does ours.

German and Sistla [16] considered global deadlocks and strong fairness properties for systems with pairwise rendezvous communication in a clique. Emerson and Kahlon [13] have shown that disjunctive guard systems can be reduced to such pairwise rendezvous systems. However, German and Sistla [16] do not provide cutoffs, nor do they consider local deadlocks, and their specifications can talk about one process only. Aminof et al. [3] have recently extended these results to more general topologies, and have shown that for some decidable PMC problems there are no cutoffs, even in cliques.

Emerson and Namjoshi provide cutoffs for systems that pass a valueless token in a ring [14], which is essentially resource allocation of a single resource with a specific allocation scheme. Their results have been extended to more general topologies [2, 10]. All of these results consider fairness of token passing in the sense that every process receives the token infinitely often.

Many of the decidability results above have recently been surveyed by Bloem et al [6]. In addition, there are many methods based on semi-algorithms.

"Dynamic cutoff" approaches [1, 18] support larger classes of systems, and try to find cutoffs for a concrete system and specification. These methods can find smaller cutoffs than those that are statically determined for a whole class of systems and specifications, but are currently limited to safety properties. The invisible invariants method [23] tries to find invariants in small systems, and applies a specialized cutoff result to prove correctness of all instances, including an extension to liveness properties [15].

Finally, there are methods that work completely without cutoffs, like regular model checking [8], network invariants [19, 21, 26], and counter abstraction [24]. They are in general incomplete, but may provide decision procedures for certain classes of systems and specifications, and support liveness to some extent.

3 Preliminaries

3.1 System Model

We consider systems $A \| B^n$, usually written $(A,B)^{(1,n)}$, consisting of one copy of a process template A and n copies of a process template B, in an interleaving parallel composition.We distinguish objects that belong to different templates by indexing them with the template. E.g., for process template $U \in \{A,B\}$, Q_U is the set of states of U. For this section, fix two disjoint finite sets Q_A, Q_B as sets of states of process templates A and B, and a positive integer n.

Processes. A *process template* is a transition system $U = (Q, \mathsf{init}, \Sigma, \delta)$ with

- Q is a finite set of states including the initial state init,
- Σ is a finite input alphabet,
- $\delta : Q \times \Sigma \times \mathcal{P}(Q_A \dot\cup Q_B) \times Q$ is a guarded transition relation.

A process template is *closed* if $\Sigma = \emptyset$, and otherwise *open*.

We define the size $|U|$ of a process template $U \in \{A,B\}$ as $|Q_U|$. A copy of template U will be called a U*-process*. Different B-processes are distinguished by subscript, i.e., for $i \in [1..n]$, B_i is the ith copy of B, and q_{B_i} is a state of B_i. A state of the A-process is denoted by q_A.

For the rest of this subsection, fix templates A and B. We assume that $\Sigma_A \cap \Sigma_B = \emptyset$. We will also write p for a process in $\{A, B_1, \ldots, B_n\}$, unless p is specified explicitly.

Disjunctive and Conjunctive Systems. In a system $(A,B)^{(1,n)}$, consider global state $s = (q_A, q_{B_1}, \ldots, q_{B_n})$ and global input $e = (\sigma_A, \sigma_{B_1}, \ldots, \sigma_{B_n})$. We also write $s(p)$ for q_p, and $e(p)$ for σ_p. A local transition $(q_p, \sigma_p, g, q'_p) \in \delta_U$ of p is *enabled for s and e* if its *guard* g is satisfied for p in s, written $(s,p) \models g$. Disjunctive and conjunctive systems are distinguished by the *interpretation of guards*:

$$\text{In disjunctive systems: } (s,p) \models g \quad \text{iff} \quad \exists p' \in \{A, B_1, \ldots, B_n\} \setminus \{p\} : \quad q_{p'} \in g.$$
$$\text{In conjunctive systems: } (s,p) \models g \quad \text{iff} \quad \forall p' \in \{A, B_1, \ldots, B_n\} \setminus \{p\} : \quad q_{p'} \in g.$$

Note that we check containment in the guard (disjunctively or conjunctively) only for local states of processes *different from p*. A process is *enabled* for s and e if at least one of its transitions is enabled for s and e, otherwise it is *disabled*.

Like Emerson and Kahlon [12], we assume that in conjunctive systems init_A and init_B are contained in all guards, i.e., they act as neutral states. Furthermore, we call a conjunctive system 1-*conjunctive* if every guard is of the form $(Q_A \dot{\cup} Q_B) \setminus \{q\}$ for some $q \in Q_A \dot{\cup} Q_B$.

Then, $(A,B)^{(1,n)}$ is defined as the transition system $(S, \mathsf{init}_S, E, \Delta)$ with

- set of global states $S = (Q_A) \times (Q_B)^n$,
- global initial state $\mathsf{init}_S = (\mathsf{init}_A, \mathsf{init}_B, \ldots, \mathsf{init}_B)$,
- set of global inputs $E = (\Sigma_A) \times (\Sigma_B)^n$,
- and global transition relation $\Delta \subseteq S \times E \times S$ with $(s, e, s') \in \Delta$ iff
  i) $s = (q_A, q_{B_1}, \ldots, q_{B_n})$,
  ii) $e = (\sigma_A, \sigma_{B_1}, \ldots, \sigma_{B_n})$, and
  iii) s' is obtained from s by replacing one local state q_p with a new local state q'_p, where p is a U-process with local transition $(q_p, \sigma_p, g, q'_p) \in \delta_U$ and $(s,p) \models g$.

We say that a system $(A,B)^{(1,n)}$ is *of type* (A,B). It is called a *conjunctive system* if guards are interpreted conjunctively, and a *disjunctive system* if guards are interpreted disjunctively. A system is *closed* if all of its templates are closed. We often denote the set $\{B_1, ..., B_n\}$ as $\mathcal{B}$.

Runs. A *configuration* of a system is a triple (s, e, p), where $s \in S$, $e \in E$, and p is either a system process, or the special symbol $\bot$. A *path* of a system is a configuration sequence $x = (s_1, e_1, p_1), (s_2, e_2, p_2), \ldots$ such that for all $m < |x|$ there is a transition $(s_m, e_m, s_{m+1}) \in \Delta$ based on a local transition of process p_m. We say that process p_m *moves* at *moment* m. Configuration $(s, e, \bot)$ appears iff all processes are disabled for s and e. Also, for every p and $m < |x|$: either $e_{m+1}(p) = e_m(p)$ or process p moves at moment m. That is, the environment keeps input to each process unchanged until the process can read it.[1]

A system *run* is a maximal path starting in the initial state. Runs are either infinite, or they end in a configuration $(s, e, \bot)$. We say that a run is *initializing* if every process that moves infinitely often also visits its init infinitely often.

Given a system path $x = (s_1, e_1, p_1), (s_2, e_2, p_2), \ldots$ and a process p, the *local path* of p in x is the projection $x(p) = (s_1(p), e_1(p)), (s_2(p), e_2(p)), \ldots$ of x onto local states and inputs of p. Similarly define the projection on two processes p_1, p_2 denoted by $x(p_1, p_2)$.

Deadlocks and Fairness. A run is *globally deadlocked* if it is finite. An infinite run is *locally deadlocked* for process p if there exists m such that p is disabled for all $s_{m'}, e_{m'}$ with $m' \geq m$. A run is *deadlocked* if it is locally or globally deadlocked. A system *has a (local/global) deadlock* if it has a (locally/globally) deadlocked run. Note that absence of local deadlocks for all p implies absence of global deadlocks, but not the other way around.

[1] By only considering inputs that are actually processed, we approximate an action-based semantics. Paths that do not fulfill this requirement are not very interesting, since the environment can violate any interesting specification that involves input signals by manipulating them when the corresponding process is not allowed to move.

A run $(s_1, e_1, p_1), (s_2, e_2, p_2), ...$ is *unconditionally-fair* if every process moves infinitely often. A run is *strong-fair* if it is infinite and for every process p, if p is enabled infinitely often, then p moves infinitely often. We will discuss the role of deadlocks and fairness in synthesis in Sect. 4.

Remark 1. Why do we consider systems $A \| B^n$? Emerson and Kahlon [12] showed how to generalize cutoffs for such systems to systems of the form $A^m \| B^n$, and further to systems with an arbitrary number of process templates $U_1^{n_1} \| \ldots \| U_m^{n_m}$. This generalization also works for our new results, except for the cutoffs for deadlock detection that are restricted to 1-conjunctive systems (see Section 5).

3.2 Specifications

Fix templates (A, B). We consider formulas in $\mathsf{LTL}\backslash\mathsf{X}$, i.e., LTL without the next-time operator X. Let $h(A, B_{i_1}, \ldots, B_{i_k})$ be an $\mathsf{LTL}\backslash\mathsf{X}$ formula over atomic propositions from $Q_A \cup \Sigma_A$ and indexed propositions from $(Q_B \cup \Sigma_B) \times \{i_1, \ldots, i_k\}$. For a system $(A, B)^{(1,n)}$ with $n \geq k$ and $i_j \in [1..n]$, satisfaction of $\mathsf{A}\, h(A, B_{i_1}, \ldots, B_{i_k})$ and $\mathsf{E}\, h(A, B_{i_1}, \ldots, B_{i_k})$ is defined in the usual way (see e.g. [5]).

Parameterized Specifications. A *parameterized specification* is a temporal logic formula with indexed atomic propositions and quantification over indices. We consider formulas of the forms $\forall i_1, \ldots, i_k.\, \mathsf{A}\, h(A, B_{i_1}, \ldots, B_{i_k})$ and $\forall i_1, \ldots, i_k.\, \mathsf{E}\, h(A, B_{i_1}, \ldots, B_{i_k})$. For given $n \geq k$,

$$(A, B)^{(1,n)} \models \forall i_1, \ldots, i_k.\, \mathsf{A}\, h(A, B_{i_1}, \ldots, B_{i_k})$$

iff

$$(A, B)^{(1,n)} \models \bigwedge_{j_1 \neq \ldots \neq j_k \in [1..n]} \mathsf{A}\, h(A, B_{j_1}, \ldots, B_{j_k}).$$

By symmetry of guarded protocols, this is equivalent (cp. [12]) to $(A, B)^{(1,n)} \models \mathsf{A}\, h(A, B_1, \ldots, B_k)$. The latter formula is denoted by $\mathsf{A}\, h(A, B^{(k)})$, and we often use it instead of the original $\forall i_1, \ldots, i_k.\, \mathsf{A}\, h(A, B_{i_1}, ..., B_{i_k})$. For formulas with path quantifier E, satisfaction is defined analogously, and equivalent to satisfaction of $\mathsf{E}\, h(A, B^{(k)})$.

Specification of Fairness and Local Deadlocks. It is often convenient to express fairness assumptions and local deadlocks as parameterized specifications. To this end, define auxiliary atomic propositions move_p and en_p for every process p of system $(A, B)^{(1,n)}$. At moment m of a given run $(s_1, e_1, p_1), (s_2, e_2, p_2), \ldots$, let move_p be true whenever $p_m = p$, and let en_p be true if p is enabled for s_m, e_m. Note that we only allow the use of these propositions to define fairness, but not in general specifications. Then, an infinite run is

- *local-deadlock-free* if it satisfies $\forall p.\, \mathsf{GF}\, \mathsf{en}_p$, abbreviated as $\Phi_{\neg dead}$,
- *strong-fair* if it satisfies $\forall p.\, \mathsf{GF}\, \mathsf{en}_p \rightarrow \mathsf{GF}\, \mathsf{move}_p$, abbreviated as Φ_{strong}, and
- *unconditionally-fair* if it satisfies $\forall p.\, \mathsf{GF}\, \mathsf{move}_p$, abbreviated as Φ_{uncond}.

If *fair* is a fairness notion and $\mathsf{A}\, h(A, B^{(k)})$ a specification, then we write $\mathsf{A}_{fair}\, h(A, B^{(k)})$ for $\mathsf{A}(\Phi_{fair} \rightarrow h(A, B^{(k)}))$. Similarly, we write $\mathsf{E}_{fair}\, h(A, B^{(k)})$ for $\mathsf{E}(\Phi_{fair} \wedge h(A, B^{(k)}))$.

3.3 Model Checking and Synthesis Problems

For a given system $(A, B)^{(1,n)}$ and specification $h(A, B^{(k)})$ with $n \geq k$,

- the *model checking problem* is to decide whether $(A, B)^{(1,n)} \models \mathsf{A}\, h(A, B^{(k)})$,
- the *deadlock detection problem* is to decide whether $(A, B)^{(1,n)}$ does not have global nor local deadlocks,
- the *parameterized model checking problem* (PMCP) is to decide whether $\forall m \geq n :\ (A, B)^{(1,m)} \models \mathsf{A}\, h(A, B^{(k)})$, and
- the *parameterized deadlock detection problem* is to decide whether for all $m \geq n$, $(A, B)^{(1,m)}$ does not have global nor local deadlocks.

For a given number $n \in \mathbb{N}$ and specification $h(A, B^{(k)})$ with $n \geq k$,

- the *template synthesis problem* is to find process templates A, B such that $(A, B)^{(1,n)} \models \mathsf{A}\, h(A, B^{(k)})$ and $(A, B)^{(1,n)}$ does not have global deadlocks.
- the *bounded template synthesis problem* for a pair of bounds $(b_A, b_B) \in \mathbb{N} \times \mathbb{N}$ is to solve the template synthesis problem with $|A| \leq b_A$ and $|B| \leq b_B$.
- the *parameterized template synthesis problem* is to find process templates A, B such that $\forall m \geq n :\ (A, B)^{(1,m)} \models \mathsf{A}\, h(A, B^{(k)})$ and $(A, B)^{(1,m)}$ does not have global deadlocks.

These definitions can be flavored with different notions of fairness (and similarly for the E path quantifier). In the next section we clarify the problems studied.

4 Reduction Method and Challenges

We show how to use existing cutoff results of Emerson and Kahlon [12] to reduce the PMCP to a standard model checking problem, and parameterized synthesis to template synthesis. We note the limitations of the existing results that are crucial in the context of synthesis.

Reduction by Cutoffs. A *cutoff* for a system type (A, B) and a specification Φ is a number $c \in \mathbb{N}$ such that:

$$\forall n \geq c : \Big((A, B)^{(1,n)} \models \Phi \quad \Leftrightarrow \quad (A, B)^{(1,c)} \models \Phi \Big) .$$

Similarly, $c \in \mathbb{N}$ is a *cutoff for (local/global) deadlock detection* if $\forall n \geq c$: $(A, B)^{(1,n)}$ has a (local/global) deadlock iff $(A, B)^{(1,c)}$ has a (local/global) deadlock. For the systems and specifications presented in this paper, cutoffs can be computed from the size of process template B and the number k of copies of B mentioned in the specification, and are given as expressions like $|B| + k + 1$.

Remark 2. Our definition of a cutoff is different from that of Emerson and Kahlon [12], and instead similar to, e.g., Emerson and Namjoshi [14]. The reason is that we want the following property to hold for any (A, B) and Φ:
if n_0 is the smallest number such that $\forall n \geq n_0 :\ (A, B)^{(1,n)} \models \Phi$,
then any $c < n_0$ is not a cutoff, any $c \geq n_0$ is a cutoff.

We call n_0 the *tight* cutoff. The definition in [12, page 2] requires that $\forall n \leq c. (A, B)^{(1,n)} \models \Phi$ if and only if $\forall n \geq 1 : (A, B)^{(1,n)} \models \Phi$, and thus allows stating $c < n_0$ as a cutoff if Φ does not hold for all n. □

In model checking, a cutoff allows us to check whether any "big" system satisfies the specification by checking it in the cutoff system. As noted by Jacobs and Bloem [17], a similar reduction applies to the parameterized synthesis problem. For guarded protocols, we obtain the following *semi-decision procedure for parameterized synthesis*:

0. set initial bound (b_A, b_B) on size of process templates;
1. determine cutoff for (b_A, b_B) and Φ;
2. solve bounded template synthesis problem for cutoff, size bound, and Φ;
3. if successful return (A, B) else increase (b_A, b_B) and goto (1).

Existing Cutoff Results. Emerson and Kahlon [12] have shown:

Theorem 1 (Disjunctive Cutoff Theorem). *For closed disjunctive systems, $|B|+2$ is a cutoff $^{(\dagger)}$ for formulas of the form $\mathsf{A}\, h(A, B^{(1)})$ and $\mathsf{E}\, h(A, B^{(1)})$, and for global deadlock detection.*

Theorem 2 (Conjunctive Cutoff Theorem). *For closed conjunctive systems, $2\,|B|$ is a cutoff $^{(\dagger)}$ for formulas of the form $\mathsf{A}\, h(A)$ and $\mathsf{E}\, h(A)$, and for global deadlock detection. For formulas of the form $\mathsf{A}\, h(B^{(1)})$ and $\mathsf{E}\, h(B^{(1)})$, $2\,|B|+1$ is a cutoff.*

Remark 3. $^{(\dagger)}$ Note that Emerson and Kahlon [12] proved these results for a different definition of a cutoff (see Remark 2). Their results also hold for our definition, except possibly for global deadlocks. For the latter case to hold with the new cutoff definition, one also needs to prove the direction "global deadlock in the cutoff system implies global deadlock in a large system" (later called Monotonicity Lemma). In Sect. 6.3 and 7.3 we prove these lemmas for the case of general deadlock (global *or* local).

Challenge: Open Systems. For any open system S there exists a closed system S' such that S and S' cannot be distinguished by LTL specifications (cp. Manna and Pnueli [22]). Thus, one approach to PMC for open systems is to use a translation between open and closed systems, and then use the existing cutoff results for closed systems.

While such an approach works in theory, it might not be feasible in practice: since cutoffs depend on the size of process templates, and the translation blows up the process template, it also blows up the cutoffs. Thus, cutoffs that directly support open systems are important.

Challenge: Liveness and Deadlocks under Fairness. We are interested in cutoff results that support liveness properties. In general, we would like to consider only runs where all processes move infinitely often, i.e., use the unconditional fairness assumption $\forall p.\, \mathsf{GF}\, \mathsf{move}_p$. However, this would mean that we accept all systems that always go into a local deadlock, since then the assumption is violated. This is especially undesirable in synthesis, because the synthesizer usually tries to violate the assumptions in order to satisfy the specification. To avoid this, we require the absence of local deadlocks under the strong fairness assumption $\forall p.(\mathsf{GF}\,\mathsf{en}_p \rightarrow \mathsf{GF}\,\mathsf{move}_p)$. Since strong fairness and absence of local deadlocks imply unconditional fairness, we can then use the latter as an assumption for the original specification.

In summary, for a parameterized specification Φ, we consider satisfaction of

$$\text{\textit{“all runs are infinite”}} \quad \wedge \quad \mathsf{A}_{strong}\,\Phi_{\neg dead} \quad \wedge \quad \mathsf{A}_{uncond}\,\Phi.$$

This is equivalent to *“all runs are infinite”* $\wedge\, \mathsf{A}_{strong}(\Phi_{\neg dead} \wedge \Phi)$, but by considering the form above we can separate the tasks of deadlock detection and of model checking $\mathsf{LTL}\backslash\mathsf{X}$-properties, and obtain modular cutoffs.

In the following, we present cutoffs for problems of the forms (i) $\mathsf{A}_{uncond}\,\Phi$, (ii) $\mathsf{A}_{strong}\,\Phi_{\neg dead}$ and no global deadlocks (and the variants with E path quantifier).

5 New Cutoff Results

We present new cutoff results that extend Theorems 1 and 2, summarized in the table below. We distinguish between disjunctive and conjunctive systems, non-fair and fair executions, as well as between the satisfaction of $\mathsf{LTL}\backslash\mathsf{X}$ properties $h(A, B^{(k)})$ and the existence of deadlocks. All results hold for open systems, and for both path quantifiers A and E. Cutoffs depend on the size of process template B and the number $k \geq 1$ of B-processes a property talks about:

	$h(A, B^{(k)})$ no fairness	deadlock detection no fairness	$h(A, B^{(k)})$ uncond. fairness	deadlock detection strong fairness
Disjunctive	$\lvert B\rvert + k + 1$	$2\lvert B\rvert - 1$	$2\lvert B\rvert + k - 1$	$2\lvert B\rvert - 1$
Conjunctive	$k + 1$	$2\lvert B\rvert - 2$ $(*)$	$k + 1$ $(*)$	$2\lvert B\rvert - 2$ $(*)$

Results marked with a $(*)$ are for a restricted class of systems: For conjunctive systems with fairness, we require infinite runs to be initializing, i.e., all non-deadlocked processes return to init infinitely often.[2] Additionally, the cutoffs for deadlock detection in conjunctive systems only support 1-conjunctive systems. The reason for this restriction will be explained in Remark 4.

All cutoffs in the table are tight – no smaller cutoff can exist for this class of systems and properties – except for the case of deadlock detection in disjunctive systems without fairness. There, the cutoff is asymptotically tight, i.e., it must increase linearly with the size of the process template.

Proof Structure

To justify the entries in the table, we first recapitulate the proof structure of the original Theorems 1 and 2. The proofs are based on two lemmas, Monotonicity

[2] This assumption is in the same flavor as the restriction that init_A and init_B appear in all conjunctive guards. Intuitively, the additional restriction makes sense since conjunctive systems model shared resources, and everybody who takes a resource should eventually release it.

and Bounding. We give some basic proof ideas of the lemmas from [12] and mention extensions to the cases with fairness and deadlock detection. For cases where this extension is not easy, we will introduce additional proof techniques and explain how to use them in Sections 6 and 7. Note that we only consider properties of the form $h(A, B^{(1)})$ — the proof ideas extend to general properties $h(A, B^{(k)})$ without difficulty. Similarly, in most cases the proof ideas extend to open systems without major difficulties — mainly because when we construct a simulating run, we have the freedom to choose the input that is needed. Only for the case of deadlock detection we have to handle open systems explicitly.

1) *Monotonicity* lemma: if a behavior is possible in a system with $n \in \mathbb{N}$ copies of B, then it is also possible in a system with one additional process:

$$(A, B)^{(1,n)} \models \mathsf{E}\, h(A, B^{(1)}) \implies (A, B)^{(1,n+1)} \models \mathsf{E}\, h(A, B^{(1)}),$$

and if a deadlock is possible in $(A, B)^{(1,n)}$, then it is possible in $(A, B)^{(1,n+1)}$.

Proof ideas. The lemma is easy to prove for properties $\mathsf{E}\, h(A, B^{(1)})$ in both disjunctive and conjunctive systems, by letting the additional process stay in its initial state init_B forever (cp. [12]). This cannot disable transitions with disjunctive guards, as these check for *existence* of a local state in another process (and we do not remove any processes), and it cannot disable conjunctive guards since they contain init_B by assumption. However, this construction violates fairness, since the new process never moves. This can be resolved in the disjunctive case by letting the additional process mimic all transitions of an existing process. But in general this does not work in conjunctive systems (due to the non-reflexive interpretation of guards). For this case and for deadlock detection, the proof is not trivial and may only work for $n \geq c$, for some lower bound $c \in \mathbb{N}$ (see Sect. 6, 7). □

2) *Bounding* lemma: for a number $c \in \mathbb{N}$, a behavior is possible in a system with c copies of B if it is possible in a system with $n \geq c$ copies of process B:

$$(A, B)^{(1,c)} \models \mathsf{E}\, h(A, B^{(1)}) \impliedby (A, B)^{(1,n)} \models \mathsf{E}\, h(A, B^{(1)}),$$

and a deadlock is possible in $(A, B)^{(1,c)}$ if it is possible in $(A, B)^{(1,n)}$.

Proof ideas. For disjunctive systems, the main difficulty is that removing processes might falsify guards of the local transitions of A or B_1 in a given run (see Sect. 6). For conjunctive systems, removing processes from a run is easy for the case of infinite runs, since a transition that was enabled before cannot become disabled. Here, the difficulty is in preserving deadlocks, because removing processes may enable processes that were deadlocked before (Sect. 7). □

6 Proof Techniques for Disjunctive Systems

6.1 LTL\X Properties without Fairness: Existing Constructions

We revisit the main technique of the original proof of Theorem 1 [12]. It constructs an infinite run y of $(A, B)^{(1,c)}$ with $y \models h(A, B^{(1)})$, based on an infinite run x of $(A,B)^{(1,n)}$ with $n > c$ and $x \models h(A,B^{(1)})$. The idea is to copy local runs $x(A)$ and $x(B_1)$ into y, and construct runs of other processes in a way that enables all transitions along $x(A)$ and $x(B_1)$. The latter is achieved with the flooding construction.

Flooding Construction [12]. Given a run $x = (s_1,e_1,p_1),(s_2,e_2,p_2)\dots$ of $(A,B)^{(1,n)}$, let $\mathsf{Visited}_{\mathcal{B}}(x)$ be the set of all local states visited by B-processes in x, i.e., $\mathsf{Visited}_{\mathcal{B}}(x) = \{q \in Q_B \mid \exists m \exists i.\ s_m(B_i) = q\}$.

For every $q \in \mathsf{Visited}_{\mathcal{B}}(x)$ there is a local run of $(A,B)^{(1,n)}$, say $x(B_i)$, that visits q first, say at moment m_q. Then, saying that process B_{i_q} of $(A,B)^{(1,c)}$ *floods* q means:

$$y(B_{i_q}) = x(B_i)[1\!:\!m_q](q)^{\omega}.$$

In words: the run $y(B_{i_q})$ is the same as $x(B_i)$ until moment m_q, and after that the process never moves.

The construction achieves the following. If we copy local runs of A and B_1 from x to y, and in y for every $q \in \mathsf{Visited}_{\mathcal{B}}(x)$ introduce one process that floods q, then: if in x at some moment m there is a process in state q', then in y at moment m there will also be a process (different from A and B_1) in state q'. Thus, every transition of A and B_1, which is enabled at moment m in x, will also be enabled in y.

Proof idea of the bounding lemma. The lemma for disjunctive systems without fairness can be proved by copying local runs $x(A)$ and $x(B_1)$, and flooding all states in $\mathsf{Visited}_{\mathcal{B}}(x)$. To ensure that at least one process moves infinitely often in y, we copy one additional (infinite) local run from x. Finally, it may happen that the resulting collection of local runs violates the interleaving semantics requirement. To resolve this, we add stuttering steps into local runs whenever two or more processes move at the same time, and we remove global stuttering steps in y. Since the only difference between $x(A,B_1)$ and $y(A,B_1)$ are stuttering steps, y and x satisfy the same $\mathsf{LTL}\backslash\mathsf{X}$-properties $h(A,B^{(1)})$. Since $|\mathsf{Visited}_{\mathcal{B}}(x)| \leq |B|$, we need at most $1 + |B| + 1$ copies of B in $(A,B)^{(1,c)}$.

6.2 LTL\X Properties with Fairness: New Constructions

The flooding construction does not preserve fairness, and also cannot be used to construct deadlocked runs since it does not preserve disabledness of transitions of processes A or B_1. For these cases, we provide new proof constructions.

Consider the proof task of the bounding lemma for disjunctive systems with fairness: given an unconditionally fair run x of $(A,B)^{(1,n)}$ with $x \models h(A,B^{(1)})$, we want to construct an unconditionally fair run y of $(A,B)^{(1,c)}$ with $y \models h(A,B^{(1)})$. In contrast to unfair systems, we need to ensure that all processes move infinitely often in y. The insight is that after a finite time all processes will start looping around some set $\mathsf{Visited}^{inf}$ of states. We construct a run y that mimics this. To this end, we introduce two constructions. *Flooding with evacuation* is similar to flooding, but instead of keeping processes in their flooding states forever it evacuates the processes into $\mathsf{Visited}^{inf}$. *Fair extension* lets all processes move infinitely often without leaving $\mathsf{Visited}^{inf}$.

Flooding with Evacuation. Given a subset $\mathcal{F} \subseteq \mathcal{B}$ and an infinite run $x = (s_1, e_1, p_1) \ldots$ of $(A,B)^{(1,n)}$, define

$$\mathsf{Visited}^{inf}_{\mathcal{F}}(x) = \{q \mid \exists \text{ infinitely many } \quad m\colon s_m(B_i) = q \text{ for some } B_i \in \mathcal{F}\} \quad (1)$$

$$\mathsf{Visited}^{fin}_{\mathcal{F}}(x) = \{q \mid \exists \text{ only finitely many } m\colon s_m(B_i) = q \text{ for some } B_i \in \mathcal{F}\} \quad (2)$$

Let $q \in \mathsf{Visited}^{fin}_{\mathcal{F}}(x)$. In run x there is a moment f_q when q is reached for the first time by some process from $\mathcal{F}$, denoted B_{first_q}. Also, in run x there is a moment l_q such that: $s_{l_q}(B_{\mathsf{last}_q}) = q$ for some process $B_{\mathsf{last}_q} \in \mathcal{F}$, and $s_t(B_i) \neq q$ for all $B_i \in \mathcal{F}$, $t > l_q$ — i.e., when some process from $\mathcal{F}$ is in state q for the last time in x. Then, saying that process B_{i_q} of $(A,B)^{(1,c)}$ *floods* $q \in \mathsf{Visited}^{fin}_{\mathcal{F}}(x)$ *and then evacuates into* $\mathsf{Visited}^{inf}_{\mathcal{F}}(x)$ means:

$$y(B_{i_q}) = x(B_{\mathsf{first}_q})[1\!:\!f_q] \;\cdot\; (q)^{(l_q - f_q + 1)} \cdot\; x(B_{\mathsf{last}_q})[l_q\!:\!m] \;\cdot\; (q')^{\omega},$$

where q' is the state in $\mathsf{Visited}^{inf}_{\mathcal{F}}(x)$ that $x(B_{\mathsf{last}_q})$ reaches first, at some moment $m \geq l_q$. In words, process B_{i_q} mimics process B_{first_q} until it reaches q, then does nothing until process B_{last_q} starts leaving q, then it mimics B_{last_q} until it reaches $\mathsf{Visited}^{inf}_{\mathcal{F}}(x)$.

The construction ensures: if we copy local runs of all processes not in $\mathcal{F}$ from x to y, then all transitions of y are enabled. This is because: for any process p of $(A,B)^{(1,c)}$ that takes a transition in y at any moment, the set of states visible to process p is a superset of the set of states visible to the original process in $(A,B)^{(1,n)}$ whose transitions process p copies.

Fair Extension. Here, we consider a path x that is the postfix of an unconditionally fair run x' of $(A,B)^{(1,n)}$, starting from the moment where no local states from $\mathsf{Visited}^{fin}_{\mathcal{B}}(x')$ are visited anymore. We construct a corresponding unconditionally-fair path y of $(A,B)^{(1,c)}$, where no local states from $\mathsf{Visited}^{fin}_{\mathcal{B}}(x')$ are visited.

Formally, let $n \geq 2|B|$, and x an unconditionally-fair path of $(A,B)^{(1,n)}$ such that $\mathsf{Visited}^{fin}_{\mathcal{B}}(x) = \emptyset$. Let $c \geq 2|B|$, and s'_1 a state of $(A,B)^{(1,c)}$ with

- $s'_1(A_1) = s_1(A_1)$, $s'_1(B_1) = s_1(B_1)$
- for every $q \in \mathsf{Visited}^{inf}_{B_2..B_n}(x) \setminus \mathsf{Visited}^{inf}_{B_1}(x)$, there are two processes $B_{i_q}, B_{i'_q}$ of $(A,B)^{(1,c)}$ that start in q, i.e., $s'_1(B_{i_q}) = s'_1(B_{i'_q}) = q$
- for every $q \in \mathsf{Visited}^{inf}_{B_2..B_n}(x) \cap \mathsf{Visited}^{inf}_{B_1}(x)$, there is one process B_{i_q} of $(A,B)^{(1,c)}$ that starts in q
- for some $q^\star \in \mathsf{Visited}^{inf}_{B_2..B_n}(x) \cap \mathsf{Visited}^{inf}_{B_1}(x)$, there is one additional process of $(A,B)^{(1,c)}$, different from any in the above, called $B_{i'_{q^\star}}$, that starts in $q^\star$.
- any other process B_i of $(A,B)^{(1,c)}$ starts in some state of $\mathsf{Visited}^{inf}_{B_2..B_n}(x)$.

Note that if $\mathsf{Visited}^{inf}_{B_2..B_n}(x) \cap \mathsf{Visited}^{inf}_{B_1}(x) = \emptyset$, then the third and fourth prerequisites are trivially satisfied.

The fair extension extends state s'_1 of $(A,B)^{(1,c)}$ to an unconditionally-fair path $y = (s'_1, e'_1, p'_1) \ldots$ with $y(A_1, B_1) = x(A_1, B_1)$ as follows:

(a) $y(A_1) = x(A_1)$, $y(B_1) = x(B_1)$
(b) for every $q \in \mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x) \setminus \mathsf{Visited}^{\mathit{inf}}_{B_1}(x)$: in run x there is $B_i \in \{B_2..B_n\}$ that starts in q and visits it infinitely often. Let B_{i_q} and $B_{i'_q}$ of $(A,B)^{(1,c)}$ mimic B_i in turns: first B_{i_q} mimics B_i until it reaches q, then $B_{i'_q}$ mimics B_i until it reaches q, and so on.
(c) arrange states of $\mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x) \cap \mathsf{Visited}^{\mathit{inf}}_{B_1}(x)$ in some order $(q^\star, q_1, \dots, q_l)$. The processes $B_{i'_{q^\star}}, B_{i_{q^\star}}, B_{i_{q_1}}, \dots, B_{i_{q_l}}$ behave as follows. Start with $B_{i'_{q^\star}}$: when B_1 enters $q^\star$ in y, it carries[3] $B_{i'_{q^\star}}$ from $q^\star$ to q_1, then carries $B_{i_{q_1}}$ from q_1 to q_2, ..., then carries $B_{i_{q_l}}$ from q_l to $q^\star$, then carries $B_{i_{q^\star}}$ from $q^\star$ to q_1, then carries $B_{i'_{q^\star}}$ from q_1 to q_2, then carries $B_{i_{q_1}}$ from q_2 to q_3, and so on.
(d) any other B_i of $(A,B)^{(1,c)}$, starting in $q \in \mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x)$, mimics B_{i_q}.

Note that parts (b) and (c) of the constrution ensure that there is always at least one process in every state from $\mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x)$. This ensures that the guards of all transitions of the construction are satisfied. Excluding processes in (d), the fair extension uses up to $2|B|$ copies of B.[4]

Proof idea of the bounding lemma. Let $c = 2\,|B|$. Given an unconditionally-fair run x of $(A,B)^{(1,n)}$ we construct an unconditionally-fair run y of the cutoff system $(A,B)^{(1,c)}$ such that $y(A,B_1)$ is stuttering equivalent to $x(A,B_1)$.

Note that in x there is a moment m such that all local states that are visited after m are in $\mathsf{Visited}^{\mathit{inf}}_{\mathcal{B}}(x)$.

The construction has two phases. In the first phase, we apply flooding for states in $\mathsf{Visited}^{\mathit{inf}}_{\mathcal{B}}(x)$, and flooding with evacuation for states in $\mathsf{Visited}^{\mathit{fin}}_{\mathcal{B}}(x)$:

(a) $y(A) = x(A)$, $y(B_1) = x(B_1)$
(b) for every $q \in \mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x) \setminus \mathsf{Visited}^{\mathit{inf}}_{B_1}(x)$, devote two processes of $(A,B)^{(1,c)}$ that flood q
(c) for some $q^\star \in \mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x) \cap \mathsf{Visited}^{\mathit{inf}}_{B_1}(x)$, devote one process of $(A,B)^{(1,c)}$ that floods $q^\star$
(d) for every $q \in \mathsf{Visited}^{\mathit{fin}}_{B_2..B_n}(x)$, devote one process of $(A,B)^{(1,c)}$ that floods q and evacuates into $\mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x)$
(e) let other processes (if any) mimic process B_1

The phase ensures that at moment m in y, there are no processes in $\mathsf{Visited}^{\mathit{fin}}_{\mathcal{B}}(x)$, and all the pre-requisites of the fair extension are satisfied.

The second phase applies the fair extension, and then establishes the interleaving semantics as in the bounding lemma in the non-fair case. The overall construction uses up to $2|B|$ copies of B.

[3] "Process B_1 starting at moment m carries process B_i from q to q'" means: process B_i mimics the transitions of B_1 starting at moment m at q until B_1 first reaches q'.
[4] A careful reader may notice that if $|\mathsf{Visited}^{\mathit{inf}}_{B_1}(x)| = 1$ and $|\mathsf{Visited}^{\mathit{inf}}_{B_2..B_n}(x)| = |B|$, then the construction uses $2|B| + 1$ copies of B. But one can slightly modify the construction for this special case, and remove process $B_{i'_{q^\star}}$ from the pre-requisites.

6.3 Detection of Local and Global Deadlocks: New Constructions

Monotonicity Lemmas. The lemma for deadlock detection, for fair and unfair cases, is proven for $n \geq |B| + 1$. In the case of local deadlocks, process B_{n+1} mimics a process that moves infinitely often in x. In the case of global deadlocks, by pigeon hole principle, in the global deadlock state there is a state q with at least two processes in it—let process B_{n+1} mimic a process that deadlocks in q.

Bounding Lemmas. For the case of global deadlocks, fairness does not affect the proof of the bounding lemma. The insight is to divide deadlocked local states into two disjoint sets, dead_1 and dead_2, as follows. Given a globally deadlocked run x of $(A,B)^{(1,n)}$, for every $q \in \mathsf{dead}_1$, there is a process of $(A,B)^{(1,n)}$ deadlocked in q with input i, that has an outgoing transition guarded "$\exists q$" – hence, adding one more process into q would unlock the process. In contrast, $q \in \mathsf{dead}_2$ if any process deadlocked in q stays deadlocked after adding more processes into q. Let us denote the set of B-processes deadlocked in dead_1 by $\mathcal{D}_1$. Finally, abuse the definition in Eq. 2 and denote by $\mathsf{Visited}^{fin}_{\mathcal{B}\setminus\mathcal{D}_1}(x)$ the set of states that are visited by B-processes not in $\mathcal{D}_1$ before reaching a deadlocked state.

Given a globally deadlocked run x of $(A,B)^{(1,n)}$ with $n \geq 2|B| - 1$, we construct a globally deadlocked run y of $(A,B)^{(1,c)}$ with $c = 2|B| - 1$ as follows:

- copy from x into y the local runs of processes in $\mathcal{D}_1 \cup \{A\}$
- flood every state of dead_2
- for every $q \in \mathsf{Visited}^{fin}_{\mathcal{B}\setminus\mathcal{D}_1}(x)$, flood q and evacuate into dead_2.

The construction ensures: (1) for any moment and any process in y, the set of local states that are visible to the process includes all the states that were visible to the corresponding process in $(A,B)^{(1,n)}$ whose transitions we copy; (2) in y, there is a moment when all processes deadlock in $\mathsf{dead}_1 \cup \mathsf{dead}_2$.

For the case of local deadlocks, the construction is similar but slightly more involved, and needs to distinguish between unfair and fair cases. In the unfair case, we also copy the behaviour of an infinitely moving process. In the strong-fair case, we continue the runs of non-deadlocked processes with the fair extension.

7 Proof Techniques for Conjunctive Systems

7.1 $\mathsf{LTL}\backslash\mathsf{X}$ Properties without Fairness: Existing Constructions

Recall that the Monotonicity lemma is proven by keeping the additional process in the initial state. To prove the bounding lemma, Emerson and Kahlon [12] suggest to simply copy the local runs $x(A)$ and $x(B_1)$ into y. In addition, we may need one more process that moves infinitely often to ensure that an infinite run of $(A,B)^{(1,n)}$ will result in an infinite run of $(A,B)^{(1,c)}$. All transitions of copied processes will be enabled because removing processes from a conjunctive system cannot disable a transition that was enabled before.

7.2 LTL\X Properties with Fairness: New Constructions

The proof of the Bounding lemma is the same as in the non-fair case, noting that if the original run is unconditional-fair, then so will be the resulting run.

Proving the Monotonicity lemma is more difficult, since the fair extension construction from disjunctive systems does not work for conjunctive systems – if an additional process mimics the transitions of an existing process then it disables transitions of the form $q \overset{\text{“}\forall\neg q\text{”}}{\rightarrow} q'$ or $q \overset{\text{“}\forall\neg q'\text{”}}{\rightarrow} q'$. Hence, we add the restriction of initializing runs, which allows us to construct a fair run as follows. The additional process B_{n+1} “shares” a local run $x(B_i)$ with an existing process B_i of $(A, B)^{(1,n+1)}$: one process stutters in init_B while the other makes transitions from $x(B_i)$, and whenever $x(B_i)$ enters init_B (this happens infinitely often), the roles are reversed. Since this changes the behavior of B_i, B_i should not be mentioned in the formula, i.e., we need $n \geq 2$ for a formula $h(A, B^{(1)})$.

7.3 Detection of Local and Global Deadlocks: New Constructions

Monotonicity lemmas for both fair and unfair cases are proven by keeping process B_{n+1} in the initial state, and copying the runs of deadlocked processes. If the run of $(A, B)^{(1,n)}$ is globally deadlocked, then process B_{n+1} may keep moving in the constructed run, i.e., it may only be locally deadlocked. In case of a local deadlock in $(A, B)^{(1,n)}$, distinguish two cases: there is an infinitely moving B-process, or all B-processes are deadlocked (and thus A moves infinitely often). In the latter case, use the same construction as in the global deadlock case (the correctness argument uses the fact that systems are 1-conjunctive, runs are initializing, and there is only one process of type A). In the former case, copy the original run, and let B_{n+1} share a local run with an infinitely moving B-process.

Bounding lemma (no fairness). In the case of global deadlock detection, Emerson and Kahlon [12] suggest to copy a subset of the original local runs. For every local state q that is present in the final state of the run, we need at most two local runs that end in this state. In the case of local deadlocks, our construction uses the fact that systems are 1-conjunctive. In 1-conjunctive systems, if a process is deadlocked, then there is a set of states *DeadGuards* that all need to be populated by other processes in order to disable all transitions of the deadlocked process. Thus, the construction copies: (i) the local run of a deadlocked process, (ii) for each $q \in \mathit{DeadGuards}$, the local run of a process that is in q at the moment of the deadlock, and (iii) the local run of an infinitely moving process.

Bounding lemma (strong fairness). We use a construction that is similar to that of properties under fairness for disjunctive systems (Sect. 6.2): in the setup phase, we populate some “safe” set of states with processes, and then we extend the runs of non-deadlocked processes to satisfy strong fairness, while ensuring that deadlocked processes never get enabled.

Let $c = 2|Q_B \backslash \{\mathsf{init}_B\}|$. Let $x = (s_1, e_1, p_1)\ldots$ be a locally deadlocked strong-fair intitializing run of $(A, B)^{(1,n)}$ with $n > c$. We construct a locally deadlocked strong-fair initializing run y of $(A, B)^{(1,c)}$.

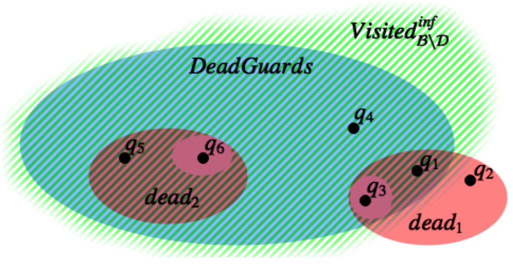


Fig. 1: Bounding lemma (strong fairness): Venn diagram for dead_1, dead_2, $DeadGuards$, $\mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x)$. States $q_1, ..., q_6$ are to illustrate that the corresponding sets may be non-empty. E.g., in x, a process may be deadlocked in $q_1 \in (DeadGuards \cap \mathsf{dead}_1 \cap \mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x))$, and another process in $q_3 \in \mathsf{dead}_1 \cap DeadGuards \setminus \mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x)$.

Let $\mathcal{D} \subseteq \mathcal{B}$ be the set of deadlocked B-processes in x. Let d be the moment in x starting from which every process in $\mathcal{D}$ is deadlocked. Let $\mathsf{dead}(x)$ be the set of states in which processes $\mathcal{D}$ of $(A, B)^{(1,n)}$ are deadlocked. Let $\mathsf{dead}_2(x) \subseteq \mathsf{dead}(x)$ be the set of deadlocked states such that: for every $q \in \mathsf{dead}_2(x)$, there is a process $B_i \in \mathcal{D}$ with $s_d(B_i) = q$ and that for input $e_{\geq d}(B_i)$ has a transition guarded with "$\forall\neg q$". Thus, a process in q is deadlocked with $e_d(B_i)$ only if there is another process in q in every moment $\geq d$. Let $\mathsf{dead}_1(x) = \mathsf{dead}(x) \setminus \mathsf{dead}_2(x)$. Define $DeadGuards$ to be the set

$$\{\ q \mid \exists B_i \in \mathcal{D} \textit{ with a transition guarded "}\forall\neg q\textit{" in } (s_d(B_i), e_d(B_i))\ \}.$$

Figure 1 illustrates properties of sets $DeadGuards$, dead_1, dead_2, $\mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x)$.

In the **setup phase**, we copy from x into y:

- the local run of A;
- for every $q \in \mathsf{dead}_1$, the local run of one process deadlocked in q;
- for every $q \in \mathsf{dead}_2$, the local runs of two[5] processes deadlocked in q;
- for every $q \in DeadGuards \setminus \mathsf{dead}$, the local run of a process that reaches q after moment d.
- Finally, we keep one B-process in init_B until moment d.

The setup phase ensures: in every state $q \in \mathsf{dead}$, there is at least one process deadlocked in q at moment d in y. Now we need to ensure that the non-deadlocked processes in $DeadGuards \setminus \mathsf{dead}$ and init_B move infinitely often, which is done using the looping extension described bellow.

Order arbitrarily $DeadGuards \setminus \mathsf{dead} = (q_1, \ldots, q_k) \subseteq \mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x)$. Let $\mathcal{P} \subseteq \{B_1, ..., B_c\}$ be the non-deadlocked processes of $(A, B)^{(1,c)}$ that we moved into $(q_1, \ldots, q_k) \dot{\cup} \{\mathsf{init}_B\}$ in the setup phase. Note that $|\mathcal{P}| = |(q_1, ..., q_k)| + 1$.

[5] Strictly speaking, in x we might not have two deadlocked processes in a state in $dead_2$ – one process may be deadlocked, others enter and exit the state infinitely often. In such case, there is always a non-deadlocked process in the state. Then, copy the local run of such infinitely moving process until it enters the deadlocked state, and then deadlock it by providing the same input as the deadlocked process receives.

The **looping phase** is: set $i = 1$, and repeat infinitely the following.

- let $B_{\mathsf{init}} \in \mathcal{P}$ be the process of $(A, B)^{(1,c)}$ that is currently in init_B, and $B_{q_i} \in \mathcal{P}$ be the process of $(A, B)^{(1,c)}$ that is currently in q_i
- let $\tilde{B}_{q_i} \in \mathsf{Visited}^{inf}_{\mathcal{B}\setminus\mathcal{D}}(x)$ be a process of $(A, B)^{(1,n)}$ that visits q_i and init_B infinitely often. Let B_{init} of $(A, B)^{(1,c)}$ copy transitions of $\tilde{B}_{q_i}$ on some path $\mathsf{init}_B \to \ldots \to q_i$, then let B_{q_i} copy transitions of $\tilde{B}_{q_i}$ on some path $q_i \to \ldots \to \mathsf{init}_B$. For copying we consider only the paths of $\tilde{B}_{q_i}$ that happen after moment d.
- $i = i \oplus 1$

Remark 4. In 1-conjunctive systems, the set *DeadGuards* is "static", i.e., there is always at least one process in *each state* of *DeadGuards* starting from the moment of the deadlock. In contrast, in general conjunctive systems where guards can overlap, there is no such set. However, there is a similar set of sets of states, such that *one state from each set* always needs to be populated to ensure the deadlock.

8 Conclusion

We have extended the cutoff results for guarded protocols of Emerson and Kahlon [12] to support local deadlock detection, fairness assumptions, and open systems. In particular, our results imply decidability of the parameterized model checking problem for this class of systems and specifications, which to the best of our knowledge was unknown before. Furthermore, the cutoff results can easily be integrated into the parameterized synthesis approach [17, 20].

Since conjunctive guards can model atomic sections and read-write locks, and disjunctive guards can model pairwise rendezvous (for some classes of specifications, cp. [13]), our results apply to a wide spectrum of systems models. But the expressivity of the model comes at a high cost: cutoffs are linear in the size of a process, and are shown to be tight (with respect to this parameter). For conjunctive systems, our new results are restricted to systems with 1-conjunctive guards, effectively only allowing to model a single shared resource. We conjecture that our proof methods can be extended to systems with more general conjunctive guards, at the price of even bigger cutoffs. We leave this extension and the question of finding cutoffs that are independent of the size of processes for future research.

Acknowledgment. We thank Roderick Bloem, Markus Rabe and Leander Tentrup for comments on drafts of this paper. This work was supported by the Austrian Science Fund (FWF) through the RiSE project (S11406-N23, S11407-N23) and grant nr. P23499-N23, as well as by the German Research Foundation (DFG) through SFB/TR 14 AVACS and project ASDPS (JA 2357/2-1).

References

1. Abdulla, P.A., Haziza, F., Holík, L.: All for the price of few. In: VMCAI. LNCS, vol. 7737, pp. 476–495. Springer (2013), http://dx.doi.org/10.1007/978-3-642-35873-9_28
2. 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), http://dx.doi.org/10.1007/978-3-642-54013-4_15
3. 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), http://dx.doi.org/10.1007/978-3-662-44584-6_9
4. Außerlechner, S., Jacobs, S., Khalimov, A.: Tight cutoffs for guarded protocols with fairness. CoRR abs/1505.03273 (2015), http://arxiv.org/abs/1505.03273
5. Baier, C., Katoen, J.P.: Principles of model checking, vol. 26202649. MIT press Cambridge (2008)
6. 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 (September 2015), http://www.morganclaypool.com/doi/10.2200/S00658ED1V01Y201508DCT013, 170 pages
7. Bouajjani, A., Habermehl, P., Vojnar, T.: Verification of parametric concurrent systems with prioritised FIFO resource management. Formal Methods in System Design 32(2), 129–172 (2008), http://dx.doi.org/10.1007/s10703-008-0048-7
8. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV. LNCS, vol. 1855, pp. 403–418. Springer (2000), http://dx.doi.org/10.1007/10722167_31
9. Clarke, E.M., Talapur, M., Veith, H.: Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In: TACAS. LNCS, vol. 4963, pp. 33–47. Springer (2008)
10. Clarke, E.M., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: CONCUR. LNCS, vol. 3170, pp. 276–291. Springer (2004)
11. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. 2(3), 241–266 (1982), http://dx.doi.org/10.1016/0167-6423(83)90017-5
12. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE. LNCS, vol. 1831, pp. 236–254. Springer (2000)
13. Emerson, E.A., Kahlon, V.: Model checking guarded protocols. In: LICS. pp. 361–370. IEEE Computer Society (2003)
14. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Foundations of Computer Science 14, 527–549 (2003)
15. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.D.: Liveness with invisible ranking. STTT 8(3), 261–279 (2006), http://dx.doi.org/10.1007/s10009-005-0193-x
16. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM 39(3), 675–735 (1992)
17. Jacobs, S., Bloem, R.: Parameterized synthesis. Logical Methods in Computer Science 10, 1–29 (2014)
18. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV. LNCS, vol. 6174, pp. 645–659. Springer (2010), http://dx.doi.org/10.1007/978-3-642-14295-6_55
19. Kesten, Y., Pnueli, A., Shahar, E., Zuck, L.D.: Network invariants in action. In: CONCUR. LNCS, vol. 2421, pp. 101–115. Springer (2002), http://dx.doi.org/10.1007/3-540-45694-5_8

20. Khalimov, A., Jacobs, S., Bloem, R.: PARTY parameterized synthesis of token rings. In: CAV. LNCS, vol. 8044, pp. 928–933. Springer (2013)
21. Kurshan, R.P., McMillan, K.L.: A structural induction theorem for processes. Inf. and Comp. 117(1), 1–11 (1995)
22. Manna, Z., Pnueli, A.: Temporal specification and verification of reactive modules. Weizmann Institute of Science Technical Report (1992)
23. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: TACAS. LNCS, vol. 2031, pp. 82–97. Springer (2001), http://dx.doi.org/10.1007/3-540-45319-9_7
24. Pnueli, A., Xu, J., Zuck, L.D.: Liveness with (0, 1, ∞)-counter abstraction. In: CAV. Lecture Notes in Computer Science, vol. 2404, pp. 107–122. Springer (2002), http://dx.doi.org/10.1007/3-540-45657-0_9
25. Suzuki, I.: Proving properties of a ring of finite state machines. Inf. Process. Lett. 28(4), 213–214 (1988)
26. Wolper, P., Lovinfosse, V.: Verifying properties of large sets of processes with network invariants. In: Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 68–80. Springer (1989), http://dx.doi.org/10.1007/3-540-52148-8_6