Global and Local Deadlock Freedom in BIP

PAUL C. ATTIE, American University of Beirut, Lebanon
SADDEK BENSALEM and MARIUS BOZGA, UJF-Grenoble 1/CNRS VERIMAG, France
MOHAMAD JABER, American University of Beirut, Lebanon
JOSEPH SIFAKIS, Ecole Polytechnique Federale, Lausanne, Switzerland
FADI A. ZARAKET, American University of Beirut, Lebanon

We present a criterion for checking local and global deadlock freedom of finite state systems expressed in BIP: a component-based framework for constructing complex distributed systems. Our criterion is evaluated by model-checking a set of subsystems of the overall large system. If satisfied in small subsystems, it implies deadlock-freedom of the overall system. If not satisfied, then we re-evaluate over larger subsystems, which improves the accuracy of the check. When the subsystem being checked becomes the entire system, our criterion becomes complete for deadlock-freedom. Hence our criterion only fails to decide deadlock freedom because of computational limitations: state-space explosion sets in when the subsystems become too large. Our method thus combines the possibility of fast response together with theoretical completeness. Other criteria for deadlock freedom, in contrast, are incomplete in principle, and so may fail to decide deadlock freedom even if unlimited computational resources are available. Also, our criterion certifies freedom from local deadlock, in which a subsystem is deadlocked while the rest of the system executes. Other criteria only certify freedom from global deadlock. We present experimental results for dining philosophers and for a multi-token-based resource allocation system, which subsumes several data arbiters and schedulers, including Milner's token-based scheduler.

CCS Concepts: • **Theory of computation → Program verification**; • **Software and its engineering → Deadlocks**; **Model checking**; **Formal software verification**; *State systems*; *Synchronization*;

Additional Key Words and Phrases: Alternation, completeness, nondeterminism

ACM Reference format:
Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. 2018. Global and Local Deadlock Freedom in BIP. *ACM Trans. Softw. Eng. Methodol.* 26, 3, Article 9 (January 2018), 48 pages.
https://doi.org/10.1145/3152910

The research leading to these results has received funding from University Research Board (URB) at the American University of Beirut.

Authors' addresses: P. C. Attie, American University of Beirut, PO Box 11-0236, Riad El Solh, Beirut, 1107 2020, Lebanon; email: paul.attie@aub.edu.lb; S. Bensalem, UJF-Grenoble 1/CNRS VERIMAG, UMR 5104, Grenoble, F-38041, France, Saddek; email: Bensalem@imag.fr; M. Bozga, UJF-Grenoble 1/CNRS VERIMAG, UMR 5104, Grenoble, F-38041, France, Marius; email: Bozga@imag.fr; M. Jaber, American University of Beirut, PO Box 11-0236, Riad El Solh, Beirut, 1107 2020, Lebanon; email: mj54@aub.edu.lb; J. Sifakis, Ecole Polytechnique Federale Lausanne, Route Cantonale, Lausanne, 1015, Switzerland; email: joseph.sifakis@imag.fr; F. A. Zaraket, American University of Beirut, PO Box 11-0236, Riad El Solh, Beirut, 1107 2020, Lebanon; email: fz11@aub.edu.lb.

1 INTRODUCTION

Deadlock freedom is a crucial property of concurrent and distributed systems. With increasing system complexity, the challenge of assuring deadlock freedom and other correct properties becomes even greater. In contrast to the alternatives of (1) deadlock detection and recovery and (2) deadlock avoidance, we advocate deadlock prevention: design the system so that deadlocks do not occur.

Deciding deadlock freedom of finite-state concurrent programs is PSPACE-complete, in general [26, chap. 19]. To achieve tractability, we present a criterion for deadlock freedom that is evaluated by model-checking a set of subsystems of the overall system. If the subsystems are small, the criterion can be checked quickly. The criterion is sound (if true, it implies deadlock freedom) but not complete (if false, then it yields no information about deadlock). If the subsystems are larger, then our criterion becomes more "accurate"—roughly speaking, there is less possibility for the criterion to evaluate to false when the system is actually deadlock-free. In the limit, when the set of subsystems includes the entire system itself, our criterion is complete, so that evaluation to false implies that the system is actually deadlock-prone. Hence, our criterion only fails to resolve the question of deadlock freedom when its evaluation exhausts available computational resources, because the subsystems being checked have become too large, and state-explosion has set in.

Our method thus combines the possibility of fast response together with theoretical completeness. All deadlock-freedom checks given in the literature to date are, to our knowledge, incomplete in principle, and so remain incomplete even if unlimited computational resources are available. Hence these criteria could fail to resolve deadlock freedom for theoretical reasons, as well as for lack of computational resources. The reason for this incompleteness is that existing criteria all characterize deadlock by the occurrence of a wait-for cycle, e.g., as stated by Reference Antonino et al. [3], discussion of related work:

> All these methods were designed, to some extent, around the principle that under reasonable assumptions about the system, any deadlock state would contain a proper cycle of ungranted requests.

In a model of concurrency that includes choice of actions (e.g., BIP, CSP, I/O automata, CCS), a wait-for cycle is an *incomplete* characterization of deadlock, since a process can be in a wait-for cycle, but not deadlocked, due to having a choice of interaction with another process not in the wait-for cycle (see Figure 5 below).

Our method, in contrast, characterizes deadlock by the occurrence of a *supercycle* [7, 8], which, very roughly, is the AND-OR analogue of a wait-for cycle: a subset of processes constitutes a supercycle SC iff every possible action of every process in SC is blocked by another process in SC. We show that supercycles are a sound and complete characterization of deadlock: a system is deadlock-prone iff a supercycle can arise in some reachable state. We then present our criterion, which prevents the occurrence of supercycles in reachable states of the system. We first present a "global" version of our criterion, which is both sound and complete w.r.t. absence of supercycles, and then a "local" version, which is sound w.r.t. absence of supercycles, and can be evaluated over small subsystems.

Our criterion guarantees freedom from local (and therefore global) deadlock. A local deadlock occurs when a subsystem is deadlocked while the rest of the system can execute. Other criteria in the literature [2, 3, 13, 17, 21, 23, 24, 29] guarantee only global deadlock freedom.

This article significantly extends a preliminary conference version [6] as follows: (1) we present an "AND-OR" criterion for deadlock freedom, which exploits the AND-OR structure of supercycles, and is therefore complete for deadlock freedom in the limit, while our preliminary work [6] gives a "linear" criterion, which is a special case in which the AND-OR structure is ignored, and (2) experimental results show that the new criterion is more efficient in practice, and also succeeds in cases where the linear criterion fails. We therefore have the best of both worlds: early stopping, and therefore efficient verification of deadlock freedom, in many cases, together with theoretical completeness. Our criterion is, to the best of our knowledge, the first criterion that is sound *and complete* for global and *local* deadlock freedom in concurrent programs with nondeterministic local choice, i.e., a process can nondeterministically choose among enabled actions.

We present experimental results for dining philosophers and for a multi-token-based resource allocation system, which generalizes Milner's token-based scheduler [25]. These show that our method compares favorably with existing approaches.

Section 2 presents BIP [14]. Section 3 characterizes local and global deadlocks as the occurrence of a pattern of wait-for edges called a supercycle (see discussion above). Section 4 considers global supercycles, i.e., those that occur in the overall system being considered, characterizes these as the greatest fixpoint of a "blocking" operator, and presents some structural properties of supercycles. Section 5 presents global conditions for the prevention of the formation of supercycles. Global means that these conditions are evaluated in the entire system. Section 6 considers local supercycles, i.e., those that occur in a given subsystem of the overall system being considered. These are characterized as the greatest fixpoint of a "local blocking" operator, which pessimistically assumes that nodes on the boundary of the subsystem are blocked. This pessimistic assumption ensures soundness, at the expense of completeness. Section 7 presents local conditions for the prevention of the formation of supercycles. These can be evaluated in (small) subsystems of the overall system, and are obtained by "projecting" the global conditions onto a subsystem. Section 8 presents the main soundness and completeness results of the article, and gives the implication relation among our various conditions for deadlock freedom. Section 9 gives algorithms to evaluate the local conditions and presents experimental evaluation. Section 10 discusses related work, further work, and concludes.

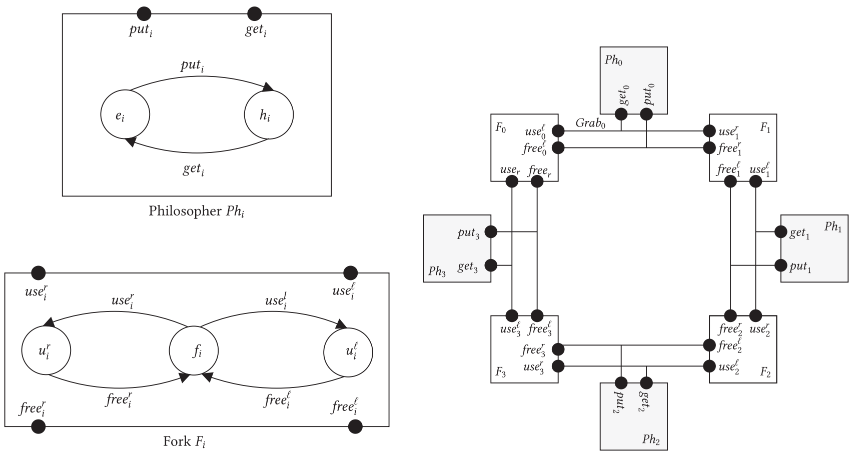
2 BIP — BEHAVIOR INTERACTION PRIORITY

BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. A technical treatment of priority is beyond the scope of this article. Adding priorities never introduces a deadlock, since priority enforces a choice between possible transitions from a state, and deadlock freedom means that there is at least one transition from every (reachable) state. Hence if a BIP system without priorities is deadlock-free, then the same system with priorities added will also be deadlock-free.

Definition 2.1 (Atomic component). An *atomic component* B_i is a labeled transition system represented by a triple $(Q_i, P_i, \rightarrow_i)$, where Q_i is a set of *states*, P_i is a set of *communication ports*, and $\rightarrow_i \subseteq Q_i \times P_i \times Q_i$ is a set of *transitions*, each labeled by some port.

For states $s_i, t_i \in Q_i$ and port $p_i \in P_i$, write $s_i \overset{p_i}{\rightarrow}_i t_i$, iff $(s_i, p_i, t_i) \in \rightarrow_i$. When p_i is clear from the context or not needed, we drop it from the transition and write $s_i \rightarrow_i t_i$. Similarly, $s_i \overset{p_i}{\rightarrow}_i$ means that there exists $t_i \in Q_i$ such that $s_i \overset{p_i}{\rightarrow}_i t_i$. In this case, p_i is *enabled* in state s_i. Ports are used for communication between different components, as discussed below.

Figure 1(a) shows atomic components for a philosopher Ph_i and a fork F_i in dining philosophers. A philosopher Ph_i that is hungry (in state h_i) can eat by executing get_i (to get its forks) and moving to state e_i (eating). From e_i, Ph_i releases its forks by executing put_i (to put down its forks) and moving back to h_i. Adding the thinking state does not change the deadlock behaviour of the system, since the thinking to hungry transition is internal to Ph_i, and so we omit it. A fork F_i is



(a) Philosopher Ph_i and fork F_i atomic components.

(b) Dining philosophers composite component with four philosophers.

Fig. 1. Dining philosophers.

taken by either: (1) the left philosopher (transition use_i^ℓ) and so moves to state u_i^ℓ (used by left philosopher), or (2) the right philosopher (transition use_i^r) and so moves to state u_i^r (used by right philosopher). From state u_i^r (resp. u_i^ℓ), F_i is released by the right philosopher (resp. left philosopher) and so moves back to state f_i (free).

In practice, we describe the transition system using some syntax, e.g., involving local variables (BIP does not have shared variables). We abstract away from issues of syntactic description since we are only interested in enablement of ports and actions. In BIP, the enablement of a port depends only on the local state of a component. In particular, it cannot depend on the state of other components. For example, state e_i in atomic component Ph_i of Figure 1(a) enables port put_i as there exists a transition from e_i labeled with port put_i. Hence, there exists a predicate $enb_{p_i}^i$ that holds in state s_i of component B_i iff port p_i is enabled in s_i, i.e., $s_i(enb_{p_i}^i) = true$ iff $s_i \xrightarrow{p_i}_i$.

Definition 2.2 (Interaction). For a given system built from a set of n atomic components $\{\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^n$, we require that their respective sets of ports are pairwise disjoint, i.e., for all i, j such that $i, j \in \{1..n\} \land i \neq j$, we have $P_i \cap P_j = \emptyset$. An *interaction* is a set of ports not containing two or more ports from the same component. That is, for an interaction a we have $\mathsf{a} \subseteq P \land (\forall i \in \{1..n\} : |\mathsf{a} \cap P_i| \leq 1)$, where $P = \bigcup_{i=1}^n P_i$ is the set of all ports in the system. When we write $\mathsf{a} = \{p_i\}_{i \in I}$, we assume that $p_i \in P_i$ for all $i \in I$, where $I \subseteq \{1..n\}$.

The connectors that connect ports in Figure 1(b) illustrate interactions. For example, the interaction $Grab_0 = \{get_0, use_0^\ell, use_1^r\}$ connects ports get_0, use_0^ℓ, and use_1^r from components Ph_0, F_0, and F_1 respectively, and corresponds to philosopher component Ph_0 acquiring both forks and moving to its eating state.

Execution of an interaction $\mathsf{a} = \{p_i\}_{i \in I}$ involves all the components that have ports in a. We denote by *components*(a) the set of atomic components participating in a. Formally, *components*(a) $= \{\mathsf{B}_i \mid p_i \in \mathsf{a}\}$.

Definition 2.3 (Composite Component). A *composite component* (or simply *component*) $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ is defined by a composition operator parameterized by a set of interactions $\gamma \subseteq 2^P$. B has a transition system $(Q, \gamma, \rightarrow)$, where $Q = Q_1 \times \cdots \times Q_n$ and $\rightarrow \subseteq Q \times \gamma \times Q$ is the least set of transitions satisfying the rule

$$\frac{\mathrm{a} = \{p_i\}_{i \in I} \in \gamma \quad \forall i \in I : s_i \xrightarrow{p_i}_i t_i \quad \forall i \notin I : s_i = t_i}{\langle s_1, \ldots, s_n \rangle \xrightarrow{\mathrm{a}} \langle t_1, \ldots, t_n \rangle}.$$

This inference rule says that a composite component $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$ can execute an interaction $\mathrm{a} \in \gamma$, iff for each port $p_i \in \mathrm{a}$, the corresponding atomic component B_i can execute a transition labeled with p_i; the states of components that do not participate in the interaction stay unchanged. Note that interactions are the only means of inter-component communication and synchronization in BIP.

Example 2.4 (Composite Component). Figure 1(b) shows a composite component consisting of four philosophers and the four forks between them. Each philosopher Ph_i $(0 \leq i \leq 3)$ and its two neighboring forks share two interactions: $Grab_i = \{get_i, use^{\ell}_i, use^r_{i+1}\}$ in which the philosopher obtains the forks, and $Rel_i = \{put_i, free^{\ell}_i, free^r_{i+1}\}$ in which the philosopher releases the forks (addition of indices being modulo 4).

Definition 2.5 (Interaction Enablement). An atomic component $\mathrm{B}_i = (Q_i, P_i, \rightarrow_i)$ enables a port $p_i \in P_i$ in state s_i iff $s_i \xrightarrow{p_i}_i$. B_i enables interaction a in state s_i iff $s_i \xrightarrow{p_i}_i$, where $\{p_i\} = P_i \cap \mathrm{a}$ is the port of B_i involved in a. That is, B_i enables a in state s_i iff B_i enables port $\mathrm{a} \cap P_i$ in state s_i.

Recall that $enb^i_{p_i}$ denotes the enablement condition for port p_i in component B_i, that is, $enb^i_{p_i}$ holds iff $s_i \xrightarrow{p_i}_i$, where s_i is the current state of B_i. Let enb^i_{a} denote the enablement condition for interaction a in component B_i, that is, $enb^i_{\mathrm{a}} = enb^i_{p_i}$ where $\{p_i\} = \mathrm{a} \cap P_i$.

Let $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$ be a composite component, let $s = \langle s_1, \ldots, s_n \rangle$ be a state of B, and let $\mathrm{a} = \{p_i\}_{i \in I}$ be an interaction of B. Then B enables a in s iff every $\mathrm{B}_i \in components(\mathrm{a})$ enables a in s_i, i.e., iff $\bigwedge_{i \in I} enb^i_{\mathrm{a}}$ holds in s.

The definition of interaction enablement is a consequence of Definition 2.3. Interaction a being enabled in state s means that executing a is one of the possible transitions that can be taken from s.

To avoid pathological cases of deadlock due solely to a single component refusing to enable any interaction at all, we assume that every component always enables at least one interaction. Structurally, this means that there is no local state with zero transitions, and every port labeling a transition is part of at least one interaction. Intuitively, component B_i in state s must enable at least one interaction a. However, a requires enablement from all components involved in it to execute. That might not be the case as a may be blocked by another component $\mathrm{B}_j \neq \mathrm{B}_i$. Therefore the assumption is not enough to guarantee deadlock freedom.

Definition 2.6 (Local Enablement Assumption). For every component $\mathrm{B}_i = (Q_i, P_i, \rightarrow_i)$, the following holds. In every $s_i \in Q_i$, B_i enables some interaction a.

Definition 2.7 (BIP-system). Let $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$ be a composite component with transition system $(Q, \gamma, \rightarrow)$, and let $Q_0 \subseteq Q$ be a set of initial states. Then (B, Q_0) is a BIP system.

Figure 1(b) gives a BIP-system with philosophers initially in state h (hungry) and forks initially in state f (free). To avoid tedious repetition, we fix, for the rest of the article, an arbitrary BIP-system (B, Q_0), with $\mathrm{B} \triangleq \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$, and transition system $(Q, \gamma, \rightarrow)$.

Definition 2.8 (Execution). Let $\rho = s_0 \mathrm{a}_1 s_1 \ldots s_{j-1} \mathrm{a}_j s_j \ldots$ be an alternating sequence of states of B and interactions of B. Then ρ is an execution of (B, Q_0) iff (1) $s_0 \in Q_0$ and (2) $\forall j > 0 : s_{j-1} \xrightarrow{\mathrm{a}_j} s_j$.

Definition 2.9 (Reachable State, Transition). A state or transition that occurs in some execution is called *reachable*. $rstates(\mathrm{B}, Q_0)$ denotes the set of reachable states of (B, Q_0).

Definition 2.10 (State Projection). Let $s = \langle s_1, \ldots, s_n \rangle$ be a state of (B, Q_0). Let $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\} \subseteq \{\mathrm{B}_1, \ldots, \mathrm{B}_n\}$. Then $s \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\} \triangleq \langle s_{i_1}, \ldots, s_{i_k} \rangle$. For a single B_i, we write $s \lceil \mathrm{B}_i = s_i$. We extend state projection to sets of states element-wise.

Definition 2.11 (Subcomponent). Let $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\} \subseteq \{\mathrm{B}_1, \ldots, \mathrm{B}_n\}$. Let $P' = P_{i_1} \cup \cdots \cup P_{i_k}$, i.e., the union of the ports of $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$. Then the subcomponent B' of B based on $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ is as follows:

(1) $\gamma' \triangleq \{\mathrm{a} \cap P' \mid \mathrm{a} \in \gamma \wedge \mathrm{a} \cap P' \neq \emptyset\}$.
(2) $\mathrm{B}' \triangleq \gamma'(\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k})$.

That is, γ' consists of those interactions in γ that have at least one participant in $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, and restricted to the participants in $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, i.e., participants not in $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ are removed.

We write $s \lceil \mathrm{B}'$ to indicate state projection onto B', and define $s \lceil \mathrm{B}' \triangleq s \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, where $\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}$ are the atomic components in B'. We say that $s \lceil \mathrm{B}'$ is a *state of* B'.

Definition 2.12 (Subsystem). Let $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\} \subseteq \{\mathrm{B}_1, \ldots, \mathrm{B}_n\}$. Then the subsystem (B', Q'_0) of (B, Q_0) based on $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ is as follows:

(1) B' is the subcomponent of B based on $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$.
(2) $Q'_0 = Q_0 \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$.

Definition 2.13 (Execution Projection). Let (B', Q'_0), with $\mathrm{B}' = \gamma'(\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k})$ be the subsystem of (B, Q_0) based on $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$. Let $P' = P_{i_1} \cup \cdots \cup P_{i_k}$, i.e., P' is the set of ports of (B', Q'_0). Let $\rho = s_0 \mathrm{a}_1 s_1 \ldots s_{j-1} \mathrm{a}_j s_j \ldots$ be an execution of (B, Q_0). Then, $\rho \lceil (\mathrm{B}', Q'_0)$, the projection of ρ onto (B', Q'_0), is the sequence resulting from

(1) replacing each s_j by $s_j \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, i.e., replacing each state by its projection onto $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, then
(2) removing all $\mathrm{a}_j s_j$ where $\mathrm{a}_j \cap P' = \emptyset$, then
(3) replacing each a_j by $\mathrm{a}_j \cap P'$, i.e., replacing each interaction by its projection onto the port set P'.

Proposition 2.14 (Execution Projection). *Let* (B', Q'_0)*, with* $\mathrm{B}' = \gamma'(\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k})$ *be the subsystem of* (B, Q_0) *based on* $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$*. Let* $P' = P_{i_1} \cup \cdots \cup P_{i_k}$*, i.e., the union of the ports of* $\{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$*. Let* $\rho = s_0 \mathrm{a}_1 s_1 \ldots s_{j-1} \mathrm{a}_j s_j \ldots$ *be an execution of* (B, Q_0)*. Then,* $\rho \lceil (\mathrm{B}', Q'_0)$ *is an execution of* (B', Q'_0).

Proof. By Definitions 2.10, 2.12, and 2.13, we have $\rho \lceil (\mathrm{B}', Q'_0) = s'_0 \mathrm{b}_1 s'_1 \mathrm{b}_2 s'_2 \ldots$ for some $s'_0, \mathrm{b}_1 s'_1 \mathrm{b}_2 s'_2 \ldots$, where $s'_j \in Q' = Q \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ for $j \geq 0$. Also by Definitions 2.10, 2.12, and 2.13, we have $s'_0 \in Q'_0 = Q_0 \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$, since $s'_0 = s_0 \lceil \mathrm{B}'$, and $s_0 \in Q_0$, by Definition 2.8.

Consider an arbitrary step $(s'_{j-1}, \mathrm{b}_j, s'_j)$ of $\rho \lceil (\mathrm{B}', Q'_0)$. Since $\mathrm{b}_j s'_j$ was not removed in Clause (2) of Definition 2.13, we have

(1) $s'_j = s_\ell \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ for some $\ell > 0$ and such that $\mathrm{a}_\ell \cap P' \neq \emptyset$.
(2) $\mathrm{b}_j = \mathrm{a}_\ell \cap P'$.
(3) $s'_{j-1} = s_m \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$ for the smallest m such that
$m < \ell$ and $\forall\, m' : m + 1 \leq m' < \ell : \mathrm{a}_{m'} \cap P' = \emptyset$.

From (3), we have $\forall\, m' : m + 1 \leq m' < \ell : \mathrm{a}_{m'} \cap P' = \emptyset$. So by Definitions 2.3 and 2.13, we have $s_m \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\} = s_{\ell-1} \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$. From (3), we have $s'_{j-1} = s_m \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$. Hence $s'_{j-1} = s_{\ell-1} \lceil \{\mathrm{B}_{i_1}, \ldots, \mathrm{B}_{i_k}\}$.

From $s_{\ell-1} \xrightarrow{a_\ell} s_\ell$, $a_\ell \cap P' \neq \emptyset$, and Definition 2.3, we have $s_{\ell-1}\restriction\{B_{i_1},\ldots,B_{i_k}\} \xrightarrow{a_\ell \cap P'} s_\ell\restriction\{B_{i_1},\ldots,B_{i_k}\}$. $s'_{j-1} = s_{\ell-1}\restriction\{B_{i_1},\ldots,B_{i_k}\}$ was established above. $s'_j = s_\ell\restriction\{B_{i_1},\ldots,B_{i_k}\}$ is from (1). $b_j = a_\ell \cap P'$ is from (2). Hence we obtain $s'_{j-1} \xrightarrow{b_j} s'_j$, i.e., that $s'_{j-1}, b_j s'_j$ is a step of (B', Q'_0).

Since (s'_{j-1}, b_j, s'_j) was arbitrarily chosen, we conclude that every step of $\rho\restriction(B', Q'_0)$ is a step of (B', Q'_0). This establishes Clause (2) of Definition 2.8. The first state of $\rho\restriction(B', Q'_0)$ is s'_0, and $s'_0 \in Q'_0$ was shown above, so we establish Clause (1) of Definition 2.8.

Since both clauses of Definition 2.8 are satisfied, we conclude that $\rho\restriction(B', Q'_0)$ is an execution of (B', Q'_0). □

Corollary 2.15. *Let (B', Q'_0) be a subsystem of (B, Q_0), and let P' be the port set of (B', Q'_0). Let s be a reachable state of (B, Q_0). Then $s\restriction B'$ is a reachable state of (B', Q'_0). Let $s \xrightarrow{a} t$ be a reachable transition of (B, Q_0), and let $a \cap P'$ be an interaction of (B', Q'_0). Then $s\restriction B' \xrightarrow{a \cap P'} t\restriction B'$ is a reachable transition of (B', Q'_0).*

Proof. Immediate corollary of Proposition 2.14. □

Example 2.16 (Execution Projection). In the dining philosophers example of Figure 1, let the (single) initial state be $(h_0, h_1, h_2, h_3, f_0, f_1, f_2, f_3)$, i.e., all philosophers are hungry and all forks are free. Also, $Grab_0 = \{get_0, use^\ell_0, use^r_1\}$ (resp. $Grab_2 = \{get_2, use^\ell_2, use^r_3\}$) is the interaction in which Ph_0 (resp. Ph_2) picks up both forks, and $Rel_0 = \{put_0, free^\ell_0, free^r_1\}$ is the interaction in which Ph_0 releases both forks. Consider the following execution: $\rho = (h_0, h_1, h_2, h_3, f_0, f_1, f_2, f_3)$ $\{get_0, use^\ell_0, use^r_1\}(e_0, h_1, h_2, h_3, u^\ell_0, u^r_1, f_2, f_3)\{get_2, use^\ell_2, use^r_3\}(e_0, h_1, e_2, h_3, u^\ell_0, u^r_1, u^\ell_2, u^r_3)\{put_0, free^\ell_0, free^r_1\}(h_0, h_1, e_2, h_3, f_0, f_1, u^\ell_2, u^r_3)\cdots$. The projection of this execution on the subsystem defined by subcomponent $\{Ph_0, Ph_1, F_0\}$, is equal to: $\rho\restriction\{Ph_0, Ph_1, F_0\}, Q'_0) = (h_0, h_1, f_0)\,\{get_0, use^\ell_0\}$ $(e_0, h_1, u^\ell_0)\,\{put_0, free^\ell_0\}\,(h_0, h_1, f_0)\cdots$. In particular, we project the states and interactions with respect to the subcomponent. Notice that interaction $Grab_2$ disappears as its ports do not belong to the subcomponent. Clearly, $\rho\restriction(\{Ph_0, Ph_1, F_0\}, Q'_0)$ is an execution of $(\{Ph_0, Ph_1, F_0\}, Q'_0)$.

3 CHARACTERIZING DEADLOCK FREEDOM

Definition 3.1 (Global Deadlock Freedom). A BIP-system (B, Q_0) is *free of global deadlock* iff in every reachable state s of (B, Q_0), some interaction a is enabled. Formally, $\forall s \in rstates(B, Q_0), \exists a : s \xrightarrow{a}_B$.

Definition 3.2 (Local Deadlock Freedom). A BIP-system (B, Q_0) is *free of local deadlock* iff for every subsystem (B', Q'_0) of (B, Q_0), and every reachable state s of (B, Q_0), (B', Q'_0) has some interaction enabled in state $s\restriction B'$. Formally,

for every subsystem (B', Q'_0) of (B, Q_0):

$$\forall s \in rstates(B, Q_0), \exists a : s\restriction B' \xrightarrow{a \cap P'}_{B'},$$

where P' is the set of ports of B'.

Note that every reachable state s' of subsystem B' (within the context of the overall system B) is a projection of a reachable state s of B, i.e., $s' = s\restriction B'$. Our definition requires that, in every reachable state s', B' is not prevented from executing *some* interaction a due to blocking relationships *within* B', which would constitute a *local deadlock*. We thus require that B', *considered in isolation from its containing system* B, must enable some interaction a. Within B overall, it is permissible for a to be disabled because some $B_i \in components(a)$ does not enable a, provided that B_i *is not a component of* B'. We now make these ideas precise.

3.1 Wait-For Graphs

The wait-for graph for a state s is a directed bipartite and-or graph which contains as nodes the atomic components $\mathrm{B}_1, \ldots, \mathrm{B}_n$, and all the interactions γ. Edges in the wait-for-graph are from a component B_i to all the interactions that B_i enables (in s), and from an interaction a to all the components that participate in a and that do not enable it (in s).

Definition 3.3 (Wait-for Graph $W_\mathrm{B}(s)$). Let $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$ be a BIP composite component, and let $s = \langle s_1, \ldots, s_n \rangle$ be an arbitrary state of B. The *wait-for graph* $W_\mathrm{B}(s)$ of s is a directed bipartite and-or graph, where

(1) the nodes of $W_\mathrm{B}(s)$ are as follows:
  (a) the and-nodes are the atomic components B_i, $i \in \{1..n\}$,
  (b) the or-nodes are the interactions $\mathrm{a} \in \gamma$,
(2) there is an edge in $W_\mathrm{B}(s)$ from B_i to every node a such that $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$ and $s_i(\mathit{enb}^i_\mathrm{a}) = \mathit{true}$, i.e., from B_i to every interaction which B_i enables in s_i,
(3) there is an edge in $W_\mathrm{B}(s)$ from a to every B_i such that $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$ and $s_i(\mathit{enb}^i_\mathrm{a}) = \mathit{false}$, i.e., from a to every component B_i which participates in a but does not enable it, in state s_i.

A component B_i is an and-node since all of its successor actions (or-nodes) must be disabled for B_i to be incapable of executing. An interaction a is an or-node since it is disabled if any of its participant components do not enable it. An edge (path) in a wait-for graph is called a wait-for edge (wait-for path).

Definition 3.4 (Subgraph of a Wait-for Graph). U is a subgraph of $W_\mathrm{B}(s)$ iff the nodes of U are a subset of the nodes of $W_\mathrm{B}(s)$ and the edges of U are the induced edges from $W_\mathrm{B}(s)$, i.e., if u, v are nodes of U and $u \longrightarrow v$ is an edge of $W_\mathrm{B}(s)$, then $u \longrightarrow v$ is an edge of U. Write $U \sqsubseteq W_\mathrm{B}(s)$ when U is a subgraph of $W_\mathrm{B}(s)$, and extend the definition of $\sqsubseteq$ to subgraphs of $W_\mathrm{B}(s)$ in the obvious manner, so that $U \sqsubseteq V$ means that U is a subgraph of V.

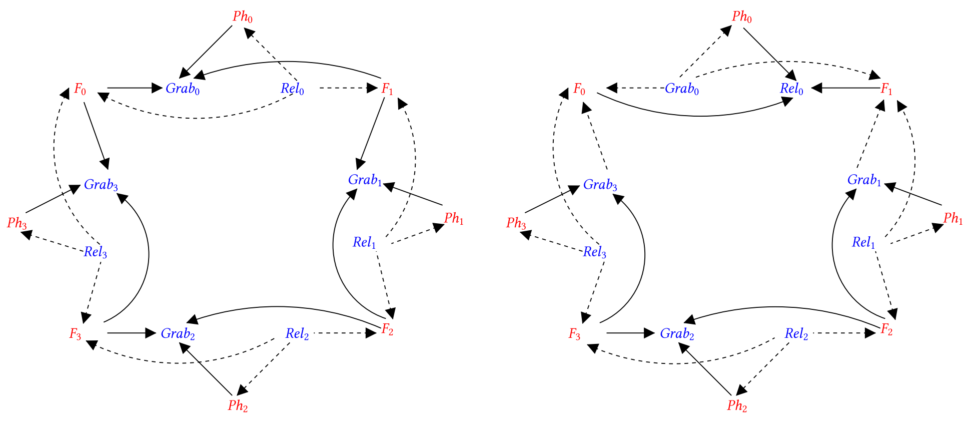
Write $\mathrm{a} \longrightarrow \mathrm{B}_i$ ($\mathrm{B}_i \longrightarrow \mathrm{a}$, respectively) for a wait-for edge from a to B_i (B_i to a, respectively). We abuse notation by writing $v \in W_\mathrm{B}(s)$ to indicate that v is a node of $W_\mathrm{B}(s)$, and $e \in W_\mathrm{B}(s)$ to indicate that e (either $\mathrm{a} \longrightarrow \mathrm{B}_i$ or $\mathrm{B}_i \longrightarrow \mathrm{a}$) is an edge in $W_\mathrm{B}(s)$. Also $\mathrm{B}_i \longrightarrow \mathrm{a} \longrightarrow \mathrm{B}'_i \in W_\mathrm{B}(s)$ for $\mathrm{B}_i \longrightarrow \mathrm{a} \in W_\mathrm{B}(s) \wedge \mathrm{a} \longrightarrow \mathrm{B}'_i \in W_\mathrm{B}(s)$, i.e., for a wait-for path of length 2, and similarly for longer wait-for paths. Likewise use $v \in U$, $e \in U$, where U is a subgraph of $W_\mathrm{B}(s)$.

Consider the dining philosophers system given in Figure 1. Figure 2(a) shows its wait-for graph in its sole initial state. Figure 2(b) shows the wait-for graph after execution of Grab_0. In all figures of wait-for graphs, we show components in red, interactions in blue, edges from components to interactions as solid, and edges from interactions to components as dashed.

A key principle of the dynamics of the change of wait-for graphs is that wait-for edges not involving some interaction a and its participants $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$ are unaffected by the execution of a. Say that edge e in a wait-for graph is B_i*-incident* iff B_i is one of the endpoints of e.

Proposition 3.5 (Wait-for Edge Preservation). *Let $s \xrightarrow{\mathrm{a}} t$ be a transition of composite component $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$, and let e be a wait-for edge in $W_\mathrm{B}(s)$ that is not B_i-incident, for every $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$. Then $e \in W_\mathrm{B}(s)$ iff $e \in W_\mathrm{B}(t)$.*

Proof. Fix e to be an arbitrary wait-for-edge that is not B_i-incident. e is either $\mathrm{B}_j \longrightarrow \mathrm{b}$ or $\mathrm{b} \longrightarrow \mathrm{B}_j$, for some component B_j of B that is not in $\mathit{components}(\mathrm{a})$, and an interaction b (different from a) that B_j participates in. Now $s \upharpoonright \mathrm{B}_j = t \upharpoonright \mathrm{B}_j$, since $s \xrightarrow{\mathrm{a}} t$ and $\mathrm{B}_j \notin \mathit{components}(\mathrm{a})$. Hence $s(\mathit{enb}^j_\mathrm{b}) = t(\mathit{enb}^j_\mathrm{b})$. It follows from Definition 3.3 that $e \in W_\mathrm{B}(s)$ iff $e \in W_\mathrm{B}(t)$. □



(a) Wait-for-graph in initial state. (b) Wait-for-graph after execution of $Grab_0$.

Fig. 2. Example wait-for-graphs for dining philosophers system of Figure 1.

3.2 Supercycles and Deadlock Freedom

We characterize a deadlock as the existence in the wait-for graph of a graph-theoretic construct that we call a *supercycle*.

Definition 3.6 (Supercycle). Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component and s be a state of B. A subgraph SC of $W_B(s)$ is a *supercycle* in $W_B(s)$ if and only if all of the following hold:

(1) SC is nonempty, i.e., contains at least one node,
(2) B_i is a node in SC, then for all interactions a such that there is an edge in $W_B(s)$ from B_i to a:
 (a) a is a node in SC, and
 (b) there is an edge in SC from B_i to a,
 that is, $B_i \to a \in W_B(s)$ implies $B_i \to a \in SC$,
(3) a is a node in SC, then there exists a B_j such that:
 (a) B_j is a node in SC, and
 (b) there is an edge from a to B_j in $W_B(s)$, and
 (c) there is an edge from a to B_j in SC,
 that is, $a \in SC$ implies $\exists B_j : a \to B_j \in W_B(s) \land a \to B_j \in SC$.

Intuitively, SC is a supercycle iff every node is SC is blocked from executing by other nodes in SC.

Definition 3.7 (Supercycle-free). $W_B(s)$ is *supercycle-free* iff there does not exist a supercycle SC in $W_B(s)$. In this case, say that state s is supercycle-free.

Figure 3 shows an example supercycle (with edges in bold) for the dining philosophers system of Figure 1. Ph_0 waits for (enables) a single interaction, $Grab_0$. $Grab_0$ waits for (is disabled by) fork F_0, which waits for interaction Rel_0. Rel_0 in turn waits for Ph_0. However, this supercycle occurs when Ph_0 is in state h_0 and F_0 is in state u_0^ℓ. This state is not reachable from the initial state.

Figure 4 shows an example of a supercycle that is not a simple cycle. The "essential" part of the supercycle, consisting of components B_1, B_2, B_3, and their interactions a, b, c, d, is in bold. The supercycle can be extended to contain B_4, but neither B_5 nor B_6: B_6 is enabled, and B_5 is ready to execute the interaction h, which waits only for B_6. Figure 5 shows that deleting the wait-for edge

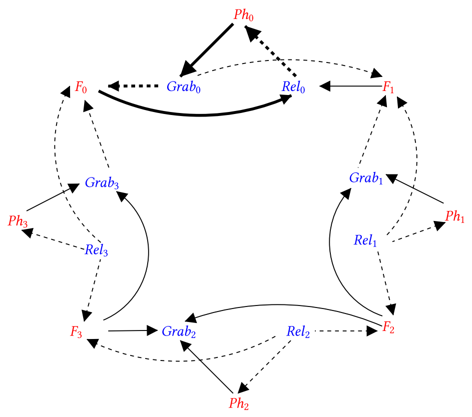


Fig. 3. Example supercycle for dining philosophers system of Figure 1.

from d to B_1 in Figure 4 results in an example where there is a cycle of wait-for edges, without there being a supercycle. This shows that a cycle does not necessarily imply a supercycle, and hence deadlock.

The existence of a supercycle is sufficient and necessary for the occurrence of a deadlock, and so checking for supercycles gives a sound and complete check for deadlocks. Proposition 3.8 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 3.8. *Let s be a state of* B. *If $SC \sqsubseteq W_B(s)$ is a supercycle, then all atomic components* B_i *in SC cannot execute a transition in any state u reachable from s, including s itself.*

Proof. Let B_i be an arbitrary component in *SC*. By Definition 3.6, every interaction that B_i enables has a wait-for edge to some other component B_j in *SC* and so cannot be executed in state s. Hence in any transition from s to another global state t, all of the components B_i in *SC* remain in the same local state. Hence $SC \sqsubseteq W_B(t)$, i.e., the same supercycle *SC* remains in global state t. Repeating this argument from state t and onwards leads us to conclude that $SC \sqsubseteq W_B(u)$ for any state u reachable from s.

Proposition 3.9 states that the existence of a supercycle is necessary for a local deadlock to occur: if a set of components, *considered in isolation*, are blocked, then there exists a supercycle consisting of exactly those components, together with the interactions that each component enables.

Proposition 3.9. *Let* B' *be a subcomponent of* B, *and let s be an arbitrary state of* B *such that* B', *when considered in isolation, has no enabled interaction in state* $s \upharpoonright B'$. *Then,* $W_B(s)$ *contains a supercycle.*

Proof. Let B_i be an arbitrary atomic component in B', and let a be any interaction that B_i enables. Since B' has no enabled interaction, it follows that a is not enabled in B', and therefore has a wait-for edge to some atomic component B_j in B'. Hence let *SC* be the subgraph of $W_B(s)$ induced by

(1) the atomic components of B',
(2) the interactions a that each atomic component B_i enables, and the edges $B_i \rightarrow$ a, and
(3) the edges a $\rightarrow B_j$ from each interaction a to some atomic component B_j in B', where B_j does not enable a.

SC satisfies Definition 3.6 and so is a supercycle.

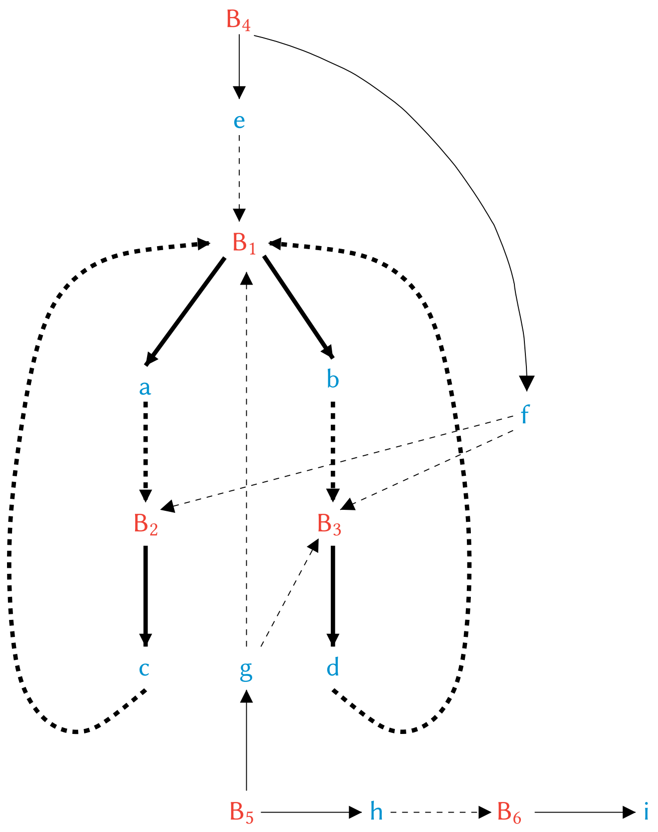


Fig. 4. Example supercycle that is not a simple cycle.

We consider subcomponent B′ in isolation to avoid other phenomena that prevent interactions from executing, e.g., conspiracies [9]. Now the contrapositive of Proposition 3.9 is that absence of supercycles in $W_B(s)$ means there is no locally deadlocked subsystem.

Corollary 3.10 (Supercycle-free Implies Free of Local Deadlock). *If, for every reachable state s of (B, Q_0), $W_B(s)$ is supercycle-free, then (B, Q_0) is free of local deadlock.*

Proof. Suppose that (B, Q_0) is not free of local deadlock. Then there exists a subsystem (B', Q'_0) of (B, Q_0), and a reachable state s of (B', Q'_0), such that B′ enables no interaction in state $s{\upharpoonright}B'$. By Proposition 3.9, $W_B(s)$ contains a supercycle. □

3.3 Subsystems and Supercycles

In the sequel, we say "deadlock-free" to mean free of local deadlock. We wish to check whether supercycles can be formed or not. In principle, we could check directly whether $W_B(s)$ contains a supercycle, for each reachable state s. However, this approach is subject to state-explosion, and so is usually unlikely to be viable in practice. Instead, we formulate global conditions for supercycle-freedom, and then "project" these conditions onto small subsystems, to obtain local versions of these conditions that are (1) efficiently checkable and (2) imply the global versions. To formulate the global conditions, we characterize the static (structural) and dynamic (formation) properties of supercycles in Sections 4 and 5, respectively. To define the projection of the global deadlock

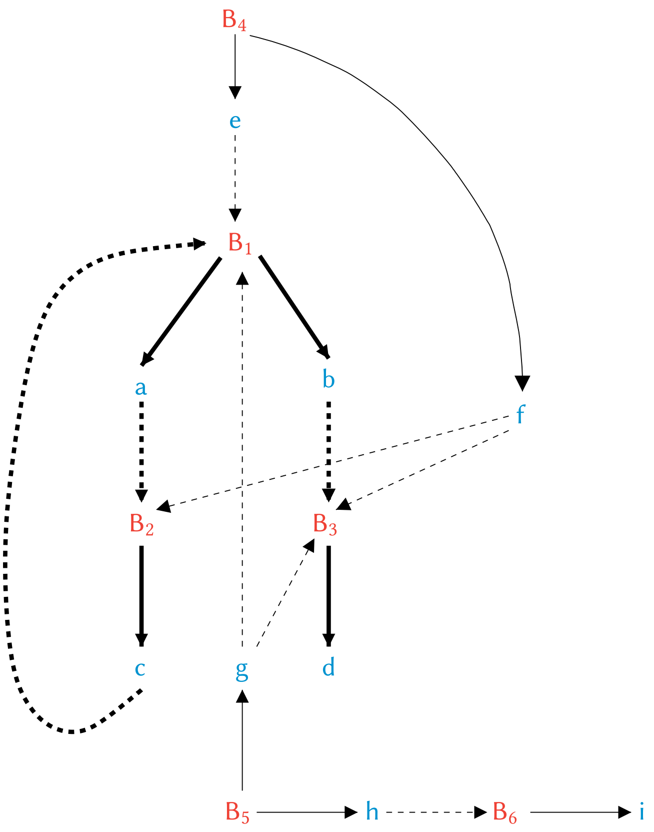


Fig. 5. Example where a wait-for cycle does not imply deadlock.

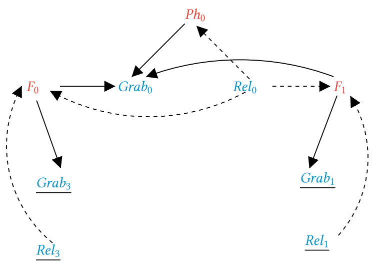


Fig. 6. Wait-for graph in the initial state of the dining philosophers subsystem for $Grab_0$ and distance 1.

freedom conditions onto small subsystems, we present the notion of *local supercycle* in Section 6. For each interaction a in the BIP-system (B, Q_0), the local check computes a subsystem which includes a and also other components/interactions at a given "distance" from a. It then checks whether any of the subsystem components is involved in a local supercycle.

Figure 6 illustrates the wait-for graph (in the initial state) of the dining philosopher subsystem corresponding to the $Grab_0$ interaction with a distance of 1, i.e., its components and their interactions. The subsystem includes the $Grab_0$ interaction, the components Ph_0, F_0, and F_1 that participate in $Grab_0$, and the interactions $Rel_0, Grab_1, Rel_1, Grab_3$, and Rel_3 which have at least one of these components as participants. We notice that no component in Figure 6 is involved in a supercycle. The interactions $Grab_1, Rel_1, Grab_3$, and Rel_3 (underlined in the figure) are "border interactions," since they have participating components that are outside the subsystem. The enablement of border interactions cannot be determined from the subsystem in isolation. Hence, to ensure soundness of our supercycle-freedom check, we must assume pessimistically that the border interactions are not enabled. This pessimism may yield a false negative, thereby causing our check to be incomplete. The further the distance is increased, the "more complete" the local check becomes, as the local states of the larger subsystem give a better over-approximation to the global states of the entire system.

3.4 Abstract Supercycle Freedom Conditions

Since we will present several conditions for supercycle freedom, we now present an abstract definition of the essential properties that all such conditions must have. The key idea is that execution of an interaction a does not create a supercycle, and so any condition which implies this for a is sufficient. If a different condition implies the same for another interaction aa, this presents no problem w.r.t. establishing deadlock freedom. Hence, it is sufficient to have one such condition for each interaction in (B, Q_0). Since each condition restricts the behavior of interaction execution, we call it a "behavioral restriction condition."

Definition 3.11 (Behavioral Restriction Condition). A *behavioral restriction condition* $\mathcal{BC}$ is a predicate $\mathcal{BC} : (\mathsf{B}, Q_0, \mathsf{a}) \rightarrow \{\mathsf{true}, \mathsf{false}\}$.

$\mathcal{BC}$ is a predicate on the effects of a particular interaction a within a given system (B, Q_0).

Definition 3.12 (Supercycle Freedom Preserving). A behavioral restriction condition $\mathcal{BC}$ is *supercycle freedom preserving* iff, for every BIP-system (B, Q_0) and interaction $\mathsf{a} \in \gamma$ of (B, Q_0):

if $\mathcal{BC}(\mathsf{B}, Q_0, \mathsf{a}) = \mathsf{true}$, then
    for every reachable transition $t \xrightarrow{\mathsf{a}} s$ of (B, Q_0)
        if t is supercycle-free, then s is supercycle-free.

Theorem 3.13 (Deadlock-freedom via Supercycle-freedom Preserving Restriction). *Assume that*

(1) for all $s_0 \in Q_0$, $W_{\mathsf{B}}(s_0)$ *is supercycle-free, and*
(2) there exists a supercycle freedom preserving restriction $\mathcal{BC}$ *such that, for all* $\mathsf{a} \in \gamma$: $\mathcal{BC}(\mathsf{B}, Q_0, \mathsf{a}) = \mathsf{true}$.

Then for every reachable state u of (B, Q_0): $W_{\mathsf{B}}(u)$ *is supercycle-free.*

Proof. Let u be an arbitrary reachable state. The proof is by induction on the length of the finite execution α that ends in u. Assumption 1 provides the base case, for α having length 0, and so $u \in Q_0$. For the induction step, we establish: for every reachable transition $t \xrightarrow{\mathsf{a}} s$, $W_{\mathsf{B}}(t)$ is supercycle-free implies that $W_{\mathsf{B}}(s)$ is supercycle-free. This is immediate from Assumption 2, and Definition 3.12.

Since the above proof does not make any use of the requirement that there is a single restriction $\mathcal{BC}$ for all interactions, we immediately have:

Corollary 3.14 (Deadlock Freedom via Several Supercycle Freedom Preserving Restrictions). *Assume that*

(1) for all $s_0 \in Q_0$, $W_B(s_0)$ is supercycle-free, and
(2) for all $a \in \gamma$, there exists a supercycle freedom preserving restriction $\mathcal{BC}$ such that $\mathcal{BC}(B, Q_0, a) = \text{true}$.

Then for every reachable state u of (B, Q_0): $W_B(u)$ is supercycle-free.

Proof. Similar to the proof of Theorem 3.13, except that, for the transition $t \xrightarrow{a} s$, use the supercycle freedom preserving restriction $\mathcal{BC}$ corresponding to a. □

4 GLOBAL SUPERCYCLES

Recall that (B, Q_0) is an arbitrary fixed BIP-system, (with $B = \gamma(B_1, \ldots, B_n)$), which we use in all definitions, theorems, and the like. We characterize a supercycle as a post-fixpoint of a blocking operator $\mathcal{S}$ (defined below) over the complete Boolean lattice formed from the subgraphs of $W_B(s)$, with $\sqsubseteq$ (Definition 3.4) as the ordering. Roughly, $\mathcal{S}$ maps a subset X of the nodes of $W_B(s)$ (i.e., some subset of the components and interactions in (B, Q_0)) to a set of nodes Y whose execution is blocked by X. An interaction a in Y is blocked by X if some participant of a is in X and does not enable a. A component B_i in Y is blocked by X if every interaction that B_i enables is in X. In terms of $W_B(s)$, a is blocked by X if there is a wait-for edge from a to some node in X, and B_i is blocked by X if every wait-for edge from B_i is to a node in X.

Since $\mathcal{S}$ is monotone, its greatest fixpoint SC exists. If $W_B(s)$ is supercycle-free, then SC is the empty wait-for graph $\emptyset$. Otherwise SC is the largest supercycle in $W_B(s)$. We define the dual $\mathcal{V}$ of $\mathcal{S}$, whose least fixpoints are the nodes that are not members of any supercycle, and we say that such nodes have a *supercycle violation.* Since $\mathcal{V}$ is monotone and continuous, and the underlying lattice is finite, its least fixpoint can be computed as usual by iterating $\mathcal{V}$, starting from $\emptyset$. This provides a method of computing the nodes with supercycle violations, which is the basis for our deadlock-freedom criterion.

4.1 A Fixpoint Characterization of Supercycles

Definition 4.1 (Set of Subgraphs). $\mathcal{P}(W_B(s)) = \{X \mid X \sqsubseteq W_B(s)\}$.

We include in $\mathcal{P}(W_B(s))$ the empty wait-for graph, which we denote by $\emptyset$. Let $nodes(B) = \{B_1, \ldots, B_n\} \cup \gamma$, i.e., $nodes(B)$ is the set of components and interactions in B, and let $\mathcal{P}(nodes(B))$ be the powerset of $nodes(B)$. Then $\mathcal{P}(W_B(s))$ is isomorphic to $\mathcal{P}(nodes(B))$, where each $X \in \mathcal{P}(W_B(s))$ is mapped to the set of nodes that it contains.

Definition 4.2 (Wait-for Lattice). Define the partially ordered set $\mathcal{L}_B(s) = \langle \mathcal{P}(W_B(s)), \sqsubseteq \rangle$ whose elements are all the subgraphs of $W_B(s)$, and where $U \sqsubseteq V$ is as in Definition 3.4.

The following proposition follows immediately from the definitions; its proof is left to the reader.

Proposition 4.3. *$\mathcal{L}_B(s) = \langle \mathcal{P}(W_B(s)), \sqsubseteq \rangle$ is a finite complete Boolean lattice as follows:*

— *meet is given by graph intersection: $X \sqcap Y$ consists of the nodes that are present in both X and Y, together with the edges induced by $W_B(s)$, i.e., if $u \in X \sqcap Y$, $v \in X \sqcap Y$, and $u \to v \in W_B(s)$, then $u \to v \in X \sqcap Y$.*
— *join is given by graph union: $X \sqcup Y$ consists of the nodes that are present in X, or in Y, or in both, together with the edges induced by $W_B(s)$. Note that $\sqcup$ is* not *disjoint graph union: it is possible for X and Y to have nodes and edges in common. Note also that $X \sqcup Y$ may contain edges not present in either X nor Y, since the edges are those induced by $W_B(s)$.*
— *$W_B(s)$ is the top element.*
— *the empty wait-for graph $\emptyset$ is the bottom element.*

— *the complement* $\overline{X}$ *of* X *is obtained by taking all the nodes of* $W_B(s)$ *that are not in* X*, together with the induced edges.*

As noted, $\sqcup, \sqcap$ and complement are determined entirely by the sets of nodes in the relevant subgraphs. The resulting edges are always those that are induced by $W_B(s)$. Let $\langle \mathcal{P}(nodes(B)), \subseteq\rangle$ be the lattice defined using the subset ordering $\subseteq$. Then $\mathcal{L}_B(s) = \langle \mathcal{P}(W_B(s)), \sqsubseteq\rangle$ is isomorphic to $\langle \mathcal{P}(nodes(B)), \subseteq\rangle$, where each $X \in \mathcal{P}(W_B(s))$ is mapped to the set of nodes that it contains.

Definition 4.4 (Blocks$_s$). Let $X \sqsubseteq W_B(s)$ and a, B_i be nodes in $W_B(s)$. Then $blocks_s(a, X) \triangleq (\exists\, B_i \in X : a \to B_i \in W_B(s))$, and $blocks_s(B_i, X) \triangleq (\forall\, a : B_i \to a \in W_B(s) \Rightarrow a \in X)$.

Hence an interaction a is blocked by a set of nodes X if some participant B_i of a is in X, and B_i does not enable a. A component B_i is blocked by X if all of the interactions that B_i enables are in X.

Definition 4.5 ($\mathcal{S}_s$). Define $\mathcal{S}_s : \mathcal{P}(W_B(s)) \to \mathcal{P}(W_B(s))$ as follows. $\mathcal{S}_s(X)$ is the subgraph with nodes $\{v \mid blocks_s(v, X)\}$, together with their induced edges.

Definition 4.6 ($\mathcal{V}_s$). Define $\mathcal{V}_s : \mathcal{P}(W_B(s)) \to \mathcal{P}(W_B(s))$ as follows. $\mathcal{V}_s(X)$ is the subgraph with nodes $\{v \mid \neg blocks_s(v, \overline{X})\}$, together with their induced edges.

Hence $\mathcal{V}_s(X) = \overline{\mathcal{S}_s(\overline{X})}$, i.e., $\mathcal{V}_s$ and $\mathcal{S}_s$ are duals. Note that $\mathcal{S}_s$ and $\mathcal{V}_s$ are defined given both a particular BIP system B and a particular state s of B. Hence we should really write $\mathcal{S}_{B,s}(X)$, $\mathcal{V}_{B,s}(X)$ to indicate this functional dependence. Since, however, B is a fixed BIP-system, we omit the B subscript to avoid notational clutter. In giving examples, we usually omit the subscript for the state, since the state will be implicitly given by the example.

Proposition 4.7. *$\mathcal{S}_s$ and $\mathcal{V}_s$ are monotone and continuous.*

Proof. We show first that $\mathcal{S}_s$ is monotone, i.e., $X \sqsubseteq Y \Rightarrow \mathcal{S}_s(X) \sqsubseteq \mathcal{S}_s(Y)$. Let v be an arbitrary node in $\mathcal{S}_s(X)$, so that $blocks_s(v, X)$ holds. There are two cases.

Case of v is an interaction a. By Definitions 4.4 and 4.5, we have $(\exists\, B_i \in X : a \to B_i \in W_B(s))$. Since $X \sqsubseteq Y$, this same B_i is also a node of Y, and so $\exists\, B_i \in Y : a \to B_i \in W_B(s)$. Hence $blocks_s(a, Y)$, and so $a \in \mathcal{S}_s(Y)$.

Case of v is a component B_i. By Definitions 4.4 and 4.5, we have $(\forall\, a : B_i \to a \in W_B(s) \Rightarrow a \in X)$. Since $X \sqsubseteq Y$, we have $(\forall\, a : B_i \to a \in W_B(s) \Rightarrow a \in Y)$. Hence $blocks_s(B_i, Y)$, and so $B_i \in \mathcal{S}_s(Y)$.

In both cases, we have $v \in \mathcal{S}_s(Y)$. Since v was chosen arbitrarily from $\mathcal{S}_s(X)$, it follows that $\mathcal{S}_s(X) \sqsubseteq \mathcal{S}_s(Y)$. Hence $\mathcal{S}_s$ is monotone. Since the dual of a monotone mapping in a complete Boolean lattice is also monotone, we have that $\mathcal{V}_s$ is monotone. Finally, since $\mathcal{L}_B(s)$ is finite, it follows that $\mathcal{S}_s$ and $\mathcal{V}_s$ are continuous. □

Hence, by the Knaster-Tarski theorem, the least and greatest fixpoints of $\mathcal{S}_s$ and $\mathcal{V}_s$ exist.

Proposition 4.8. *Let $X \neq \emptyset$ and $X \sqsubseteq W_B(s)$, i.e., X is a non-empty subgraph of $W_B(s)$. Then X is a supercycle in $W_B(s)$ iff $X \sqsubseteq \mathcal{S}_s(X)$.*

Proof. Let X be a supercycle in $W_B(s)$. By Definition 3.6, every node in X is blocked by X, i.e., $(\forall\, x \in X : blocks_s(x, X))$. By Definition 4.5, $X \sqsubseteq \mathcal{S}_s(X)$. Conversely, suppose $X \sqsubseteq \mathcal{S}_s(X)$ for some subgraph X of $W_B(s)$. Hence $(\forall\, x \in X : x \in \mathcal{S}_s(X))$, so by Definition 4.5, $(\forall\, x \in X : blocks_s(x, X))$. Hence every node in X is blocked by X, and so X satisfies Definition 3.6, and is therefore a supercycle. □

Thus, the supercycles of $W_B(s)$ are exactly the post-fixpoints of $\mathcal{S}$.

Proposition 4.9. *Let SC, SC' be supercycles in $W_B(s)$. Then $SC \sqcup SC'$ is a supercycle in $W_B(s)$.*

Proof. By Proposition 4.8, SC and SC' are post-fixpoints of $\mathcal{S}_s$. Since the join of post-fixpoints is a post-fixpoint, the proposition follows by applying Proposition 4.8 again. □

Proposition 4.10. *Let SC be the greatest post-fixpoint of $\mathcal{S}_s$. Then either (a) $W_B(s)$ is supercycle-free and $SC = \emptyset$, or (b) $W_B(s)$ contains supercycles, and SC is the largest supercycle in $W_B(s)$.* □

Proof. By the Knaster-Tarski theorem, the greatest post-fixpoint is the join of all the post-fixpoints. If $W_B(s)$ is supercycle-free, then by Proposition 4.8, the only post-fixpoint of $\mathcal{S}_s$ is $\emptyset$. Hence $SC = \emptyset$. If $W_B(s)$ contains supercycles, then by Proposition 4.8, the set of post-fixpoints of $\mathcal{S}_s$ is exactly the set of supercycles of $W_B(s)$. Hence SC is the join of all these supercycles. By Proposition 4.9, SC is itself a supercycle. Hence SC is the largest supercycle in $W_B(s)$. □

Let *lfp*, *gfp* denote the least fixpoint and greatest fixpoint operators, respectively.

Proposition 4.11. *$v \in lfp(\mathcal{V}_s)$ iff v is not a node in any supercycle of $W_B(s)$.*

Proof. From the Park conjugate (dual) fixpoint theorem in complete Boolean lattices [27], we have $lfp(\mathcal{V}_s) = \overline{gfp(\mathcal{S}_s)}$. By Proposition 4.10, $gfp(\mathcal{S}_s)$ is the largest supercycle in $W_B(s)$. Hence the nodes not in $gfp(\mathcal{S}_s)$ are exactly the nodes that are not in any supercycle. These are exactly the nodes in $lfp(\mathcal{V}_s)$. □

Define $\mathcal{V}_s^1(X) = \mathcal{V}_s(X)$, and for $d > 1$, $\mathcal{V}_s^d(X) = \mathcal{V}_s(\mathcal{V}_s^{d-1}(X))$, i.e., a superscript indicates functional iteration of $\mathcal{V}$. Also let $\bigsqcup$ be the "quantifier" version of $\sqcup$. Note that $\mathcal{V}_s^d(\emptyset) \sqsubseteq \mathcal{V}_s^{d'}(\emptyset)$ when $d \leq d'$, since $\mathcal{V}$ is monotone. Hence $\mathcal{V}_s^1(\emptyset), \mathcal{V}_s^2(\emptyset), \ldots$ is a non-decreasing sequence.

Proposition 4.12. *$lfp(\mathcal{V}_s) = \bigsqcup_{d \geq 1} \mathcal{V}_s^d(\emptyset)$.*

Proof. By Proposition 4.7, $\mathcal{V}_s$ is continuous. Follows by standard results, e.g., see the CPO fixpoint theorem I in Reference [20]. □

Definition 4.13 (Supercycle Violation, $\mathsf{viol}_B(v,s)$, $\mathsf{viol}_B(v,s,d)$). Let v be a node of $W_B(s)$. Define $\mathsf{viol}_B(v,s) \triangleq v \in lfp(\mathcal{V}_s)$ and, for $d \geq 1$, $\mathsf{viol}_B(v,s,d) \triangleq v \in \mathcal{V}_s^d(\emptyset)$.[1]

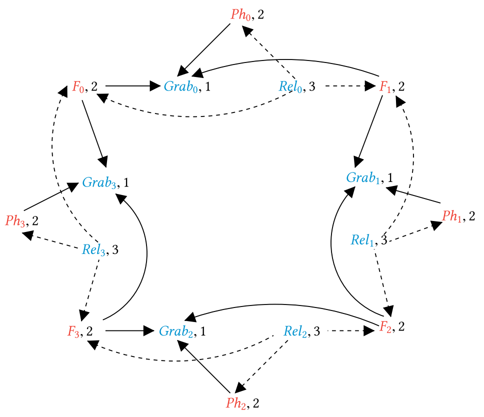
Proposition 4.14. $\mathsf{viol}_B(v,s)$ *iff* $(\exists\, d \geq 1 : \mathsf{viol}_B(v,s,d))$.

Proof. By Definition 4.13, $\mathsf{viol}_B(v,s) \equiv v \in lfp(\mathcal{V}_s)$. By Proposition 4.12, $v \in lfp(\mathcal{V}_s) \equiv v \in \bigsqcup_{d \geq 1} \mathcal{V}_s^d(\emptyset)$. By Definition 4.13, $(\forall\, d \geq 1 : \mathsf{viol}_B(v,s,d) \equiv v \in \mathcal{V}_s^d(\emptyset))$. Chaining these equivalences establishes the proposition. □

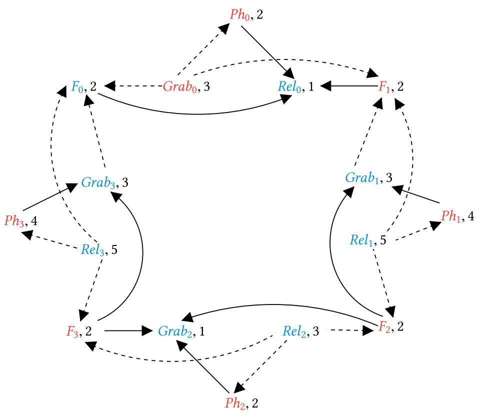
It follows from Proposition 4.11 that $\mathsf{viol}_B(v,s)$ iff there does not exist SC such that SC is a supercycle and $v \in SC$. We say that a node v of $W_B(s)$ has a *supercycle violation* iff v is not a node in any supercycle of $W_B(s)$, i.e., iff $\mathsf{viol}_B(v,s)$ holds. By Proposition 4.12, we can compute $lfp(\mathcal{V}_s)$ (and therefore $\mathsf{viol}_B(v,s)$) by iterating $\mathcal{V}_s$, starting from $\emptyset$, until there is no more change. $\mathsf{viol}_B(v,s,d)$ defines a supercycle violation that can be confirmed within d iterations of $\mathcal{V}_s$, which we call a *level-d supercycle violation*. $\mathsf{viol}_B(v,s)$ requires, in general, the entire least fixed point of $\mathcal{V}_s$.

Example 4.15 (Supercycle Violation). For example, consider the wait-for graph in Figure 4. We show the set of nodes in each $\mathcal{V}^d(\emptyset)$, since the induced subgraph is easily inferred from Figure 4.

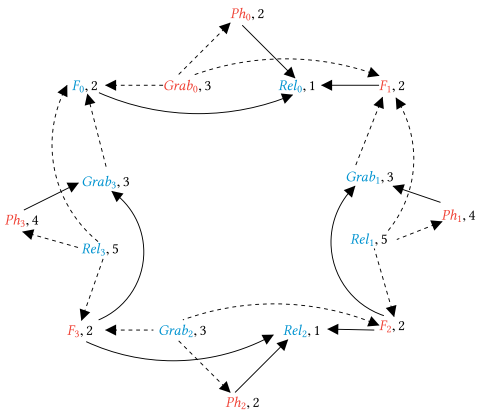
[1] Note that we abuse notation by overloading viol, but no ambiguity arises since the two versions have different parameter lists.



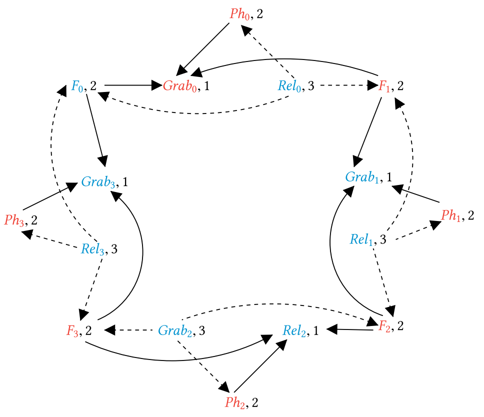
(a) Supercycle violations in initial state.



(b) Supercycle violations after execution of $Grab_0$.



(c) Supercycle violations after execution of $Grab_0; Grab_2$.



(d) Supercycle violations after execution of $Grab_0; Grab_2; Rel_0$.

Fig. 7. Example supercycle violations for dining philosophers system of Figure 1.

$\mathcal{V}^1(\emptyset) = \{\mathsf{i}\}$, $\mathcal{V}^2(\emptyset) = \{\mathsf{B}_6, \mathsf{i}\}$, $\mathcal{V}^3(\emptyset) = \{\mathsf{h}, \mathsf{B}_6, \mathsf{i}\}$, $\mathcal{V}^4(\emptyset) = \{\mathsf{B}_5, \mathsf{h}, \mathsf{B}_6, \mathsf{i}\}$, $\mathcal{V}^5(\emptyset) = \{\mathsf{B}_5, \mathsf{h}, \mathsf{B}_6, \mathsf{i}\}$, as so $\mathit{lfp}(\mathcal{V}) = \{\mathsf{B}_5, \mathsf{h}, \mathsf{B}_6, \mathsf{i}\}$. For Figure 5, it is easy to verify that $\mathit{lfp}(\mathcal{V})$ consists of all the nodes in the system, i.e., the wait-for graph shown is supercycle-free.

Example 4.16 (Supercycle Violations in Dining Philosophers). Figure 7 illustrates supercycle violations in four global states of the dining philosophers system of Figure 1. The states shown are the initial state, and the states resulting after execution of the indicated sequences of interactions. For each node v (interaction or component), we include a small positive integer after its name, giving the smallest d such that $v \in \mathcal{V}^d(\emptyset)$, i.e., the supercyle violation level.

Definition 4.17 (Supercycle Membership, $scyc_{\mathrm{B}}(v, s)$). Let v be a node of $W_{\mathrm{B}}(s)$. Then $scyc_{\mathrm{B}}(v, s)$ holds iff there exists a supercycle $SC \sqsubseteq W_{\mathrm{B}}(s)$ such that $v \in SC$.

Proposition 4.18. *Let v be a node of $W_{\mathrm{B}}(s)$. Then $\neg scyc_{\mathrm{B}}(v, s)$ iff $\mathsf{viol}_{\mathrm{B}}(v, s)$ That is, a node is not in any supercycle iff it has a supercycle violation.*

Proof. Immediate from Definition 4.13, Definition 4.17, and Proposition 4.11. □

4.2 Structural Properties of Supercycles

We present some structural properties of supercycles, which are central to our deadlock-freedom conditions.

Define $preds_{\mathrm{B}}(v, s) = \{w \mid w \rightarrow v \in W_{\mathrm{B}}(s)\}$ and $succs_{\mathrm{B}}(v, s) = \{w \mid v \rightarrow w \in W_{\mathrm{B}}(s)\}$. The definition of a supercycle (Definition 3.6) imposes certain constraints on supercycle membership of a node w.r.t. its predecessors and successors in the wait-for-graph, as follows:

Proposition 4.19 (Supercycle Membership Constraints). *Let* a, B_i *be nodes of* $W_{\mathrm{B}}(s)$. *Then*

(1) $scyc_{\mathrm{B}}(\mathrm{B}_i, s) \equiv (\forall\, \mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$.
(2) $scyc_{\mathrm{B}}(\mathrm{B}_i, s) \Rightarrow (\forall\, \mathrm{a} \in preds_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$.
(3) $scyc_{\mathrm{B}}(\mathrm{a}, s) \equiv (\exists\, \mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s) : scyc_{\mathrm{B}}(\mathrm{B}_i, s))$.
(4) $scyc_{\mathrm{B}}(\mathrm{a}, s) \Leftarrow (\exists\, \mathrm{B}_i \in preds_{\mathrm{B}}(\mathrm{a}, s) : scyc_{\mathrm{B}}(\mathrm{B}_i, s))$.

Proof. We deal with each clause in turn.

Proof of Clause 1. Assume $scyc_{\mathrm{B}}(\mathrm{B}_i, s)$, and let $SC \sqsubseteq W_{\mathrm{B}}(s)$ be the supercycle containing B_i. By Definition 3.6, Clause 2, $(\forall\, \mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$. We conclude $scyc_{\mathrm{B}}(\mathrm{B}_i, s) \Rightarrow (\forall\, \mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$. Now assume $(\forall\, \mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$, and let SC be the join of all the supercycles containing all the $\mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s)$. By Proposition 4.9, $SC \sqsubseteq W_{\mathrm{B}}(s)$ is a supercycle. Let SC' be SC with edge $\mathrm{B}_i \rightarrow \mathrm{a}$ added, for all $\mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s)$. Then SC' is a supercycle by Definition 3.6, and also $SC' \sqsubseteq W_{\mathrm{B}}(s)$. Hence $scyc_{\mathrm{B}}(\mathrm{a}, s)$. We conclude $scyc_{\mathrm{B}}(\mathrm{B}_i, s) \Leftarrow (\forall\, \mathrm{a} \in succs_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$.

Proof of Clause 2. Assume $scyc_{\mathrm{B}}(\mathrm{B}_i, s)$, so that $SC \sqsubseteq W_{\mathrm{B}}(s)$ is the supercycle containing B_i. Let $\mathrm{a} \in preds_{\mathrm{B}}(\mathrm{B}_i, s)$, and let SC' be SC with $\mathrm{a} \rightarrow \mathrm{B}_i$ added. Hence SC' is a supercycle by Definition 3.6, Clause 3. Since a was chosen arbitrarily, we conclude $(\forall\, \mathrm{a} \in preds_{\mathrm{B}}(\mathrm{B}_i, s) : scyc_{\mathrm{B}}(\mathrm{a}, s))$.

Proof of Clause 3. Assume $scyc_{\mathrm{B}}(\mathrm{a}, s)$, and let $SC \sqsubseteq W_{\mathrm{B}}(s)$ be the supercycle containing a. By Definition 3.6, Clause 3, there exists some $\mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s)$ such that $\mathrm{B}_i \in SC$. Hence $scyc_{\mathrm{B}}(\mathrm{B}_i, s)$. We conclude $scyc_{\mathrm{B}}(\mathrm{a}, s) \Rightarrow (\exists\, \mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s) : scyc_{\mathrm{B}}(\mathrm{B}_i, s))$. Now assume $(\exists\, \mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s) : scyc_{\mathrm{B}}(\mathrm{B}_i, s))$, and let $SC \sqsubseteq W_{\mathrm{B}}(s)$ be the supercycle containing some $\mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s)$. Let SC' be SC with $\mathrm{a} \rightarrow \mathrm{B}_i$ added. Then SC' is a supercycle by Definition 3.6, and also $SC' \sqsubseteq W_{\mathrm{B}}(s)$. Hence $scyc_{\mathrm{B}}(\mathrm{a}, s)$. We conclude $scyc_{\mathrm{B}}(\mathrm{a}, s) \Leftarrow (\exists\, \mathrm{B}_i \in succs_{\mathrm{B}}(\mathrm{a}, s) : scyc_{\mathrm{B}}(\mathrm{B}_i, s))$.

Proof of Clause 4. Assume $\neg scyc_{\mathrm{B}}(\mathrm{a}, s)$, so that a is not in any supercycle of $W_{\mathrm{B}}(s)$. Let $\mathrm{B}_i \in preds_{\mathrm{B}}(\mathrm{a}, s)$. By Definition 3.6, Clause 2, B_i cannot be in any supercycle of $W_{\mathrm{B}}(s)$, since all $\mathrm{aa} \in succs_{\mathrm{B}}(\mathrm{B}_i, s)$ must also be in the supercycle. Hence $\neg scyc_{\mathrm{B}}(\mathrm{B}_i, s)$. Since B_i was chosen arbitrarily, we conclude $\neg scyc_{\mathrm{B}}(\mathrm{a}, s) \Rightarrow (\forall\, \mathrm{B}_i \in preds_{\mathrm{B}}(\mathrm{a}, s) : \neg scyc_{\mathrm{B}}(\mathrm{B}_i, s))$, the contrapositive of Clause 4. □

Note that Clause 2 cannot be strengthened to an equivalence: if all the interactions that wait for a component B_i are in a supercycle, then B_i itself may or may not be in a supercycle, depending on whether B_i is waiting for some other interaction aa that is not in a supercycle. Likewise, Clause 4 cannot be strengthened to an equivalence: if a is in a supercycle, then any component B_i that waits for a may or may not be in a supercycle, depending on whether B_i is waiting for some other interaction aa that is not in a supercycle.

While Proposition 4.19 gives relationships between supercycle membership of a node and both its successors and predecessors, nevertheless Definition 3.6 implies that the "causality" of supercycle-membership of a node v is from the successors of v to v, i.e., membership of v in a supercycle is caused only by membership of v's successors in a supercycle. Repeating this step, we infer that v's supercycle-membership is caused by the subgraph of the wait-for graph that is reachable from v.

Proposition 4.20. *Every supercycle SC contains at least two nodes.*

Proof. By Definition 3.6, *SC* is nonempty, and so contains at least one node v. If v is an interaction a, then by Definition 3.6, *SC* also contains some component B_i such that $\mathrm{a} \rightarrow \mathrm{B}_i$. If v is a component B_i, then, by assumption, B_i enables at least one interaction a, and by Definition 3.6, every interaction that B_i enables must be in *SC*. Hence in both cases, *SC* contains at least two nodes. □

Proposition 4.21. *Every supercycle SC contains a maximal strongly connected component CC such that (1) CC is itself a supercycle and (2) there is no wait-for edge from a node in CC to a node outside of CC.*

Proof. *SC* is a directed graph, and so consider the decomposition of *SC* into its maximal strongly connected components (MSCC). Let *mscc*(*SC*) be the graph resulting from replacing each MSCC by a single node. By its construction, *mscc*(*SC*) is acyclic, and so contains at least one node x with no outgoing edges. Let *CC* be the MSCC corresponding to x. It follows from the construction of *CC* that no node in *CC* has a wait-for edge going to a node outside of *CC*, and so Clause (2) of the proposition is established.

It also follows from the construction of *CC* that *CC* is nonempty, and hence *CC* satisfies Clause (1) of Definition 3.6. Let v be an arbitrary node in *CC*. Since $CC \sqsubseteq SC$, v is a node of *SC*. Let w be an arbitrary successor of v in *SC*. Since no node in *CC* has an edge going to a node outside of *CC*, it follows that w is a node of *CC*. Hence v has the same successors in *CC* as in *SC*. Now since *SC* is a supercycle, every vertex v in *SC* has enough successors in *SC* to satisfy Clauses (2) and (3) of Definition 3.6. It follows that every vertex v in *CC* has enough successors in *CC* to satisfy Clauses (2) and (3) of Definition 3.6. Hence, by Definition 3.6, *CC* is itself a supercycle, and so Clause (1) of the proposition is established.

Note also that by Proposition 4.20, *CC* contains at least two nodes. Hence *CC* is not a trivial strongly connected component.

Definition 4.22 (Path, Path Length). Let G be a directed graph and v a vertex in G. A path π in G is a *finite* sequence $v_0, v_1, \ldots, v_n$ such that (v_i, v_{i+1}) is an edge in G for all $i \in \{0, \ldots, n-1\}$. Write $path_G(\pi)$ iff π is a path in G. Define $first(\pi) = v_0$ and $last(\pi) = v_n$. Let $|\pi|$ denote the length of π, which we define as follows:

— if π is simple, i.e., all v_i, $0 \le i \le n$, are distinct, then $|\pi| = n$, i.e., the number of edges in π
— if π contains a cycle, i.e., there exist v_i, v_j such that $i \neq j$ and $v_i = v_j$, then $|\pi| = \omega$ (ω for "infinity").

Definition 4.23 (In-depth, Out-depth). Let G be a directed graph and v a vertex in G. Define the in-depth of v in G, notated as $in\_depth_G(v)$, as follows:

— if there exists a path π in G that contains a cycle and ends in v, i.e., $|\pi| = \omega \wedge last(\pi) = v$, then $in\_depth_G(v) = \omega$,
— otherwise, let π be a longest (simple) path ending in v. Then $in\_depth_G(v) = |\pi|$.

Formally, $in\_depth_G(v) = (\text{MAX}\ \pi : path_G(\pi) \wedge last(\pi) = v : |\pi|)$.

Likewise, define the out-depth of v in G, notated as $out\_depth_G(v)$, as follows:

— if there exists a path π in G that contains a cycle and starts in v, i.e., $|\pi| = \omega \wedge first(\pi) = v$, then $out\_depth_G(v) = \omega$,
— otherwise, let π be a longest (simple) path starting in v. Then $out\_depth_G(v) = |\pi|$.

Formally, $out\_depth_G(v) = (\text{MAX}\ \pi : path_G(\pi) \wedge first(\pi) = v : |\pi|)$.

We use $in\_depth_{\mathrm{B}}(v,s)$ for $in\_depth_{W_{\mathrm{B}}(s)}(v)$, and also $out\_depth_{\mathrm{B}}(v,s)$ for $out\_depth_{W_{\mathrm{B}}(s)}(v)$. A node with finite in-depth is not reachable from any non-trivial (i.e., consisting of more than one node) MSCC, and a node with finite out-depth cannot reach any non-trivial MSCC.

Proposition 4.24. *Assume that node v of $W_{\mathrm{B}}(s)$ has a finite out-depth of $d \geq 0$ in $W_{\mathrm{B}}(s)$, i.e., $out\_depth_{\mathrm{B}}(v,s) = d$. Then* $\mathsf{viol}_{\mathrm{B}}(v,s,d+1)$.

Proof. Proof is by induction on d.

Base case, $d = 0$. Hence by $out\_depth_{\mathrm{B}}(v,s) = 0$ and Definitions 4.22 and 4.23, v has no outgoing wait-for edges in $W_{\mathrm{B}}(s)$. Hence $\neg blocks_s(v, W_{\mathrm{B}}(s))$, i.e., v is not blocked by the entire set of nodes in $W_{\mathrm{B}}(s)$. Hence $\neg blocks_s(v,\overline{\emptyset})$, since $W_{\mathrm{B}}(s) = \overline{\emptyset}$. So by Definition 4.6, $v \in \mathcal{V}_s(\emptyset)$. By Definition 4.13, $\mathsf{viol}_{\mathrm{B}}(v,s,1)$.

Inductive step, $d > 0$. Let w be an arbitrary successor of v, i.e., a node w such that $v \longrightarrow w \in W_{\mathrm{B}}(s)$. By Definitions 4.22 and 4.23, w has an out-depth d' that is less than d. That is, $out\_depth_{\mathrm{B}}(u,s) = d' < d$. By the induction hypothesis applied to d', we obtain $\mathsf{viol}_{\mathrm{B}}(w,s,d'+1)$, and so $w \in \mathcal{V}_s^{d'+1}(\emptyset)$ by Definition 4.13. Hence $w \in \mathcal{V}_s^{d}(\emptyset)$, since, by monotonicity of $\mathcal{V}_s$, we have $\mathcal{V}_s^{n'}(\emptyset) \sqsubseteq \mathcal{V}_s^{n}(\emptyset)$ when $n' \leq n$. Since w is an arbitrary successor of v, it follows that v is only blocked by nodes in $\mathcal{V}_s^{d}(\emptyset)$. Hence $\neg blocks_s(v, \overline{\mathcal{V}_s^{d}(\emptyset)})$. By Definition 4.6, $v \in \mathcal{V}_s(\mathcal{V}_s^{d}(\emptyset))$, i.e., $v \in \mathcal{V}_s^{d+1}(\emptyset)$. By Definition 4.13, $\mathsf{viol}_{\mathrm{B}}(v,s,d+1)$.

Corollary 4.25. *A supercycle SC contains no nodes with finite out-depth.*

Proof. Let v be a node in SC with finite out-depth d. By Proposition 4.24, $\mathsf{viol}_{\mathrm{B}}(v,s,d+1)$. By Definition 4.13, $\mathsf{viol}_{\mathrm{B}}(v,s)$. By Proposition 4.18 $\neg scyc_{\mathrm{B}}(v,t)$. Hence v cannot be a node of any supercycle, and we have a contradiction. □

Proposition 4.26. *Every supercycle SC contains at least one cycle.*

Proof. By contradiction. Suppose that SC is a supercycle and is also acyclic. Then every path in SC is simple, and therefore finite. Hence every node in SC has finite out-depth. By Proposition 4.24, SC cannot be a supercycle. □

Proposition 4.27. *Let SC be a supercycle in $W_{\mathrm{B}}(s)$, and let SC' be the graph obtained from SC by removing all vertices of finite in-depth and their incident edges. Then SC' is also a supercycle in $W_{\mathrm{B}}(s)$.*

Proof. A vertex with finite in-depth cannot lie on a cycle in SC. Hence by Proposition 4.26, $SC' \neq \emptyset$. Thus SC' satisfies Cause (1) of the supercycle Definition 3.6. Let v be an arbitrary vertex of SC'. Thus $v \in SC$ and $in\_depth_{SC}(v) = \omega$ by definition of SC'. Let w be an arbitrary successor of v in SC, i.e., $v \longrightarrow w \in SC$. Hence $in\_depth_{SC}(w) = \omega$ by Definition 4.23. Hence $w \in SC'$, by definition of SC'. Furthermore, $v \longrightarrow w \in SC'$, since SC' contains *all* nodes of SC with infinite in-depth. Hence the successors of v in SC' are the same as the successors of v in SC Now since SC is a supercycle, every vertex v in SC has enough successors in SC to satisfy Clauses (2) and (3) of the supercycle Definition (3.6). It follows that every vertex v in SC' has enough successors in SC' to satisfy Clauses (2) and (3) of the supercycle Definition 3.6. Hence SC' is a supercycle in $W_{\mathrm{B}}(s)$. □

5 GLOBAL CONDITIONS FOR DEADLOCK FREEDOM

5.1 The Supercycle Formation Condition

We use the structural properties of supercycles (Section 4.2) and the dynamics of wait-for graphs (Proposition 3.5) to define a condition that must hold whenever a supercycle is created. Negating this condition then implies the absence of supercycles.

Proposition 5.1 (Supercycle Formation Condition). *Assume that $t \xrightarrow{a} s$ is a transition of BIP-composite component $B = \gamma(B_1, \ldots, B_n)$, $W_B(t)$ is supercycle-free, and that $W_B(s)$ contains a supercycle. Then, in $W_B(s)$, there exists a CC such that*

1. *CC is a subgraph of $W_B(s)$, i.e., $CC \sqsubseteq W_B(s)$,*
2. *CC is strongly connected,*
3. *CC is a supercycle,*
4. *in $W_B(s)$, there is no wait-for edge from a node in CC to a node outside of CC, and*
5. *there exists a component $B_i \in components(a)$ such that B_i is in CC.*

Proof. By assumption, there is a supercycle SC that is a subgraph of $W_B(s)$. By Proposition 4.21, SC contains a subgraph CC that is strongly connected, is itself a supercycle, and such that there is no wait-for-edge from a node in CC to a node outside of CC. This establishes Clauses (1)–(4).

Now suppose $B_i \notin CC$ for every $B_i \in components(a)$. Then, no edge in CC is B_i-incident. Hence, by Proposition 3.5, every edge in CC is an edge in $W_B(t)$. Hence CC is a subgraph of $W_B(t)$. Now let v be an arbitrary node in CC. Suppose v is a component B_j. By assumption, $B_j \notin components(a)$, and so $s{\restriction}B_j = t{\restriction}B_j$ by Definition 2.3. Hence B_j enables the same set of interactions in state t as in state s. Also, in $W_B(s)$, all of B_j's wait-for edges must end in an interaction that is in CC, since CC is a supercycle in $W_B(s)$. Hence the same holds in $W_B(t)$. If v is an interaction, it must have a wait-for edge e' to some component $B_j \in CC$, since CC is a supercycle in $W_B(s)$. Hence this also holds in $W_B(t)$, since $s{\restriction}B_j = t{\restriction}B_j$. Hence v has enough successors in CC to satisfy the supercycle definition (Definition 3.6). We conclude that CC by itself is a supercycle in $W_B(t)$, which contradicts the assumption that $W_B(t)$ is supercycle-free. Hence, $B_i \in CC$ for some $B_i \in components(a)$, and so Clause (5) is established.

The supercycle formation condition (Proposition 5.1) tells us that, when a supercycle SC is created, some component B_i that participates in the interaction a whose execution created SC, must be a node of a strongly connected component CC of SC, and moreover CC is itself a supercycle in its own right. In a sense, CC is the "essential" part of SC. We use this to formulate a condition that prevents the formation of supercycles. For transition $t \xrightarrow{a} s$, we determine for every component $B_i \in components(a)$ whether it is possible for B_i to be a node in a strongly connected supercycle CC in $W_B(s)$. There are two ways for B_i to not be a node in a strongly connected supercycle:

1. *No supercycle membership*: B_i is not a node of any supercycle, i.e., $\neg scyc_B(B_i, s)$.
2. *No strong-connectedness*: B_i is a node in a supercycle, but not a node in a *strongly connected* supercycle.

Hence, for a BIP system (B, Q_0), our fundamental criterion for the prevention of supercycles is that, for every reachable transition $t \xrightarrow{a} s$ resulting from execution of a, in the resulting state s, every component B_i of a must violate at least one of the above two conditions. Condition (1) is just supercycle violation, as in Definition 4.13. Condition (2) is violation with respect to a strongly connected supercycle, i.e., non-membership in a strongly connected supercycle. Technically, this is implied by supercycle violation, and so the disjunction of the two conditions is equivalent to Condition (1). It is, however, convenient to use the disjunction of the two conditions, since we will formulate local versions of these violation conditions, and the implication does not necessarily hold for the local versions.

For a given BIP system (B, Q_0) and interaction a, we use $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ to denote the deadlock-freedom criterion based on the disjunction of Conditions (1) and (2) above. This criterion is, in a sense, the "most general" criterion for supercycle freedom. If $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds, global state t is supercycle-free, and $t \xrightarrow{\mathsf{a}} s$, then it follows (as we establish below) that global state s is also supercycle-free. So, by requiring (1) that all initial states are supercycle-free and (2) that $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds for all interactions $\mathsf{a} \in \gamma$, we obtain, by straightforward induction on length of executions, that every reachable state is supercycle-free.

It also follows that any condition which implies $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ is also sufficient to guarantee supercycle freedom, and hence deadlock freedom. We exploit this in two ways:

(1) To define a "linear" condition, $\mathcal{GLIN}$, that is easier to evaluate than $\mathcal{GALT}$, since it requires only the evaluation of lengths of wait-for paths, i.e., it does not have the "alternating" character of $\mathcal{GALT}$. It also implies $\mathcal{GALT}$.
(2) To define "local variants" of $\mathcal{GALT}$ and $\mathcal{GLIN}$, which we denote as $\mathcal{LALT}$ and $\mathcal{LLIN}$, respectively. $\mathcal{LALT}$ and $\mathcal{LLIN}$ can be evaluated in small subsystems of (B, Q_0). When either $\mathcal{LALT}$ or $\mathcal{LLIN}$ holds in a small subsystem, we confirm deadlock freedom of (B, Q_0) without state-explosion. The local conditions imply the corresponding global ones, i.e., they are sufficient but not necessary for deadlock freedom.

We therefore now have four deadlock-freedom conditions: $\mathcal{GALT}$ (global alternating), $\mathcal{LALT}$ (local alternating), $\mathcal{GLIN}$ (global linear), and $\mathcal{LLIN}$ (local linear). These are all concrete instances of the abstract version of the deadlock-freedom condition given in Section 3.4.

5.2 A Global AND-OR Condition for Deadlock-Freedom

We wish to show that the transition $t \xrightarrow{\mathsf{a}} s$ does not create a supercycle in state s. Towards this end, we first formalize violation of strong connectedness (Condition 2 above) as follows.

Definition 5.2 (Strong Connectedness Violation, $\mathsf{sConnViol_B}(v, s)$*).* Let v be a node of $W_\mathsf{B}(s)$. Then $\mathsf{sConnViol_B}(v, s)$ holds iff there does not exist a strongly connected supercycle SSC such that $v \in SSC$ and $SSC \sqsubseteq W_\mathsf{B}(s)$.

The general supercycle violation condition is then a disjunction of the supercycle violation condition and the strong connectedness violation conditions.

Definition 5.3 (General Supercycle Violation, $\mathsf{genViol_B}(v, s)$*).* Let v be a node of $W_\mathsf{B}(s)$. Then $\mathsf{genViol_B}(v, s) \triangleq \mathsf{viol_B}(v, s) \lor \mathsf{sConnViol_B}(v, s)$.

Let $t \xrightarrow{\mathsf{a}} s$ be a reachable transition. If, for every $\mathsf{B}_i \in$ *components*(a), $\mathsf{genViol_B}(v, s)$ holds, then, as we show below, $t \xrightarrow{\mathsf{a}} s$ does not introduce a supercycle, i.e., if t is supercycle-free, then so is s. As stated above, we formulate below a "local" version of the general condition, which can be evaluated in "small subsystems," and so we often avoid state-explosion. We reiterate that $\mathsf{viol_B}(v, s)$ implies that v cannot be in a supercycle. Hence v cannot be in a strongly connected supercycle. Hence $\mathsf{viol_B}(v, s) \Rightarrow \mathsf{sConnViol_B}(v, s)$, so that $\mathsf{viol_B}(v, s) \lor \mathsf{sConnViol_B}(v, s) \equiv \mathsf{sConnViol_B}(v, s)$. We give the formation violation condition in this manner, since the implication does not hold for the local versions of $\mathsf{viol_B}(v, s)$ and $\mathsf{sConnViol_B}(v, s)$.

This discussion leads to the formal definition of $\mathcal{GALT}$: after execution of interaction a, all $\mathsf{B}_i \in$ *components*(a) exhibit a general supercycle violation, as given by $\mathsf{genViol_B}(\mathsf{B}_i, s)$ above.

Definition 5.4 ($\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$)*.* Let $t \xrightarrow{\mathrm{a}} s$ be a reachable transition of (B, Q_0). Then, for every component $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$, the formation violation condition holds in state s. Formally,

$$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}), \mathsf{genViol}_{\mathrm{B}}(\mathrm{B}_i, s).$$

THEOREM 5.5. *$\mathcal{GALT}$ is supercycle freedom preserving.*

PROOF. We establish for every reachable transition $t \xrightarrow{\mathrm{a}} s$, $W_{\mathrm{B}}(t)$ is supercycle-free implies that $W_{\mathrm{B}}(s)$ is supercycle-free. Our proof is by contradiction, so we assume the existence of a reachable transition $t \xrightarrow{\mathrm{a}} s$ such that $W_{\mathrm{B}}(t)$ is supercycle-free and $W_{\mathrm{B}}(s)$ contains a supercycle. By Proposition 5.1 there exists a component $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$ such that B_i is in CC, where CC is a strongly connected supercycle that is a subgraph of $W_{\mathrm{B}}(s)$. Since CC is a strongly connected supercycle, we have, by Definition 5.2, that $\neg\mathsf{sConnViol}_{\mathrm{B}}(\mathrm{B}_i, s)$ holds. Since CC is a supercycle, we have, by Proposition 4.18, that $\neg\mathsf{viol}_{\mathrm{B}}(\mathrm{B}_i, s)$ holds. Hence, by Definition 5.3, $\neg\mathsf{genViol}_{\mathrm{B}}(\mathrm{B}_i, s)$. But, by Definition 5.4, we have $\mathsf{genViol}_{\mathrm{B}}(\mathrm{B}_i, s)$. Hence, we have the desired contradiction, and so the theorem holds. □

5.3 A Global Linear Condition for Deadlock-Freedom

In some cases, a simpler condition suffices to guarantee deadlock freedom. This simpler condition is "linear", i.e., it lacks the AND-OR alternation aspect of $\mathcal{GALT}$. After execution of a reachable transition $t \xrightarrow{\mathrm{a}} s$ of (B, Q_0), we consider the in-depth and out-depth of the components $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$. Suppose that there exists a supercycle $SC \sqsubseteq W_{\mathrm{B}}(s)$. There are three cases:

— *Case 1.* B_i has finite in-depth in $W_{\mathrm{B}}(s)$: then, if $\mathrm{B}_i \in SC$, it can be removed and still leave a supercycle SC', by Proposition 4.27. Hence $SC' \sqsubseteq W_{\mathrm{B}}(t)$, and so B_i is not essential to the creation of a supercycle.
— *Case 2.* B_i has finite out-depth in $W_{\mathrm{B}}(s)$: by Corollary 4.25, B_i cannot be part of a supercycle, and so $SC \sqsubseteq W_{\mathrm{B}}(t)$.
— *Case 3.* B_i has infinite in-depth and infinite out-depth in $W_{\mathrm{B}}(s)$: in this case, B_i is possibly an essential part of SC, i.e., SC was created in going from t to s.

We thus impose a condition which guarantees that only Case 1 or Case 2 occur.

Definition 5.6 ($\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$)*.* Let $t \xrightarrow{\mathrm{a}} s$ be a reachable transition of (B, Q_0). Then, in $W_{\mathrm{B}}(s)$, every component B_i of $\mathit{components}(\mathrm{a})$ either has finite in-depth, or has finite out-depth. Formally,
$\forall\, \mathrm{B}_i \in \mathit{components}(a) : \mathit{in\_depth}_{\mathrm{B}}(\mathrm{B}_i, s) < \omega \lor \mathit{out\_depth}_{\mathrm{B}}(\mathrm{B}_i, s) < \omega$.

PROPOSITION 5.7. *Assume that node v of $W_{\mathrm{B}}(s)$ has a finite in-depth of d in $W_{\mathrm{B}}(s)$, i.e., $\mathit{in\_depth}_{\mathrm{B}}(v, s) = d$. Then* $\mathsf{sConnViol}_{\mathrm{B}}(v, s)$.

PROOF. A node with finite in-depth cannot be in a wait-for cycle (i.e., a cycle of wait-for edges), and therefore cannot be in a strongly connected supercycle. □

LEMMA 5.8. *For all interactions* $\mathrm{a} \in \gamma : \mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$ *implies* $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$.

PROOF. Assume, for arbitrary $\mathrm{a} \in \gamma$, that $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$ holds. That is,
For every reachable transition $t \xrightarrow{\mathrm{a}} s$ of (B, Q_0),
$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathit{in\_depth}_{\mathrm{B}}(\mathrm{B}_i, s) < \omega \lor \mathit{out\_depth}_{\mathrm{B}}(\mathrm{B}_i, s) < \omega$.
By Propositions 4.24 and 5.7,
For every reachable transition $t \xrightarrow{\mathrm{a}} s$ of (B, Q_0),
$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathsf{sConnViol}_{\mathrm{B}}(\mathrm{B}_i, s) \lor (\exists\, d \geq 1 : \mathsf{viol}_{\mathrm{B}}(\mathrm{B}_i, s, d))$.
Hence by Proposition 4.14 and Definition 5.3,

For every reachable transition $t \xrightarrow{a} s$ of (B, Q_0),
$\forall \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathrm{genViol}_{\mathrm{B}}(\mathrm{B}_i, s)$
Hence $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$ holds. □

Theorem 5.9. *$\mathcal{GLIN}$ is supercycle freedom preserving*

Proof. Follows immediately from Theorem 5.5 and Lemma 5.8. □

5.4 Deadlock Freedom Using Global Restrictions

Corollary 5.10 (Deadlock-freedom via $\mathcal{GALT}, \mathcal{GLIN}$). *Assume that*

(1) *for all $s_0 \in Q_0$, $W_{\mathrm{B}}(s_0)$ is supercycle-free, and*
(2) *for all interactions a of B (i.e., $\mathrm{a} \in \gamma$): $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a}) \lor \mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$ holds.*

Then for every reachable state u of (B, Q_0): $W_{\mathrm{B}}(u)$ is supercycle-free, and so (B, Q_0) is free of local deadlock.

Proof. Immediate from Corollary 3.14, Theorem 5.5, and Theorem 5.9. □

6 LOCAL SUPERCYCLES

Evaluating the global restrictions $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$, $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$ requires checking all reachable transitions of (B, Q_0), which are, in general, subject to state-explosion. We need restrictions which imply $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$, $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$, and which can be checked efficiently. To this end, we first develop some terminology, and a projection result, for relating the waiting-behavior in a subsystem of (B, Q_0) to that in (B, Q_0) overall.

6.1 Projection onto Subsystems

Definition 6.1 (Projection of a Wait-for Graph). Let (B', Q_0') be a subsystem of (B, Q_0). $W_{\mathrm{B}}(s)\lceil \mathrm{B}'$ is the wait-for graph whose nodes are the components and interactions in B′, and whose edges are the induced edges from $W_{\mathrm{B}}(s)$, i.e., for nodes v, w of $W_{\mathrm{B}}(s)\lceil \mathrm{B}'$, $v \to w$ is an edge in $W_{\mathrm{B}}(s)\lceil \mathrm{B}'$ iff $v \to w$ is an edge in $W_{\mathrm{B}}(s)$.

Write $W_{\mathrm{B}'}(s)$ for $W_{\mathrm{B}}(s)\lceil \mathrm{B}'$. Also, if $s' = s\lceil \mathrm{B}'$, then define $W_{\mathrm{B}'}(s') \triangleq W_{\mathrm{B}}(s)\lceil \mathrm{B}'$, since $W_{\mathrm{B}}(s)\lceil \mathrm{B}'$ depends only on the projection of state s onto B′.

We now show that wait-for behavior in B "projects down" to any subcomponent B′, and that wait-for behavior in B′ "projects up" to B.

Proposition 6.2 (Wait-for Edge Projection). *Let B′ be a subcomponent of B. Let s be a state of B, and $s' = s\lceil \mathrm{B}'$. Let a be an interaction of B′, and $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$ be an atomic component of B′. Then (1) $\mathrm{a} \to \mathrm{B}_i \in W_{\mathrm{B}}(s)$ iff $\mathrm{a} \to \mathrm{B}_i \in W_{\mathrm{B}'}(s')$ and (2) $\mathrm{B}_i \to \mathrm{a} \in W_{\mathrm{B}}(s)$ iff $\mathrm{B}_i \to \mathrm{a} \in W_{\mathrm{B}'}(s')$.*

Proof. By Definition 3.3, $\mathrm{a} \to \mathrm{B}_i \in W_{\mathrm{B}}(s)$ iff $s\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = \mathit{false}$. Since $s' = s\lceil \mathrm{B}'$, we have $s'\lceil \mathrm{B}_i = s\lceil \mathrm{B}_i$. Hence $s\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = s'\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i})$. By Definition 3.3, $a \to \mathrm{B}_i \in W_{\mathrm{B}'}(s')$ iff $s'\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = \mathit{false}$. Putting together these three equalities gives us Clause (1).

By Definition 3.3, $\mathrm{B}_i \to \mathrm{a} \in W_{\mathrm{B}}(s)$ iff $s\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = \mathit{true}$. Since $s' = s\lceil \mathrm{B}'$, we have $s'\lceil \mathrm{B}_i = s\lceil \mathrm{B}_i$. Hence $s\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = s'\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i})$. By Definition 3.3, $\mathrm{B}_i \to \mathrm{a} \in W_{\mathrm{B}'}(s')$ iff $s'\lceil \mathrm{B}_i(\mathit{enb}_{\mathrm{a}}^{\mathrm{B}_i}) = \mathit{true}$. Putting the above three equalities together gives us Clause (2). □

Definition 6.3 (Structure Graph G_{B}, G_{a}^{ℓ}). The structure graph G_{B} of composite component $\mathrm{B} = \gamma(\mathrm{B}_1, \dots, \mathrm{B}_n)$ is a bipartite graph whose nodes are the $\mathrm{B}_1, \dots, \mathrm{B}_n$ and all the $\mathrm{a} \in \gamma$. There is an edge between B_i and interaction a iff B_i participates in a, i.e., $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$. Define the *distance* between two nodes to be the number of edges in a shortest path between them. Let G_{a}^{ℓ} be the subgraph of G_B that contains a and all nodes of G_B that have a distance to a which is less than or equal to ℓ.

Definition 6.4 (Deadlock-checking Subsystem, D_a^ℓ*).* Define D_a^ℓ, the *deadlock-checking subsystem for interaction* a *and radius* ℓ, to be the subsystem of (B, Q_0) based on the set of components in $G_a^{2\ell}$. (See Definition 2.12).

Definition 6.5 (Border Node, Interior Node of D_a^ℓ*).* A node v of D_a^ℓ is a *border-node* iff it has an edge in G_B to a node outside of D_a^ℓ. If node v of D is not a border node, then it is an *internal node.*

Note that all border nodes of D_a^ℓ are interactions, since 2ℓ is even. Hence all component nodes of D_a^ℓ are interior nodes.

In the sequel, we fix a particular subsystem D_a^ℓ, which we refer to simply as D, with a and ℓ being implicit (to avoid notational clutter with double-sub/superscripts). We write D.*action* = a and D.*radius* = ℓ. Also, let $Q_0^D = Q_0 \upharpoonright D$, i.e., Q_0^D is the set of initial states of D, and let s_D be an arbitrary state of D. As given above, for a state s_D of D, the wait-for graph for D only (i.e., ignoring the components and interactions of B that are not in D) is denoted as $W_D(s_D)$. Note also that "the nodes of D" and "the nodes of $W_D(s_D)$" denote the same set of components and interactions. We will use either expression, depending on context.

6.2 Fixpoint Characterization of Local Supercycles in a Subsystem

We now develop a local version of the sequence of definitions and propositions given in Section 4.1. Local means that they apply to any subsystem (B', Q_0') of (B, Q_0). A subsystem has, in general, border nodes, i.e., those nodes with a neighbor outside of the subsystem. The supercycle membership of these nodes cannot be determined with certainty, by inspecting just the subsystem. Hence we pessimistically assume that border nodes are in a supercycle. When false, this assumption may produce a false negative, and so we sacrifice completeness of our deadlock-freedom criterion. We do, however, avoid false positives (that may result if we assume a border node is not in a supercycle when, in fact, it is), and so we maintain soundness of our criterion.

Definition 6.6 (Local Supercycle). Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component, D a subsystem of B, and s_D a state of D. A subgraph SC of $W_D(s_D)$ is a local supercycle in $W_D(s_D)$ if and only if all of the following hold:

(1) SC is nonempty, i.e., contains at least one node,
(2) if B_i is a node in SC, then for all interactions a such that there is an edge in $W_D(s_D)$ from B_i to a:
    (a) a is a node in SC, and
    (b) there is an edge in SC from B_i to a,
    that is, $B_i \rightarrow a \in W_D(s_D)$ implies $B_i \rightarrow a \in SC$,
(3) if a is a node in SC, then, either a is a border interaction of D, or there exists a B_j such that:
    (a) B_j is a node in SC, and
    (b) there is an edge from a to B_j in $W_D(s_D)$, and
    (c) there is an edge from a to B_j in SC,
    that is, $a \in SC$ implies $(\exists B_j : a \rightarrow B_j \in W_D(s_D) \wedge a \rightarrow B_j \in SC)$ or (a is a border interaction of D).

Intuitively, SC is a supercycle iff every node in SC is blocked from executing by other nodes in SC, or is a border interaction. We pessimistically consider a border interaction a to be blocked, since the subsystem D cannot provide information about the participant components of a that are outside of D. In particular, one or more border interactions necessarily form a local supercycle. Yet, it is important to notice that a blocked border interaction a does not necessarily imply a global supercycle.

We carry over the definition of subgraph $\sqsubseteq$ from Section 4.1, and develop the analogous definitions for the subsystem D of B.

Definition 6.7 (Set of Subgraphs). $\mathcal{P}(W_D(s_D)) \triangleq \{X \mid X \sqsubseteq W_D(s_D)\}$.

Definition 6.8 (Wait-for Lattice). Define the partially ordered set $\mathcal{L}_D(s_D) = \langle \mathcal{P}(W_D(s_D)), \sqsubseteq \rangle$ whose elements are all the subgraphs of $W_D(s_D)$, and where $U \sqsubseteq V$ is as in Definition 3.4.

Proposition 6.9. *$\mathcal{L}_D(s_D) = \langle \mathcal{P}(W_D(s_D)), \sqsubseteq \rangle$ is a finite complete Boolean lattice, with $\sqcap$, $\sqcup$, and complementation as in Proposition 4.3, top element $W_D(s_D)$, and bottom element $\emptyset$.*

Definition 6.10 ($lblocks_{s_D}$). Let $X \sqsubseteq W_D(s_D)$ and a, B_i be nodes in $W_D(s_D)$. Then $lblocks_{s_D}(\mathrm{a}, X) \triangleq [(\exists\, \mathrm{B}_i \in X : \mathrm{a} \rightarrow \mathrm{B}_i \in W_D(s_D))$ or a is a border interaction of D], and $lblocks_{s_D}(\mathrm{B}_i, X) \triangleq (\forall\, \mathrm{a} : \mathrm{B}_i \rightarrow \mathrm{a} \in W_D(s_D) \Rightarrow \mathrm{a} \in X)$.

Hence a border interaction a is pessimistically considered to be always blocked, since the subsystem D does not contain enough information about the enablement of a. A non-border interaction a is (as usual) blocked by a set of nodes X if some participant B_i of a is in X, and B_i does not enable a. A component B_i is blocked by X if all of the interactions that B_i enables are in X.

Definition 6.11 ($\mathcal{SL}_{s_D}$). Define $\mathcal{SL}_{s_D} : \mathcal{P}(W_D(s_D)) \rightarrow \mathcal{P}(W_D(s_D))$ as follows. $\mathcal{SL}_{s_D}(X)$ is the subgraph with nodes $\{v \mid lblocks_{s_D}(v, X)\}$, together with the edges induced by $W_D(s_D)$.

Definition 6.12 ($\mathcal{VL}_{s_D}$). Define $\mathcal{VL}_{s_D} : \mathcal{P}(W_D(s_D)) \rightarrow \mathcal{P}(W_D(s_D))$ as follows. $\mathcal{VL}_{s_D}(X)$ is the subgraph with nodes $\{v \mid \neg lblocks_{s_D}(v, \widehat{X})\}$, together with the edges induced by $W_D(s_D)$, where we take the complement $\widehat{X}$ with respect to D.

Implicit in writing $\widehat{X}$ is that $X \sqsubseteq$ D. Then $\widehat{X}$ contains all nodes that are in D and not in X, together with the edges induced by $W_D(s_D)$. Hence $\mathcal{VL}_{s_D}(X) = \widehat{\mathcal{SL}_{s_D}(\widehat{X})}$, i.e., $\mathcal{VL}_{s_D}$ and $\mathcal{SL}_{s_D}$ are duals. Note that (as for $\mathcal{S}_s$ and $\mathcal{V}_s$) $\mathcal{SL}_{s_D}$ and $\mathcal{VL}_{s_D}$ are defined given a particular subsystem D of the system B, and a particular state s_D of D. Since the subsystem D is fixed, we will omit D from the subscript of $\mathcal{SL}_{s_D}$ and $\mathcal{VL}_{s_D}$.

Proposition 6.13. *$\mathcal{SL}_{s_D}$ and $\mathcal{VL}_{s_D}$ are monotone and continuous.*

Proof. We show first that $\mathcal{SL}_{s_D}$ is monotone, i.e., $X \sqsubseteq Y \Rightarrow \mathcal{SL}_{s_D}(X) \sqsubseteq \mathcal{SL}_{s_D}(Y)$. Let v be an arbitrary node in $\mathcal{SL}_{s_D}(X)$, so that $lblocks_{s_D}(v, X)$ holds. There are three cases.

Case of v is a border interaction of D. Then $lblocks_{s_D}(v, Y)$ by Definition 6.10, and so $v \in \mathcal{SL}_{s_D}(Y)$ by Definition 6.11.

Case of v is a non-border interaction a. By Definitions 6.10 and 6.11, we have $\exists\, \mathrm{B}_i \in X : \mathrm{a} \rightarrow \mathrm{B}_i \in W_D(s_D)$. Since $X \sqsubseteq Y$, this same B_i is also a node of Y, and so $\exists\, \mathrm{B}_i \in Y : \mathrm{a} \rightarrow \mathrm{B}_i \in W_D(s_D)$. Hence $lblocks_{s_D}(\mathrm{a}, Y)$, and so $\mathrm{a} \in \mathcal{SL}_{s_D}(Y)$.

Case of v is a component B_i. By Definitions 6.10 and 6.11, we have $(\forall\, \mathrm{a} : \mathrm{B}_i \rightarrow \mathrm{a} \in W_D(s_D) \Rightarrow \mathrm{a} \in X)$. Since $X \sqsubseteq Y$, we have $(\forall\, \mathrm{a} : \mathrm{B}_i \rightarrow \mathrm{a} \in W_D(s_D) \Rightarrow \mathrm{a} \in Y)$. Hence, $lblocks_{s_D}(\mathrm{B}_i, Y)$, and so $\mathrm{B}_i \in \mathcal{SL}_{s_D}(Y)$.

In all three cases, we have $v \in \mathcal{SL}_{s_D}(Y)$. Since v was chosen arbitrarily from $\mathcal{SL}_{s_D}(X)$, it follows that $\mathcal{SL}_{s_D}(X) \sqsubseteq \mathcal{SL}_{s_D}(Y)$. Hence, $\mathcal{SL}_{s_D}$ is monotone. Since the dual of a monotone mapping in a complete Boolean lattice is also monotone, we have that $\mathcal{VL}_{s_D}$ is monotone. Finally, since $\mathcal{L}_D(s_D)$ is finite, it follows that $\mathcal{SL}_{s_D}$ and $\mathcal{VL}_{s_D}$ are continuous.

Proposition 6.14. *Let $X \neq \emptyset$ and $X \sqsubseteq W_D(s_D)$, i.e., X is a non-empty subgraph of $W_D(s_D)$. Then, X is a local supercycle in $W_D(s_D)$ iff $X \sqsubseteq \mathcal{SL}_{s_D}(X)$.*

Proof. Let X be a local supercycle in $W_D(s_D)$. By Definition 6.6, every node in X is blocked by X or is a border interaction, i.e., $(\forall x \in X : lblocks_{s_D}(x, X))$. By Definition 4.5, $X \sqsubseteq \mathcal{SL}_{s_D}(X)$.

Conversely, suppose $X \sqsubseteq \mathcal{SL}_{s_D}(X)$ for some subgraph X of $W_D(s_D)$. Hence $(\forall x \in X : x \in \mathcal{SL}_{s_D}(X))$, so by Definition 6.11, $(\forall x \in X : lblocks_{s_D}(x, X))$. Hence every node in X is blocked by X or is a border interaction, and so X satisfies Definition 6.6, and is therefore a local supercycle.

Proposition 6.15. *Let SC, SC' be local supercycles in $W_D(s_D)$. Then $SC \sqcup SC'$ is a local supercycle in $W_D(s_D)$.* □

Proof. By Proposition 6.14, SC and SC' are post-fixpoints of $\mathcal{SL}_{s_D}$. Since the join of post-fixpoints is a post-fixpoint, the proposition follows by applying Proposition 6.14 again. □

Proposition 6.16. *Let SC be the greatest fixpoint of $\mathcal{SL}_{s_D}$. Then either (a) $W_D(s_D)$ is supercycle-free and $SC = \emptyset$, or (b) $W_D(s_D)$ contains local supercycles, and SC is the largest local supercycle in $W_D(s_D)$.*

Proof. By the Knaster-Tarski theorem, the greatest fixpoint is the join of all the post-fixpoints. If $W_D(s_D)$ is supercycle-free, then by Proposition 6.14, the only post-fixpoint of $\mathcal{SL}_{s_D}$ is $\emptyset$. Hence $SC = \emptyset$ (this is possible since there may be no border interactions). If $W_D(s_D)$ contains supercycles, then by Proposition 6.14, the set of post-fixpoints of $\mathcal{SL}_{s_D}$ is exactly the set of local supercycles of $W_D(s_D)$. Hence SC is the join of all these local supercycles. By Proposition 6.15, SC is the largest local supercycle in $W_D(s_D)$. □

Proposition 6.17. *$v \in lfp(\mathcal{VL}_{s_D})$ iff v is not a node in any local supercycle of $W_D(s_D)$.*

Proof. From the Park conjugate (dual) fixpoint theorem in complete Boolean lattices [27], we have $lfp(\mathcal{V}_{s_D}) = \overline{gfp(\mathcal{S}_{s_D})}$. By Proposition 4.10, $gfp(\mathcal{S}_{s_D})$ is the largest local supercycle in $W_D(s_D)$. Hence the nodes not in $gfp(\mathcal{S}_{s_D})$ are exactly the nodes that are not in any local supercycle. These are exactly the nodes in $lfp(\mathcal{V}_{s_D})$. □

Define $\mathcal{VL}^1_{s_D}(X) = \mathcal{VL}_{s_D}(X)$, and for $d > 1$, $\mathcal{VL}^d_{s_D}(X) = \mathcal{VL}_{s_D}(\mathcal{VL}^{d-1}_{s_D}(X))$, i.e., a superscript indicates functional iteration of $\mathcal{VL}_{s_D}$. Note that $\mathcal{VL}^d_{s_D}(\emptyset) \sqsubseteq \mathcal{VL}^{d'}_{s_D}(\emptyset)$ when $d \leq d'$, since $\mathcal{VL}_{s_D}$ is monotone. Hence $\mathcal{VL}^1_{s_D}(\emptyset), \mathcal{VL}^2_{s_D}(\emptyset), \ldots$ is a non-decreasing sequence.

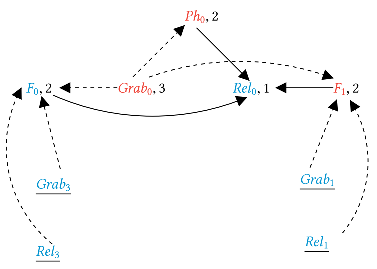
Proposition 6.18. *$lfp(\mathcal{VL}_{s_D}) = \bigsqcup_{d \geq 1} \mathcal{VL}^d_{s_D}(\emptyset)$.*

Proof. $\mathcal{VL}_{s_D}$ is continuous. Follows by standard results, e.g., see the CPO fixpoint theorem I in Reference [20]. □

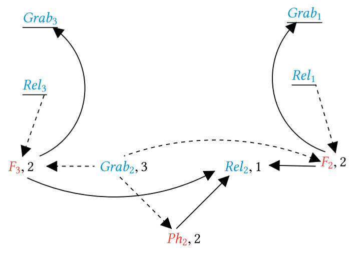
Definition 6.19 (Local Supercycle Violation, $\mathsf{violLoc}_D(v, s_D), \mathsf{violLoc}_D(v, s_D, d)$*).* Let s_D be a state of D and v be a node of D. Define $\mathsf{violLoc}_D(v, s_D) \triangleq v \in lfp(\mathcal{VL}_{s_D})$, and, for $d \geq 1$, $\mathsf{violLoc}_D(v, s_D, d) \triangleq v \in \mathcal{VL}^d_{s_D}(\emptyset)$.

Proposition 6.20. *$\mathsf{violLoc}_D(v, s_D)$ iff $(\exists d \geq 1 : \mathsf{violLoc}_D(v, s_D, d))$.*

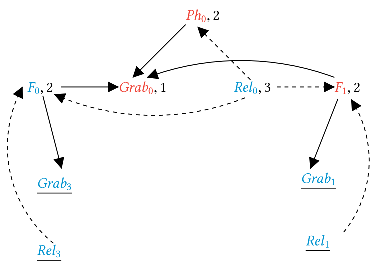
Proof. By Definition 6.19, $\mathsf{violLoc}_D(v, s_D) \equiv v \in lfp(\mathcal{VL}_{s_D})$. By Proposition 6.18, $v \in lfp(\mathcal{VL}_{s_D}) \equiv v \in \bigsqcup_{d \geq 1} \mathcal{VL}^d_{s_D}(\emptyset)$. By Definition 6.19, $(\forall d \geq 1 : \mathsf{violLoc}_D(v, s_D, d) \equiv v \in \mathcal{VL}^d_{s_D}(\emptyset))$. Chaining these equivalences establishes the proposition. □



(a) Local supercycle violations after execution of $Grab_0$.



(b) Local supercycle violations after execution of $Grab_0; Grab_2$.



(c) Local supercycle violations after execution of $Grab_0; Grab_2; Rel_0$.

Fig. 8. Example local supercycle violations for dining philosophers system of Figure 1.

$\mathsf{violLoc}_{\mathrm{D}}(v, s_{\mathrm{D}}, d)$ defines a local supercycle violation that can be confirmed within d iterations of $\mathcal{VL}_{s_{\mathrm{D}}}$, which we call a *level-d local supercycle violation*. $\mathsf{violLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$ requires, in general, the entire least fixpoint of $\mathcal{VL}_{s_{\mathrm{D}}}$.

Example 6.21 (Local Supercycle Violations in Dining Philosophers). Figures 8(a), 8(b), and 8(c) illustrate local supercycle violations corresponding to Figures 7(b), 7(c), and 7(d), respectively. The subsystem used in each case is based on the last interaction executed, i.e., $Grab_0$, $Grab_2$, and Rel_0, respectively, and with a radius of 1 in all cases. The border interactions are shown underlined, and for each node v (interaction or component), we include a small positive integer after its name, giving the smallest d such that $v \in \mathcal{VL}^d(\emptyset)$, i.e., the local supercycle violation level.

We now show that a local supercycle violation implies (global) supercycle violation.

Proposition 6.22. *Let s be an arbitrary global state of* B, *and let* $s_{\mathrm{D}} = s{\upharpoonright}\mathrm{D}$.
(a) Let $X \sqsubseteq W_{\mathrm{D}}(s_{\mathrm{D}})$. Then $\mathcal{VL}_{s_{\mathrm{D}}}(X) \sqsubseteq \mathcal{V}_s(X)$.
(b) Let $d \geq 1$. Then $\mathcal{VL}^d_{s_{\mathrm{D}}}(\emptyset) \sqsubseteq \mathcal{V}^d_s(\emptyset)$.

Proof. For (a), let $v \in \mathcal{VL}_{s_{\mathrm{D}}}(X)$. By Definition 6.12, $\neg lblocks_{s_{\mathrm{D}}}(v, \widehat{X})$. Now v is either an interaction a or a component B_i.

By Definition 6.10, if v is an interaction a, then it is not a border interaction, and furthermore there is no component $\mathsf{B}_i \in \widehat{X}$ such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathrm{D}}(s_{\mathrm{D}})$. Since $\widehat{X} \sqsubseteq \overline{X}$, we conclude $\neg blocks_s(v, \overline{X})$, and so $v \in \mathcal{V}_s(X)$.

By Definition 6.10, if v is a component B_i, then there exists an interaction a such that $\mathsf{B}_i \to \mathsf{a} \in W_{\mathrm{D}}(s_{\mathrm{D}})$ and $\mathsf{a} \notin \widehat{X}$. Hence $\mathsf{a} \in X$, and so $\mathsf{a} \notin \overline{X}$. Hence $\neg blocks_s(v, \overline{X})$, and so $v \in \mathcal{V}_s(X)$.

In both cases, the arbitrary element v of $\mathcal{VL}_{s_{\mathrm{D}}}(X)$ is also an element of $\mathcal{V}_s(X)$, and so $\mathcal{VL}_{s_{\mathrm{D}}}(X) \sqsubseteq \mathcal{V}_s(X)$.

We establish (b) by induction on d. The base case is $d = 1$, which is given by (a). For the induction step, $d > 1$, we have the induction hypothesis $\mathcal{VL}^{d-1}_{s_D}(\emptyset) \sqsubseteq \mathcal{V}^{d-1}_s(\emptyset)$. Hence $\mathcal{VL}_{s_D}(\mathcal{VL}^{d-1}_{s_D}(\emptyset)) \sqsubseteq \mathcal{VL}_{s_D}(\mathcal{V}^{d-1}_s(\emptyset))$ since $\mathcal{VL}_{s_D}$ is monotone. By (a) $\mathcal{VL}_{s_D}(\mathcal{V}^{d-1}_s(\emptyset)) \sqsubseteq \mathcal{V}_s(\mathcal{V}^{d-1}_s(\emptyset))$. Hence $\mathcal{VL}_{s_D}(\mathcal{VL}^{d-1}_{s_D}(\emptyset)) \sqsubseteq \mathcal{V}_s(\mathcal{V}^{d-1}_s(\emptyset))$, i.e., $\mathcal{VL}^{d}_{s_D}(\emptyset) \sqsubseteq \mathcal{V}^{d}_s(\emptyset)$, and so (b) is established. □

Proposition 6.23. *Let s be an arbitrary global state of* B. *For all interactions* $a \in \gamma$ *and* $\ell \geq 1$, *let* $D = D^\ell_a$ *and* $s_D = s \upharpoonright D$. *Then,*
(a) For all $d \geq 1$: $\mathsf{violLoc}_D(v, s_D, d)$ *implies* $\mathsf{viol}_B(v, s, d)$, *and*
(b) $\mathsf{violLoc}_D(v, s_D)$ *implies* $\mathsf{viol}_B(v, s)$.

Proof. For (a), assume $\mathsf{violLoc}_D(v, s_D, d)$ for some arbitrary $d \geq 1$. By Definition 6.19, $v \in \mathcal{VL}^d_{s_D}(\emptyset)$. By Proposition 6.22, $v \in \mathcal{V}^d_s(\emptyset)$. By Definition 4.13, $\mathsf{viol}_B(v, s, d)$.
For (b), assume $\mathsf{violLoc}_D(v, s_D)$. By Proposition 6.20, $\mathsf{violLoc}_D(v, s_D, d)$ for some $d \geq 1$. By (a), we have $\mathsf{viol}_B(v, s, d)$. By Proposition 4.14, we have $\mathsf{viol}_B(v, s)$. □

7 LOCAL CONDITIONS FOR DEADLOCK FREEDOM

7.1 A Local AND-OR Condition for Deadlock Freedom

We now seek a local condition, which we evaluate in D, and which implies $\mathcal{GALT}$. We define local versions of both $\mathsf{viol}_B(v, s, d)$ and $\mathsf{sConnViol}_B(v, s)$.

To achieve a local and conservative approximation of $\mathsf{viol}_B(v, s, d)$, we make the "pessimistic" assumption that the violation status of border nodes of D cannot be known, since it depends on nodes outside of D. Now, if an internal node v of D can be marked with a level-d *local* supercycle violation, by applying Definition 6.19 to D, and with the border nodes marked as non-violating, then it is also the case, as we show below, that v also has a level-d *global* supercycle violation, as per Definition 4.13.

To achieve a local and conservative approximation of $\mathsf{sConnViol}_B(v, s)$, we project onto the subsystem D.

7.1.1 Local Strong Connectedness Condition. We now present the local version of the strong connectedness violation condition, given above in Definition 5.2.

Definition 7.1 (Local Strong Connectedness Violation, $\mathsf{sConnViolLoc}_D(v, s_D)$*).* Let L be the nodes of $W_D(s_D)$ that have no local supercycle violation, i.e., $L = \{v \mid v \in D \wedge \neg\mathsf{violLoc}_D(v, s_D)\}$. Let $WL = W_D(s_D) \upharpoonright L$, i.e., WL is the subgraph of $W_D(s_D)$ consisting of the nodes with no local supercycle violation, and the edges between those nodes that are also edges in $W_D(s_D)$.
Let v be an arbitrary node in WL. Then, $\mathsf{sConnViolLoc}_D(v, s_D)$ holds iff

(1) there does not exist a nontrivial strongly connected supercycle SSC such that $v \in SSC$ and $SSC \sqsubseteq WL$, and
(2) either
  (a) there is no path in WL from v to a border node of D
  or
  (b) there is no path in WL from a border node of D to v.

Note that Clause (2a) means that every wait-for path π in $W_D(s_D)$ from v to a border node of D contains at least one node w with a local supercycle violation, i.e., $\mathsf{violLoc}_D(w, s_D)$. Also Clause (2b) means that every wait-for path π' in $W_D(s_D)$ from a border node of D to v contains at least one node w with a local supercycle violation, i.e., $\mathsf{violLoc}_D(w, s_D)$.

Table 1. Summary of Predicates

$\mathsf{viol}_{\mathsf{B}}(v, s, d)$	v determined at depth d to not be in supercycle
$\mathsf{violLoc}_{\mathsf{D}}(v, s_{\mathsf{D}}, d)$	v locally determined at depth d to not be in a supercycle
$\mathsf{sConnViol}_{\mathsf{B}}(v, s)$	v not in a strongly connected supercycle
$\mathsf{sConnViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$	v locally determined to not be in a strongly connected supercycle
$\mathsf{genViol}_{\mathsf{B}}(v, s)$	v does not contribute to a supercycle
$\mathsf{genViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$	v locally determined to not contribute to a supercycle

Table 1 summarizes the main predicates that we have defined. We show that the local strong connectedness condition implies the global strong connectedness condition.

Proposition 7.2. *Let* s *be an arbitrary state of* B. *For all interactions* $\mathsf{a} \in \gamma$, *and* $\ell \geq 1$, *let* $\mathsf{D} = \mathsf{D}^{\ell}_{\mathsf{a}}$, $s_{\mathsf{D}} = s{\restriction}\mathsf{D}$, *and let* v *be an arbitrary node in* D. *Then*

$\mathsf{sConnViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$ *implies* $\mathsf{sConnViol}_{\mathsf{B}}(v, s)$.

Proof. By contradiction. Assume for some state s of B and some node v in D that $\mathsf{sConnViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}}) \wedge \neg\mathsf{sConnViol}_{\mathsf{B}}(v, s)$ holds. By $\neg\mathsf{sConnViol}_{\mathsf{B}}(v, s)$ and Definition 5.2, there exists a strongly connected supercycle SSC such that $v \in SSC$ and $SSC \sqsubseteq W_{\mathsf{B}}(s)$. Then, there are two cases:

(1) $SSC \sqsubseteq W_{\mathsf{D}}(s_{\mathsf{D}})$: let x be any node in SSC. Since x is a node in a supercycle, we have by Definition 4.17 and Proposition 4.18, that $\neg\mathsf{viol}_{\mathsf{B}}(x, s)$. Hence, by Proposition 6.23, we have $\neg\mathsf{violLoc}_{\mathsf{D}}(x, s_{\mathsf{D}})$. Let WL be as given in Definition 7.1. Then $x \in WL$, and since x is an arbitrary node of SSC, we have $SSC \sqsubseteq WL$. Thus Clause (1) of Definition 7.1 is violated.

(2) $SSC \not\sqsubseteq W_{\mathsf{D}}(s_{\mathsf{D}})$: then there exists a node $x \in SSC - W_{\mathsf{D}}(s_{\mathsf{D}})$. Now $v \in SSC$ and SSC is strongly connected. Hence there must exist a wait-for path π in $W_{\mathsf{D}}(s_{\mathsf{D}})$ from v to x and a wait-for path π' in $W_{\mathsf{D}}(s_{\mathsf{D}})$ from x to v. Since $v \in \mathsf{D}$ and $x \notin \mathsf{D}$, it follows that both π and π' cross a border node of D. Furthermore, since π, π' are paths in SSC, every node w that is in π or in π' must be in a supercycle, and so cannot have a supercycle violation, i.e., $\neg\mathsf{viol}_{\mathsf{B}}(w, s)$. By Proposition 6.23, every node w that is in π or in π' cannot have a local supercycle violation, i.e., $\neg\mathsf{violLoc}_{\mathsf{D}}(w, s_{\mathsf{D}})$. Hence, Clauses (2a) and (2b) of Definition 7.1 are violated, since they require that at least one node in π and at least one node in π' has a local supercycle violation.

In both cases, Definition 7.1 is violated. But Definition 7.1 must hold, since we have $\mathsf{sConnViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$. Hence, the desired contradiction. □

7.1.2 *General Local Violation Condition.*

We showed above that local supercycle violation implies global supercycle violation, and local strong connectedness violation implies global strong connectedness violation. The general global supercycle violation condition is the disjunction of global supercycle violation and global strong connectedness violation. Hence, we formulate the general local supercycle violation condition as the disjunction of local supercycle violation and local strong connectedness violation. It follows that the general local supercycle violation condition implies the general global supercycle violation condition.

Definition 7.3 (General Local Supercycle Violation, $\mathsf{genViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$*).* Let v be an arbitrary node of D and s_{D} be an arbitrary state of D. Then $\mathsf{genViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}}) \triangleq \mathsf{violLoc}_{\mathsf{D}}(v, s_{\mathsf{D}}) \vee \mathsf{sConnViolLoc}_{\mathsf{D}}(v, s_{\mathsf{D}})$.

PROPOSITION 7.4 (LOCAL VIOLATION IMPLIES GLOBAL VIOLATION). *Let s be an arbitrary state of BIP composite component* B. *For all interactions* $\mathrm{a} \in \gamma$, *and* $\ell \geq 1$, *let* $\mathrm{D} = \mathrm{D}^{\ell}_{\mathrm{a}}$ *and* $s_{\mathrm{D}} = s{\restriction}\mathrm{D}$. *Also let v be an arbitrary node of* D. *Then,*

$\mathsf{genViolLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$ *implies* $\mathsf{genViol}_{\mathrm{B}}(v, s)$.

PROOF. Assume that $\mathsf{genViolLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$ holds. Then, by Definition 7.3, $\mathsf{violLoc}_{\mathrm{D}}(v, s_{\mathrm{D}}) \vee \mathsf{sConnViolLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$. We proceed by cases:

(1) $\mathsf{violLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$: hence $\mathsf{viol}_{\mathrm{B}}(v, s)$ by Proposition 6.23.
(2) $\mathsf{sConnViolLoc}_{\mathrm{D}}(v, s_{\mathrm{D}})$: hence $\mathsf{sConnViol}_{\mathrm{B}}(v, s)$ by Proposition 7.2.

By Definition 5.3, $\mathsf{genViol}_{\mathrm{B}}(v, s) \triangleq \mathsf{viol}_{\mathrm{B}}(v, s) \vee \mathsf{sConnViol}_{\mathrm{B}}(v, s)$. Hence we conclude that $\mathsf{genViol}_{\mathrm{B}}(v, s)$ holds. □

7.1.3 Local AND-OR Condition. The actual local condition, $\mathcal{LALT}$, is given by applying the general local supercycle violation condition to every reachable transition of the subsystem D being considered, and to every component $B_i \in \mathit{components}(\mathrm{a})$.

Definition 7.5 ($\mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$). Let $\ell \geq 1$, $\mathrm{D} = \mathrm{D}^{\ell}_{\mathrm{a}}$, $Q^{\mathrm{D}}_0 = Q_0{\restriction}\mathrm{D}$. Let $t_{\mathrm{D}} \xrightarrow{\mathrm{a}} s_{\mathrm{D}}$ be an arbitrary reachable transition of the subsystem $(\mathrm{D}, Q^{\mathrm{D}}_0)$. Then, in s_{D}, the following holds. For every $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$: B_i has a general local supercycle violation that can be confirmed within D. Formally,

$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathsf{genViolLoc}_{\mathrm{D}}(\mathrm{B}_i, s_{\mathrm{D}})$.

To summarize, $\mathcal{LALT}$ depends on two main ideas:

— Verification of supercycle violation can be done within a small subsystem, as given by Proposition 7.4, and
— Dynamic formation of supercycles, as given by Proposition 5.1, means that verification of supercycle violation is required only for participants of the last interaction that was executed.

We showed previously that $\mathcal{GALT}$ implies deadlock freedom, and so it remains to establish that $\mathcal{LALT}$ implies $\mathcal{GALT}$.

LEMMA 7.6. *Let* $\mathrm{a} \in \gamma$ *be an interaction of BIP-system* (B, Q_0). *Then*

$(\exists\, \ell \geq 1 : \mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell))$ *implies* $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$.

PROOF. Assume $\mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ for some $\ell \geq 1$, and let $\mathrm{D} = \mathrm{D}^{\ell}_{\mathrm{a}}$, $Q^{\mathrm{D}}_0 = Q_0{\restriction}\mathrm{D}$. Let $t \xrightarrow{\mathrm{a}} s$ be an arbitrary reachable transition of BIP-system (B, Q_0), and let $t_{\mathrm{D}} = t{\restriction}\mathrm{D}$, $s_{\mathrm{D}} = s{\restriction}\mathrm{D}$, so that $t_{\mathrm{D}} \xrightarrow{\mathrm{a}} s_{\mathrm{D}}$ is the projection of $t \xrightarrow{\mathrm{a}} s$ onto D. By Corollary 2.15, $t_{\mathrm{D}} \xrightarrow{\mathrm{a}} s_{\mathrm{D}}$ is a reachable transition of $(\mathrm{D}, Q^{\mathrm{D}}_0)$. By Definition 7.5,

for every reachable transition $t_{\mathrm{D}} \xrightarrow{\mathrm{a}} s_{\mathrm{D}}$ of $(\mathrm{D}, Q^{\mathrm{D}}_0)$:
$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathsf{genViolLoc}_{\mathrm{D}}(\mathrm{B}_i, s_{\mathrm{D}})$.

From this and Proposition 7.4,

for every reachable transition $t \xrightarrow{\mathrm{a}} s$ of (B, Q_0):
$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathsf{genViol}_{\mathrm{B}}(\mathrm{B}_i, s)$

Hence, by Definition 5.4, $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$ holds. □

THEOREM 7.7. $\mathcal{LALT}$ *is supercycle-freedom preserving.*

PROOF. Follows immediately from Theorem 5.5 and Lemma 7.6. □

Notice that Definition 7.5 calls $\mathsf{genViolLoc}_\mathrm{D}(v, s_\mathrm{D})$ on components, which by definition should be connected to at least one non-border interaction. As such, the trivial local supercycles, i.e., consisting only of border interactions, have no effect on supercycle formation.

7.2 A Local Linear Condition for Deadlock-Freedom

We now formulate a local version of $\mathcal{GLIN}$. Observe that if $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < \omega \vee \mathit{out\_depth}_\mathrm{B}(\mathrm{B}_i, s) < \omega$, then there is some finite ℓ such that $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) = \ell \vee \mathit{out\_depth}_\mathrm{B}(\mathrm{B}_i, s) = \ell$.

Definition 7.8 ($\mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$). Let $\ell \geq 1$, $\mathrm{D} = \mathrm{D}_\mathrm{a}^\ell$, $Q_0^\mathrm{D} = Q_0 \lceil \mathrm{D}$. Let $t_\mathrm{D} \xrightarrow{\mathrm{a}} s_\mathrm{D}$ be an arbitrary reachable transition of the subsystem $(\mathrm{D}, Q_0^\mathrm{D})$. Then, in s_D, the following holds. For every $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$: either B_i has in-depth less than $2\ell - 1$, or out-depth less than $2\ell - 1$, in $W_\mathrm{D}(s_\mathrm{D})$. Formally,

$$\forall\, \mathrm{B}_i \in \mathit{components}(\mathrm{a}) : \mathit{in\_depth}_\mathrm{D}(\mathrm{B}_i, s_\mathrm{D}) < 2\ell - 1 \vee \mathit{out\_depth}_\mathrm{D}(\mathrm{B}_i, s_\mathrm{D}) < 2\ell - 1.$$

To infer deadlock freedom in (B, Q_0) by checking $\mathcal{LLIN}(\mathrm{B}, Q_0, a, \ell)$, we use Proposition 6.2: since wait-for edges project up and down, it follows that wait-for paths project up and down, provided that the subsystem contains the entire wait-for path.

Proposition 7.9 (In-projection, Out-projection). *Let $\ell \geq 1$, let B_i be an atomic component of B, and let (B', Q_0') be a subsystem of (B, Q_0) which is based on a superset of $G_\mathrm{a}^{2\ell}$. Let s be a state of (B, Q_0), and $s' = s \lceil B'$. Then (1) $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1$ iff $\mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') < 2\ell - 1$, and (2) $\mathit{out\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1$ iff $\mathit{out\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') < 2\ell - 1$.*

Proof. We establish Clause (1). The proof of Clause (2) is analogous, except we replace paths ending in B_i by paths starting from B_i. The proof of Clause (1) is by double implication.

$\underline{\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1 \text{ implies } \mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') < 2\ell - 1}$: Assume that $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1$. Let π be an arbitrary wait-for path in $W_{\mathrm{B}'}(s')$ that ends in B_i. Since (B', Q_0') is a subsystem of (B, Q_0), by Definition 3.3 and $s' = s \lceil \mathrm{B}'$, $W_{\mathrm{B}'}(s')$ is a subgraph of $W_\mathrm{B}(s)$, i.e., $W_{\mathrm{B}'}(s') \sqsubseteq W_\mathrm{B}(s)$. Hence π is a wait-for path in $W_\mathrm{B}(s)$. By $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1$, we have $|\pi| < 2\ell - 1$. Hence $\mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') < 2\ell - 1$ since π was arbitrarily chosen.

$\underline{\mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') < 2\ell - 1 \text{ implies } \mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1}$: Assume that $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) \geq 2\ell - 1$. Then there exists a wait-for path π in $W_\mathrm{B}(s)$ such that $|\pi| \geq 2\ell - 1$ and π ends in B_i. Let ρ be the suffix of π with length $2\ell - 1$. Since (B', Q_0') is based on a superset of $G_a^{2\ell}$, and since the distance from B_i to the border of $G_\mathrm{a}^{2\ell}$ is $2\ell - 1$, we conclude that ρ is a wait-for path that is wholly contained in $W_{\mathrm{B}'}(s')$. Hence we have $\mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') \geq 2\ell - 1$. We have thus established $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) \geq 2\ell - 1$ implies $\mathit{in\_depth}_{\mathrm{B}'}(\mathrm{B}_i, s') \geq 2\ell - 1$. The contrapositive is the desired result.

We now show that $\mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ implies $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$, which in turn implies deadlock freedom.

Lemma 7.10. *Let a be an interaction of BIP-system (B, Q_0), i.e., $\mathrm{B} = \gamma(\mathrm{B}_1, \ldots, \mathrm{B}_n)$ and $\mathrm{a} \in \gamma$. If $\mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ holds for some finite $\ell \geq 1$, then $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$ holds.*

Proof. Let $t \xrightarrow{\mathrm{a}} s$ be a reachable transition of (B, Q_0) and let $\mathrm{B}_i \in \mathit{components}(\mathrm{a})$. Let $\mathrm{D} = \mathrm{D}_\mathrm{a}^\ell$ and $Q_0^\mathrm{D} = Q_0 \lceil \mathrm{D}$. Let $s_\mathrm{D} = s \lceil \mathrm{D}$, $t_\mathrm{D} = t \lceil \mathrm{D}$. Then $t_\mathrm{D} \xrightarrow{\mathrm{a}} s_\mathrm{D}$ is a reachable transition of $(\mathrm{D}, Q_0^\mathrm{D})$ by Corollary 2.15. By $\mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$, $\mathit{in\_depth}_\mathrm{D}(\mathrm{B}_i, s_\mathrm{D}) < 2\ell - 1 \vee \mathit{out\_depth}_\mathrm{D}(\mathrm{B}_i, s_\mathrm{D}) < 2\ell - 1$. Hence by Proposition 7.9, $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1 \vee \mathit{out\_depth}_\mathrm{B}(\mathrm{B}_i, s) < 2\ell - 1$. So $\mathit{in\_depth}_\mathrm{B}(\mathrm{B}_i, s) < \omega \vee \mathit{out\_depth}_\mathrm{B}(\mathrm{B}_i, s) < \omega$. Hence $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$. □

PROPOSITION 7.11. *Let $d < \ell$ and assume that node v of $W_D(s_D)$ has finite out-depth of $d \geq 1$ in $W_D(s_D)$, i.e., $out\_depth_D(v, s_D) = d$. Then* $\mathsf{violLoc}_D(v, s_D, d + 1)$.

PROOF. Proof is by induction on d.

Base case, $d = 0$. Hence by $out\_depth_D(v, s_D) = 0$, and Definitions 4.22 and 4.23, v has no outgoing wait-for edges in $W_D(s_D)$. By Definition 6.10, $\neg lblocks_{s_D}(v, W_D(s_D))$. By Definition 6.12, $v \in \mathcal{VL}_{s_D}(\emptyset)$. Hence $\mathsf{violLoc}_D(v, s_D, 1)$ by Definition 6.19.

Induction step $d > 0$. Assume $(out\_depth_D(v, s_D) = d)$. Let w be an arbitrary successor of v, i.e., a node w such that $v \rightarrow w \in W_D(s_D)$. By Definitions 4.22 and 4.23, w has an out-depth d' that is less than d.

By the induction hypothesis applied to d', we obtain $\mathsf{violLoc}_D(w, s_D, d' + 1)$, and so $w \in \mathcal{VL}^{d'+1}_{s_D}(\emptyset)$ by Definition 6.19. Hence $w \in \mathcal{VL}^{d}_{s_D}(\emptyset)$, since, by monotonicity of $\mathcal{VL}_{s_D}$, we have $\mathcal{VL}^{n'}_{s_D}(\emptyset) \sqsubseteq \mathcal{VL}^{n}_{s_D}(\emptyset)$ when $n' \leq n$. Since w is an arbitrary successor of v, it follows that v is only blocked by nodes in $\mathcal{VL}^{d}_{s_D}(\emptyset)$. Hence $\neg lblocks_{s_D}(v, \widehat{\mathcal{VL}^{d}_{s_D}(\emptyset)})$. By Definition 6.12, $v \in \mathcal{VL}_{s_D}(\mathcal{VL}^{d}_{s_D}(\emptyset))$, i.e., $v \in \mathcal{VL}^{d+1}_{s_D}(\emptyset)$. By Definition 6.19, $\mathsf{violLoc}_D(v, s_D, d + 1)$. □

LEMMA 7.12. *For all interactions* a *of* B*, i.e.,* $\mathsf{a} \in \gamma$*,*
$\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ *implies* $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$.

PROOF. Assume $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. Let $t_D \xrightarrow{\mathsf{a}} s_D$ be an arbitrary reachable transition of D, and let B_i be an arbitrary component of $components(\mathsf{a})$. Then, from Definition 7.8, we have:

$$in\_depth_D(\mathsf{B}_i, s_D) < 2\ell - 1 \vee out\_depth_D(\mathsf{B}_i, s_D) < 2\ell - 1.$$

The proof proceeds by two cases.

$in\_depth_D(\mathsf{B}_i, s_D) < 2\ell - 1$: Hence B_i cannot be in a strongly connected supercycle, because B_i would then lie on at least one wait-for cycle, and so would have infinite in-depth. Hence $\mathsf{sConnViolLoc}_D(\mathsf{B}_i, s_D)$ by Definition 7.1, Clause (1). Hence by Definition 7.3, $\mathsf{genViolLoc}_D(\mathsf{B}_i, s_D)$.

$out\_depth_D(\mathsf{B}_i, s_D) < 2\ell - 1$: Hence $out\_depth_D(\mathsf{B}_i, s_D) = d$ for some $d < 2\ell - 1$. By Proposition 7.11, $\mathsf{violLoc}_D(\mathsf{B}_i, s_D, d + 1)$. Hence by Definition 7.3, $\mathsf{genViolLoc}_D(\mathsf{B}_i, s_D)$.

In both cases, we have $\mathsf{genViolLoc}_D(\mathsf{B}_i, s_D)$. Since B_i is an arbitrarily chosen component of $components(\mathsf{a})$, we have $\forall\, \mathsf{B}_i \in components(\mathsf{a}) : \mathsf{genViolLoc}_D(\mathsf{B}_i, s_D)$. Hence, by Definition 7.5, we conclude $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. □

THEOREM 7.13. *$\mathcal{LLIN}$ is supercycle-freedom preserving*

PROOF. Follows immediately from Theorem 5.9 and Lemma 7.10. Also follows immediately from Theorem 5.5 and Lemma 7.12.

8 OVERALL SOUNDNESS, COMPLETENESS, AND IMPLICATION RESULTS

Figure 9 gives the implication relations between our four deadlock-freedom conditions. Each implication arrow is labeled by the lemma that provides the corresponding result.

We can use the four conditions together: if, for each interaction, we verify one of the conditions, then we can infer deadlock freedom, i.e., combining the conditions in this manner is still sound w.r.t. deadlock freedom.

THEOREM 8.1 (DEADLOCK FREEDOM VIA $\mathcal{GALT}$, $\mathcal{GLIN}$, $\mathcal{LALT}$, $\mathcal{LLIN}$). *Assume that*

(1) for all $s_0 \in Q_0$, $W_B(s_0)$ is supercycle-free, and
(2) for all interactions a *of* B *(i.e.,* $\mathsf{a} \in \gamma$*), one of the following holds:*

$$\begin{tikzcd}
\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a}) \arrow[r, "\text{Lemma 5.8}"] & \mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a}) \\
\mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell) \arrow[u, "\text{Lemma 7.10}"] \arrow[r, "\text{Lemma 7.12}"] & \mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell) \arrow[u, "\text{Lemma 7.6}"']
\end{tikzcd}$$

Fig. 9. Implication relations between deadlock-freedom conditions.

(a) $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$
(b) $\mathcal{GLIN}(\mathrm{B}, Q_0, \mathrm{a})$
(c) $\exists \ell \geq 1 : \mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$
(d) $\exists \ell \geq 1 : \mathcal{LLIN}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$

Then for every reachable state u of (B, Q_0): $W_{\mathrm{B}}(u)$ is supercycle-free, and so (B, Q_0) is free of local and global deadlock.

Proof. Immediate from Theorems 5.5, 5.9, 7.7, 7.13 and Corollary 3.14.

Finally, we establish that $\mathcal{GALT}$ is complete w.r.t. deadlock freedom: any system that is free of local and global deadlock will satisfy $\mathcal{GALT}$.

Theorem 8.2 (Completeness of $\mathcal{GALT}$ w.r.t. Deadlock Freedom). *Assume that (B, Q_0) is free from local and global deadlock. Then, for all interactions* a *of* B *(i.e.,* $\mathrm{a} \in \gamma$*),* $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$ *holds.*

Proof. Let a be an arbitrary interaction in γ, and let $t \xrightarrow{\mathrm{a}} s$ be a reachable transition of (B, Q_0). Hence s is a reachable state of (B, Q_0). Suppose that $W_{\mathrm{B}}(s)$ contains a supercycle SC. Then, by Proposition 3.8, the subcomponent B' consisting of all the atomic components $\mathrm{B}_i \in SC$ cannot execute a transition from any state reachable from s, and so is deadlocked. Hence (B, Q_0) has a local deadlock in reachable state s, contrary to assumption. Hence $W_{\mathrm{B}}(s)$ is supercycle-free.

Let v be an arbitrary node in $W_{\mathrm{B}}(s)$. By Definition 4.17, $\neg scyc_{\mathrm{B}}(v, s)$ holds. Hence by Proposition 4.18, $\mathsf{viol}_{\mathrm{B}}(v, s)$ holds. By Definition 5.3, $\mathsf{genViol}_{\mathrm{B}}(v, s)$ holds. Since v is an arbitrary node in $W_{\mathrm{B}}(s)$, and all $\mathrm{B}_i \in components(\mathrm{a})$ are nodes in $W_{\mathrm{B}}(s)$, we have $(\forall \mathrm{B}_i \in components(\mathrm{a}), \mathsf{genViol}_{\mathrm{B}}(\mathrm{B}_i, s))$. By Definition 5.4, $\mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a})$ holds. Since a is an arbitrary interaction in γ, we have $(\forall\, \mathrm{a} \in \gamma : \mathcal{GALT}(\mathrm{B}, Q_0, \mathrm{a}))$, and the theorem is established. □

9 IMPLEMENTATION AND EXPERIMENTS

To implement our deadlock-freedom conditions, we must:

(1) Check that all initial states are supercycle-free.
(2) Evaluate $\mathcal{LALT}$.
(3) Evaluate $\mathcal{LLIN}$.

Tasks 1 and 2 require the computation of $lfp(\mathcal{VL}_{s_{\mathrm{D}}})$. Figures 10, 11, and 12 present an algorithm that does this. Its correctness follows immediately from Definitions 6.10 and 6.12.

Note that $Y \sqcup \{v\}$ is the join of the wait-for-subgraph Y with the wait-for-subgraph v consisting of the single node v. Recall that the edges of $Y \sqcup \{v\}$ are induced from $W_{\mathrm{D}}(s_{\mathrm{D}})$.

```
COMPUTE-LFP(D, $s_D$)
▷ Precondition: $s_D$ is a state of D
▷ Postcondition: returns least fixpoint of $\mathcal{VL}_{s_D}$ i.e., $lfp(\mathcal{VL}_{s_D}) = \bigsqcup_{d \geq 1} \mathcal{VL}^d_{s_D}(\emptyset)$
1.  $X \leftarrow \emptyset$;
2.  while ($true$)
3.     ▷loop invariant: at end of $i$'th iteration, $X = \mathcal{VL}^i_{s_D}(\emptyset) = \bigsqcup_{1 \leq d \leq i} \mathcal{VL}^d_{s_D}(\emptyset)$
4.     $Y \leftarrow$ COMPUTE-VL(D, $s_D$, $X$);
5.     if ($X = Y$) return($X$);                                   ▷reached fixpoint
6.     $X \leftarrow Y$                                    ▷$X \neq Y$, so keep iterating
7.  endwhile
```

Fig. 10. Procedure to compute $lfp(\mathcal{VL}_{s_D})$.

```
COMPUTE-VL(D, $s_D$, $X$)
▷ precondition: $s_D$ is a state of D and $X \sqsubseteq W_D(s_D)$
▷ postcondition: returns $\mathcal{VL}_{s_D}(X)$
1.  compute $W_D(s_D)$;
2.  $Y \leftarrow \emptyset$;
3.  forall $v \in W_D(s_D)$
4.     if ($\neg$COMPUTE-LBLOCKS($v$, D, $s_D$, $\widehat{X}$))
5.        $Y \leftarrow Y \sqcup \{v\}$                          ▷satisfies Definition 6.12
6.  endfor;
7.  return($Y$)
```

Fig. 11. Procedure to compute $\mathcal{VL}_{s_D}(X)$.

```
COMPUTE-LBLOCKS($v$, D, $s_D$, $X$)
▷ precondition: $s_D$ is a state of D, $X \sqsubseteq W_D(s_D)$, and $v \in W_D(s_D)$
▷ postcondition: returns $lblocks_{s_D}(v, X)$
1.     if ($v$ is a border interaction of D)
2.        return($true$)            ▷border interaction pessimistically assumed to be blocked
3.     else if ($v$ is an interior interaction a and ($\exists\, B_i \in X : a \rightarrow B_i \in W_D(s_D)$))
4.        return($true$)              ▷some outgoing wait-for edge goes to a component in $X$
5.     else if ($v$ is a component $B_i$ and ($\forall\, a : B_i \rightarrow a \in W_D(s_D) \Rightarrow a \in X$))
6.        return($true$)          ▷every outgoing wait-for edge goes to some interaction in $X$
7.     else
8.        return($false$)              ▷all cases for blocking do not hold, cf. Definition 6.10
9.     fi
```

Fig. 12. Procedure to compute $lblocks_{s_D}(v, X)$.

Let Deg_v be the outdegree of v in $W_D(t_D)$, $|nodes(D)|$ be the number of components and interactions in D, i.e., the number of nodes in $W_D(s_D)$ and $|D|$ be the size of the syntactic description of D. We assume that membership in X and Y can be determined in constant time, e.g., by using Boolean arrays, and that evaluation of transition guards in components takes time proportional to the length of the guards. Then, the time complexity of COMPUTE-LBLOCKS(v, D, s_D, X) is $O(Deg_v)$. Hence the time complexity of COMPUTE-VL(D, s_D, X) is $O(|D| + |W_D(t_D)|)$, since each edge in $W_D(t_D)$ is examined at most once, over all calls of COMPUTE-LBLOCKS(v, D, s_D, X), and computing $W_D(t_D)$ can be done in time $O(|D|)$. Since $|W_D(t_D)|$ is $O(|D|)$, this is just $O(|D|)$. The time complexity of COMPUTE-LFP(D, s_D) is then $O(Fix \cdot |D|)$ where Fix is the number of iterations

```
checkInitSupercycleFree($Q_0$)
▷ returns true iff all initial states are supercycle-free
1. forall $s_0 \in Q_0$
2.    compute $W_B(s_0)$
3.    $U \leftarrow$ Compute-LFP(B, $s_0$)
4.    if ($U \neq W_B(s_0)$) then return(false)          ▷ $s_0$ not supercycle-free, so return false
5. endfor;
6. return(true)                                          ▷all initial states are supercycle-free
```

Fig. 13. Procedure to check that all initial states are supercycle-free.

that Compute-LFP(D, s_D) takes to reach a fixpoint. *Fix* is $O(|nodes(D)|)$, since each iteration either adds a node or reaches the least fixpoint. Thus the time complexity of Compute-LFP(D, s_D) is $O(|nodes(D)| \cdot |D|)$.

Theorem 8.1 requires that all initial states be supercycle-free. We assume that the number of initial states is small, so that we can check each explicitly. Figure 13 presents an algorithm which checks that all initial states are supercycle-free.

Proposition 9.1. checkInitSupercycleFree*(Q_0) returns true iff all initial states are supercycle-free.*

Proof. Consider the execution of checkInitSupercycleFree(Q_0) for an arbitrary $s_0 \in Q_0$.

By its construction, $U \sqsubseteq W_B(s_0)$. Suppose that $U \neq W_B(s_0)$. By Propositions 4.18 and 4.9, the nodes of $W_B(s_0)$ that are not in U constitute a supercycle. Hence s_0 not supercycle-free, and so false is the correct result in this case.

Now suppose that $U = W_B(s_0)$. Hence every node in $W_B(s_0)$ has a supercycle violation, and so by Proposition 4.18, no node of $W_B(s_0)$ is in a supercycle. Hence $W_B(s_0)$ does not contain a supercycle, and so s_0 is supercycle-free. Hence the for loop continues on to the next initial state. If all initial states are supercycle-free, the for loop terminates, and checkInitSupercycleFree(Q_0) returns true, as required. □

Let |B| be the size of the syntactic description of B, $|nodes(B)|$ be the number of components and interactions in B, which is also the number of nodes in $W_B(s_0)$, and $|Q_0|$ be the number of states in Q_0, i.e., the number of initial states, Then time complexity of checkInitSupercycleFree(Q_0) is $O(|Q_0| \cdot |nodes(B)| \cdot |B|)$.

9.1 Implementation of the Linear Condition $\mathcal{LLIN}$

Figure 14 presents the pseudocode for our algorithm LLin(B, Q_0) to evaluate $\mathcal{LLIN}$. LLin(B, Q_0) iterates over each interaction a of (B, Q_0), and invokes LLinInt(B, Q_0, a) to evaluate $(\exists \ell \geq 1 : \mathcal{LLIN}(B, Q_0, a, \ell))$. LLinInt(B, Q_0, a) starts with $\ell = 1$ and increments ℓ until either $\mathcal{LLIN}(B, Q_0, a, \ell)$ is found to hold, or D has become the entire system and $\mathcal{LLIN}(B, Q_0, a, \ell)$ does not hold. In the latter case, $\mathcal{LLIN}(B, Q_0, a, \ell)$ does not hold for any finite ℓ, and, in practice, computation would halt before D had become the entire system, due to exhaustion of resources. Evaluation of $\mathcal{LLIN}(B, Q_0, a, \ell)$ is done by LLinIntDist(B, Q_0, a, ℓ), which examines every reachable transition that executes a, and checks that the final state satisfies Definition 7.8.

Time complexity. Let ℓ_a be the smallest value of ℓ for which $\mathcal{LLIN}(B, Q_0, a, \ell)$ holds, M_a^ℓ be the transition system of D_a^ℓ, $|M_a^\ell|$ be the size (number of nodes plus number of edges) of M_a^ℓ, and $|D_a^\ell|$ be the size of the syntactic description of D_a^ℓ. Then the running time of LLinIntDist(B, Q_0, a, ℓ) is $O(|M_a^\ell| \cdot |D_a^\ell|)$, since computing $W_{D_a^\ell}(s_D)$ can be done in time $O(|D_a^\ell|)$, and $W_{D_a^\ell}(s_D)$ has size in

```
LLIN$(B, Q_0)$, where $B \triangleq \gamma(B_1, \ldots, B_n)$
$\triangleright$ returns true iff $(\forall a \in \gamma, \exists \ell \geq 1 : \mathcal{LLIN}(B, Q_0, a, \ell))$
1.  forall interactions $a \in \gamma$
2.    if (LLININT$(B, Q_0, a)$ = false) return(false) fi
3.  endfor;
4.  return(true)                                   $\triangleright$ return true if check succeeds for all $a \in \gamma$

LLININT$(B, Q_0, a)$, where $B \triangleq \gamma(B_1, \ldots, B_n), a \in \gamma$
$\triangleright$ returns true iff $(\exists \ell \geq 1 : \mathcal{LLIN}(B, Q_0, a, \ell))$
1.   $\ell \leftarrow 1$;                                                    $\triangleright$ start with $\ell = 1$
2.   while (true)
3.     if (LLININTDIST$(B, Q_0, a, \ell)$ = true) return(true) fi;          $\triangleright$ success, so return true
4.     if $(D_a^\ell = \gamma(B_1, \ldots, B_n))$ return(false) fi;     $\triangleright$ exhausted all subsystems, return false
5.     $\ell \leftarrow \ell + 1$           $\triangleright$ increment $\ell$ until success (true) or intractable or failure (false)
6.   endwhile

LLININTDIST$(B, Q_0, a, \ell)$
$\triangleright$ returns true iff $\mathcal{LLIN}(B, Q_0, a, \ell))$
1.  let $D = D_a^\ell$
2.  forall reachable transitions $t_D \xrightarrow{a} s_D$ of D
3.    compute $W_D(s_D)$;
4.    if $(\neg(\forall B_i \in components(a) : in\_depth_D(B_i, s_D) < 2\ell - 1 \vee out\_depth_D(B_i, s_D) < 2\ell - 1))$
5.      return(false)                                              $\triangleright$ check Definition 7.8
6.    fi
7.  endfor;
8.  return(true)                              $\triangleright$ return true if check succeeds for all transitions
```

Fig. 14. Pseudocode for the implementation of $\mathcal{LLIN}$.

$O(|D_a^\ell|)$, and computing in-depth and out-depth in $W_{D_a^\ell}(s_D)$ can be done in linear time using depth-first graph search. The running time of LLININT(B, Q_0, a), is $O(\Sigma_{1 \leq \ell \leq \ell_a} |M_a^\ell| \cdot |D_a^\ell|)$. The running time of LLIN(B, Q_0) is $O(\Sigma_{a \in \gamma} \Sigma_{1 \leq \ell \leq \ell_a} |M_a^\ell| \cdot |D_a^\ell|)$.

9.2 Implementation of the AND-OR Condition $\mathcal{LALT}$

Figure 15 presents the pseudocode for our algorithm LALT(B, Q_0) to evaluate $\mathcal{LALT}$. This uses the COMPUTE-LFP(D, s_D) algorithm for computing local supercycle violations in D, given in Figure 10.

LALT(B, Q_0) iterates over each interaction a of (B, Q_0), and invokes LALTINT(B, Q_0, a) to evaluate $(\exists \ell \geq 1 : \mathcal{LALT}(B, Q_0, a, \ell))$. LALTINT$(B, Q_0, a)$ starts with $\ell = 1$ and increments ℓ until either $\mathcal{LALT}(B, Q_0, a, \ell)$ is found to hold, or D has become the entire system and $\mathcal{LALT}(B, Q_0, a, \ell)$ does not hold. In the latter case, $\mathcal{LALT}(B, Q_0, a, \ell)$ does not hold for any finite ℓ, and, in practice, computation would halt before D had become the entire system, due to exhaustion of resources. Note that D is the smallest system in which a supercycle-violation can be confirmed.

Evaluation of $\mathcal{LALT}(B, Q_0, a, \ell)$ is done by LALTINTDIST(B, Q_0, a, ℓ), which invokes COMPUTE-LFP(D, t_D) to compute the local supercycle violations and GENLOCVIOL(B_i, V, D, s_D) to compute the general local supercycle violations. GENLOCVIOL(B_i, V, D, s_D) invokes LOCSCONNSCVIOL(B_i, V, D, s_D) to compute the local strong connectedness violation. The pseudocode is a straightforward translation of Definitions 7.1 and 7.3. Table 2 shows a summary of the procedures.

```
LALT(B, $Q_0$), where B $\triangleq \gamma(B_1, \ldots, B_n)$
$\triangleright$ returns true iff ($\forall$ a $\in \gamma, \exists \ell \geq 1 : \mathcal{LALT}(B, Q_0, a, \ell)$)
1.  forall interactions a $\in \gamma$
2.     if (LALTINT(B, $Q_0$, a) = false) return(false) fi
3.  endfor;
4.  return(true)                                   $\triangleright$ return true if check succeeds for all $a \in \gamma$

LALTINT(B, $Q_0$, a), where B $\triangleq \gamma(B_1, \ldots, B_n)$, a $\in \gamma$
$\triangleright$ returns true iff ($\exists \ell \geq 1 : \mathcal{LALT}(B, Q_0, a, \ell)$)
1.  $\ell \leftarrow 1$;                                                   $\triangleright$ start with $\ell = 1$
2.  while (true)
3.     if (LALTINTDIST(B, $Q_0$, a, $\ell$) = true) return(true) fi;       $\triangleright$ success, so return true
4.     if ($D^\ell_a = \gamma(B_1, \ldots, B_n)$) return(false) fi;     $\triangleright$ exhausted all subsystems, return false
5.     $\ell \leftarrow \ell + 1$                          $\triangleright$ increment $\ell$ until success or intractable or failure
6.  endwhile

LALTINTDIST(B, $Q_0$, a, $\ell$)
$\triangleright$ returns true iff $\mathcal{LALT}(B, Q_0, a, \ell)$
1.  let D = $D^\ell_a$
2.  forall reachable transitions $t_D \xrightarrow{a} s_D$ of D
3.     compute $W_D(s_D)$;
4.     $V \leftarrow$ COMPUTE-LFP(D, $s_D$);                                    $\triangleright$see Figure 10
5.     forall $B_i \in$ components(a)
6.        if ($\neg$GENLOCVIOL($B_i$, $V$, D, $s_D$)) return(false) fi             $\triangleright$no violation for $B_i$
7.     endfor
8.  endfor;
9.  return(true)              $\triangleright$all $B_i \in$ components(a) have a general local supercycle violation

GENLOCVIOL($B_i$, $V$, D, $s_D$)
$\triangleright$ returns true iff genViolLoc$_D$($B_i$, $s_D$) holds (Definition 7.3)
$\triangleright$ i.e., $B_i$ has a general local supercycle violation in state $s_D$ of subsystem D
1.  return($B_i \in V \vee$ LOCSCONNSCVIOL($B_i$, $V$, D, $s_D$))

LOCSCONNSCVIOL($B_i$, $V$, D, $s_D$)
$\triangleright$ returns true iff sConnViolLoc$_D$($B_i$, $s_D$) holds (Definition 7.1)
$\triangleright$ i.e., $B_i$ has a local strong connectedness violation in state $s_D$ of subsystem D
1.  $WL \leftarrow W_D(s_D) - V$, i.e., remove from $W_D(s_D)$ all nodes with a local supercycle violation;
2.  compute maximal strongly connected components of $WL$;
3.  forall maximal strongly connected components $C$ of $WL$
4.     if ($C$ contains a non-trivial strongly connected supercycle which contains $B_i$ as a node)
5.        return(false) fi;                             $\triangleright$Definition 7.1, Clause 1 violated
6.  if (there is no path in $WL$ from $B_i$ to a border node of D)
7.     return(true) fi;                                   $\triangleright$Definition 7.1, Clause 2a holds
8.  if (there is no path in $WL$ from some border node of D to $B_i$)
9.     return(true) fi;                                   $\triangleright$Definition 7.1, Clause 2b holds
10. return(false)                         $\triangleright$Definition 7.1, Clauses 1 and 2 both violated
```

Fig. 15. Pseudocode for the implementation of $\mathcal{LALT}$.

Table 2. Summary of Procedures

$\textsc{Lalt}(\mathrm{B}, Q_0)$	true iff $(\forall\, \mathrm{a} \in \gamma, \exists\, \ell \geq 1 : \mathcal{LALT}(B, Q_0, \mathrm{a}, \ell))$
$\textsc{LaltInt}(\mathrm{B}, Q_0, \mathrm{a})$	true iff $(\exists\, \ell \geq 1 : \mathcal{LALT}(B, Q_0, \mathrm{a}, \ell))$
$\textsc{LaltIntDist}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$	true iff $\mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$
$\textsc{GenLocViol}(\mathrm{B}_i, V, \mathrm{D}, s_\mathrm{D})$	true iff B_i has local sc-formation violation in state s_D of D, i.e., $\mathrm{genViolLoc}_\mathrm{D}(B_i, s_\mathrm{D})$ holds
$\textsc{locSconnScViol}(\mathrm{B}_i, V, \mathrm{D}, s_\mathrm{D})$	true iff B_i has local strong connectedness violation in t_a, i.e., $\mathrm{sConnViolLoc}_\mathrm{D}(B_i, s_\mathrm{D})$ holds
$\textsc{Compute}\text{-LFP}(\mathrm{D}, s_\mathrm{D})$	compute local supercycle violations in state s_D of D, i.e., $\mathrm{violLoc}_\mathrm{D}(v, s_\mathrm{D}, d)$ for all v, d

```
> java -jar ldfc.jar [options] input.bip
and options are:
-condition <s> LLIN (local linear check) or LALT (local and/or check - default)
               (optional)
-debug         Prints useful information at each iteration of checking.
               Example: selected interaction, depth length, etc.
               This information could be useful in case when the condition fails.

Examples:
  java -jar ldfc.jar -debug input.bip # deadlock freedom using default LALT
  java -jar ldfc.jar -condition=LLIN -debug input.bip # deadlock freedom using LLIN
```

Fig. 16. LALT-BIP Command Line Interface.

Time complexity. Let ℓ_a be the smallest value of ℓ for which $\mathcal{LALT}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ holds, M_a^ℓ be the transition system of $\mathrm{D}_\mathrm{a}^\ell$, $|M_\mathrm{a}^\ell|$ be the size (number of nodes plus number of edges) of M_a^ℓ, $|\mathrm{D}_\mathrm{a}^\ell|$ be the size of the syntactic description of $\mathrm{D}_\mathrm{a}^\ell$, and $|nodes(\mathrm{D}_\mathrm{a}^\ell)|$ be the number of components and interactions in $\mathrm{D}_\mathrm{a}^\ell$. The time complexity of $\textsc{locSconnScViol}(\mathrm{B}_i, V, \mathrm{D}_\mathrm{a}^\ell, s_\mathrm{D})$ is $O(|nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$, since maximal strongly connected components are computable in linear time using Tarjan's algorithm [30], and the existence of paths in WL from B_i to the border of $\mathrm{D}_\mathrm{a}^\ell$ can be checked in linear time by using depth-first graph search. Also, checking for supercycles (via least-fixpoint computation) within the strongly connected component C can be done in time $O(|nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$, amortized over all such C. The time complexity of $\textsc{GenLocViol}(\mathrm{B}_i, V, \mathrm{D}_\mathrm{a}^\ell, s_\mathrm{D})$ is also $O(|nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$. The running time of $\textsc{LaltIntDist}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ is $O(|M_\mathrm{a}^\ell| \cdot |nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$, since $\textsc{Compute}\text{-LFP}(\mathrm{D}_\mathrm{a}^\ell, s_\mathrm{D})$ has time complexity in $O(|nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$, and computing $W_{\mathrm{D}_\mathrm{a}^\ell}(s_\mathrm{D})$ can be done in time $O(|\mathrm{D}_\mathrm{a}^\ell|)$. The running time of $\textsc{LaltInt}(\mathrm{B}, Q_0, \mathrm{a})$ is $O(\Sigma_{1 \leq \ell \leq \ell_\mathrm{a}} |M_\mathrm{a}^\ell| \cdot |nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$ since $\textsc{LaltInt}(\mathrm{B}, Q_0, \mathrm{a})$ iterates $\textsc{LaltIntDist}(\mathrm{B}, Q_0, \mathrm{a}, \ell)$ until $\ell = \ell_\mathrm{a}$. The running time of $\textsc{Lalt}(\mathrm{B}, Q_0)$ is $O(\Sigma_{\mathrm{a} \in \gamma} \Sigma_{1 \leq \ell \leq \ell_\mathrm{a}} |M_\mathrm{a}^\ell| \cdot |nodes(\mathrm{D}_\mathrm{a}^\ell)| \cdot |\mathrm{D}_\mathrm{a}^\ell|)$ since $\textsc{Lalt}(\mathrm{B}, Q_0)$ calls $\textsc{LaltInt}(\mathrm{B}, Q_0, \mathrm{a})$ for every $\mathrm{a} \in \gamma$.

9.3 Toolset

We provide LALT-BIP , a suite of supporting tools that implement our method. LALT-BIP consists of about 2500 Java lines of code. LALT-BIP is equipped with a command line interface (see Figure 16) that accepts a set of configuration options. It takes the name of the input BIP file and other optional flags.

9.4 Experimentation

We evaluated LALT-BIP using several case studies including the dining philosopher example and multiple instances of a configurable generalized *Resource Allocation System* that comprises a configurable multi token-based scheduler. The different configurations of our resource allocation

Table 3. Benchmarks: Dining Philosopher (Seconds or Minutes : Seconds)

Size	$\mathcal{LALT}$	$\mathcal{LLIN}$	D-Finder
1,000	0.46	0.7	15
2,000	1.4	1.9	60
3,000	2.9	4	2 : 41
4,000	4.8	7	5 : 37
5,000	8.3	12	12 : 38
6,000	13.0	17	17 : 48
7,000	17.2	25	30 : 18
8,000	25.6	34	–
9,000	34.1	55	–
10,000	47	62	–

system subsume problems like the Milner's scheduler, data arbiters, and the dining philosopher with a butler problem. We benchmarked the performance of LALT-BIP against DFinder [13] on two benchmarks: *Dining Philosopher* with an increasing number of philosophers and a deadlock free resource allocation system with an increasing number of clients and resources.

All experiments were conducted on a machine with Intel (R) 8-Cores (TM) $i7$-6700, CPU @ 3.40GHZ, 32GB RAM, running a CentOS Linux distribution.

9.4.1 Dining Philosophers Case Study. We consider the traditional dining philosopher problem as depicted in Figure 1, which shows four philosophers and four forks modeled in BIP.

Each philosopher component has two states, and each fork component has three states. Thus, the total number of states is $2^n \times 3^n$. We evaluated LALT-BIP by increasing n and applying both $\mathcal{LALT}$ and $\mathcal{LLIN}$ methods and compared against the best configuration we could compute with DFinder2. DFinder2 allows for several techniques to be applied. The most efficient one is the Incremental Positive Mapping (IPM) technique [13]. IPM requires a manual partitioning of the system to exploit its efficiency. We applied IPM on all structural partitions and we report on the best result which is consistent with the results reported in Reference Bensalem et al. [13].

Table 3 shows the results. Both $\mathcal{LALT}$ and $\mathcal{LLIN}$ outperform the best performance of DFinder2 by several orders of magnitude for $n \leq 3{,}000$. Both $\mathcal{LALT}$ and $\mathcal{LLIN}$ successfully completed the deadlock freedom check for $3{,}000 \leq n \leq 10{,}000$ in less than 1m, where DFinder2 timed out (1h). The sole exception being that $\mathcal{LLIN}$ required 62s for $n = 10{,}000$.

Even though $\mathcal{LLIN}$ is asymptotically more efficient than $\mathcal{LALT}$, $\mathcal{LALT}$ outperforms $\mathcal{LLIN}$ in all cases. This is due to the following:

— The largest subsystem that $\mathcal{LALT}$ had to consider was with depth $\ell = 1$. This corresponds to $18 = 2^1 \times 3^2$ states regardless of n, the number of philosophers.
— The largest subsystem that $\mathcal{LLIN}$ had to consider was with depth $\ell = 2$. This corresponds to $648 = 2^3 \times 3^4$ states regardless of n.
— For a given depth ℓ, $\mathcal{LLIN}$ is more efficient to compute than $\mathcal{LALT}$. Since $\mathcal{LALT}$ performs a stronger check, it often terminates for smaller depths, which makes it more efficient than $\mathcal{LLIN}$ in many cases.

9.4.2 Resource Allocation System Case Studies. We evaluated LALT-BIP with a multi token-based resource allocation system. The system consists of n clients, m resources, k tokens. The number of tokens specifies the maximum number of resources that can be in use at a given time.

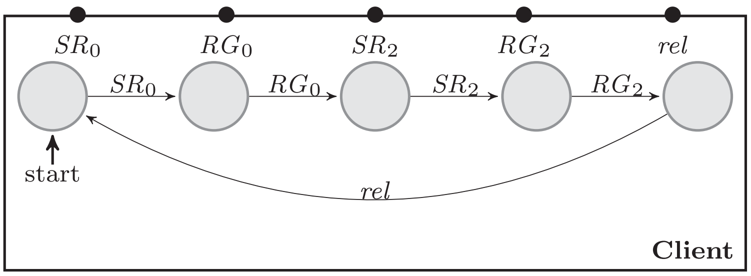


Fig. 17. Client.

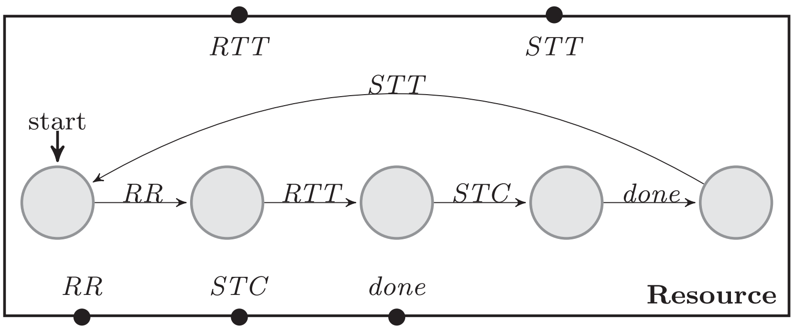


Fig. 18. Resource.

The system allows us to specify conflicting resources. Only one resource out of a set of conflicting resources can be in use at a given time. For each set of conflicting resources, we create a resource manager. Resource managers are connected in a ring where they pass tokens to neighboring resource managers or to resources.

Given a configuration specifying n, m, k, a map of requests between clients and resources, and a set of sets of conflicting resources, we automatically generate a corresponding BIP model. Figures 17, 18, and 19 show BIP atomic components for client, resource, and manager components. The client in Figure 17 requests resources R_0 and R_2 in sequence. It has five ports. Ports SR_0 and SR_2 send requests for resources R_0 and R_2, respectively. Ports RG_0 and RG_2 receive grants for resources R_0 and R_2, respectively. Port *rel* releases all resources. The behavior of the client depends on its request sequence.

Figure 18 shows a resource component. A resource component waits for a request from a connected client on port *RR*. Once a request is received, the resource component transitions to a state where it is ready to receive a token from the corresponding resource manager using port *RTT*. The resource transitions to a state where it grants the client request using port *STC* and waits until it is released on port *done*. There, it returns the token back to the resource manager and transitions to the start state.

Figure 19 shows a resource manager. A resource manager M has four states.

— State T denotes that M has a token. M may send the token to either (1) a resource on port *STR* and transition to state *TwR* (token with resource) or (2) the next resource manager on port *STT* and transition to state N (no token).
— State N denotes that N has no token. It may receive a token from a neighboring resource manager in the ring on port *RTT* and transition to state T.

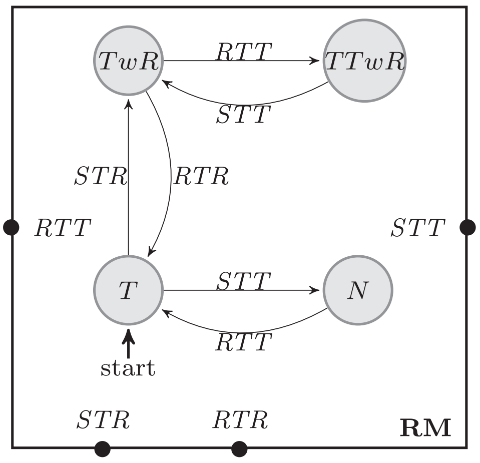


Fig. 19. Token Resource Manager.

— State *TwR* denotes that *M* has already passed a token to one of its resources. *M* may either receive (1) the assigned token back from the resource using port *RTR* and transition to state *T* or (2) another token from a neighboring manager using port *RTT* and transition to state *TTwR* (token and token with resource).
— State *TTwR* denotes that *M* has a token and has already passed a token to one of its resources. In this state, *M* cannot send the token it has to a resource it manages to respect the conflict constraint. *M* may send the token to the next manager on port *STT* and transition back to state *TwR*.

The connections between a resource manager *M* and its resources on ports *STR* and *RTR* specify that the resources are conflicting. A system should have at least x resource managers where x is the maximum between the number of sets of conflicting resources and k. Note that k resource managers start at state *T* to denote the k tokens; the rest start at state *N*.

Figure 20 shows a configuration system with five clients and five resources where

— Client C_0 requires resource R_0 then R_2,
— Client C_1 requires resource R_2 then R_0,
— Client C_2 requires resource R_1,
— Client C_3 requires resource R_3, and
— Client C_4 requires resource R_4.

The system has three resource managers to specify the conflicting resources. RM_{01} manages conflicting resources $\{R_0, R_1\}$. RM_{23} manages conflicting resources $\{R_2, R_3\}$. RM_4 manages resource R_4.

We evaluated LALT-BIP with various configurations. We highlight several lessons learned for specific systems as follows.

Lesson 1. $\mathcal{LALT}$ verifies freedom from global and local deadlock where DFinder2 can only verify freedom from global deadlock. Consider a system with five clients, three tokens, and five resources. Clients request resources $\langle 0, 2\rangle, \langle 2, 0\rangle, \langle 1\rangle, \langle 3\rangle$, and $\langle 4\rangle$, respectively. Resource sets $\{0, 1\}, \{2, 3\}$ are conflicting. This system is clearly global deadlock-free. It has a local deadlock where client C_0 has resource 0 and client C_1 has resource 2. DFinder qualitatively can not detect such a local deadlock while $\mathcal{LALT}$ successfully does.

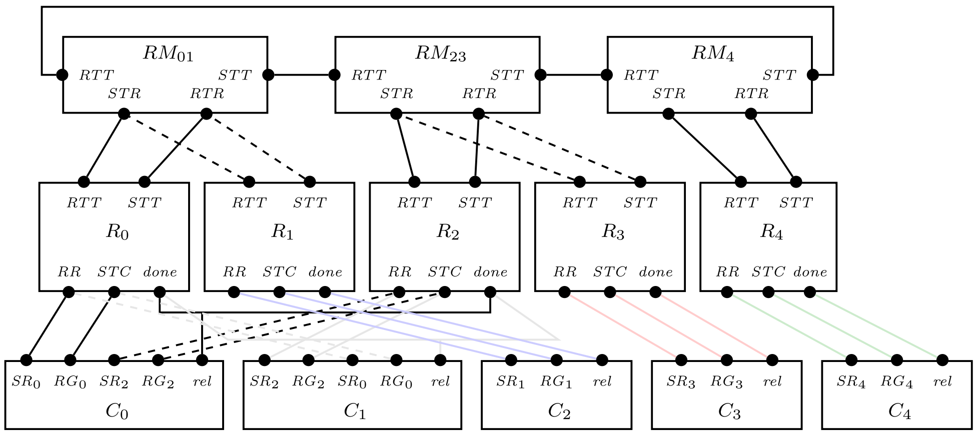


Fig. 20. Conflict-Resource Allocation System.

Table 4. Benchmarks: Time Required for $\mathcal{LALT}$ on the Resource Allocation System

Size	10	12	14	16	18	20	22	24	26	28	30
Time (sec)	148	169	189	230	254	277	298	318	351	374	430

Lesson 2. $\mathcal{LALT}$ is more complete than both $\mathcal{LLIN}$ and DFinder2. For example, it can verify global and local deadlock freedom in cases where $\mathcal{LLIN}$ fails. Consider a system with five clients, two tokens, and five resources. Clients request resources $\langle 0,2\rangle, \langle 0,2\rangle, \langle 1\rangle, \langle 3\rangle$, and $\langle 4\rangle$, respectively. Resource sets $\{0,1\}, \{2,3,4\}$ are conflicting. This system is global and local deadlock-free. Both DFinder2 and $\mathcal{LLIN}$ report that the system might contain a deadlock. $\mathcal{LALT}$ successfully reports that the system is both global and local deadlock-free.

Benchmarking: We evaluated the performance of $\mathcal{LALT}$ on a deadlock-free system with the following configuration.

— n clients each with three states, n resources each with five states, and n tokens,
— Client $C_i, 0 \le i < n$ requests resource i, and
— No resources are in conflict, hence we have n resource managers each with four states.

The system has a total of $4^n \times 3^n \times 5^n$ states. DFinder2 timed out within 7h for $n = 10$. $\mathcal{LLIN}$ had to increase the subsystem up to the whole system and also timed out within 7h for $n = 10$. $\mathcal{LALT}$ was able to verify deadlock freedom. It has to check subsystems with 12 components out of $3 \times n$ components regardless of n. This resulted from inspecting subsystems corresponding to a depth $\ell = 2$ with $\le 23,040,000 = 4^6 \times 3^2 \times 5^4$ states regardless of n. The numbers in Table 4 show a *linear increase in time* required to check deadlock freedom using $\mathcal{LALT}$ with respect to n. This indicates that the number of subsystems to check is proportional to n.

Our resource allocation system subsumes the token based Milner scheduler [25], which is essentially a token ring with precisely one token present [3].

10 RELATED WORK, DISCUSSION, AND FURTHER WORK

The notions of wait-for-graph and supercycle [7, 8] were initially defined for a shared memory program $P = P_1 \| \cdots \| P_K$ in *pairwise normal form* [4, 5]: a binary symmetric relation I specifies the directly interacting pairs ("neighbors") $\{P_i, P_j\}$ If P_i has neighbors P_j and P_k, then the code in

P_i that interacts with P_j is expressed separately from the code in P_i that interacts with P_k. These synchronization codes are executed synchronously and atomically, so the grain of atomicity is proportional to the degree of I. Attie and Chockler [7] give two polynomial time methods for (local and global) deadlock freedom. The first checks subsystems consisting of three processes. The second computes the wait-for-graphs of all pair subsystems $P_i \parallel P_j$, and takes their union, for all pairs and all reachable states of each pair. The first method considers only wait-for-paths of length ≤ 2. The second method is prone to false negatives, because wait-for edges generated by different states are all merged together, which can result in spurious supercycles.

Gössler and Sifakis [21] use a BIP-like formalism, Interaction Models. They present a criterion for global deadlock freedom, based on an and-or graph with components and constraints as the two sets of nodes. A constraint gives the condition under which a component is blocked. Edges are labeled with conjuncts of the constraints. Deadlock freedom is checked by traversing every cycle, taking the conjunction of all the conditions labeling its edges, and verifying that this conjunction is always false, i.e., verifying the absence of cyclical blocking. No complexity bounds are given. Martens and Majster-Cederbaum [23] present a polynomial time checkable deadlock freedom condition based on structural restrictions: "the communication structure between the components is given by a tree." This restriction allows them to analyze only pair systems. Aldini and Bernardo [2] use a formalism based on process algebra. They check deadlock by analyzing cycles in the connections between software components, and claim scalability, but no complexity bounds are given.

Roscoe and Dathi [29] present several rules for freedom of global deadlock of "triple disjoint" (no action involves >2 processes) CSP concurrent programs. The basis for these rules is to first check that each individual process is deadlock free (i.e., the network is "busy"), and then to define a "variant function" that maps the state of each process to a partially ordered set. The first rule requires to establish that, if P_i waits for P_j, then the value of P_i's state is greater than the value of P_j's state. Since every process is blocked in a global deadlock, one can then construct an infinite sequence of processes with strictly decreasing values, which are therefore all distinct. This cannot happen in a finite network, and hence some process is not blocked. They treat several examples, including a self-timed systolic array (in two and three dimensions), dining philosophers, and a message-switching network. They generalize the first rule to exploit "disconnecting edges" (whose removal partitions the network into disconnected components) to decompose the proof of deadlock freedom into showing that each disconnected component is deadlock-free, and also to weaken the restriction on the variant function so that it only has to decrease for at least one edge on each wait-for cycle. Brookes and Roscoe [17] also provide criteria for deadlock freedom of triple-disjoint CSP programs, and use the same technical framework as Reference [29]. However, they do not use variant functions, but show that, in a busy network, a deadlock implies the existence of a wait-for cycle. They give many examples, and demonstrate the absence of wait-for cycles in each example, by ad hoc reasoning. Finally, they give a deadlock freedom rule that exploits disconnecting edges, similar to that of [29]. In both of these papers, the wait-for relations are defined by examining a pair of processes at a time: P_i waits for P_j iff P_i offers an action to P_j which P_j is not willing to participate in.

Martin [24] applies the results of References [29] and [17] to formulate deadlock-freedom design rules for several classes of CSP concurrent programs: cyclic processes, client-server protocols, and resource allocation protocols. He also introduces the notion of State Dependence Digraph (SDD), whose nodes are local states of individual processes, and whose edges are wait-for relations between processes in particular local states. An acyclic SDD implies deadlock freedom. A cyclic SDD does not imply deadlock, however, since the cycle may be "spurious"—the local states along the cycle may not be reachable at the same time, and so the cycle cannot give rise to an actual deadlock during execution. Hence the SDD approach cannot deal with "non-hereditary" deadlock freedom, i.e., a deadlock-free system that contains a deadlock-prone subsystem. Consider, e.g., the dining philosophers with a butler solution; removing the butler leaves a deadlock-prone subsystem. Antonino et al. [3] takes the SDD approach and improves its accuracy by checking for mutual reachability of pairs of local states, and also eliminating local states and pairs of local states, where action enablement can be verified locally. These checks are formulated as a Boolean formula which is then sent to a SAT solver. Their method is able to verify deadlock freedom of dining philosophers with a butler, whereas our method timed out, since the subsystems on which $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ is evaluated becomes the entire system. On the other hand, our approach succeeded in quickly verifying deadlock freedom of the resource allocation example, whereas the method of Reference [3] failed for Milner's token based scheduler, which is a special case of our resource allocation example. An intriguing topic for future work is to attempt to combine the two methods, to obtain the advantages of both.

We compared our implementation LALT-BIP to D-Finder 2 [13]. D-Finder 2 computes a finite-state abstraction for each component, which it uses to compute a global invariant I. It then checks if I implies deadlock freedom. Unlike LALT-BIP, D-Finder 2 handles infinite state systems. However, LALT-BIP had superior running time for dining philosophers and resource controller (both finite-state).

All the above methods (except Reference [7]) verify global (and not local) deadlock freedom. Our method verifies local deadlock freedom, which subsumes global deadlock freedom as a special case. Also, our approach makes no structural restriction at all on the system being checked for deadlock. Our method checks for the absence of supercycles, which are a sound and complete characterization of deadlock. Moreover, the $\mathcal{LALT}$ condition is complete w.r.t. the occurrence of a supercycle wholly within the subsystem being checked, and the $\mathcal{GALT}$ condition is complete w.r.t. freedom from local and global deadlock, as given by Theorem 8.2. None of the above papers give a completeness result similar to Theorem 8.2. Hence, the only source of incompleteness in our method is that of computational limitation: if the subsystem being checked becomes too large before the $\mathcal{LALT}$ condition is verified. If computational resources are not exhausted, then our method can keep checking until the subsystem being checked is the entire system, at which point $\mathcal{LALT}$ coincides with $\mathcal{GALT}$, which is sound and complete for local deadlock (Proposition 4.18, Definition 5.3, and Definition 5.4).

Related to methods that specifically check for deadlock freedom are methods that check for general safety properties, e.g., using abstraction and compositional reasoning. van Glabbeek et al. [31] extend CTL* with constructs to support expressing and distinguishing between deadlock, livelock, and successful termination. This is key since the standard semantics of CTL* requires that Kripke structures be total, i.e., every state has at least one outgoing transition, and so deadlock cannot be modeled. They provide a semantics for CTL* in which deadlock, livelock, and successful termination can all be distinguished.

Abstraction methods related to compositional reasoning [1, 19, 28] that target safety properties can be used to prove global deadlock freedom. The aforementioned papers target parameterized systems composed of N communicating processes $P_i, 1 \leq i \leq N$. They overapproximate the reachable state space of a system with N processes, using symbolic states from a system of size $K < N$. If the property holds for the system of size K, then it holds for any arbitrary N. Crucial to Reference [1] is a bound on the number of processes involved at each state, such that the post image (typically infinite) can be computed using successor operations of size $K + \ell$, where ℓ is a small constant. This limits the completeness of the technique to systems with specific array, ring, and treelike topologies. Cohen and Namjoshi [19] provide a global proof using several local proofs. It splits a target system invariant into local process invariants across local and shared variables and attempts to prove these invariants. The derived local invariants are symbolic overapproximations of the reachable state space of the system. The abstraction refinement step refines the invariants with predicates reasoning about additional local variables.

Pnueli et al. [28] target a specific type of bounded data parameterized systems with parameter N, where N is the number of processes, and where safety is expressed using a specific type of assertions called R-assertions. They show that, for a given system, there exists an N_0 such that an R−assertion ϕ is preserved by any step of the system for every $N > 1$ iff ϕ is preserved by any step of the system for every $N \leq N_0$. They show how to handle such systems with model checking and deductive reasoning techniques. The survey paper Reference [18] describes several abstraction techniques that use counterexamples to guide the refinement steps. It also describes a localization reduction technique [22]. The first abstraction is the property itself. If the model check fails, then an error "track" is produced, and either the track is feasible and the property fails, or the track is analyzed and linked to a group of blocking variables that could not be assigned to satisfy the track. The blocking variables lead, via dependency graph paths, to active variables that have full assignments in the error track. All these variables constitute the next refinement step, where the border variables are considered free and are called the "free fence.' Key to the efficacy of the technique is the choice of the blocking variables, so that they minimize the free fence at each abstraction step. Localization refinement is also used synergistically with input reparameterization, to attain maximal input reduction in sequential netlists, using min-cut analysis in a structural manner [12]. The reduced netlists are then subject to verification using several techniques, including decomposing the netlist into several sub-netlists, each with a bounded state transition diameter, and then applying bounded model checking [10, 11] to each sub-netlist. Our work differs from the above techniques in that (1) we do not limit our technique to parameterized systems, (2) we characterize deadlock freedom with a structural supercycle property that governs the wait-for-graph of the system interactions, (3) we compute our local subsystems based on interactions, (4) we establish deadlock freedom by performing the structural supercycle violation check for each interaction using its local subsystems, and (5) our technique is complete for BIP systems. In the future, we would like to explore whether the local supercycle violation check is enough to prove deadlock freedom of parameterized systems. We would also like to consider characterizing other interesting safety properties with similar structural checks in the context of BIP.

10.1 Discussion

Our approach has the following advantages:

— *Local and global deadlock*: our method shows that no subset of processes can be deadlocked, i.e., absence of both local and global deadlock.
— *Check works for realistic formalism*: by applying the approach to BIP, we provide an efficient deadlock-freedom check within a formalism from which efficient distributed implementations can be generated [15].
— *Locality*: if a component B_i is modified, or is added to an existing system, then $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ only has to be rechecked for B_i and components within distance ℓ of B_i. A condition whose evaluation considers the entire system at once, e.g., [2, 13, 21] would have to be rechecked for the entire system.
— *Easily parallelizable*: since the checking of each subsystem D is independent of the others, the checks can be carried out in parallel. Hence our method can be easily parallelized and distributed for speedup, if needed. Alternatively, performing the checks sequentially minimizes the amount of memory needed.

— *Framework aspect*: supercycles and in/out-depth provide a *framework* for deadlock freedom. Conditions more general and/or discriminating than the one presented here should be devisable in this framework. This is a topic for future work. In addition, our approach is applicable to any model of concurrency in which our notions of wait-for graph and supercycle can be defined. For example, Attie and Chockler [7] give two methods for verifying global and local deadlock freedom of shared-memory concurrent programs in pairwise normal form, as noted above. Hence, our methods are applicable to other formalisms such as CSP, CCS, I/O Automata, and so on.

10.2 Further Work

Our implementation uses explicit state enumeration. Using BDDs may improve the running time when $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ holds only for large ℓ. Another potential method for improving the running time is to use SAT solving, cf. [3]. An enabled port p enables all interactions containing p. Deadlock-freedom conditions based on ports could exploit this interdependence among interaction enablement. Our implementation should produce *counterexamples* when a system fails to satisfy $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. These can be used to manually modify the system to eliminate a possible deadlock. Also, when $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ fails to verify deadlock freedom, we increment ℓ, in effect extending the subsystem being checked "in all directions" away from a (in the structure graph). A counterexample may provide guidance to a more discriminating extension, when adding only a few components, so we now consider subsystems whose boundary has varying distance from a, in the structure graph. This has the benefit that we might verify deadlock freedom using a smaller subsystem than with our current approach. *Design rules* for ensuring $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ will help users to produce deadlock-free systems, and also to interpret counterexamples. A *fault* may create a deadlock, i.e., a supercycle, by creating wait-for-edges that would not normally arise. Tolerating a fault that creates up to f such spurious wait-for-edges requires that there do not arise during normal (fault-free) operation subgraphs of $W_{\mathsf{B}}(s)$ that can be made into a supercycle by adding f edges. We will investigate criteria for preventing formation of such subgraphs. Methods for evaluating $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ on *infinite state* systems will be devised, e.g., by extracting proof obligations and verifying using SMT solvers. We will extend our method to *Dynamic BIP* [16], where participants can add and remove interactions at runtime.

REFERENCES

[1] Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. 2013. All for the price of few. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'13).* Springer-Verlag, 476–495. DOI: http://dx.doi.org/10.1007/978-3-642-35873-9_28

[2] Alessandro Aldini and Marco Bernardo. 2003. A general approach to deadlock freedom verification for software architectures. *FME* 2805 (2003), 658–677.

[3] Pedro Antonino, Thomas Gibson-Robinson, and A. W. Roscoe. 2016. Efficient deadlock-freedom checking using local analysis and SAT solving. In *Proceedings of the 12th International Conference on Integrated Formal Methods (IFM'16).* Springer-Verlag, 345–360. DOI: http://dx.doi.org/10.1007/978-3-319-33693-0_22

[4] Paul C. Attie. 2016. Finite-state concurrent programs can be expressed in pairwise normal form. *Theor. Comp. Sci.* 619 (2016), 1–31. DOI: http://dx.doi.org/10.1016/j.tcs.2015.11.032

[5] Paul C. Attie. 2016. Synthesis of large dynamic concurrent programs from dynamic specifications. *Formal Methods in System Design* 48, 1–2 (2016), 1–54. DOI: http://dx.doi.org/10.1007/s10703-016-0252-9

[6] Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. 2013. An abstract framework for deadlock prevention in BIP. In *Proceedings of the Formal Techniques for Distributed Systems - Joint IFIP WG 6.1 International Conference, FMOODS/FORTE 2013, held as part of the 8th International Federated Conference on Distributed Computing Techniques (DisCoTec'13).* Springer-Verlag, 161–177. DOI: http://dx.doi.org/10.1007/978-3-642-38592-6_12

[7] Paul C. Attie and Hana Chockler. 2005. Efficiently verifiable conditions for deadlock freedom of large concurrent programs. In *VMCAI (Lecture Notes in Computer Science)*, Radhia Cousot (Ed.), Vol. 3385. Springer, 465–481.

[8] Paul C. Attie and E. Allen Emerson. 1998. Synthesis of concurrent systems with many similar processes. *TOPLAS* 20, 1 (Jan. 1998), 51–115.
[9] Paul C. Attie, Nissim Francez, and Orna Grumberg. 1993. Fairness and hyperfairness in multiparty interactions. *Distrib. Comput.* 6 (1993), 245–254.
[10] Jason Baumgartner and Andreas Kuehlmann. 2004. Enhanced diameter bounding via structural transformation. In *Design, Automation and Test in Europe Conference and Exposition (DATE'04)*. IEEE, 36–41. DOI: http://dx.doi.org/10.1109/DATE.2004.1268824
[11] Jason Baumgartner, Andreas Kuehlmann, and Jacob A. Abraham. 2002. Property checking via structural analysis. In *Computer Aided Verification (CAV'02)*. Springer-Verlag, 151–165. DOI: http://dx.doi.org/10.1007/3-540-45657-0_12
[12] Jason Baumgartner and Hari Mony. 2005. Maximal input reduction of sequential netlists via synergistic reparameterization and localization strategies. In *Correct Hardware Design and Verification Methods, CHARME*. Springer-Verlag, 222–237. DOI: http://dx.doi.org/10.1007/11560548_18
[13] Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. 2011. D-finder 2: Towards efficient correctness of incremental design. In *NASA Formal Methods*. Springer-Verlag, Pasadena, CA, 453–458.
[14] Simon Bliudze and Joseph Sifakis. 2008. The algebra of connectors—structuring interaction in BIP. *IEEE Trans. Comput.* 57, 10 (2008), 1315–1330. DOI: http://dx.doi.org/10.1109/TC.2008.26
[15] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. 2010. From high-level component-based models to distributed implementations. In *EMSOFT*. ACM, 209–218.
[16] Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. 2012. Modeling dynamic architectures using Dy-BIP. In *Software Composition*. Springer-Verlag, 1–16.
[17] Stephen Brookes and Andrew William Roscoe. 1991. Deadlock analysis in networks of communicating processes. *Distrib. Comput.* 4, 4 (1991), 209–230.
[18] Edmund M. Clarke, Robert P. Kurshan, and Helmut Veith. 2010. The localization reduction and counterexample-guided abstraction refinement. In *Time for Verification, Essays in Memory of Amir Pnueli*. Springer-Verlag, New York, NY, 61–71. DOI: http://dx.doi.org/10.1007/978-3-642-13754-9_4
[19] Ariel Cohen and Kedar S. Namjoshi. 2009. Local proofs for global safety properties. *Form. Methods Syst. Des.* 34, 2 (May 2009), 104–125. DOI: http://dx.doi.org/10.1007/s10703-008-0063-8
[20] Brian Davey and Hilary Priestly. 2002. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK.
[21] Gregor Gössler and Joseph Sifakis. 2003. Component-based construction of deadlock-free systems. In *FSTTCS*. Springer,420–433.
[22] Robert P. Kurshan. 1994. *Computer-Aided Verification of Coordinating Processes: The Automata-theoretic Approach*. Princeton University Press, Princeton, NJ.
[23] Moritz Martens and Mila Majster-Cederbaum. 2012. Deadlock-freedom in component systems with architectural constraints. *FMSD* 41, 2 (2012), 129–177. DOI: http://dx.doi.org/10.1007/s10703-012-0160-6
[24] Jeremy Malcolm Randolph Martin. 1996. *The Design and Construction of Deadlock-Free Concurrent Systems*. Ph.D. Dissertation. The University of Buckingham, Buckingham MK18 1EG, United Kingdom.
[25] Robin Milner. 1989. *Communication and Concurrency*. Prentice Hall, New Jersey.
[26] Christos H. Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley, Boston, MA.
[27] David Park. 1969. Fixpoint induction and proofs of program properties. *Mach. Intell.* 5 (1969), 59–78.
[28] Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. 2001. Automatic deductive verification with invisible invariants. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*. Springer-Verlag, Genova, Italy, 82–97. DOI: http://dx.doi.org/10.1007/3-540-45319-9_7
[29] Andrew William Roscoe and Naiem Dathi. 1987. The pursuit of deadlock freedom. *Inf. Comput.* 75, 3 (1987), 289–327. DOI: http://dx.doi.org/10.1016/0890-5401(87)90004-6
[30] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. DOI: http://dx.doi.org/10.1137/0201010
[31] Rob J. van Glabbeek, Bas Luttik, and Nikola Trcka. 2009. Computation tree logic with deadlock detection. *Log. Methods Comp. Sci.* 5, 4 (Oct. 2009), 1–24.

Received October 2016; revised October 2017; accepted October 2017