Global and Local Deadlock Freedom in BIP

Journal:	*Transactions on Software Engineering and Methodology*
Manuscript ID	Draft
Manuscript Type:	Journal-First Paper
Date Submitted by the Author:	n/a
Complete List of Authors:	Attie, Paul; American University of Beirut, Computer Science Bensalem, Saddek; VERIMAG Bozga, Marius; Verimag, DCS Jaber, Mohamad; American University of Beirut Sifakis, Joseph; Ecole Polytechnique Federale de Lausanne Zaraket, Fadi; American University of Beirut, ECE
Computing Classification Systems:	Theory of computation~Program verification, Software and its engineering~Deadlocks, Software and its engineering~Model checking, Software and its engineering~Formal software verification, Software and its engineering~State systems

October 6, 2016

Prof. David S. Rosenblum
Editor-in-Chief
ACM Transactions on Software Engineering and Methodology

Dear Professor Rosenblum,

I would like to submit our paper "Global and Local Deadlock Freedom in BIP", by Paul C Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A Zaraket, for consideration for publication in ACM TOSEM. The paper presents a sound and complete criterion for deadlock-freedom in concurrent programs with local non-determinism. Completeness is obtained by treating the wait-for graph as an AND-OR graph. In the positive case of deadlock freedom, the criterion can often be verified quickly, given a fast method for verifying deadlock-freedom.

I would also like to submit this paper as a "journal first" submission. A very preliminary version of this paper appeared in the 2013 IFIP Joint International Conference on Formal Techniques for Distributed Systems (33rd FORTE / 15th FMOODS), with the title "An Abstract Framework for Deadlock Prevention in BIP". The conference version gives a restricted "linear" version of the criterion, which is *not* complete, and which fails in cases where the complete AND-OR criterion succeeds. We also provide experiments that are new and different from the conference version, and which, among other results, give an example where the linear criterion fails while the AND-OR criterion succeeds. Hence the contributions of this paper are different, and extend the conference paper in new ways.

Sincerely,

Paul C Attie
Associate Professor
American University of Beirut
Department of Computer Science
Beirut 1107 2020, Lebanon

pa07@aub.edu.lb
http://www.cs.aub.edu.lb/pa07/

A

Global and Local Deadlock Freedom in BIP[1]

PAUL C ATTIE, Department of Computer Science, American University of Beirut, Beirut, Lebanon
SADDEK BENSALEM, UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France
MARIUS BOZGA, UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France
MOHAMAD JABER, Department of Computer Science, American University of Beirut, Beirut, Lebanon
JOSEPH SIFAKIS, Rigorous System Design Laboratory, EPFL, Lausanne, Switzerland
FADI A ZARAKET, Department of Electrical and Computer Engineering, American University of Beirut, 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 the construction of 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 can only fail to decide deadlock-freedom because of computational limitations: state-space explosion sets in when the subsystems being checked 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. 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: Nondeterminism, Completeness

ACM Reference Format:
Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, Fadi A Zaraket, 2016. Global and Local Deadlock Freedom in BIP. *ACM Trans. Softw. Eng. Methodol.* V, N, Article A (January YYYY), 43 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

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 cor-

Author's addresses: Paul C. Attie and Mohamad Jaber, Department of Computer Science, American University of Beirut, Beirut, Lebanon, Saddek Bensalem and Marius Bozga, VERIMAG, Grenoble, France, Joseph Sifakis, EPFL, Lausanne, Switzerland, Fadi A Zaraket, Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon.

DOI: http://dx.doi.org/10.1145/0000000.0000000

rectness 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 [Papadimitriou 1994, chapter 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, and 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 it's 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 Antonino et al. [2016], 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 un-granted requests.

In a model of concurrency which 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).

Our method, in contrast, characterizes deadlock by the occurrence of a *supercycle* [Attie and Emerson 1998; Attie and Chockler 2005], 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.

In addition 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 [Antonino et al. 2016; Martin 1996; Roscoe and Dathi 1987; Bensalem et al. 2011; Brookes and Roscoe 1991; Martens and Majster-Cederbaum 2012; Gössler and Sifakis 2003; Aldini and Bernardo 2003] guarantee only global deadlock freedom.

This paper significantly extends a preliminary conference version [Attie et al. 2013] 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 [Attie et al. 2013] 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 local and global 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 [Milner 1989]. These show that our method compares favorably with existing approaches.

Section 2 presents BIP. Section 3 characterizes local and global deadlocks as the occurrence of a pattern of wait-for edges called a *supercycle*, and presents some structural properties of supercycles. Section 4 considers how a supercyle can be formed, and analyzes the consequences of supercycle formation. 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 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 7 presents the main soundness and completeness results of the paper, and gives the implication relation among our various conditions for deadlock-freedom. Section 8 gives algorithms to evaluate the local conditions, and presents experimental evaluation. Section 9 discusses related work, further work, and concludes.

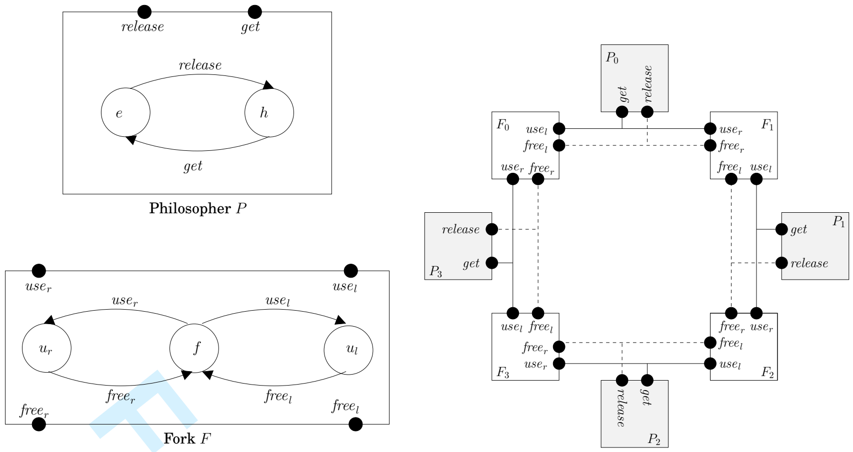
2. BIP — BEHAVIOR INTERACTION PRIORITY

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 paper. 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* possible transitions*, each labeled by some port.*

For states $s_i, t_i \in Q_i$ and port $p_i \in P_i$, write $s_i \xrightarrow{p_i}_i t_i$, iff $(s_i, p_i, t_i) \in \rightarrow_i$. When p_i is irrelevant, write $s_i \rightarrow_i t_i$. Similarly, $s_i \xrightarrow{p_i}_i$ means that there exists $t_i \in Q_i$ such that $s_i \xrightarrow{p_i}_i t_i$. In this case, p_i is *enabled* in state s_i. Ports are used for communication between different components, as discussed below.

In practice, we describe the transition system using some syntax, e.g., involving variables. We abstract away from issues of syntactic description since we are only



(a) Philosopher P and fork F atomic components.

(b) Dining philosophers composite component with four philosophers.

Fig. 1. Dining philosophers.

interested in enablement of ports and actions. We assume that enablement of a port depends only on the local state of a component. In particular, it cannot depend on the state of other components. This is a restriction on BIP, and we defer to subsequent work how to lift this restriction. So, we assume the existence of a predicate $enb^i_{p_i}$ that holds in state s_i of component B_i iff port p_i is enabled in s_i, i.e., $s_i(enb^i_{p_i}) = true$ iff $s_i \xrightarrow{p_i}_i$.

Figure 1(a) shows atomic components for a philosopher P and a fork F in dining philosophers. A philosopher P that is hungry (in state h) can eat by executing get and moving to state e (eating). From e, P releases its forks by executing $release$ and moving back to h. Adding the thinking state does not change the deadlock behaviour of the system, since the thinking to hungry transition is internal to P, and so we omit it. A fork F is taken by either: (1) the left philosopher (transition get_l) and so moves to state u_l (used by left philosopher), or (2) the right philosopher (transition get_r) and so moves to state u_r (used by right philosopher). From state u_r (resp. u_l), F is released by the right philosopher (resp. left philosopher) and so moves back to state f (free).

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 $i, j \in \{1..n\} \wedge i \neq j$, 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 $\mathsf{a} \subseteq P \wedge (\forall i \in \{1..n\} : |\mathsf{a} \cap P_i| \leq 1)$, 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 $I \subseteq \{1..n\}$.*

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

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*

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

This inference rule says that a composite component $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ can execute an interaction $\mathsf{a} \in \gamma$, iff for each port $p_i \in \mathsf{a}$, 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. Figure 1(b) shows a composite component consisting of four philosophers and the four forks between them. Each philosopher and its two neighboring forks share two interactions: $Get = \{get, use_l, use_r\}$ in which the philosopher obtains the forks, and $Rel = \{release, free_l, free_r\}$ in which the philosopher releases the forks.

Definition 2.4 (Interaction enablement) *An atomic component* $\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)$ *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* $\{p_i\} = P_i \cap \mathsf{a}$ *is the port of* B_i *involved in* a. *That is,* B_i *enables* a *in state* s_i *iff* B_i *enables port* $\mathsf{a} \cap P_i$ *in state* s_i.

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

Let $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *be a composite component, and let* $s = \langle s_1, \ldots, s_n \rangle$ *be a state of* B. *Then* B *enables* a *in* s *iff every* $\mathsf{B}_i \in components(\mathsf{a})$ *enables* a *in* s_i.

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 zero transitions, and every port labeling a transition is part of at least one interaction.

Definition 2.5 (Local Enablement Assumption) *For every component* $\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)$*, the following holds. In every* $s_i \in Q_i$*,* B_i *enables some interaction* a.

Definition 2.6 (BIP System) *Let* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *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 paper, an arbitrary BIP-system (B, Q_0), with $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and transition system $(Q, \gamma, \rightarrow)$.

Definition 2.7 (Execution) *Let* (B, Q_0) *be a BIP system with transition system* $(Q, \gamma, \rightarrow)$. *Let* $\rho = s_0 \mathsf{a}_1 s_1 \ldots s_{j-1} \mathsf{a}_j s_j \ldots$ *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)* $\forall j > 0 : s_{j-1} \xrightarrow{\mathsf{a}_j} s_j$.

Definition 2.8 (Reachable state, transition) *A state or transition that occurs in some execution is called* reachable.

Definition 2.9 (State Projection) *Let* (B, Q_0) *be a BIP system where* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *and let* $s = \langle s_1, \ldots, s_n \rangle$ *be a state of* (B, Q_0)*. Let* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\} \subseteq \{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$*. Then* $s \upharpoonright \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\} \triangleq \langle s_{i_1}, \ldots, s_{i_k} \rangle$*. For a single* B_i*, we write* $s \upharpoonright \mathsf{B}_i = s_i$*. We extend state projection to sets of states element-wise.*

Definition 2.10 (Subcomponent) *Let* $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *be a composite component, and let* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ *be a subset of* $\{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$*. Let* $P' = P_{i_1} \cup \cdots \cup P_{i_k}$*, i.e., the union of the ports of* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$*. Then the subcomponent* B' *of* B *based on* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ *is as follows:*

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

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

We write $s \upharpoonright \mathsf{B}'$ to indicate state projection onto B', and define $s \upharpoonright \mathsf{B}' \triangleq s \upharpoonright \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$, where $\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}$ are the atomic components in B'.

Definition 2.11 (Subsystem) *Let* (B, Q_0) *be a BIP system where* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$*, and let* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ *be a subset of* $\{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$*. Then the subsystem* (B', Q'_0) *of* (B, Q_0) *based on* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ *is as follows:*

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

Definition 2.12 (Execution Projection) *Let* (B, Q_0) *be a BIP system where* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$*, and let* (B', Q'_0)*, with* $\mathsf{B}' = \gamma'(\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k})$ *be the subsystem of* (B, Q_0) *based on* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$*. Let* $P' = P_{i_1} \cup \cdots \cup P_{i_k}$*, i.e.,* P' *is the set of ports of* (B', Q'_0)*. Let* $\rho = s_0 \mathsf{a}_1 s_1 \ldots s_{j-1} \mathsf{a}_j s_j \ldots$ *be an execution of* (B, Q_0)*. Then,* $\rho \upharpoonright (\mathsf{B}', Q'_0)$*, the projection of* ρ *onto* (B', Q'_0)*, is the sequence resulting from:*

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

Proposition 2.13 (Execution Projection) *Let* (B, Q_0) *be a BIP system where* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$*, and let* (B', Q'_0)*, with* $\mathsf{B}' = \gamma'(\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k})$ *be the subsystem of* (B, Q_0) *based on* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$*. Let* $P' = P_{i_1} \cup \cdots \cup P_{i_k}$*, i.e., the union of the ports of* $\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$*. Let* $\rho = s_0 \mathsf{a}_1 s_1 \ldots s_{j-1} \mathsf{a}_j s_j \ldots$ *be an execution of* (B, Q_0)*. Then,* $\rho \upharpoonright (\mathsf{B}', Q'_0)$ *is an execution of* (B', Q'_0)*.*

Proof. By Definitions 2.9, 2.11, and 2.12, we have $\rho \upharpoonright (\mathsf{B}', Q'_0) = s'_0 \mathsf{b}_1 s'_1 \mathsf{b}_2 s'_2 \ldots$ for some $s'_0, \mathsf{b}_1 s'_1 \mathsf{b}_2 s'_2 \ldots$, where $s'_j \in Q' = Q \upharpoonright \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ for $j \geq 0$. Also by Definitions 2.9, 2.11, and 2.12, we have $s'_0 \in Q'_0 = Q_0 \upharpoonright \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$, since $s'_0 = s_0 \upharpoonright \mathsf{B}'$, and $s_0 \in Q_0$, by Definition 2.7.

Consider an arbitrary step $(s'_{j-1}, \mathsf{b}_j, s'_j)$ of $\rho \upharpoonright (\mathsf{B}', Q'_0)$. Since $\mathsf{b}_j s'_j$ was not removed in Clause 2 of Definition 2.12, we have

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

From (3) we have $\forall\, m' : m+1 \leq m' < \ell : \mathsf{a}_{m'} \cap P' = \emptyset$. So by Definitions 2.3 and 2.12, we have $s_m \restriction \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\} = s_{\ell-1} \restriction \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$. From (3) we have $s'_{j-1} = s_m \restriction \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$. Hence $s'_{j-1} = s_{\ell-1} \restriction \{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$.

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

Since $(s'_{j-1}, \mathsf{b}_j, s'_j)$ was arbitrarily chosen, we conclude that every step of $\rho \restriction (\mathsf{B}', Q'_0)$ is a step of (B', Q'_0). This establishes Clause (2) of Definition 2.7. The first state of $\rho \restriction (\mathsf{B}', Q'_0)$ is s'_0, and $s'_0 \in Q'_0$ was shown above, so we establish Clause (1) of Definition 2.7.

Since both clauses of Definition 2.7 are satisfied, we conclude that $\rho \restriction (\mathsf{B}', Q'_0)$ is an execution of (B', Q'_0). $\square$

COROLLARY 2.14. *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* $s \restriction \mathsf{B}'$ *is a reachable state of* (B', Q'_0)*. Let* $s \xrightarrow{\mathsf{a}} t$ *be a reachable transition of* (B, Q_0)*, and let* a *be an interaction of* (B', Q'_0)*. Then* $s \restriction \mathsf{B}' \xrightarrow{\mathsf{a} \cap P'} t \restriction \mathsf{B}'$ *is a reachable transition of* (B', Q'_0).

Proof. Immediate corollary of Proposition 2.13. $\square$

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,* $\forall\, s \in \mathit{rstates}(\mathsf{B}, Q_0), \exists\, \mathsf{a} : s \xrightarrow{\mathsf{a}}_{\mathsf{B}}$.

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* $s \restriction \mathsf{B}'$*. Formally:*
for every subsystem (B', Q'_0) *of* (B, Q_0)*:*
$$\forall\, s \in \mathit{rstates}(\mathsf{B}, Q_0), \exists\, \mathsf{a} : s \restriction \mathsf{B}' \xrightarrow{\mathsf{a}}_{\mathsf{B}'}.$$

Proposition 3.7 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 3.8 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.

3.1. Wait-for graphs

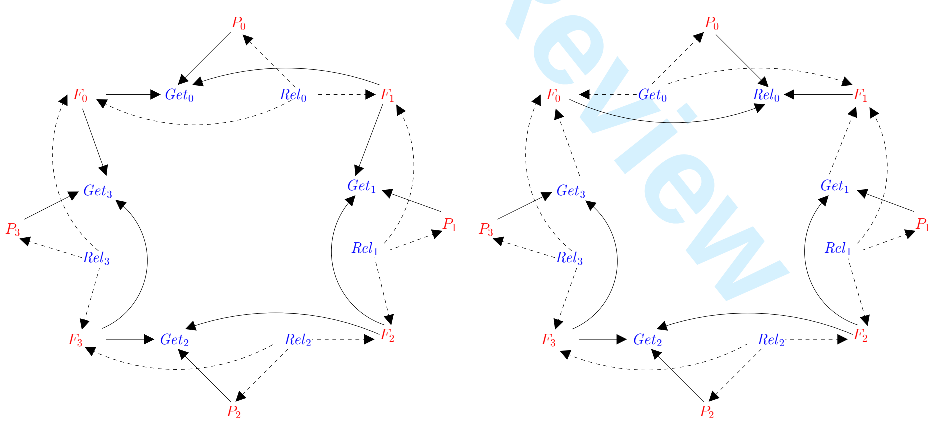
The wait-for-graph for a state s is a directed bipartite and-or graph which contains as nodes the atomic components $\mathsf{B}_1, \ldots, \mathsf{B}_n$, and all the interactions γ. Edges in the wait-for-graph are from a B_i to all the interactions that B_i enables (in s), and from an interaction a to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

Definition 3.3 (Wait-for-graph $W_{\mathsf{B}}(s)$) *Let $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ 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 $W_{\mathsf{B}}(s)$ *of s is a directed bipartite and-or graph, where*

1. *the nodes of $W_{\mathsf{B}}(s)$ are as follows:*
   (a) *the and-nodes are the atomic components B_i, $i \in \{1..n\}$,*
   (b) *the or-nodes are the interactions $\mathsf{a} \in \gamma$,*
2. *there is an edge in $W_{\mathsf{B}}(s)$ from B_i to every node a such that $\mathsf{B}_i \in components(\mathsf{a})$ and $s_i(enb^i_{\mathsf{a}}) = true$, i.e., from B_i to every interaction which B_i enables in s_i,*
3. *there is an edge in $W_{\mathsf{B}}(s)$ from a to every B_i such that $\mathsf{B}_i \in components(\mathsf{a})$ and $s_i(enb^i_{\mathsf{a}}) = false$, 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). Write $\mathsf{a} \to \mathsf{B}_i$ ($\mathsf{B}_i \to \mathsf{a}$ respectively) for a wait-for-edge from a to B_i (B_i to a respectively). We abuse notation by writing $e \in W_{\mathsf{B}}(s)$ to indicate that e (either $\mathsf{a} \to \mathsf{B}_i$ or $\mathsf{B}_i \to \mathsf{a}$) is an edge in $W_{\mathsf{B}}(s)$. Also $\mathsf{B}_i \to \mathsf{a} \to \mathsf{B}'_i \in W_{\mathsf{B}}(s)$ for $B_i \to \mathsf{a} \in W_{\mathsf{B}}(s) \wedge \mathsf{a} \to B'_i \in W_{\mathsf{B}}(s)$, i.e., for a wait-for-path of length 2, and similarly for longer wait-for-paths.

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 get_0. Edges from components to interactions are shown solid, and edges from interactions to components are shown dashed.



(a) Wait-for-graph in initial state. (b) Wait-for-graph after execution of get_0.

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

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 $\mathsf{B}_i \in components(\mathsf{a})$ 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 3.4 (Wait-for edge preservation) *Let $s \xrightarrow{\mathsf{a}} t$ be a transition of composite component $B = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and let e be a wait-for edge in $W_\mathsf{B}(s)$ that is not B_i-incident, for every $\mathsf{B}_i \in components(\mathsf{a})$. Then $e \in W_\mathsf{B}(s)$ iff $e \in W_\mathsf{B}(t)$.*

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

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 3.5 (Supercycle) *Let $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ be a composite component and s be a state of B. A subgraph SC of $W_\mathsf{B}(s)$ is a supercycle in $W_\mathsf{B}(s)$ 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 $W_\mathsf{B}(s)$ 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, $\mathsf{B}_i \to \mathsf{a} \in W_\mathsf{B}(s)$ implies $\mathsf{B}_i \to \mathsf{a} \in SC$,
3. *if 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 $W_\mathsf{B}(s)$, and*
   - *(c) there is an edge from a to B_j in SC,*

   that is, $\mathsf{a} \in SC$ implies $\exists\, \mathsf{B}_j : \mathsf{a} \to \mathsf{B}_j \in W_\mathsf{B}(s) \wedge \mathsf{a} \to \mathsf{B}_j \in SC$,

where $\mathsf{a} \in SC$ means that a is a node in SC, etc. Also, write $SC \subseteq W_\mathsf{B}(s)$ when SC is a subgraph of $W_\mathsf{B}(s)$.

Definition 3.6 (Supercycle-free) *$W_\mathsf{B}(s)$ is* supercycle-free *iff there does not exist a supercycle SC in $W_\mathsf{B}(s)$. In this case, say that state s is supercycle-free. Formally, we define the predicate $sc\_free_B(s) \triangleq \neg\exists\, SC : SC \subseteq W_\mathsf{B}(s)$ and SC is a supercycle.*

Figure 3 shows an example supercycle (with boldened edges) for the dining philosophers system of Figure 1. P_0 waits for (enables) a single interaction, Get_0. Get_0 waits for (is disabled by) fork F_0, which waits for interaction Rel_0. Rel_0 in turn waits for P_0. However, this supercycle occurs in a state where P_0 is in h and F_0 is in u_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 actions a, b, c, d, is boldened. The supercycle can be extended to contain B_4, but beither B_5 nor B_6: B_6 is enabled, and B_5 has is ready to execute h, which waits only for B_6. Figure 5 shows that deleting the wait-for edge from d to B_1 in Figure 4 results in an example where

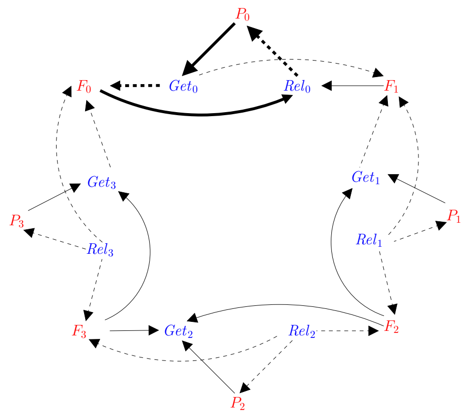


Fig. 3. Example supercycle for dining philosophers system of Figure 1.

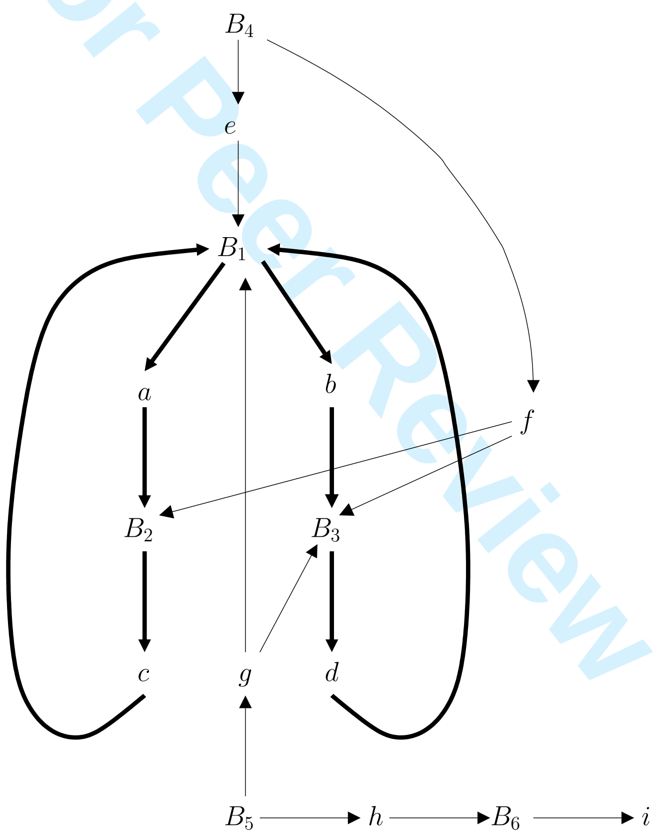


Fig. 4. Example supercycle that is not a simple cycle

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.

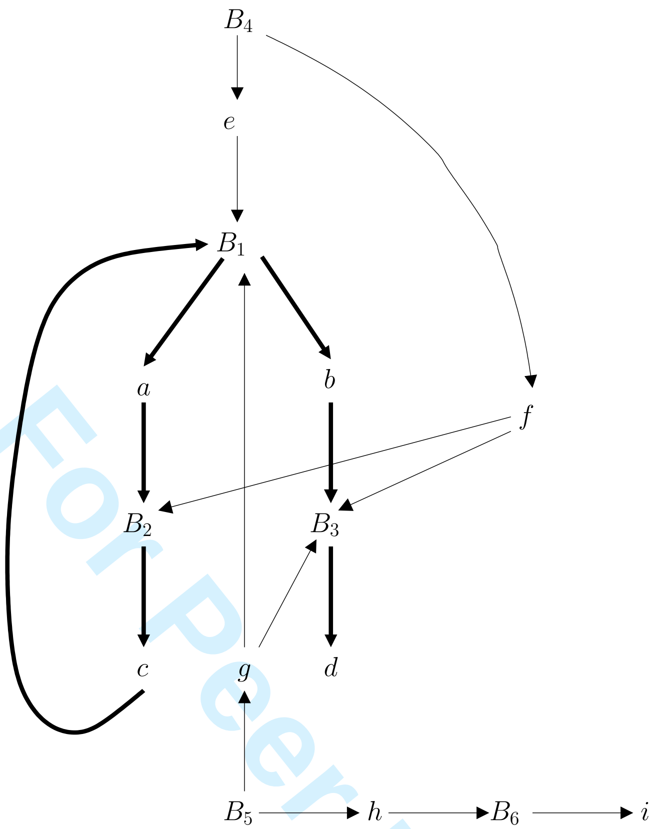


Fig. 5. Example where a wait-for cycle does not imply deadlock

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 3.7 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

PROPOSITION 3.7. *Let s be a state of* B. *If $SC \subseteq W_{\mathsf{B}}(s)$ is a supercycle, then all components B_i in SC cannot execute a transition in any state reachable from s, including s itself.*

Proof. Let B_i be an arbitrary component in SC. By Definition 3.5, 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 $SC \subseteq W_{\mathsf{B}}(t)$, i.e., the same supercycle SC remains in global state t. Repeating this argument from state t and onwards leads us to conclude that $SC \subseteq W_{\mathsf{B}}(u)$ for any state u reachable from s. □

Proposition 3.8 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 3.8. *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 $s{\upharpoonright}\mathsf{B}'$. Then, $W_{\mathsf{B}}(s)$ contains a supercycle.*

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

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

SC satisfies Definition 3.5 and so is a supercycle. □

We consider subcomponent B' in isolation to avoid other phenomena that prevent interactions from executing, e.g., conspiracies [Attie et al. 1993]. Now the converse of Proposition 3.8 is that absence of supercycles in $W_{\mathsf{B}}(s)$ means there is no locally deadlocked subsystem.

Corollary 3.9 (Supercycle-free implies free of local deadlock) *If, for every reachable state s of (B, Q_0), $W_{\mathsf{B}}(s)$ is supercycle-free, then (B, Q_0) is free of local deadlock.*

Proof. We establish the contrapositive. 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 $s{\upharpoonright}\mathsf{B}'$. By Proposition 3.8, $W_{\mathsf{B}}(s)$ contains a supercycle. □

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 $W_{\mathsf{B}}(t)$ contains a supercycle, for each reachable state t. 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 these conditions, we need to characterize the static (structural) and dynamic (formation) properties of supercycles.

3.3. Structural properties of supercycles

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

Definition 3.10 (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 3.11 (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 $in\_depth_{\mathsf{B}}(v, s)$ for $in\_depth_{W_{\mathsf{B}}(s)}(v)$, and also $out\_depth_{\mathsf{B}}(v, s)$ for $out\_depth_{W_{\mathsf{B}}(s)}(v)$.

PROPOSITION 3.12. *A supercycle SC contains no nodes with finite out-depth.*

Proof. By contradiction. Let v be a node in SC with finite out-depth. Hence by Definition 3.11 all outgoing paths from v are simple (and finite), and end in a sink node w, so w has no outgoing wait-for-edges. By assumption, all atomic components are individually deadlock-free, i.e., they always enable at least one interaction. So if w is an atomic component B_i, we have a wait-for-edge $\mathsf{B}_i \rightarrow \mathsf{a}$ for some interaction a, contradicting the fact that w is a sink node. Hence w is some interaction a. Since a has no outgoing edges, it violates clause 3 in Definition 3.5, contradicting the assumption that SC is a supercycle. □

PROPOSITION 3.13. *Every supercycle SC contains at least two nodes.*

Proof. By Definition 3.5, SC is nonempty, and so contains at least one node v. If v is an interaction a, then by Definition 3.5, SC also contains some component B_i such that $\mathsf{a} \rightarrow \mathsf{B}_i$. If v is a component B_i, then, by assumption, B_i enables at least one interaction a, and by Definition 3.5, every interaction that B_i enables must be in SC. Hence in both cases, SC contains at least two nodes. □

PROPOSITION 3.14. *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 3.12, SC cannot be a supercycle. □

PROPOSITION 3.15. *Let $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ be a composite component and s a state of B. Let SC be a supercycle in $W_{\mathsf{B}}(s)$, 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 $W_{\mathsf{B}}(s)$.*

Proof. A vertex with finite in-depth cannot lie on a cycle in SC. Hence by Proposition 3.14, $SC' \neq \emptyset$. Thus SC' satisfies clause (1) of the supercycle definition (3.5). 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. $in\_depth_{SC}(w) = \omega$ by Definition 3.11. Hence $w \in SC'$, by definition of SC'. Furthermore, w is a successor of v in SC', since

SC' consists of *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 (3.5). It follows that every vertex v in SC' has enough successors in SC' to satisfy clauses (2) and (3) of the supercycle definition (3.5). □

PROPOSITION 3.16. *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 $\overline{SC}$ be the graph resulting from replacing each MSCC by a single node. By its construction, $\overline{SC}$ is acyclic, and so contains at least one node x with no outgoing edges. Let CC be the MSCC corresponding to x. It follows that CC is nonempty, and hence CC satisfies clause (1) of the supercycle definition (3.5). It also 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.

Let v be an arbitrary node in CC. Since $CC \subseteq SC$, 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 the supercycle definition (3.5). It follows that every vertex v in CC has enough successors in CC to satisfy clauses (2) and (3) of the supercycle definition (3.5).

Hence, by Definition 3.5, CC is itself a supercycle, and so Clause (1) of the Proposition is established. □

Note also that by Proposition 3.13, CC contains at least two nodes. Hence CC is not a trivial strongly connected component.

PROPOSITION 3.17. *Let SC, SC' be supercycles in $W_B(s)$. Then $SC \cup SC'$ is a supercycle in $W_B(s)$.*

Proof. Straightforward, since each node in $SC \cup SC'$ has enough successors that it waits for to satisfy Def. 3.5. □

4. SUPERCYCLE FORMATION AND ITS CONSEQUENCES

4.1. Supercycle Membership

Definition 4.1 (Supercycle membership, $scyc_{\mathsf{B}}(s, v)$) *Let v be a node of $W_{\mathsf{B}}(s)$. Then $scyc_{\mathsf{B}}(s, v)$ holds iff there exists a supercycle $SC \subseteq W_{\mathsf{B}}(s)$ such that $v \in SC$.*

If a component or interaction is not a node of a supercycle, then we say that it has a *SC-violation*, i.e., a supercycle-violation.

Define $preds_{\mathsf{B}}(s, v) = \{w \mid w \to v \in W_{\mathsf{B}}(s)\}$ and $succs_{\mathsf{B}}(s, v) = \{w \mid v \to w \in W_{\mathsf{B}}(s)\}$. The definition of a supercycle (Def. 3.5) imposes certain constraints on supercycle membership of a node w.r.t. its predecessors and successors in the wait-for-graph, as follows:

Proposition 4.2 (Supercycle-membership constraints) *Let a, B_i be nodes of $W_{\mathsf{B}}(s)$. Then*

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

PROOF. We deal with each clause in turn.

Proof of Clause 1. Assume $scyc_{\mathsf{B}}(s, \mathsf{B}_i)$, and let $SC \subseteq W_B(s)$ be the supercycle containing B_i. Let $\mathsf{aa} \in succs_{\mathsf{B}}(s, \mathsf{B}_i)$. By Def. 3.5, Clause 2, $\mathsf{aa} \in SC$. Hence $(\forall\, \mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i) : scyc_{\mathsf{B}}(s, \mathsf{a}))$. We conclude $scyc_{\mathsf{B}}(s, \mathsf{B}_i) \Rightarrow (\forall\, \mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i) : scyc_{\mathsf{B}}(s, \mathsf{a}))$. Now assume $(\forall\, \mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i) : scyc_{\mathsf{B}}(s, \mathsf{a}))$, and let SC be the union of all the supercycles containing all the $\mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i)$. By Prop. 3.17, $SC \subseteq W_B(s)$ is a supercycle. Let SC' be SC with $\mathsf{B}_i \rightarrow \mathsf{a}$ added, for all $\mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i)$. Then SC' is a supercycle by Def. 3.5, and also $SC' \subseteq W_B(s)$. Hence $scyc_{\mathsf{B}}(s, \mathsf{a})$. We conclude $scyc_{\mathsf{B}}(s, \mathsf{B}_i) \Leftarrow (\forall\, \mathsf{a} \in succs_{\mathsf{B}}(s, \mathsf{B}_i) : scyc_{\mathsf{B}}(s, \mathsf{a}))$.

Proof of Clause 2. Assume $scyc_{\mathsf{B}}(s, \mathsf{B}_i)$, so that $SC \subseteq W_B(s)$ is the supercycle containing B_i. Let $\mathsf{a} \in preds_{\mathsf{B}}(s, \mathsf{B}_i)$, and let SC' be SC with $\mathsf{a} \rightarrow \mathsf{B}_i$ added. Hence SC' is a supercycle by Definition 3.5, Clause 3. Since a was chosen arbitrarily, we conclude $(\forall\, \mathsf{a} \in preds_{\mathsf{B}}(s, \mathsf{B}_i) : scyc_{\mathsf{B}}(s, \mathsf{a}))$.

Proof of Clause 3. Assume $scyc_{\mathsf{B}}(s, \mathsf{a})$, and let $SC \subseteq W_B(s)$ be the supercycle containing a. By Def. 3.5, Clause 3, there exists a $\mathsf{B}_i \in succs_{\mathsf{B}}(s, \mathsf{a})$ such that $\mathsf{B}_i \in SC$. Hence $scyc_{\mathsf{B}}(s, \mathsf{B}_i)$. We conclude $scyc_{\mathsf{B}}(s, \mathsf{a}) \Rightarrow (\exists\, \mathsf{B}_i \in succs_{\mathsf{B}}(s, \mathsf{a}) : scyc_{\mathsf{B}}(s, \mathsf{B}_i))$. Now assume $(\exists\, \mathsf{B}_i \in succs_{\mathsf{B}}(s, \mathsf{a}) : scyc_{\mathsf{B}}(s, \mathsf{B}_i))$, and let $SC \subseteq W_B(s)$ be the supercycle containing some $\mathsf{B}_i \in succs_{\mathsf{B}}(s, \mathsf{a})$. Let SC' be SC with $\mathsf{a} \rightarrow \mathsf{B}_i$ added. Then SC' is a supercycle by Def. 3.5, and also $SC' \subseteq W_B(s)$. Hence $scyc_{\mathsf{B}}(s, \mathsf{a})$. We conclude $scyc_{\mathsf{B}}(s, \mathsf{a}) \Leftarrow (\exists\, \mathsf{B}_i \in succs_{\mathsf{B}}(s, \mathsf{a}) : scyc_{\mathsf{B}}(s, \mathsf{B}_i))$.

Proof of Clause 4. Assume $\neg scyc_{\mathsf{B}}(s, \mathsf{a})$, so that a is not in any supercycle of $W_B(s)$. Let $\mathsf{B}_i \in preds_{\mathsf{B}}(s, \mathsf{a})$. By Def. 3.5, Clause 2, B_i cannot be in any supercycle of $W_B(s)$, since all $\mathsf{aa} \in succs_{\mathsf{B}}(s, \mathsf{B}_i)$ must also be in the supercycle. Hence $\neg scyc_{\mathsf{B}}(s, \mathsf{B}_i)$. Since B_i was chosen arbitrarily, we conclude $\neg scyc_{\mathsf{B}}(s, \mathsf{a}) \Rightarrow (\forall\, \mathsf{B}_i \in preds_{\mathsf{B}}(s, \mathsf{a}) : \neg scyc_{\mathsf{B}}(s, \mathsf{B}_i))$, 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 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 aa that is not in a supercycle.

While Prop. 4.2 gives relationships between supercycle membership of a node and both its successors and predecessors, nevertheless Def. 3.5 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.

Hence, we follow outgoing wait-for edges in computing supercycle-membership. Actually, it turns out to be easier to compute the negation of supercycle membership, which we call *supercycle violation*. This is because supercycle-violation has a base case: when a node has no outgoing wait-for edges. We need a base case, and an inductive definition, because a node that is not in any supercycle may nevertheless be a node of a wait-for cycle, since a cycle of wait-for-edges does not necessarily imply a supercycle.

$\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$

0. **if** $(d = 1 \wedge v = \mathsf{a} \wedge \neg(\exists\, \mathsf{B}_i : \mathsf{a} \rightarrow \mathsf{B}_i \in W_{\mathsf{B}}(t)))$ **return**(tt) **fi** ▷base case for tt result
1. **if** $(v = \mathsf{B}_i \wedge (\exists\, \mathsf{a} : \mathsf{B}_i \rightarrow \mathsf{a} \in W_{\mathsf{B}}(t) : (\exists\, d' : 1 \leq d' < d : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{a}, d', t))))$ **return**(tt) **fi**
2. **if** $(v = \mathsf{a} \wedge (\forall\, \mathsf{B}_i : \mathsf{a} \rightarrow \mathsf{B}_i \in W_{\mathsf{B}}(t) : (\exists\, d' : 1 \leq d' < d : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{B}_i, d', t))))$ **return**(tt) **fi**
3. **return**(ff) ▷no case for tt result, so result is ff

Fig. 6. Formal definition of $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$

Hence, to compute supercycle violation properly, we introduce a notion of the *level* of a violation. A node with no outgoing wait-for edges has a level-1 violation. A node whose violation is based on outgoing edges to neighbors whose violation level is at most $d-1$, has itself a level-d violation. We formalize the notion of *level-d supercycle violation* as the predicate $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$, defined by induction on d.

Definition 4.3 (Supercycle violation, $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$**)** *Let t be a state of (B, Q_0), v be a node of $W_{\mathsf{B}}(t)$, and d an integer ≥ 1. We define the predicate $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$ by induction on d, as follows. We indicate the justification for each clause of the definition.*

Base case, $d = 1$. $\mathsf{scViolate}_{\mathsf{B}}(v, 1, t)$ *iff v is an interaction* a *and it has no outgoing wait-for-edges, otherwise* $\neg\mathsf{scViolate}_{\mathsf{B}}(v, 1, t)$*. Justification: if v has no outgoing wait-for-edges, then it cannot be in a supercycle. Note that v must be an interaction in this case, since a component must have at least one outgoing wait-for edge at all times.*

Inductive step, $d > 1$. $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$ *iff any of the following cases hold. Otherwise* $\neg\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$.

(1) *v is a component* B_i *and there exists interaction* a *such that* $\mathsf{B}_i \rightarrow \mathsf{a} \in W_{\mathsf{B}}(t)$ *and* $(\exists\, d' : 1 \leq d' < d : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{a}, d', t))$*. That is,* B_i *enables an interaction* a *which has a level-d' supercycle-violation, for some $d' < d$. Justification is Prop. 4.2, Clause 1.*
(2) *v is an interaction* a *and for all components* B_i *such that* $\mathsf{a} \rightarrow \mathsf{B}_i \in W_{\mathsf{B}}(t)$*, we have* $(\exists\, d' : 1 \leq d' < d : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{B}_i, d', t))$*. That is, each component* B_i *that* a *waits for has a level-d' supercycle-violation, for some $d' < d$. Justification is Prop. 4.2, Clause 3.*

Figure 6 gives a formal, recursive definition of $\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$. The notation $v = \mathsf{B}_i$ means that v is some component B_i. Likewise, $v = \mathsf{a}$ means that v is some interaction a. Line 0 corresponds to the base case, line 1 corresponds to item 1 of the inductive case, and line 2 corresponds to item 2 of the inductive case. Line 3 handles all cases that do not return true.

In the sequel, we say sc-violation rather than "supercycle violation." The crucial result is that, if v has a level-d sc-violation, for some $d \geq 1$, then v cannot be a node of a supercycle.

Proposition 4.4 (Soundness of supercycle violation w.r.t. supercycle non-membership) *If* $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ *then* $\neg scyc_{\mathsf{B}}(t, v)$*, i.e., supercycle violation implies supercycle non-membership.*

Proof. Proof is by induction in d.

Base case, $d = 1$. v has no outgoing edges. Hence v cannot be in a supercycle.

Induction step, $d > 1$. Assume that v has a level d SC-violation. We have two cases.

Case 1: v is a component B_i. Hence there exists an interaction a such that $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(t)$ and a has a level-$(d-1)$ SC-violation. By the induction hypothesis, $\neg scyc_{\mathsf{B}}(t, \mathsf{a})$. By Prop. 4.2, Clause 1, $\neg scyc_{\mathsf{B}}(t, \mathsf{B}_i)$.

Case 2: v is an interaction a. Hence for all components B_i such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t)$, B_i has a level-$(d-1)$ SC-violation. By the induction hypothesis, $(\forall\, \mathsf{B}_i : \mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t) : \neg scyc_{\mathsf{B}}(t, \mathsf{B}_i))$. By Prop. 4.2, Clause 3, $\neg scyc_{\mathsf{B}}(t, \mathsf{a})$. □

Proposition 4.5 (Completeness of supercycle violation w.r.t. supercycle non-membership) *If* $\neg scyc_{\mathsf{B}}(t, v)$ *then* $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$*, i.e., supercycle non-membership implies supercycle violation.*

Proof. We establish the contrapositive $(\forall\, d \geq 1 : \neg\mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ then $scyc_{\mathsf{B}}(t, v)$. Let V be the set of nodes in $W_{\mathsf{B}}(t)$ with a supercycle-violation, i.e., $V = \{w \mid w \in W_{\mathsf{B}}(t) \wedge (\exists\, d : \mathsf{scViolate}_{\mathsf{B}}(w, d, t))\}$. Let $\overline{V}$ be the remaining nodes, i.e., all nodes in $W_{\mathsf{B}}(t)$ that do not have a supercycle-violation, so $\overline{V} = \{w \mid w \in W_{\mathsf{B}}(t) \wedge (\forall\, d \geq 1 : \neg\mathsf{scViolate}_{\mathsf{B}}(v, d, t))\}$.

If $\overline{V}$ is empty then the proposition holds vacuously and we are done. So assume that $\overline{V}$ is non-empty and let v be an arbitrary node in $\overline{V}$.

Case 1: v is a component B_i. Suppose that there is a wait-for-edge from v to some interaction a that is in V. Then, by Definition 4.3, v has a supercycle violation, which contradicts the choice of v as a member of $\overline{V}$. Hence all wait-for-edges starting in v must end in a node in $\overline{V}$.

Case 2: v is an interaction a. Suppose that every wait-for-edge from v to some component B_i that is in V. Then, by Definition 4.3, v has a supercycle violation, which contradicts the choice of v as a member of $\overline{V}$. Hence some wait-for-edge starting in v must end in a node in $\overline{V}$.

Hence we have that $\overline{V}$ satisfies all three clauses of Definition 3.5: it is nonempty, each component in $\overline{V}$ has all its enabled interactions also in $\overline{V}$, and each interaction in $\overline{V}$ waits for a component in $\overline{V}$. Hence $\overline{V}$ as a whole is a supercycle. Since the nodes of $\overline{V}$ are, by definition of $\overline{V}$, exactly the nodes v such that $(\forall\, d \geq 1 : \neg\mathsf{scViolate}_{\mathsf{B}}(v, d, t))$, we have that any such node v is a node of a supercycle in $W_{\mathsf{B}}(t)$, i.e., $scyc_{\mathsf{B}}(t, v)$. Hence the Proposition is established. □

PROPOSITION 4.6. $\neg scyc_{\mathsf{B}}(t, v)$ *iff* $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$.

Proof. Immediate from Propositions 4.4 and 4.5. □

4.2. The supercycle formation condition

We use the structural properties of supercycles (Sect. 3.3) and the dynamics of wait-for graphs (Prop. 3.4) to define a condition that must hold whenever a supercycle is created. Negating this condition then implies the absence of supercycles.

Proposition 4.7 (Supercycle formation condition) *Assume that* $s \xrightarrow{\mathsf{a}} t$ *is a transition of* (B, Q_0)*,* $W_{\mathsf{B}}(s)$ *is supercycle-free, and that* $W_{\mathsf{B}}(t)$ *contains a supercycle. Then, in* $W_{\mathsf{B}}(t)$*, there exists a* CC *such that*

(*1*) CC *is a subgraph of* $W_{\mathsf{B}}(t)$
(*2*) CC *is strongly connected*
(*3*) CC *is a supercycle*

(4) *in* $W_{\mathsf{B}}(t)$*, there is no wait-for edge from a node in* CC *to a node outside of* CC
(5) *there exists a component* $\mathsf{B}_i \in components(\mathsf{a})$ *such that* B_i *is in* CC

PROOF. By assumption, there is a supercycle SC that is a subgraph of $W_{\mathsf{B}}(t)$. By Proposition 3.16, 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 $\mathsf{B}_i \notin CC$ for every $\mathsf{B}_i \in components(\mathsf{a})$. Then, no edge in CC is B_i-incident. Hence, by Proposition 3.4, every edge in CC is an edge in $W_{\mathsf{B}}(s)$. Hence CC is a subgraph of $W_{\mathsf{B}}(s)$. Now let v be an arbitrary node in CC. Suppose v is a component B_j. By assumption, $\mathsf{B}_j \notin components(\mathsf{a})$, and so $s{\upharpoonright}\mathsf{B}_j = t{\upharpoonright}\mathsf{B}_j$ by Definition 2.3. Hence B_j enables the same set of interactions in state s as in state t. Also, in $W_{\mathsf{B}}(t)$, all of B_j's wait-for edges must end in an interaction that is in CC, since CC is a supercycle in $W_{\mathsf{B}}(t)$. Hence the same holds in $W_{\mathsf{B}}(s)$. If v is an interaction, it must also have a wait-for-edge e' to some component $\mathsf{B}_j \in CC$, since CC is a supercycle in $W_{\mathsf{B}}(t)$. Hence this also holds in $W_{\mathsf{B}}(s)$. Hence v has enough successors in CC to satisfy the supercycle definition (Def. 3.5). We conclude that CC by itself is a supercycle in $W_{\mathsf{B}}(s)$, which contradicts the assumption that $W_{\mathsf{B}}(s)$ is supercycle-free. Hence, $\mathsf{B}_i \in CC$ for some $\mathsf{B}_i \in components(\mathsf{a})$, and so Clause 5 is established. □

4.3. General supercycle violation condition

We use Prop. 4.7 to formulate a condition that prevents the formation of supercycles. For transition $s \xrightarrow{\mathsf{a}} t$, we determine for every component $\mathsf{B}_i \in components(\mathsf{a})$ whether it is possible for B_i to be a node in a strongly-connected supercycle CC in $W_{\mathsf{B}}(t)$. 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., $\neg scyc_{\mathsf{B}}(s, \mathsf{B}_i)$.
(2) *no strong-connectedness*: B_i is a node in a supercycle, but not a node in a *strongly-connected* supercycle.

We formalize the second condition as follows.

Definition 4.8 (Strong connectedness violation, $\mathsf{sConnViolate}_{\mathsf{B}}(v,t)$**)** *Let* v *be a node of* $W_{\mathsf{B}}(t)$*. Then* $\mathsf{sConnViolate}_{\mathsf{B}}(v,t)$ *holds iff there does not exist a strongly connected supercycle* SSC *such that* $v \in SSC$ *and* $SSC \subseteq W_B(t)$*.*

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

Definition 4.9 (General supercycle violation, $\mathsf{genViolate}_{\mathsf{B}}(v,t)$**)** *Let* v *be a node of* $W_{\mathsf{B}}(t)$*. Then* $\mathsf{genViolate}_{\mathsf{B}}(v,t) \triangleq (\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v,d,t)) \vee \mathsf{sConnViolate}_{\mathsf{B}}(v,t)$.

Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition. If, for every $\mathsf{B}_i \in components(\mathsf{a})$, $\mathsf{genViolate}_{\mathsf{B}}(v,t)$ holds, then, as we show below, $s \xrightarrow{\mathsf{a}} t$ does not introduce a supercycle, i.e., if s is supercycle-free, then so is t. However, evaluating this condition over all global transitions is subject to state explosion, and so we formulate below a "local" version of the general condition, which can be evaluated in "small subsystems", and so we often avoid state-explosion. Hence the advantage of the local versions is that they are usually efficiently computable, as we show in the sequel. We also formulate a "linear" condition (both global and local), which is simpler (but "more incomplete") than the general condition, and so is easier to evaluate.

We remark that, as shown above $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ implies that v cannot be in a supercycle. Hence, v cannot be in a strongly-connected supercycle. Hence $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ implies $\mathsf{sConnViolate}_{\mathsf{B}}(v, t)$. It is however convenient to state the formation violation condition in this manner, since we will formulate a local version for each of $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ and $\mathsf{sConnViolate}_{\mathsf{B}}(v, t)$, and the implication does not necessarily hold for the local versions.

We therefore now have four deadlock-freedom conditions: global general, local general, global linear, and local linear. We therefore define an abstract version of the deadlock-freedom condition first.

4.4. 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 4.10 (Behavioral restriction condition) *A* behavioral restriction condition *$\mathcal{BC}$ is a predicate $\mathcal{BC} : (\mathsf{B}, Q_0, \mathsf{a}) \to \{\mathtt{tt}, \mathtt{ff}\}$.*

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

Definition 4.11 (Supercycle-freedom preserving) *A behavioral restriction condition $\mathcal{BC}$ is* supercycle-freedom preserving *iff, for every system (B, Q_0) and $\mathsf{a} \in \gamma$ such that $\mathcal{BC}(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{tt}$, the following holds:*

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

Theorem 4.12 (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 $\mathsf{a} \in \gamma$: $\mathcal{BC}(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{tt}$*

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 $s \xrightarrow{\mathsf{a}} t$, $W_{\mathsf{B}}(s)$ is supercycle-free implies that $W_{\mathsf{B}}(t)$ is supercycle-free. This is immediate from Assumption 2, and Definition 4.11. $\square$

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 4.13 (Deadlock-freedom via several supercycle-freedom preserving restrictions)

Assume that

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

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

PROOF. Similar to the proof of Th. 4.12, except that, for the transition $s \xrightarrow{\mathsf{a}} t$, use the supercycle-freedom preserving restriction $\mathcal{BC}$ corresponding to a. □

4.5. Overview of the four supercycle-freedom preserving restrictions

The supercycle formation condition (Proposition 4.7) 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.

Hence, for a BIP system (B, Q_0), our fundamental condition for the prevention of supercycles is that for every reachable transition $s \xrightarrow{\mathsf{a}} t$ resulting from execution of a, every component B_i of a must exhibit a supercycle-violation (Definition 4.3) in state t (the state resulting from the execution of a). For a given BIP system (B, Q_0) and interaction a, we denote that condition $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$, and define it formally below. This condition is, in a sense, the "most general" condition for supercycle-freedom.

If $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds, and global state s is supercycle-free, and $s \xrightarrow{\mathsf{a}} t$, then it follows (as we establish below) that global state t 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 provide 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 $\mathcal{GALT}$.
(2) To provide "local variants" of $\mathcal{GALT}$ and $\mathcal{GLIN}$, which can often be evaluated in small subsystems of (B, Q_0), thereby avoiding state-explosion. The local conditions imply the corresponding global ones, i.e., they are sufficient but not necessary for deadlock-freedom.

5. GLOBAL CONDITIONS FOR DEADLOCK FREEDOM

5.1. A Global AND-OR Condition for Deadlock Freedom

Our first global condition is the most general possible: simply assert that, after execution of interaction a, some $\mathsf{B}_i \in components(\mathsf{a})$ exhibits a supercycle-violation, as given by $\mathsf{scViolate}_\mathsf{B}(\mathsf{B}_i, d, t)$ (Definition 4.3).

Definition 5.1 **($\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$)** *Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0). Then, in t, the following holds. For every component $\mathsf{B}_i \in components(\mathsf{a})$, the formation violation condition holds. Formally,*

$\forall\, \mathsf{B}_i \in components(\mathsf{a}), \mathsf{genViolate}_\mathsf{B}(\mathsf{B}_i, t)$.

We now show that $\mathcal{GALT}$ is supercycle-freedom preserving.

THEOREM 5.2. *$\mathcal{GALT}$ is supercycle-freedom preserving.*

Proof. We must establish: for every reachable transition $s \xrightarrow{\mathsf{a}} t$, $W_\mathsf{B}(s)$ is supercycle-free implies that $W_\mathsf{B}(t)$ is supercycle-free. Our proof is by contradiction, so we assume the existence of a reachable transition $s \xrightarrow{\mathsf{a}} t$ such that $W_\mathsf{B}(s)$ is supercycle-free and $W_\mathsf{B}(t)$ contains a supercycle.

By Proposition 4.7 there exists a component $\mathsf{B}_i \in components(\mathsf{a})$ such that B_i is in CC, where CC is a strongly connected supercycle that is a subgraph of $W_\mathsf{B}(t)$.

Since CC is a strongly connected supercycle, we have, by Definition 4.8, that $\neg\mathsf{sConnViolate}_\mathsf{B}(\mathsf{B}_i, t)$ holds.

Since CC is a supercycle, we have, by Proposition 4.6, that $\neg(\exists\, d \geq 1 : \mathsf{scViolate}_\mathsf{B}(\mathsf{B}_i, d, t))$ holds.

Hence, by Definition 4.9, $\neg\mathsf{genViolate}_\mathsf{B}(\mathsf{B}_i, t)$ But, by Definition 5.1, we have $\mathsf{genViolate}_\mathsf{B}(\mathsf{B}_i, t)$. Hence, we have the desired contradiction, and so the theorem holds. □

5.2. 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 $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0), we consider the in-depth and out-depth of the components $\mathsf{B}_i \in components(\mathsf{a})$. There are three cases:

Case 1. B_i has finite in-depth in $W_\mathsf{B}(t)$: then, if $\mathsf{B}_i \in SC$, it can be removed and still leave a supercycle SC', by Proposition 3.15. Hence SC' exists in $W_\mathsf{B}(s)$, and so B_i is not essential to the creation of a supercycle.
Case 2. B_i has finite out-depth in $W_\mathsf{B}(t)$: by Proposition 3.12, B_i cannot be part of a supercycle, and so $SC \subseteq W_\mathsf{B}(s)$.
Case 3. B_i has infinite in-depth and infinite out-depth in $W_\mathsf{B}(t)$: in this case, B_i is possibly an essential part of SC, i.e., SC was created in going from s to t.

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

Definition 5.3 **($\mathcal{GLIN}(B, Q_0, \mathsf{a})$)** *$\mathcal{GLIN}(B, Q_0, \mathsf{a})$ holds iff, for every reachable transition $s \xrightarrow{\mathsf{a}} t$ of BIP-system (B, Q_0), the following holds in state t:*

$$\forall \mathsf{B}_i \in components(a) : in\_depth_{\mathsf{B}}(\mathsf{B}_i, t) < \omega \vee out\_depth_{\mathsf{B}}(\mathsf{B}_i, t) < \omega.$$

That is, for every component B_i *of* $components(\mathsf{a})$*: either* B_i *has finite in-depth, or finite out-depth, in* $W_{\mathsf{B}}(t)$.

PROPOSITION 5.4. *Assume that node* v *of* $W_{\mathsf{B}}(t)$ *has a finite in-depth of* d *in* $W_{\mathsf{B}}(t)$*, i.e.,* $in\_depth_{\mathsf{B}}(v, t) = d$*. Then* $\mathsf{sConnViolate}_{\mathsf{B}}(v, t)$.

Proof. A node with finite in-depth cannot be in a wait-for-cycle, and therefore cannot be in a strongly connected supercycle. □

PROPOSITION 5.5. *Assume that node* v *of* $W_{\mathsf{B}}(t)$ *has a finite out-depth of* d *in* $W_{\mathsf{B}}(t)$*, i.e.,* $out\_depth_{\mathsf{B}}(v, t) = d$*. Then* $\mathsf{scViolate}_{\mathsf{B}}(v, d + 1, t)$.

Proof. Proof is by induction on d.

Base case, $d = 0$. Hence by $out\_depth_{\mathsf{B}}(v, t) = 0$ and Definitions 3.10 and 3.11, v has no outgoing wait-for-edges in $W_{\mathsf{B}}(t)$. Hence by Definition 4.3, $\mathsf{scViolate}_{\mathsf{B}}(v, 1, t)$.

Inductive step, $d > 0$. Let u be an arbitrary successor of v, i.e., a node u such that $v \to u \in W_{\mathsf{B}}(t)$. By Definitions 3.10 and 3.11, u has an out-depth d' that is less than d. That is, $out\_depth_{\mathsf{B}}(u, t) = d' < d$. By the induction hypothesis applied to d', we obtain $\mathsf{scViolate}_{\mathsf{B}}(u, d' + 1, t)$. Hence by Definition 4.3, Clauses 1 and 2, $\mathsf{scViolate}_{\mathsf{B}}(v, d + 1, t)$. □

LEMMA 5.6. $\forall \mathsf{a} \in \gamma : \mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a}) \Rightarrow \mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$.

Proof. Assume, for arbitrary $\mathsf{a} \in \gamma$, that $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$ holds. That is,

For every reachable transition $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0),
$\forall \mathsf{B}_i \in components(\mathsf{a}) : in\_depth_{\mathsf{B}}(\mathsf{B}_i, t) < \omega \vee out\_depth_{\mathsf{B}}(\mathsf{B}_i, t) < \omega$.

By Propositions 5.4 and 5.5,

For every reachable transition $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0),
$\forall \mathsf{B}_i \in components(\mathsf{a}) : \mathsf{sConnViolate}_{\mathsf{B}}(\mathsf{B}_i, t) \vee (\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{B}_i, d, t))$.

Hence by Definition 4.9,
For every reachable transition $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0),
$\forall \mathsf{B}_i \in components(\mathsf{a}) : \mathsf{genViolate}_{\mathsf{B}}(\mathsf{B}_i, t)$

Hence $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds. □

THEOREM 5.7. $\mathcal{GLIN}$ *is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 5.6 and Theorem 5.2. □

5.3. Deadlock freedom using global restrictions

Corollary 5.8 (Deadlock-freedom via $\mathcal{GALT}$, $\mathcal{GLIN}$) *Assume that*

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

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

Proof. Immediate from Theorems 5.2, 5.7 and Corollary 4.13. □

6. LOCAL CONDITIONS FOR DEADLOCK FREEDOM

Evaluating the global restrictions $\mathcal{GALT}(B, Q_0, \mathsf{a})$, $\mathcal{GLIN}(B, Q_0, \mathsf{a})$ requires checking all reachable transitions of (B, Q_0), which is, in general, subject to state-explosion. We need restrictions which imply a global restriction, 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 (Structure Graph G_B, G^ℓ_a) *The structure graph G_B of composite component $B = \gamma(\mathsf{B}_1, \dots, \mathsf{B}_n)$ is a bipartite graph whose nodes are the $\mathsf{B}_1, \dots, \mathsf{B}_n$ and all the $\mathsf{a} \in \gamma$. There is an edge between B_i and interaction a iff B_i participates in a, i.e., $\mathsf{B}_i \in components(\mathsf{a})$. Define the* distance *between two nodes to be the number of edges in a shortest path between them. Let G^ℓ_a be the subgraph of G_B that contains a and all nodes of G_B that have a distance to a less than or equal to ℓ.*

Definition 6.2 (Deadlock-checking subsystem, D^ℓ_a) *Define D^ℓ_a, the* deadlock-checking subsystem *for interaction* a *and* depth ℓ*, to be the subsystem of (B, Q_0) based on the set of components in $G^{2\ell}_\mathsf{a}$.*

Definition 6.3 (Border node, interior node of D^ℓ_a) *A node v of D^ℓ_a is a* border-node *iff it has an edge in G_B to a node outside of D^ℓ_a. If node v of D^ℓ_a is not a border node, then it is an* internal node.

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

Proposition 6.4 (Wait-for-edge projection) *Let (B', Q'_0) be a subsystem of (B, Q_0). Let s be a state of (B, Q_0), and $s' = s{\upharpoonright}\mathsf{B}'$. Let a be an interaction of (B', Q'_0), and $\mathsf{B}_i \in components(\mathsf{a})$ an atomic component of B'. Then (1) $\mathsf{a} \to \mathsf{B}_i \in W_\mathsf{B}(s)$ iff $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}'}(s')$, and (2) $\mathsf{B}_i \to \mathsf{a} \in W_\mathsf{B}(s)$ iff $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}'}(s')$.*

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

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

6.2. A Local AND-OR Condition for Deadlock Freedom

We now seek a local condition, which we evaluate in D^ℓ_a, and which implies $\mathcal{GALT}$. We define local versions of both $\mathsf{scViolate}_\mathsf{B}(v, d, t)$ and $\mathsf{sConnViolate}_\mathsf{B}(v, t)$.

To achieve a local and conservative approximation of $\mathsf{scViolate}_\mathsf{B}(v, d, t)$, we make the "pessimistic" assumption that the violation status of border nodes of D^ℓ_a cannot be known, since it depends on nodes outside of D^ℓ_a. Now, if an internal node v of D^ℓ_a can be marked with a level d sc-violation, by applying Definition 4.3 only within D^ℓ_a, and with the border nodes marked as non-violating, then it is also the case, as we show below, that v has a level d sc-violation overall.

To achieve a local and conservative approximation of $\mathsf{sConnViolate}_{\mathsf{B}}(v,t)$, we project onto a subsystem.

6.2.1. Local supercycle violation condition. We define the predicate $\mathsf{scViolateLoc}(v,d,t,D^{\ell}_{\mathsf{a}})$ to hold iff node v in $W_B(t)$ has a level-d supercycle-violation *that can be confirmed within* D^{ℓ}_{a}.

Definition 6.5 (Local supercycle violation, $\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$**)** *Let t_{a} be a state of D^{ℓ}_{a} and v be a node of D^{ℓ}_{a}. We define $\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$ by induction on d, as follows.*

Base case, $d=1$. $\mathsf{scViolateLoc}(v,1,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$ *iff v is an interaction aa and aa is an interior node of D^{ℓ}_{a} that has no outgoing wait-for edges in $W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}})$. Otherwise* $\neg\mathsf{scViolateLoc}(v,1,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$.

Inductive step, $d>1$. $\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$ *iff either of the following two cases hold. Otherwise* $\neg\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$.

(*1*) *v is a component B_i and there exists an interaction aa such that $B_i \rightarrow \mathsf{aa} \in W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}})$ and $(\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{aa},d',t_{\mathsf{a}},D^{\ell}_{\mathsf{a}}))$. That is, B_i enables an interaction aa which has a level-d' supercycle-violation in D^{ℓ}_{a}, for some $d' < d$.*

(*2*) *v is an interaction aa and an internal node of D^{ℓ}_{a} and for all components B_i such that $\mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}})$, we have $(\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{B}_i,d',t_{\mathsf{a}},D^{\ell}_{\mathsf{a}}))$. That is, each component B_i that aa waits for has a level-d' supercycle-violation in D^{ℓ}_{a}, for some $d' < d$*

Note that if v is an interaction aa and a border node, then $\mathsf{scViolateLoc}(\mathsf{aa},d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$ is false, for all d. This is because aa has some component that is outside D^{ℓ}_{a}, and so this component cannot be checked. A component cannot have a level-1 supercycle-violation since it must have at least one outgoing wait-for edge at all times. Figure 7 gives a formal, recursive definition of $\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$. The notation $v=\mathsf{B}_i$ means that v is some component B_i. Likewise, $v=\mathsf{aa}$ means that v is some interaction a, and "$v=\mathsf{aa}$ is interior" means that v is an interaction a and also an internal node. Line 0 corresponds to the base case, line 1 corresponds to item 1 of the inductive case, and line 2 corresponds to item 2 of the inductive case. Line 3 handles all cases that do not return true.

$\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$
$\triangleright$ Precondition: v is a node of D^{ℓ}_{a} and $d \ge 1$
0. if $(d=1 \wedge v=\mathsf{aa}$ is interior $\wedge \neg(\exists\, \mathsf{B}_i : \mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}})))$ **return**(tt);
1. if $(v=\mathsf{aa}$ is interior $\wedge (\forall\, \mathsf{B}_i : \mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}}) : (\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{B}_i,d',t_{\mathsf{a}},D^{\ell}_{\mathsf{a}}))))$
   return(tt);
2. if $(v=\mathsf{B}_i \wedge (\exists\, \mathsf{aa} : \mathsf{B}_i \rightarrow \mathsf{aa} \in W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}}) : (\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{aa},d',t_{\mathsf{a}},D^{\ell}_{\mathsf{a}}))))$ **return**(tt);
3. **return**(ff)

Fig. 7. Formal definition of $\mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}})$.

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

PROPOSITION 6.6. *Let t be an arbitrary reachable state of BIP-system (B,Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell \ge 1$, let $t_{\mathsf{a}} = t\lceil D^{\ell}_{\mathsf{a}}$. Then*
$\forall\, d \ge 1 : \mathsf{scViolateLoc}(v,d,t_{\mathsf{a}},D^{\ell}_{\mathsf{a}}) \Rightarrow \mathsf{scViolate}_{\mathsf{B}}(v,d,t)$.

Proof. Proof is by induction on d.

Base case, $d = 1$. Assume $\mathsf{scViolateLoc}(v, 1, t_\mathsf{a}, D^\ell_\mathsf{a})$ for some node v. Then, by Figure 7, v is an interior node and an interaction aa of D^ℓ_a, and has no outgoing wait-for edges. Therefore, in $W_\mathsf{B}(t)$, it is still the case that v has no outgoing wait-for edges. Hence $\mathsf{scViolate}_\mathsf{B}(v, 1, t)$ holds.

Inductive step, $d > 1$. Assume $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$ for some node v and some $d > 1$. We proceed by cases on Figure 7.

(1) v is an interior interaction aa and $(\forall\, \mathsf{B}_i : \mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : (\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{B}_i, d', t_\mathsf{a}, D^\ell_\mathsf{a})))$.
Choose an arbitrary B_i such that $\mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$. By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{B}_i, d', t_\mathsf{a}, D^\ell_\mathsf{a})$, we have $\mathsf{scViolate}_\mathsf{B}(\mathsf{B}_i, d', t)$ for some $d' < d$. Since $W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction, we have $\mathsf{aa} \to B_i \in W_\mathsf{B}(t)$ and $\mathsf{scViolate}_\mathsf{B}(\mathsf{B}_i, d', t)$. Hence by Definition 4.3, Clause 1, we have $\mathsf{scViolate}_\mathsf{B}(v, d, t)$.

(2) v is a component B_i and $(\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : (\exists\, d' : 1 \le d' < d : \mathsf{scViolateLoc}(\mathsf{aa}, d', t_\mathsf{a}, D^\ell_\mathsf{a})))$.
By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{aa}, d', t_\mathsf{a}, D^\ell_\mathsf{a})$, we have $\mathsf{scViolate}_\mathsf{B}(\mathsf{aa}, d', t)$ for some $d' < d$. Since $W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction, we have $\mathsf{B}_i \to \mathsf{aa} \in W_\mathsf{B}(t)$ and $\mathsf{scViolate}_\mathsf{B}(\mathsf{aa}, d', t)$. Hence by Definition 4.3, Clause 1, we have $\mathsf{scViolate}_\mathsf{B}(v, d, t)$.

□

6.2.2. Local strong connectedness condition. We now present the local version of the strong connectedness violation condition, given above in Definition 4.8.

Definition 6.7 (Local strong connectedness violation, $\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D^\ell_\mathsf{a})$**)**

Let L be the nodes of $W_{D^\ell_a}(t_\mathsf{a})$ that have no local supercycle violation, i.e., $L = \{v \mid v \in D^\ell_a \wedge \neg(\exists\, d \ge 1 : \mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a}))\}$. Let v be an arbitrary node in L. Let $WL = W_{D^\ell_a}(t_\mathsf{a})\lceil L$, i.e., WL is the subgraph of $W_{D^\ell_a}(t_\mathsf{a})$ consisting of the nodes in L, and the edges between those nodes that are also edges in $W_{D^\ell_a}(t_\mathsf{a})$.

Then, $\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D^\ell_\mathsf{a})$ holds iff:

(1) *there does not exist a nontrivial strongly connected supercycle SSC such that $v \in SSC$ and $SSC \subseteq WL$, and*
(2) *either*
  (a) *every wait-for path π from v to a border node of D^ℓ_a contains at least one node with a local supercycle violation*
  or
  (b) *every wait-for path π' from a border node of D^ℓ_a to v contains at least one node with a local supercycle violation*

We show that the local strong connectedness condition implies the global strong connectedness condition.

PROPOSITION 6.8. *Let t be an arbitrary reachable state of BIP-system (B, Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell > 0$, let $t_\mathsf{a} = t\lceil D^\ell_\mathsf{a}$. Then*
$\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D^\ell_\mathsf{a}) \Rightarrow \mathsf{sConnViolate}_\mathsf{B}(v, t)$.

PROOF. By contradiction. Assume there exists a node v in D_a^ℓ such that $\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell) \wedge \neg\mathsf{sConnViolate}_\mathsf{B}(v,t)$. By $\neg\mathsf{sConnViolate}_\mathsf{B}(v,t)$ and Definition 4.8, there exists a strongly connected supercycle SSC such that $v \in SSC$ and $SSC \subseteq W_B(t)$. Then, there are two cases:

(1) $SSC \subseteq W_{D_a^\ell}(t_\mathsf{a})$: let x be any node in SSC. Since x is a node in a supercycle, we have by Proposition 4.4, that $\neg(\exists\, d \geq 1 : \mathsf{scViolate}_\mathsf{B}(x,d,t))$. Hence $(\forall\, d \geq 1 : \neg\mathsf{scViolate}_\mathsf{B}(x,d,t))$. Hence by Proposition 6.6, we have $(\forall\, d \geq 1 : \neg\mathsf{scViolateLoc}(x,d,t_\mathsf{a},D_\mathsf{a}^\ell))$. Let L, WL be as given in Definition 6.7. Then $x \in L$, and since x is an arbitrary node of SSC, we have $SSC \subseteq WL$. Thus Clause 1 of Definition 6.7 is violated.

(2) $SSC \not\subseteq W_{D_a^\ell}(t_\mathsf{a})$: then there exists a node $x \in SSC - D_a^\ell$. Since $v \in SSC$, there must exist a wait-for path π from v to x and a wait-for path π' from x to v. Since $v \in D_a^\ell$ and $x \notin D_a^\ell$, it follows that both π, π' cross a border node of D_a^ℓ. Furthermore, since π, π' are part of SSC, every node along π, π' is in a supercycle, and so cannot have a supercycle violation. By Proposition 6.6, the nodes on π, π' cannot have a local supercycle violation. Hence Clauses 2a and 2b of Definition 6.7 are violated, since they require that at least one node along π, π' respectively, have a local supercycle violation.

In both cases, Definition 6.7 is violated. But Definition 6.7 must hold, since we have $\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$. Hence the desired contradiction. □

6.2.3. General local violation condition. We showed above that local supercycle violation implies global supercycle violation, and local strong connectedness violation implies global string 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 local supercycle formation condition implies the global supercycle formation condition.

Definition 6.9 (General local supercycle violation, $\mathsf{genViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$**)** *Let v be a node of D_a^ℓ. Then* $\mathsf{genViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell) \triangleq (\exists\, d \geq 1 : \mathsf{scViolateLoc}(v,d,t_\mathsf{a},D_\mathsf{a}^\ell)) \vee \mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$.

PROPOSITION 6.10. *Let t be an arbitrary reachable state of BIP-system (B, Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell > 0$, let $t_\mathsf{a} = t {\upharpoonright} D_\mathsf{a}^\ell$. Then*
$\mathsf{genViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell) \Rightarrow \mathsf{genViolate}_\mathsf{B}(v,t)$.

PROOF. Assume that $\mathsf{genViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$ holds. Then, by Definition 4.9, $(\exists\, d \geq 1 : \mathsf{scViolateLoc}(v,d,t_\mathsf{a},D_\mathsf{a}^\ell)) \vee \mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$. We proceed by cases:

(1) $(\exists\, d \geq 1 : \mathsf{scViolateLoc}(v,d,t_\mathsf{a},D_\mathsf{a}^\ell))$: hence $(\exists\, d \geq 1 : \mathsf{scViolate}_\mathsf{B}(v,d,t))$ by Proposition 6.6.

(2) $\mathsf{sConnViolateLoc}(v, t_\mathsf{a}, D_\mathsf{a}^\ell)$: hence $\mathsf{sConnViolate}_\mathsf{B}(v,t)$ by Proposition 6.8.

By Definition 4.9, $\mathsf{genViolate}_\mathsf{B}(v,t) \triangleq (\exists\, d \geq 1 : \mathsf{scViolate}_\mathsf{B}(u,d,t)) \vee \mathsf{sConnViolate}_\mathsf{B}(v,t)$. Hence we conclude that $\mathsf{genViolate}_\mathsf{B}(v,t)$ holds. □

$\mathsf{scViolate}_{\mathsf{B}}(v, d, t)$	v confirmed at depth d to not be in supercycle
$\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}})$	v locally determined to not be in a supercycle
$\mathsf{sConnViolate}_{\mathsf{B}}(v, t)$	v not in a strongly connected supercycle
$\mathsf{sConnViolateLoc}(v, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}})$	v locally determined to not be in a strongly connected supercycle
$\mathsf{genViolate}_{\mathsf{B}}(v, t)$	v does not contribute to a supercycle
$\mathsf{genViolateLoc}(v, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}})$	v locally determined to not contribute to a supercycle

Fig. 8. Summary of predicates

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

Definition 6.11 **($\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$)** *Let $\ell > 0$, and let $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ be an arbitrary reachable transition of D^{ℓ}_{a}. Then, in t_{a}, the following holds. For every component B_i of $\mathit{components}(\mathsf{a})$: B_i has a supercycle formation violation that can be confirmed within D^{ℓ}_{a}. Formally,*

$$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathsf{genViolateLoc}(\mathsf{B}_i, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}}).$$

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

LEMMA 6.12. *Let $\mathsf{a} \in \gamma$ be an interaction of BIP-system (B, Q_0). Then* $(\exists\, \ell > 0 : \mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell))$ *implies* $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$

PROOF. Assume $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ for some $\ell > 0$. Let $s \xrightarrow{\mathsf{a}} t$ be an arbitrary reachable transition of BIP-system (B, Q_0), and let $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ be the projection of $s \xrightarrow{\mathsf{a}} t$ onto D^{ℓ}_{a}. By Corollary 2.14, $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ is a reachable transition of D^{ℓ}_{a}.

By Definition 6.11, we have for some $\ell > 0$:

for every reachable transition $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ of D^{ℓ}_{a}:

$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathsf{genViolateLoc}(\mathsf{B}_i, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}})$.

From this and Proposition 6.10,

for every reachable transition $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0):

$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathsf{genViolate}_{\mathsf{B}}(B_i, t)$

Hence, by Definition 5.1, $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds. $\square$

THEOREM 6.13. *$\mathcal{LALT}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 6.12 and Theorem 5.2. $\square$

6.3. A Local Linear Condition for Deadlock Freedom

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

Definition 6.14 **($\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$)** *Let $\ell > 0$ and $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ be an arbitrary reachable transition of D^{ℓ}_{a}. Then, in t_{a}, the following holds. For every component B_i of $\mathit{components}(\mathsf{a})$: either B_i has in-depth less than $2\ell - 1$, or out-depth less than $2\ell - 1$, in $W_{D^{\ell}_{\mathsf{a}}}(t_{\mathsf{a}})$. Formally,*

$$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathit{in\_depth}_{D^{\ell}_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1 \lor \mathit{out\_depth}_{D^{\ell}_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1.$$

To infer deadlock-freedom in (B, Q_0) by checking $\mathcal{LLIN}(\mathsf{B}, Q_0, a, \ell)$, we show that wait-for behavior in B "projects down" to any subcomponent B', and that wait-for behavior in B' "projects up" to B.

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 6.15 (In-projection, Out-projection) *Let $\ell > 0$, 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 $G_a^{2\ell}$. Let s be a state of (B, Q_0), and $s' = s{\upharpoonright}B'$. Then (1) $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1$ iff $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') < 2\ell - 1$, and (2) $\mathit{out\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1$ iff $\mathit{out\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') < 2\ell - 1$.*

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.

$\underline{\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1 \text{ implies } \mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') < 2\ell - 1}$: Assume $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1$. Let π be an arbitrary wait-for path in $W_{\mathsf{B}'}(s')$ that ends in B_i. Since (B', Q'_0) is a subsystem of (B, Q_0), by Definition 3.3 and $s' = s{\upharpoonright}B'$, $W_{\mathsf{B}'}(s')$ is a subgraph of $W_{\mathsf{B}}(s)$. Hence π is a wait-for-path in $W_{\mathsf{B}}(s)$. By $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1$, we have $|\pi| < 2\ell - 1$. Hence $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') < 2\ell - 1$ since π was arbitrarily chosen.

$\underline{\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') < 2\ell - 1 \text{ implies } \mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) < 2\ell - 1}$: Assume $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) \geq 2\ell - 1$. Then there exists a wait-for path π in $W_{\mathsf{B}}(s)$ such that $|\pi| \geq 2\ell - 1$. Let ρ be the prefix 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 $G_a^{2\ell}$ is $2\ell - 1$, we conclude that ρ is a wait-for path that is wholly contained in $W_{\mathsf{B}'}(s')$. Hence we have $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \geq 2\ell - 1$. We have thus established $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, s) \geq 2\ell - 1$ implies $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \geq 2\ell - 1$. The contrapositive is the desired result. □

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

LEMMA 6.16. *Let a be an interaction of B, i.e., $\mathsf{a} \in \gamma$. If $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ holds for some finite $\ell > 0$, then $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$ holds.*

Proof. Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0) and let $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, $s_{\mathsf{a}} = s{\upharpoonright}D_{\mathsf{a}}^{\ell}$, $t_{\mathsf{a}} = t{\upharpoonright}D_{\mathsf{a}}^{\ell}$. Then $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ is a reachable transition of D_{a}^{ℓ} by Corollary 2.14. By $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$, $\mathit{in\_depth}_{D_{\mathsf{a}}^{\ell}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1 \vee \mathit{out\_depth}_{D_{\mathsf{a}}^{\ell}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1$. Hence by Proposition 6.15, $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, t) < 2\ell - 1 \vee \mathit{out\_depth}_{\mathsf{B}}(\mathsf{B}_i, t) < 2\ell - 1$. So $\mathit{in\_depth}_{\mathsf{B}}(\mathsf{B}_i, t) < \omega \vee \mathit{out\_depth}_{\mathsf{B}}(\mathsf{B}_i, t) < \omega$. Hence $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$. □

THEOREM 6.17. *$\mathcal{LLIN}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 6.16 and Theorem 5.7. □

Proposition 6.18 (Finite out-depth implies local supercycle-violation) *For $d < \ell$: $(\mathit{out\_depth}_{D_{\mathsf{a}}^{\ell}}(v, t_{\mathsf{a}}) = d) \Rightarrow \mathsf{scViolateLoc}(v, d+1, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$.*

Proof. Proof is by induction on d.

<u>Base case, $d = 0$.</u> Then v has no outgoing wait-for edges. Hence $\mathsf{scViolateLoc}(v, 1, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ by Definition 6.5.

<u>Induction step, $d > 0$.</u> Assume $(\mathit{out\_depth}_{D_{\mathsf{a}}^{\ell}}(v, t_{\mathsf{a}}) = d)$. Then, every outgoing wait-for edge of v is to some v' such that $(\mathit{out\_depth}_{D_{\mathsf{a}}^{\ell}}(v', t_{\mathsf{a}}) = d' < d)$. By the induction

hypothesis, $\mathsf{scViolateLoc}(v', d' + 1, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence, by Definition 6.5, $\mathsf{scViolateLoc}(v, d + 1, t_\mathsf{a}, D^\ell_\mathsf{a})$. □

LEMMA 6.19. *Let* a *be an interaction of* B*, i.e.,* $\mathsf{a} \in \gamma$*. Then* $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ *implies* $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$.

Proof. Assume $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. Let $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ be an arbitrary reachable transition of D^ℓ_a, and let B_i be an arbitrary component of $components(\mathsf{a})$. Then, from Definition 6.14, we have:

$$in\_depth_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1 \vee out\_depth_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1.$$

The proof proceeds by two cases.

$\underline{in\_depth_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1}$: 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 $\mathsf{sConnViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$ by Definition 6.7, Clause 1. Hence by Definition 6.9, $\mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$.

$\underline{out\_depth_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1}$: Hence $out\_depth_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) = d$ for some $d < 2\ell - 1$. By Proposition 6.18, $\mathsf{scViolateLoc}(\mathsf{B}_i, d + 1, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence by Definition 6.9, $\mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$.

In both cases, we have $\mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$. Since B_i is an arbitrarily chosen component of $components(\mathsf{a})$, we have $\forall \mathsf{B}_i \in components(\mathsf{a}) : \mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence, by Definition 6.11, we conclude $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. □

7. 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.

$$\begin{tikzcd}
\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a}) \arrow[r, "\text{Lemma 5.6}"] & \mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a}) \\
\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell) \arrow[u, "\text{Lemma 6.16}"] \arrow[r, "\text{Lemma 6.19}"] & \mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell) \arrow[u, "\text{Lemma 6.12}"]
\end{tikzcd}$$

Fig. 9. Implication relations between deadlock-freedom conditions

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 7.1 (Deadlock-freedom via $\mathcal{GALT}$, $\mathcal{GLIN}$, $\mathcal{LALT}$, $\mathcal{LLIN}$) *Assume that*

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

(*a*) $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$
(*b*) $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$
(*c*) $\exists \ell > 0 : \mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
(*d*) $\exists \ell > 0 : \mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$

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

Proof. Immediate from Theorems 5.2, 5.7, 6.13, 6.17 and Corollary 4.13. □

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 7.2 (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., $\mathsf{a} \in \gamma$), $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds.*

Proof. Let a be an arbitrary interaction in γ, and let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0). Hence t is a reachable state of (B, Q_0). Suppose that $W_{\mathsf{B}}(t)$ contains a supercycle SC. Then, by Proposition 3.7, the subcomponent B' consisting of all the atomic components $B_i \in SC$ cannot execute a transition from any state reachable from t, and so is deadlocked. Hence (B, Q_0) has a local deadlock in reachable state t, contrary to assumption. Hence $W_{\mathsf{B}}(t)$ is supercycle-free.

Let v be an arbitrary node in $W_{\mathsf{B}}(t)$. By Definition 4.1, $\neg scyc_{\mathsf{B}}(s, v)$ holds. Hence by Proposition 4.5, $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ holds. By Definition 4.9, $\mathsf{genViolate}_{\mathsf{B}}(v, t)$ holds. Since v is an arbitrary node in $W_{\mathsf{B}}(t)$, and all $\mathsf{B}_i \in components(\mathsf{a})$ are nodes in $W_{\mathsf{B}}(t)$, we have $(\forall\, \mathsf{B}_i \in components(\mathsf{a}), \mathsf{genViolate}_{\mathsf{B}}(\mathsf{B}_i, t))$. By Definition 5.1, $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds. Since a is an arbitrary interaction in γ, we have $(\forall\, \mathsf{a} \in \gamma : \mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a}))$, and the theorem is established. □

8. IMPLEMENTATION AND EXPERIMENTS

8.1. Checking that initial states are supercycle-free

Our deadlock-freedom theorem require that all initial states be sueprcycle-free. We assume that the number of initial states is small, so that we can check each explicitly.

```
checkInitSupercycleFree(Q_0)
▷ returns true iff all initial states are supercycle-free
forall s_0 ∈ Q_0
    compute W_B(s_0)
    let U be the result of removing from W_B(s_0) all nodes v such that (∃ d ≥ 1 : scViolate_B(v, d, t))
    if (U is nonempty) then return(ff)    ▷ s_0 not supercycle-free, so return false
else return(tt)
```

Fig. 10. Procedure to check that all initial states are supercycle-free

PROPOSITION 8.1. CHECKINITSUPERCYCLEFREE(Q_0) *returns true iff all initial states are supercycle-free.*

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

Suppose that U is nonempty. By Proposition 4.5, U is a supercycle. Since $U \subseteq W_{\mathsf{B}}(s_0)$, we conclude that s_0 not supercycle-free, so false is the correct result in this case.

Now suppose that U is empty. Hence every node in $W_{\mathsf{B}}(s_0)$ has a supercycle violation, and so by Proposition 4.4, no node of $W_{\mathsf{B}}(s_0)$ can be in a strongly-connected supercycle. Hence $W_{\mathsf{B}}(s_0)$ does not contain a strongly-connected supercycle. So, by Proposition 3.16, $W_{\mathsf{B}}(s_0)$ does not contain a supercycle. □

8.2. Implementation of the Linear Condition

LLIN(B, Q_0) iterates over each interaction a of (B, Q_0), and checks $(\exists \ell > 0 : \mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell))$ by starting with $\ell = 1$ and incrementing ℓ until either $\mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell)$ is found to hold, or D^ℓ_{a} has become the entire system and $\mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell)$ does not hold. In the latter case, $\mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell)$ does not hold for any finite ℓ, and, in practice, computation would halt before D^ℓ_{a} had become the entire system, due to exhaustion of resources.

LLININTDIST$(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ checks $\mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell)$ by examining every reachable transition that executes a, and checking that the final state satisfies Definition 6.14.

LLIN(B, Q_0), where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$
forall interactions $\mathsf{a} \in \gamma$
    if (LLININT$(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{ff}$) **return**($\mathtt{ff}$) **fi**
endfor;
return($\mathtt{tt}$) ▷ return $\mathtt{tt}$ if check succeeds for all $\mathsf{a} \in \gamma$

LLININT$(\mathsf{B}, Q_0, \mathsf{a})$, where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n), \mathsf{a} \in \gamma$
▷ check $(\exists \ell > 0 : \mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell))$
$\ell \leftarrow 1$; ▷ start with $\ell = 1$
while ($\mathtt{tt}$)
    if (LLININTDIST$(\mathsf{a}, \ell) = \mathtt{tt}$) **return**($\mathtt{tt}$) **fi**; ▷ success, so return true
    if ($D^\ell_{\mathsf{a}} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$) **return**($\mathtt{ff}$) **fi**; ▷ exhausted all subsystems, return false
    $\ell \leftarrow \ell + 1$ ▷ increment ℓ until success or intractable or failure
endwhile

LLININTDIST$(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
forall reachable transitions $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ of D^ℓ_{a}
    if $(\neg(\forall \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathit{in\_depth}_{D^\ell_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1 \lor \mathit{out\_depth}_{D^\ell_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) < 2\ell - 1))$
        return($\mathtt{ff}$) ▷ check Definition 6.14
    fi
endfor;
return($\mathtt{tt}$) ▷ return $\mathtt{tt}$ if check succeeds for all transitions

Fig. 11. Pseudocode for the implementation of the linear condition.

Complexity. The running time of our implementation is also $O(\Sigma_{a \in \gamma} |M^{\ell_a}_{\mathsf{a}}| * |D^{\ell_a}_{\mathsf{a}}|)$, where ℓ_a is the smallest value of ℓ for which $\mathcal{LLIN}(B, Q_0, \mathsf{a}, \ell)$ holds, and where $|D^{\ell_a}_{\mathsf{a}}|$, and $|M^{\ell_a}_{\mathsf{a}}|$ are as above.

LALT(B, Q_0)	true iff $(\forall\, \mathsf{a} \in \gamma, \exists\, \ell > 0 : \mathcal{LALT}(B, Q_0, \mathsf{a}, \ell))$
LALTINT($\mathsf{B}, Q_0, \mathsf{a}$)	true iff $(\exists\, \ell > 0 : \mathcal{LALT}(B, Q_0, \mathsf{a}, \ell))$
LALTINTDIST($\mathsf{B}, Q_0, \mathsf{a}, \ell$)	true iff $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
LOCFORMVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$)	true iff B_i has local sc-formation violation in state t_a of D^ℓ_a, i.e., $\mathsf{genViolateLoc}(B_i, t_\mathsf{a}, D^\ell_\mathsf{a})$ holds
LOCSCONNSCVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$)	true iff B_i has local strong connectedness violation in t_a, i.e., $\mathsf{sConnViolateLoc}(B_i, t_\mathsf{a}, D^\ell_\mathsf{a})$ holds
LOCSCVIOL($D^\ell_\mathsf{a}, t_\mathsf{a}$)	compute local supercycle violations in state t_a of D^ℓ_a, i.e., $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$ for all v, d

Fig. 12. Summary of procedures

8.3. Implementation of the AND-OR Condition

Our implementation evaluates $\mathcal{LALT}$. Figure 13 presents the pseudocode, and Figure 14 presents the pseudocode for computing supercycle violations based on D^ℓ_a.

LALT(B, Q_0) verifies $\mathcal{LALT}$ by iterating over all $\mathsf{a} \in \gamma$. LALTINT($\mathsf{B}, Q_0, \mathsf{a}$) checks $(\exists\, \ell > 0 : \mathcal{LALT}(B, Q_0, \mathsf{a}, \ell))$, i.e., if $\mathcal{LALT}$ for a can be verified in some D^ℓ_a. We start with $\ell = 1$ since D^1_a is the smallest system, in which a supercycle-violation can be confirmed. LALTINTDIST($\mathsf{B}, Q_0, \mathsf{a}, \ell$) checks $\mathcal{LALT}(B, Q_0, \mathsf{a}, \ell)$ for a particular ℓ. Figure 12 shows a summary of the procedures.

Complexity. The running time of our implementation is $O(\Sigma_{a \in \gamma}\, |M^{\ell_a}_\mathsf{a}| * |D^{\ell_a}_\mathsf{a}|)$, where $M^{\ell_a}_\mathsf{a}$ is the transition system of $D^{\ell_a}_\mathsf{a}$, and $|M^{\ell_a}_\mathsf{a}|$ is the size (number of nodes plus number of edges) of $M^{\ell_a}_\mathsf{a}$, $|D^{\ell_a}_\mathsf{a}|$ is the size of the syntactic description of $D^{\ell_a}_\mathsf{a}$, and ℓ_a is the smallest value of ℓ for which $\mathcal{LALT}(B, Q_0, \mathsf{a}, \ell)$ holds.

$\textsc{LAlt}(\mathsf{B}, Q_0)$, where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$
$\triangleright$ returns $\mathtt{tt}$ iff $(\forall\, \mathsf{a} \in \gamma, \exists\, \ell > 0 : \mathcal{LALT}(\mathsf{a}, \ell))$
1. **forall** interactions $\mathsf{a} \in \gamma$
2.   **if** $(\textsc{LAltInt}(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{ff})$ **return**$(\mathtt{ff})$ **fi**
3. **endfor**;
4. **return**$(\mathtt{tt})$ $\triangleright$ return $\mathtt{tt}$ if check succeeds for all $a \in \gamma$

$\textsc{LAltInt}(\mathsf{B}, Q_0, \mathsf{a})$, where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n), \mathsf{a} \in \gamma$
$\triangleright$ returns $\mathtt{tt}$ iff $(\exists\, \ell > 0 : \mathcal{LALT}(B, Q_0, \mathsf{a}, \ell))$
1. $\ell \leftarrow 1$; $\triangleright$ start with $\ell = 1$
2. **while** $(\mathtt{tt})$
3.   **if** $(\textsc{LAltIntDist}(\mathsf{a}, \ell) = \mathtt{tt})$ **return**$(\mathtt{tt})$ **fi**; $\triangleright$ success, so return true
4.   **if** $(D_{\mathsf{a}}^{\ell} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n))$ **return**$(\mathtt{ff})$ **fi**; $\triangleright$ exhausted all subsystems, return false
5. $\ell \leftarrow \ell + 1$ $\triangleright$ increment ℓ until success or intractable or failure
6. **endwhile**

$\textsc{LAltIntDist}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
$\triangleright$ returns $\mathtt{tt}$ iff $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
1. **forall** reachable transitions $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ of D_{a}^{ℓ}
2.   **forall** $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$
3.   **if** $\neg\textsc{locFormViol}(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$ **then return**$(\mathtt{ff})$ **fi** $\triangleright$return $\mathtt{ff}$ if no violation for B_i
4.   **endfor**
5. **endfor**;
6. **return**$(\mathtt{tt})$ $\triangleright$return $\mathtt{tt}$ if all $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ violate local supercycle formation

$\textsc{locFormViol}(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$
$\triangleright$returns true iff $\mathsf{genViolateLoc}(B_i, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ holds (Definition 6.9)
$\triangleright$i.e., B_i has a local supercycle formation violation in state t_{a} of subsystem D_{a}^{ℓ}
1. $\textsc{LocScViol}(D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$
2. **return**$(V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[B_i] \vee \textsc{locSconnScViol}(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}))$

$\textsc{locSconnScViol}(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$
$\triangleright$returns true iff $\mathsf{sConnViolateLoc}(B_i, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ holds (Definition 6.7)
$\triangleright$i.e., B_i has a local strong connectedness supercycle formation violation in state t_{a} of subsystem D_{a}^{ℓ}
1. remove all nodes with local supercycle violation
2. compute maximal strongly connected components of remaining wait-for graph
3. forall maximal strongly connected components C
4.   if C contains a non-trivial strongly connected supercycle which contains B_i as a node
5.   **then return**$(\mathtt{ff})$ **fi** $\triangleright$Definition 6.7, Clause 1 holds here
6. forall wait-for paths π from B_i to the border of D_{a}^{ℓ}
7.   if some node of π has a local supercycle violation **then return**$(\mathtt{tt})$ **fi**$\triangleright$Clause 2a holds
8. forall wait-for paths π' from the border of D_{a}^{ℓ} to B_i
9.   if some node of π' has a local supercycle violation **then return**$(\mathtt{tt})$ **fi**$\triangleright$Clause 2b holds
10. **return**$(\mathtt{ff})$ $\triangleright$Definition 6.7, Clause 2 does not hold

Fig. 13. Pseudocode for the implementation of the local AND-OR condition.

```
> 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. 15. LALT-BIP Command Line Interface

LocScViol($D^\ell_{\mathsf{a}}, t_{\mathsf{a}}$)
▷ compute supercycle violations in state t_{a} of D^ℓ_{a}
▷ Postcondition: $\forall v \in D^\ell_{\mathsf{a}} : V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[v] = \begin{cases} \mathtt{tt} & \text{if } \exists d \geq 1 : \mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) \\ \mathtt{ff} & \text{otherwise} \end{cases}$
1. $foundScViolate \leftarrow \mathtt{ff}$

2. **forall** $v \in D^\ell_{\mathsf{a}}$
3. **if** (v is an interior interaction aa and $\neg(\exists\, \mathsf{B}_i : \mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}}))$)
4. $V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[v] \leftarrow \mathtt{tt}$ ▷Base case: interaction with no outgoing wait-for-edges
5. $foundScViolate \leftarrow \mathtt{tt}$
6. **fi**
7. **endfor**

8. **while** ($foundScViolate$)
9. $foundScViolate \leftarrow \mathtt{ff}$
10. **forall** $v \in D^\ell_{\mathsf{a}} : \neg V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[v]$
11. **if** (v is an interior interaction aa and ($\forall\, \mathsf{B}_i : \mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}}) : V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[B_i]$))
12. $V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[v] \leftarrow \mathtt{tt}$
13. $foundScViolate \leftarrow \mathtt{tt}$
14. **else if** (v is a component B_i and ($\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}}) : V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[\mathsf{aa}]$))
15. $V_{D^\ell_{\mathsf{a}},t_{\mathsf{a}}}[vd] \leftarrow \mathtt{tt}$
16. $foundScViolate \leftarrow \mathtt{tt}$
17. **fi**
18. **endfor**
19. **endwhile**

Fig. 14. Procedure to compute all supercycle-violations in state t_{a} of D^ℓ_{a}

8.4. Tool-set

We provide LALT-BIP, a suite of supporting tools that implement our method. LALT-BIP is around ~ 2500 Java LOC. LALT-BIP is equipped with a command line interface (see Figure 15) that accepts a set of configuration options. It takes the name of the input BIP file and other optional flags.

8.5. 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 config-

Table I. Benchmarks: Dining Philosopher

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

urations of our resource allocation 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 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.

8.5.1. Dining philosophers case study. We consider the traditional dining philosopher problem as depicted in Figure 1. The Figure shows n philosophers competing on n forks modeled in BIP.

Each philosopher component has 2 states, and each fork component has 3 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 [Bensalem et al. 2011]. 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 Bensalem et al. [2011].

Table I 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 $\mathcal{LALT}$ successfully completed the deadlock freedom check for $3,000 \leq n \leq 10,000$ in less than one minute, where DFinder2 timed out (1 Hour). $\mathcal{LLIN}$ required 62 seconds for $n = 10,000$.

Even though $\mathcal{LLIN}$ is asymptotically more efficient than $\mathcal{LALT}$, $\mathcal{LALT}$ outperforms $\mathcal{LLIN}$ in all cases. This 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 effectively more efficient than $\mathcal{LLIN}$.

8.5.2. 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. The system allows 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 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 16, 17, and 18 show BIP atomic components for client, resource and manager components.

The client in Figure 16 requests resources R_0 and R_2 in sequence. It has 5 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.

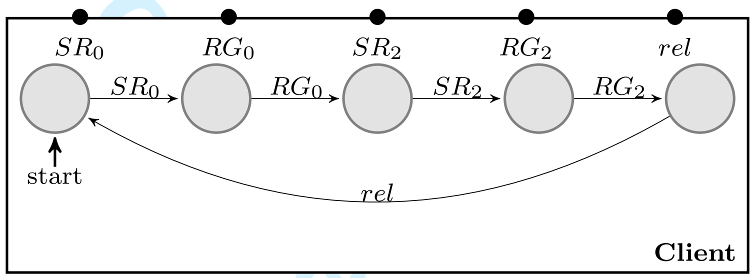


Fig. 16. Client

Figure 17 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.

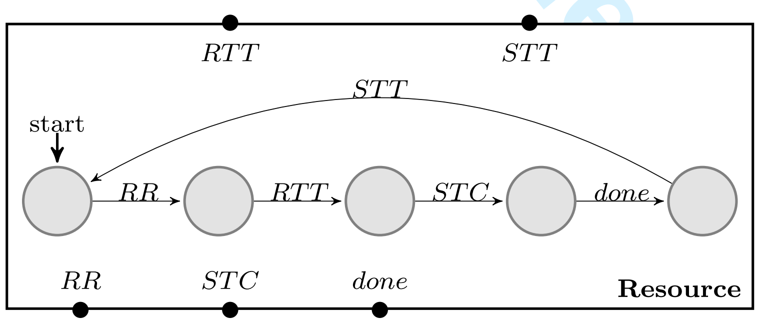


Fig. 17. Resource

Figure 18 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.
— 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 can not 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.

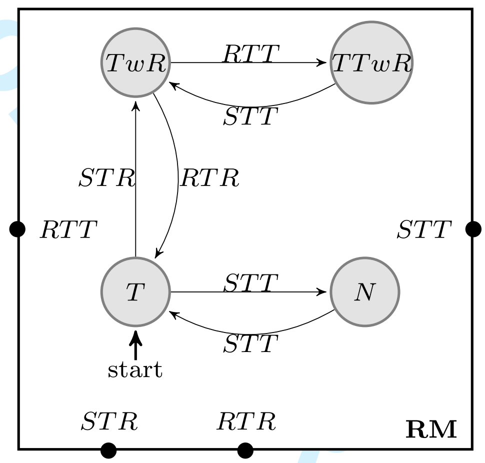


Fig. 18. Token Resource Manager

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 19 shows a configuration system with 5 clients and 5 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} managers conflicting resources $\{R_2, R_3\}$. RM_4 manages resource R_4.

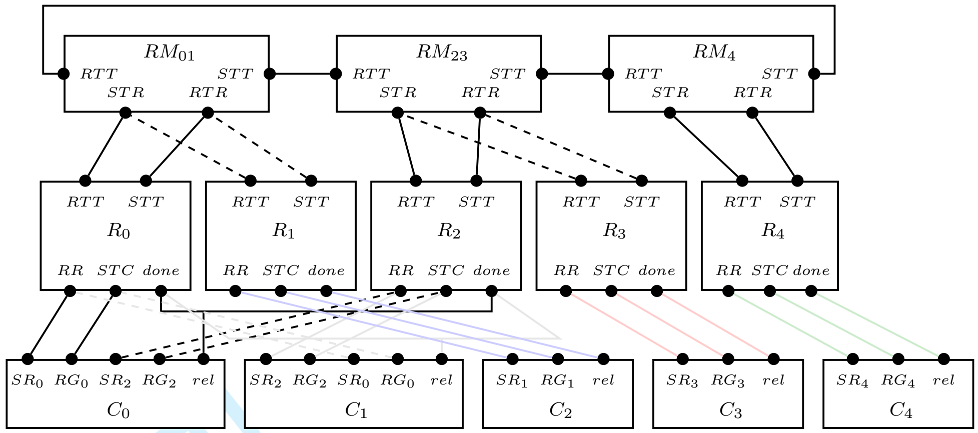


Fig. 19. Conflict-Resource Allocation System

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 5 clients, 3 tokens, and 5 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 clearly is a 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.

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 5 clients, 2 tokens, and 5 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.

Lesson 3:. Our work can be extended to detect conspiracies [Attie et al. 1993]. For example, consider a system with 5 clients, 2 tokens, and 5 resources. Clients request resources $\langle 0,1\rangle, \langle 1,0\rangle, \langle 2\rangle, \langle 3\rangle$, and $\langle 4\rangle$, respectively. Resource sets $\{0,1\}, \{2,3,4\}$ are conflicting. Client C_0 may block forever in case it acquires resource 0 because resource 0 is conflicting with resource 1. However, it is not possible to find a deadlocked subsystem containing C_0 and resources 0 and 1 since that will also have to include the resource manager M_{01} managing conflicting resources 0 and 1. The latter can always exchange the second token with the neighboring resource managers.

An extension of our work that consider subsystem boundaries at ports and abstracts port enablement conditions with free Boolean variables can help detect such scenarios.

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

Table II. 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

— n clients each with 3 states, n resources each with 5 states, and n tokens,
— Client $C_i, 0 \leq i < n$ requests resource i, and
— No resources are in conflict, hence we have n resource managers each with 4 states.

The system has a total of $4^n \times 3^n \times 5^n$ states. DFinder2 timed out within seven hours for $n = 10$. $\mathcal{LLIN}$ had to increase the subsystem up to the whole system and also timed out within seven hours 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 $\leq 23,040,000 = 4^6 \times 3^2 \times 5^4$ states regardless of n. The numbers in Table II 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 [Milner 1989] which is essentially a token ring with precisely one token present [Antonino et al. 2016]. [Antonino et al. 2016] present a technique that fails to prove deadlock freedom for Milner Scheduler because it requires a large subset of the system, while $\mathcal{LALT}$ succeeds.

9. DISCUSSION, RELATED WORK, AND FURTHER WORK

9.1. Related work.

The notions of wait-for-graph and supercycle [Attie and Chockler 2005; Attie and Emerson 1998] were initially defined for a shared memory program $P = P_1 \parallel \cdots \parallel P_K$ in *pairwise normal form* [Attie 2016b; Attie 2016a]: 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 [2005] 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 [2003] 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 [2012] 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 [2003] 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 [1987] 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 2 and 3 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 [1991] also provide criteria for deadlock freedom of triple-disjoint CSP programs, and use the same technical framework as Roscoe and Dathi [1987]. 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 Roscoe and Dathi [1987]. 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 [1996] applies the results of Roscoe and Dathi [1987] and Brookes and Roscoe [1991] 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. [2016] 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 $\mathcal{LALT}(B, Q_0, a, \ell)$ 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 Antonino et al. [2016] 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 [Bensalem et al. 2011]. 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 Attie and Chockler [2005]) 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 7.2. None of the above papers give a completeness result similar to Theorem 7.2. 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 (Prop. 4.6, Def. 4.9, and Def. 5.1).

9.2. 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 [Bonakdarpour et al. 2010].
— *Locality*: if a component B_i is modified, or is added to an existing system, then $\mathcal{LALT}(B, Q_0, a, \ell)$ only has to be re-checked for B_i and components within distance ℓ of B_i. A condition whose evaluation considers the entire system at once, e.g., [Aldini and Bernardo 2003; Bensalem et al. 2011; Gössler and Sifakis 2003] would have to be re-checked for the entire system.
— *Easily parallelizable*: since the checking of each subsystem D_a^ℓ 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 [2005] 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, etc.

9.3. Further work.

Our implementation uses explicit state enumeration. Using BDD's may improve the running time when $\mathcal{LALT}(B, Q_0, a, \ell)$ holds only for large ℓ. Another potential method for improving the running time is to use SAT solving, cf. Antonino et al. [2016]. 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 $\mathcal{LALT}(B, Q_0, a, \ell)$. These can be used to manually modify the system to eliminate a possible deadlock. Also, when $\mathcal{LALT}(B, Q_0, a, \ell)$ 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 adds 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 $\mathcal{LALT}(B, Q_0, a, \ell)$ 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 $W_B(s)$ that can be made into a supercycle by adding f edges. We will investigate criteria for preventing formation of such subgraphs. Methods for evaluating $\mathcal{LALT}(B, Q_0, a, \ell)$ 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*, [Bozga et al. 2012], where participants can add and remove interactions at run time.

REFERENCES

Alessandro Aldini and Marco Bernardo. 2003. A General Approach to Deadlock Freedom Verification for Software Architectures. *FME* 2805 (2003), 658–677.

Pedro Antonino, Thomas Gibson-Robinson, and A. W. Roscoe. 2016. Efficient Deadlock-Freedom Checking Using Local Analysis and SAT Solving. In *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*. 345–360. DOI:http://dx.doi.org/10.1007/978-3-319-33693-0_22

P.C. Attie, N. Francez, and O. Grumberg. 1993. Fairness and Hyperfairness in Multiparty Interactions. *Distributed Computing* 6 (1993), 245–254.

Paul C. Attie. 2016a. Finite-state concurrent programs can be expressed in pairwise normal form. *Theoretical Computer Science* 619 (2016), 1 – 31. DOI:http://dx.doi.org/10.1016/j.tcs.2015.11.032

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

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 *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 2013, Florence, Italy, June 3-5, 2013. Proceedings*. 161–177. DOI:http://dx.doi.org/10.1007/978-3-642-38592-6_12

Paul C. Attie and H. Chockler. 2005. Efficiently Verifiable Conditions for Deadlock-freedom of Large Concurrent Programs. In *VMCAI*. France.

Paul C. Attie and E. Allen Emerson. 1998. Synthesis of Concurrent Systems with Many Similar Processes. *TOPLAS* 20, 1 (Jan. 1998), 51–115.

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*. 453–458.

Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. 2010. From High-level Component-based Models to Distributed Implementations. In *EMSOFT*. 209–218.

Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. 2012. Modeling Dynamic Architectures Using Dy-BIP. In *Software Composition*. 1–16.

S.D. Brookes and A.W. Roscoe. 1991. Deadlock analysis in networks of communicating processes. *Distributed Computing* 4 (1991), 209–230. Issue 4.

Gregor Gössler and Joseph Sifakis. 2003. Component-based Construction of Deadlock-free Systems. In *FSTTCS*. Springer, 420–433.

Moritz Martens and Mila Majster-Cederbaum. 2012. Deadlock-freedom in component systems with architectural constraints. *FMSD* 41 (2012), 129–177. Issue 2. DOI:http://dx.doi.org/10.1007/s10703-012-0160-6

Jeremy Malcolm Randolph Martin. 1996. *The Design and Construction of Deadlock-Free Concurrent Systems*. Ph.D. Dissertation. The University of Buckingham.

Robin Milner. 1989. *Communication and concurrency*. Prentice Hall.

Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley.

A.W. Roscoe and Naiem Dathi. 1987. The pursuit of deadlock freedom. *Information and Computation* 75, 3 (1987), 289 – 327. DOI:http://dx.doi.org/10.1016/0890-5401(87)90004-6

Differences between the conference version and the submission.

- The conference version gives a restricted "linear" criterion for local and global deadlock freedom which is *not* complete for local and global deadlock-freedom. The submission gives an "alternating" AND-OR criterion, which is complete for local and global deadlock freedom. The linear criterion can fail in cases where the alternating criterion succeeds in verifying deadlock freedom. The linear condition is actually a special case of the AND-OR condition.

- The submission provides new results concerning the graph-theoretic properties of the waiting patterns that constitute a local or global deadlock.

- Submission provides experiments that are new and different from the conference version, and which, among other results, give an example where the linear criterion fails while the AND-OR criterion succeeds. Experiments also deal with more challenging examples than in the conference version, including a generalization of Milner's token-based scheduler.

An Abstract Framework for Deadlock Prevention in BIP*

Paul C. Attie[1], Saddek Bensalem[2], Marius Bozga[2], Mohamad Jaber[1], Joseph Sifakis[3], and Fadi A. Zaraket[4]

[1] Department of Computer Science, American University of Beirut, Beirut, Lebanon
[2] UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France
[3] Rigorous System Design Laboratory, EPFL, Lausanne, Switzerland
[4] Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon

Abstract. We present a sound but incomplete criterion for checking deadlock freedom of finite state systems expressed in BIP: a component-based framework for the construction of complex distributed systems. Since deciding deadlock-freedom for finite-state concurrent systems is PSPACE-complete, our criterion gives up completeness in return for tractability of evaluation. Our criterion can be evaluated by model-checking subsystems of the overall large system. The size of these subsystems depends only on the local topology of direct interaction between components, and *not* on the number of components in the overall system.

We present two experiments, in which our method compares favorably with existing approaches. For example, in verifying deadlock freedom of dining philosphers, our method shows linear increase in computation time with the number of philosophers, whereas other methods (even those that use abstraction) show super-linear increase, due to state-explosion.

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 correctness 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 [15, chapter 19]. To achieve tractability, we can either make our deadlock freedom check incomplete (sufficient but not necessary), or we can restrict the systems that we check to special cases. We choose the first option: a system meeting our condition is free of both local and global deadlocks, while a system which fails to meet our condition may or may not be deadlock free.

* The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 288175 (CERTAINTY) and no 257414 (ASCENS).

D. Beyer and M. Boreale (Eds.): FMOODS/FORTE 2013, LNCS 7892, pp. 161–177, 2013.

We generalize previous works [2–4] by removing the requirement that interaction between processes be expressed pairwise, and also by applying to BIP [6], a framework from which efficient distributed code can be generated. In contrast, the model of concurrency in [2–4] requires shared memory read-modify-write operations with a large grain of atomicity. The full paper, including proofs for all theorems, is available on-line, as is our implementation of the method.

2 BIP – Behavior Interaction Priority

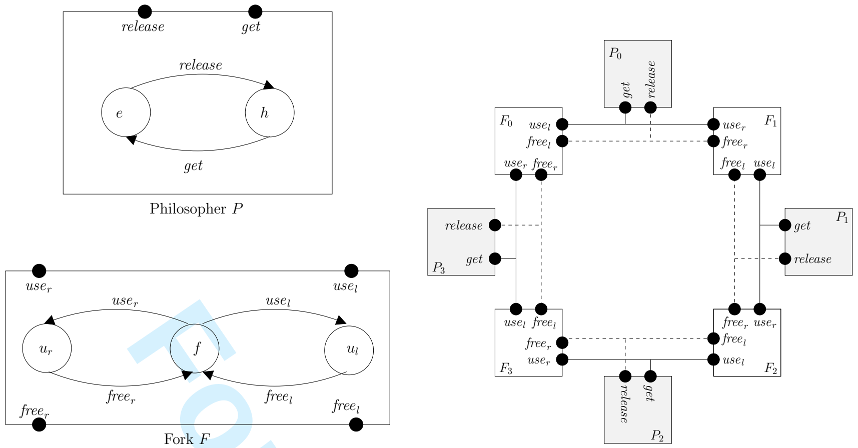
BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. A technical treatment of priority is beyond the scope of this paper. Adding priorities never introduces a deadlock, since priority enforces a choice between possible transitions 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 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* possible transitions, *each labeled by some port.*

For states $s_i, t_i \in Q_i$ and port $p_i \in P_i$, write $s_i \xrightarrow{p_i}_i t_i$, iff $(s_i, p_i, t_i) \in \rightarrow_i$. When p_i is irrelevant, write $s_i \rightarrow_i t_i$. Similarly, $s_i \xrightarrow{p_i}_i$ means that there exists $t_i \in Q_i$ such that $s_i \xrightarrow{p_i}_i t_i$. In this case, p_i is *enabled* in state s_i. Ports are used for communication between different components, as discussed below.

In practice, we describe the transition system using some syntax, e.g., involving variables. We abstract away from issues of syntactic description since we are only interested in enablement of ports and actions. We assume that enablement of a port depends only on the local state of a component. In particular, it cannot depend on the state of other components. This is a restriction on BIP, and we defer to subsequent work how to lift this restriction. So, we assume the existence of a predicate $enb^i_{p_i}$ that holds in state s_i of component B_i iff port p_i is enabled in s_i, i.e., $s_i(enb^i_{p_i}) = true$ iff $s_i \xrightarrow{p_i}_i$.

Figure 1(a) shows atomic components for a philospher P and a fork F in dining philosophers. A philosopher P that is hungry (in state h) can eat by executing *get* and moving to state e (eating). From e, P releases its forks by executing *release* and moving back to h. Adding the thinking state does not change the deadlock behaviour of the system, since the thinking to hungry transition is internal to P, and so we omit it. A fork F is taken by either: (1) the left philosopher (transition get_l) and so moves to state u_l (used by left philosopher), or (2) the right philosopher (transition get_r) and so moves to state u_r (used by right philosopher). From state u_r (resp. u_l), F is released by the right philosopher (resp. left philosopher) and so moves back to state f (free).



(a) Philosopher P and fork F atomic components.

(b) Dining philosophers composite component with four philosophers.

Fig. 1. Dining philosophers

Definition 2 (Interaction). *For a given system built from a set of n atomic components $\{B_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^{n}$, we require that their respective sets of ports are pairwise disjoint, i.e., for all i, j such that $i, j \in \{1..n\} \wedge i \neq j$, 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 $a \subseteq P \wedge (\forall i \in \{1..n\} : |a \cap P_i| \leq 1)$, where $P = \bigcup_{i=1}^{n} P_i$ is the set of all ports in the system. When we write $a = \{p_i\}_{i \in I}$, we assume that $p_i \in P_i$ for all $i \in I$, where $I \subseteq \{1..n\}$.*

Execution of an interaction a involves all the components which have ports in a.

Definition 3 (Composite Component). *A* composite component *(or simply* component*) $B \triangleq \gamma(B_1, \ldots, B_n)$ 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*

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

This inference rule says that a composite component $B = \gamma(B_1, \ldots, B_n)$ can execute an interaction $a \in \gamma$, iff for each port $p_i \in a$, 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. Given an interaction $a = \{p_i\}_{i \in I}$, we denote by C_a the set of atomic components participating in a, formally: $C_a = \{B_i \mid p_i \in a\}$. Figure 1(b) shows a composite component consisting of four philosophers and the four forks between them. Each philosopher and its two neighboring forks share two interactions: $Get = \{get, use_l, use_r\}$ in which the philosopher obtains the forks, and $Rel = \{release, free_l, free_r\}$ in which the philosopher releases the forks.

Definition 4 (Interaction enablement). *An atomic component $B_i = (Q_i, P_i, \rightarrow_i)$ enables interaction a in state s_i iff $s_i \xrightarrow{p_i}_i$, where $p_i = P_i \cap a$ is the port of B_i involved in a. Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component, and let $s = \langle s_1, \ldots, s_n \rangle$ be a state of B. Then B enables a in s iff every $B_i \in C_a$ enables a in s_i.*

The definition of interaction enablement is a consequence of Definition 3. Interaction a being enabled in state s means that executing a is one of the possible transitions that can be taken from s. Let enb_a^i denote the enablement condition for interaction a in component B_i. By definition, $enb_a^i = enb_{p_i}^i$ where $p_i = a \cap P_i$.

Definition 5 (BIP System). *Let $B = \gamma(B_1, \ldots, B_n)$ 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).

Definition 6 (Execution). *Let (B, Q_0) be a BIP system with transition system $(Q, \gamma, \rightarrow)$. Let $\rho = s_0 a_1 s_1 \ldots s_{i-1} a_i s_i \ldots$ 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) $\forall i > 0 : s_{i-1} \xrightarrow{a_i} s_i$.*

A state or transition that occurs in some execution is called *reachable*.

Definition 7 (State Projection). *Let (B, Q_0) be a BIP system where $B = \gamma(B_1, \ldots, B_n)$ and let $s = \langle s_1, \ldots, s_n \rangle$ be a state of (B, Q_0). Let $\{B_{j_1}, \ldots, B_{j_k}\} \subseteq \{B_1, \ldots, B_n\}$. Then $s \restriction \{B_{j_1}, \ldots, B_{j_k}\} \triangleq \langle s_{j_1}, \ldots, s_{j_k} \rangle$. For a single B_i, we write $s \restriction B_i = s_i$. We extend state projection to sets of states element-wise.*

Definition 8 (Subcomponent). *Let $B \triangleq \gamma(B_1, \ldots, B_n)$ be a composite component, and let $\{B_{j_1}, \ldots, B_{j_k}\}$ be a subset of $\{B_1, \ldots, B_n\}$. Let $P' = P_{j_1} \cup \cdots \cup P_{j_k}$, i.e., the union of the ports of $\{B_{j_1}, \ldots, B_{j_k}\}$. Then the subcomponent B' of B based on $\{B_{j_1}, \ldots, B_{j_k}\}$ is as follows:*

1. $\gamma' \triangleq \{a \cap P' \mid a \in \gamma \wedge a \cap P' \neq \emptyset\}$
2. $B' \triangleq \gamma'(B_{j_1}, \ldots, B_{j_k})$

That is, γ' consists of those interactions in γ that have at least one participant in $\{B_{j_1}, \ldots, B_{j_k}\}$, and restricted to the participants in $\{B_{j_1}, \ldots, B_{j_k}\}$, i.e., participants not in $\{B_{j_1}, \ldots, B_{j_k}\}$ are removed.

We write $s \restriction B'$ to indicate state projection onto B', and define $s \restriction B' \triangleq s \restriction \{B_{j_1}, \ldots, B_{j_k}\}$, where $B_{j_1}, \ldots, B_{j_k}$ are the atomic components in B'.

Definition 9 (Subsystem). *Let* (B, Q_0) *be a BIP system where* $B = \gamma(B_1, \ldots, B_n)$*, and let* $\{B_{j_1}, \ldots, B_{j_k}\}$ *be a subset of* $\{B_1, \ldots, B_n\}$*. Then the subsystem* (B', Q'_0) *of* (B, Q_0) *based on* $\{B_{j_1}, \ldots, B_{j_k}\}$ *is as follows:*

1. B' *is the subcomponent of* B *based on* $\{B_{j_1}, \ldots, B_{j_k}\}$
2. $Q'_0 = Q_0 \upharpoonright \{B_{j_1}, \ldots, B_{j_k}\}$

Definition 10 (Execution Projection). *Let* (B, Q_0) *be a BIP system where* $B = \gamma(B_1, \ldots, B_n)$*, and let* (B', Q'_0)*, with* $B' = \gamma'(B_{j_1}, \ldots, B_{j_k})$ *be the subsystem of* (B, Q_0) *based on* $\{B_{j_1}, \ldots, B_{j_k}\}$*. Let* $\rho = s_0 a_1 s_1 \ldots s_{i-1} a_i s_i \ldots$ *be an execution of* (B, Q_0)*. Then,* $\rho \upharpoonright (B', Q'_0)$*, the projection of* ρ *onto* (B', Q'_0)*, is the sequence resulting from:*

1. *replacing each* s_i *by* $s_i \upharpoonright \{B_{j_1}, \ldots, B_{j_k}\}$*, i.e., replacing each state by its projection onto* $\{B_{j_1}, \ldots, B_{j_k}\}$
2. *removing all* $a_i s_i$ *where* $a_i \notin \gamma'$

Proposition 1 (Execution Projection). *Let* (B, Q_0) *be a BIP system where* $B = \gamma(B_1, \ldots, B_n)$*, and let* (B', Q'_0)*, with* $B' = \gamma'(B_{j_1}, \ldots, B_{j_k})$ *be the subsystem of* (B, Q_0) *based on* $\{B_{j_1}, \ldots, B_{j_k}\}$*. Let* $\rho = s_0 a_1 s_1 \ldots s_{i-1} a_i s_i \ldots$ *be an execution of* (B, Q_0)*. Then,* $\rho \upharpoonright (B', Q'_0)$ *is an execution of* (B', Q'_0)*.*

Corollary 1. *Let* (B', Q'_0) *be a subsystem of* (B, Q_0)*. Let* s *be a reachable state of* (B, Q_0)*. Then* $s \upharpoonright B'$ *is a reachable state of* (B', Q'_0)*. Let* $s \xrightarrow{a} t$ *be a reachable transition of* (B, Q_0)*, and let* a *be an interaction of* (B', Q'_0)*. Then* $s \upharpoonright B' \xrightarrow{a} t \upharpoonright B'$ *is a reachable transition of* (B', Q'_0)*.*

To avoid tedious repetition, we fix, for the rest of the paper, an arbitrary BIP-system (B, Q_0), with $B \triangleq \gamma(B_1, \ldots, B_n)$, and transition system $(Q, \gamma, \rightarrow)$.

3 Characterizing Deadlock-Freedom

Definition 11 (Deadlock-freedom). *A BIP-system* (B, Q_0) *is* deadlock-free *iff in every reachable state* s *of* (B, Q_0)*, some interaction* a *is enabled.*

We assume in the sequel that each individual component B_i is deadlock-free, when considered in isolation, with respect to the set of initial states $Q_0 \upharpoonright B_i$.

3.1 Wait-For Graphs

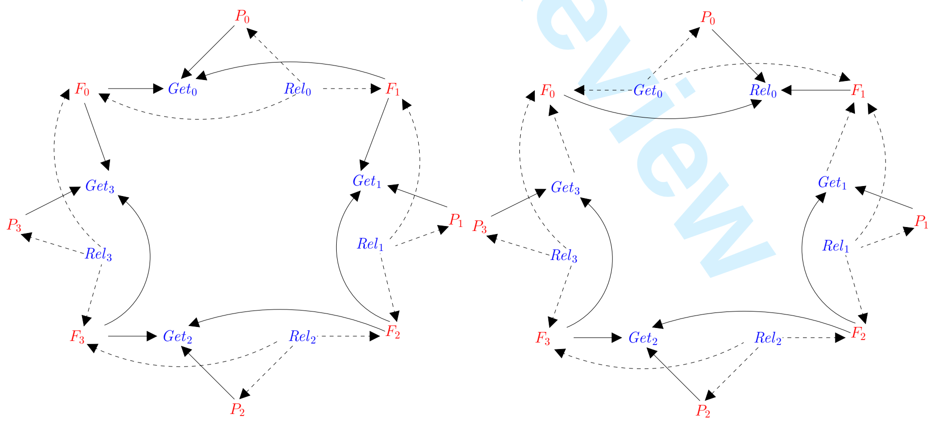
The wait-for-graph for a state s is a directed bipartite and-or graph which contains as nodes the atomic components $B_1, \ldots, B_n$, and all the interactions γ. Edges in the wait-for-graph are from a B_i to all the interactions that B_i enables (in s), and from an interaction a to all the components that participate in a and which do not enable it (in s).

Definition 12 (Wait-for-graph $W_B(s)$). *Let $B = \gamma(B_1, \dots, B_n)$ be a BIP composite component, and let $s = \langle s_1, \dots, s_n \rangle$ be an arbitrary state of B. The* wait-for-graph $W_B(s)$ *of s is a directed bipartite and-or graph, where*

1. *the nodes of $W_B(s)$ are as follows:*
   (a) *the and-nodes are the atomic components B_i, $i \in \{1..n\}$,*
   (b) *the or-nodes are the interactions $a \in \gamma$,*
2. *there is an edge in $W_B(s)$ from B_i to every node a such that $B_i \in C_a$ and $s_i(enb^i_a) = true$, i.e., from B_i to every interaction which B_i enables in s_i,*
3. *there is an edge in $W_B(s)$ from a to every B_i such that $B_i \in C_a$ and $s_i(enb^i_a) = false$, 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). Write $a \rightarrow B_i$ ($B_i \rightarrow a$ respectively) for a wait-for-edge from a to B_i (B_i to a respectively). We abuse notation by writing $e \in W_B(s)$ to indicate that e (either $a \rightarrow B_i$ or $B_i \rightarrow a$) is an edge in $W_B(s)$. Also $B \rightarrow a \rightarrow B' \in W_B(s)$ for $B \rightarrow a \in W_B(s) \wedge a \rightarrow B' \in W_B(s)$, i.e., for a wait-for-path of length 2, and similarly for longer wait-for-paths.

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 get_0. Edges from components to interactions are shown solid, and edges from interactions to components are shown dashed.



(a) Wait-for-graph in initial state. (b) Wait-for-graph after execution of get_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 13 (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. *if 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. *if 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 $\exists B_j : a \to B_j \in W_B(s) \wedge a \to B_j \in SC$,

where $a \in SC$ means that a is a node in SC, etc. $W_B(s)$ 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.

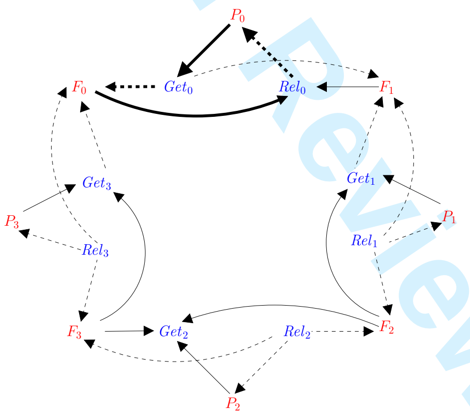


Fig. 3. Example supercycle for dining philosophers system of Figure 1

Figure 3 shows an example supercycle (with boldened edges) for the dining philosophers system of Figure 1. P_0 waits for (enables) a single interaction, Get_0. Get_0 waits for (is disabled by) fork F_0, which waits for interaction Rel_0. Rel_0 in turn waits for P_0. However, this supercycle occurs in a state where P_0 is in h and F_0 is in u_l. This state is not reachable from the initial state.

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. Write $SC \subseteq W_B(s)$ when SC is a subgraph of $W_B(s)$. Proposition 2 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 2. *Let s be a state of B. If $SC \subseteq W_B(s)$ is a supercycle, then all components B_i in SC cannot execute a transition in any state reachable from s, including s itself.*

Proof sketch. Every interaction a that B_i enables is not enabled by some participant. By Defintion 4, a cannot be executed. Hence B_i cannot execute any transition.

Proposition 3 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. *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 sketch. Every atomic component B_i in B' is individually deadlock free, by assumption, and so there is at least one interaction a_i which B_i enables. Now a_i is not enabled in B', by the antecedent of the proposition. Hence a_i has some outgoing wait-for-edge in $W_B(s)$. The subgraph of $W_B(s)$ induced by all the B_i and all their (locally) enabled interactions is therefore a supercycle.

We consider subcomponent B' in isolation to avoid other phenomena that prevent interactions from executing, e.g., conspiracies [5]. Now the converse of Proposition 3 is that absence of supercycles in $W_B(s)$ means there is no locally deadlocked subsystem. Taking $B' = B$, this implies that B is not deadlocked, and so there is at least one interaction of B which is enabled in state s.

Corollary 2. *If, for every reachable state s of (B, Q_0), $W_B(s)$ is supercycle-free, then (B, Q_0) is deadlock-free.*

Proof sketch. Immediate from Proposition 3 (with $B' = B$) and Definition 11.

3.3 Structural Properties of Supercycles

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

Definition 14 (Path, path length). *Let G be a directed graph and v a vertex in G. A path π in G is a* finite *sequence $v_1, v_2, \ldots, v_n$ such that (v_i, v_{i+1}) is an edge in G for all $i \in \{1, \ldots, n-1\}$. Write $path_G(\pi)$ iff π is a path in G. Define $first(\pi) = v_1$ 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, $1 \le i \le n$, are distinct, then $|\pi| = n - 1$, 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 15 (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 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 $out\_depth_G(v) = (\text{MAX } \pi : path_G(\pi) \wedge first(\pi) = v : |\pi|)$, the out-depth of v in G, i.e., we consider paths starting (rather than ending) in v.

We use $in\_depth_B(v, s)$ for $in\_depth_{W_B(s)}(v)$, and also $out\_depth_B(v, s)$ for $out\_depth_{W_B(s)}(v)$.

Proposition 4. *A supercycle SC contains no nodes with finite out-depth.*

Proof sketch. By contradiction. Let v be a node in SC with finite out-depth. Hence all outgoing paths from v end in a sink node. By assumption, all atomic components are individually deadlock-free, i.e., they always enable at least one interaction. Hence these sink nodes are all interactions, and therefore they violate clause 3 in Definition 13.

Proposition 5. *Every supercycle SC contains at least one cycle.*

Proof sketch. Suppose not. Then SC is an acyclic supercycle. Hence every node in SC has finite out-depth, which contradicts Proposition 4.

Proposition 6. *Let $B = \gamma(B_1, \dots, B_n)$ be a composite component and s a state of B. Let SC be a supercycle in $W_B(s)$, 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 $W_B(s)$.*

Proof sketch. By Proposition 5, SC' is nonempty. Thus SC' satisfies clause (1) of Definition 13. Let v be an arbitrary vertex of SC'. Hence v has infinite in-depth, and therefore so do all of v's sucessors in SC. Hence all of these successors are in SC'. Hence every vertex v in SC' has successors in SC' that satisfy clauses (2) and (3) of Definition 13.

4 A Global Condition for Deadlock Freedom

Consider a reachable transition $s \xrightarrow{a} t$ of (B, Q_0). Suppose that the execution of this transition creates a supercycle SC, i.e., $SC \not\subseteq W_B(s) \wedge SC \subseteq W_B(t)$. The only components that can change state along this transition are the participants of a, i.e., the $B_i \in C_a$, and so they are the only components that can cause a supercycle to be created in going from s to t. There are three relevant possibilities for each $B_i \in C_a$:

1. B_i has finite in-depth in $W_B(t)$: then, if $B_i \in SC$, it can be removed and still leave a supercycle SC', by Proposition 6. Hence SC' exists in $W_B(s)$, and so B_i is not essential to the creation of a supercycle.
2. B_i has finite out-depth in $W_B(t)$: by Proposition 4, B_i cannot be part of a supercycle, and so $SC \subseteq W_B(s)$.
3. B_i has infinite in-depth and infinite out-depth in $W_B(t)$: in this case, B_i is possibly an essential part of SC, i.e., SC was created in going from s to t.

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

Definition 16 ($\mathcal{DFC}(a)$). *Let $s \xrightarrow{a} t$ be a reachable transition of BIP-system (B, Q_0). Then, in t, the following holds. For every component B_i of C_a: either B_i has finite in-depth, or finite out-depth, in $W_B(t)$. Formally,*

$$\forall B_i \in C_a : \mathit{in\_depth}_B(B_i, t) < \omega \vee \mathit{out\_depth}_B(B_i, t) < \omega.$$

To proceed, we show that wait-for-edges not involving some interaction a and its participants $B_i \in C_a$ 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 7 (Wait-for-edge preservation). *Let $s \xrightarrow{a} t$ be a transition of composite component $B = \gamma(B_1, \ldots, B_n)$, and let e be a wait-for edge that is not B_i-incident, for every $B_i \in C_a$. Then $e \in W_B(s)$ iff $e \in W_B(t)$.*

Proof sketch. Components not involved in the execution of a do not change state along $s \xrightarrow{a} t$. Hence the endpoint of e that is a component has the same state in s as in t. The proposition then follows from Definition 12.

We show, by induction on the length of finite exeuctions, that every reachable state is supercycle-free. Assume that every initial state is supercycle-free, for the base case. Assuming $\mathcal{DFC}(a)$ for all $a \in \gamma$ provides, by the above discussion, the induction step.

Theorem 1 (Deadlock-freedom). *If (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., $a \in \gamma$), $\mathcal{DFC}(a)$ holds, then for every reachable state u of (B, Q_0): $W_B(u)$ is supercycle-free.*

Proof. We only need show the induction step: for every reachable transition $s \xrightarrow{a} t$, $W_B(s)$ is supercycle-free implies that $W_B(t)$ is supercycle-free. We establish the contrapositive: if $W_B(t)$ contains a supercycle, then so does $W_B(s)$.

Let SC be a supercycle in $W_B(t)$, and let SC' be SC with all nodes of finite in-depth removed. SC' is a supercycle in $W_B(t)$ by Proposition 6. Let e be an arbitrary edge in SC'. Hence $e \in W_B(t)$. Also, both nodes of e have infinite in-depth (by construction of SC') and infinite out-depth (by Proposition 4) in $W_B(t)$. Let B_i be an arbitrary component in C_a. By $\mathcal{DFC}(a)$, B_i has finite in-depth or finite out-depth in $W_B(t)$: $\mathit{in\_depth}_B(B_i, t) < \omega \vee \mathit{out\_depth}_B(B_i, t) < \omega$. Hence e is not B_i-incident. So, $e \in W_B(s)$, by Proposition 7. Hence $SC' \subseteq W_B(s)$, and so $W_B(s)$ contains a supercycle.

5 A Local Condition for Deadlock Freedom

Evaluating $\mathcal{DFC}(a)$ requires checking all reachable transitions of (B, Q_0), which is subject to state-explosion. We need a condition which implies $\mathcal{DFC}(a)$ and can be checked efficiently. Observe that if $\mathit{in\_depth}_B(B_i, t) < \omega \vee \mathit{out\_depth}_B(B_i, t) < \omega$, then there is some finite ℓ such that $\mathit{in\_depth}_B(B_i, t) = \ell \vee \mathit{out\_depth}_B(B_i, t) = \ell$. This can be verified in a subsystem whose size depends on ℓ, as follows.

Definition 17 (Structure Graph G_B, G_i^ℓ, G_a^ℓ). *The structure graph G_B of composite component $B = \gamma(B_1, \ldots, B_n)$ is a bipartite graph whose nodes are the $B_1, \ldots, B_n$ and all the $a \in \gamma$. There is an edge between B_i and interaction a iff B_i participates in a, i.e., $B_i \in C_a$. Define the* distance *between two nodes to be the number of edges in a shortest path between them. Let G_i^ℓ (G_a^ℓ respectively) be the subgraph of G_B that contains B_i (a respectively) and all nodes of G_B that have a distance to B_i (a respectively) less than or equal to ℓ.*

Then $\mathit{in\_depth}_B(B_i, t) = \ell \vee \mathit{out\_depth}_B(B_i, t) = \ell$ can be verified in the wait-for-graph of $G_i^{\ell+1}$, since we verify either that all wait-for-paths ending in B_i have length $\leq \ell$, or that all wait-for-paths starting in B_i have length $\leq \ell$. These conditions can be checked in $G_i^{\ell+1}$, since $G_i^{\ell+1}$ contains every node in a wait-for-path of length $\ell + 1$ or less and which starts or ends in B_i. Since $G_i^{\ell+1} \subseteq G_a^{\ell+2}$ for $B_i \in C_a$, we use $G_a^{\ell+2}$ instead of the set of subsystems $\{G_i^{\ell+1} : B_i \in C_a\}$. We leave analysis of the tradeoff between using one larger system ($G_a^{\ell+2}$) versus several smaller ones ($G_i^{\ell+1}$) to another paper. Define D_a^ℓ, the *deadlock-checking subsystem for interaction a and depth ℓ*, to be the subsystem of (B, Q_0) based on $G_a^{\ell+2}$.

Definition 18 ($\mathcal{LDFC}(a, \ell)$). *Let $s_a \xrightarrow{a} t_a$ be a reachable transition of D_a^ℓ. Then, in t_a, the following holds. For every component B_i of C_a: either B_i has in-depth at most ℓ, or out-depth at most ℓ, in $W_{D_a^\ell}(t_a)$. Formally,*

$$\forall B_i \in C_a : \mathit{in\_depth}_{D_a^\ell}(B_i, t_a) \leq \ell \vee \mathit{out\_depth}_{D_a^\ell}(B_i, t_a) \leq \ell.$$

To infer deadlock-freedom in (B, Q_0) by checking $\mathcal{LDFC}(a, \ell)$, we 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 8 (Wait-for-edge projection). *Let (B', Q_0') be a subsystem of (B, Q_0). Let s be a state of (B, Q_0), and $s' = s{\restriction}B'$. Let a be an interaction of (B', Q_0'), and $B_i \in C_a$ an atomic component of B'. Then (1) $a \to B_i \in W_B(s)$ iff $a \to B_i \in W_{B'}(s')$, and (2) $B_i \to a \in W_B(s)$ iff $B_i \to a \in W_{B'}(s')$.*

Proof sketch. Since $s' = s{\restriction}B'$, all port enablement conditions of components in B' have the same value in s and in s'. The proposition then follows by straightforward application of Definition 12.

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 9 (In-projection, Out-projection). *Let $\ell \geq 0$, 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 $G_i^{\ell+1}$. Let s be a state of (B, Q_0), and $s' = s{\restriction}B'$. Then (1) $in\_depth_B(B_i, s) \leq \ell$ iff $in\_depth_{B'}(B_i, s') \leq \ell$, and (2) $out\_depth_B(B_i, s) \leq \ell$ iff $out\_depth_{B'}(B_i, s') \leq \ell$.*

Proof sketch. Follows from Defintion 15, Proposition 8, and the observation that $W_{B'}(s')$ contains all wait-for-paths of length $\leq \ell$ that start or end in B_i.

We now show that $\mathcal{LDFC}(a, \ell)$ implies $\mathcal{DFC}(a)$, which in turn implies deadlock-freedom.

Lemma 1. *Let a be an interaction of B, i.e., $a \in \gamma$. If $\mathcal{LDFC}(a, \ell)$ holds for some finite $\ell \geq 0$, then $\mathcal{DFC}(a)$ holds.*

Proof sketch. Let $s \xrightarrow{a} t$ be a reachable transition of (B, Q_0) and let $s_a = s{\restriction}D_a^\ell$, $t_a = t{\restriction}D_a^\ell$. Then $s_a \xrightarrow{a} t_a$ is a reachable transition of D_a^ℓ by Corollary 1. By $\mathcal{LDFC}(a, \ell)$, $in\_depth_{D_a^\ell}(B_i, t_a) \leq \ell \vee out\_depth_{D_a^\ell}(B_i, t_a) \leq \ell$. Hence by Proposition 9, $in\_depth_B(B_i, t) \leq \ell \vee out\_depth_B(B_i, t) \leq \ell$. So $in\_depth_B(B_i, t) < \omega \vee out\_depth_B(B_i, t) < \omega$. Hence $\mathcal{DFC}(a)$ holds.

Theorem 2 (Deadlock-freedom). *If (1) for all $s_0 \in Q_0$, $W_B(s_0)$ is supercycle-free, and (2) for all interactions a of B $(a \in \gamma)$, $\mathcal{LDFC}(a, \ell)$ holds for some $\ell \geq 0$, then for every reachable state u of (B, Q_0): $W_B(u)$ is supercycle-free.*

Proof sketch. Immediate from Lemma 1 and Theorem 1.

6 Implementation and Experimentation

LDFC-BIP, ($\sim$ 1500 LOC Java) implements our method for finite-state BIP-systems. Pseudocode for LDFC-BIP is shown in Figure 4. checkDF(B, Q_0) iterates over each interaction a of (B, Q_0), and checks $(\exists \ell \geq 0 : \mathcal{LDFC}(a, \ell))$ by starting with $\ell = 0$ and incrementing ℓ until either $\mathcal{LDFC}(a, \ell)$ is found to hold, or D_a^ℓ has become the entire system and $\mathcal{LDFC}(a, \ell)$ does not hold. In the latter case, $\mathcal{LDFC}(a, \ell)$ does not hold for any finite ℓ, and, in practice, computation would halt before D_a^ℓ had become the entire system, due to exhaustion of resources.

locLDFC(a, ℓ) checks $\mathcal{LDFC}(a, \ell)$ by examining every reachable transition that executes a, and checking that the final state satisfies Definition 18.

The running time of our implementation is $O(\Sigma_{a \in \gamma} |D_a^{\ell_a}|)$, where ℓ_a is the smallest value of ℓ for which $\mathcal{LDFC}(a, \ell)$ holds, and where $|D_a^{\ell_a}|$ denotes the size of the transition system of $D_a^{\ell_a}$.

6.1 Experiment: Dining Philosophers

We consider n philosophers in a cycle, based on the components of Figure 1. Figure 5(a) provides experimental results. The x axis gives the number n of philosophers (and also the number of forks), and the y axis gives the verification time (in milliseconds). We verified that $\mathcal{LDFC}(a, \ell)$ holds for $\ell = 1$ and all interactions a. Hence dining philosophers is deadlock-free. We increase n and plot the

$\mathsf{checkDF}(B, Q_0)$, where $B \triangleq \gamma(B_1, \ldots, B_n)$
forall interactions $a \in \gamma$
  //check $(\exists \ell \geq 0 : \mathcal{LDFC}(a, \ell))$
  $\ell \leftarrow 0$; //start with $\ell = 0$
  while (`true`)
    if $(\mathsf{locLDFC}(a, \ell) = \texttt{true})$ **break endif**; //success, so go on to next a
    if $(D_a^\ell = \gamma(B_1, \ldots, B_n))$ **return**(`false`) **endif**;
    $\ell \leftarrow \ell + 1$ //increment ℓ until success or intractable or failure
  endwhile
endfor;
return(`true`) //return `true` if check succeeds for all $a \in \gamma$

$\mathsf{locLDFC}(a, \ell)$
forall reachable transitions $s_a \xrightarrow{a} t_a$ of D_a^ℓ
  if $(\neg(\forall B_i \in C_a : \mathit{in\_depth}_{D_a^\ell}(B_i, t_a) = \ell \vee \mathit{out\_depth}_{D_a^\ell}(B_i, t_a) = \ell))$
    return(`false`) //check Definition 18
endfor;
return(`true`) //return `true` if check succeeds for all transitions

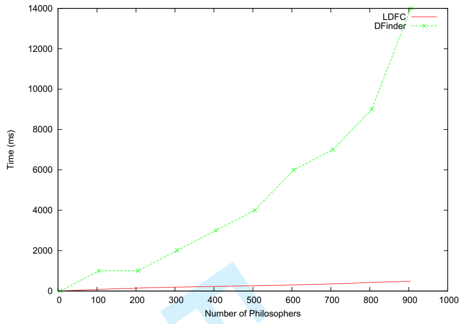
Fig. 4. Pseudocode for the implementation of our method

verification time for both LDFC-BIP and D-Finder 2 [8]. D-Finder 2 implements a compositional and incremental method for the verification of BIP-systems. D-Finder (the precursor of D-Finder 2) has been compared favorably with NuSmv and SPIN, outperforming both NuSmv and SPIN on dining philosophers, and outperforming NuSmv on the gas station example [7], treated next. Our results show that LDFC-BIP has a linear increase of computation time with the system size (n), and so outperforms D-Finder 2.

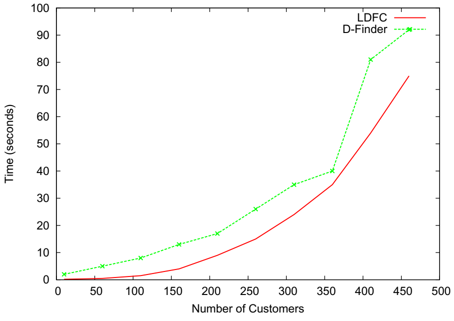
6.2 Experiment: Gas Station

A gas station [13] consists of an operator, a set of pumps, and a set of customers. Before using a pump, a customer has to prepay. Then the customer uses the pump, collects his change and starts a new transaction. Before being used by a customer, a pump has to be activated by the operator. When a pump is shut off, it can be re-activated for the next operation.

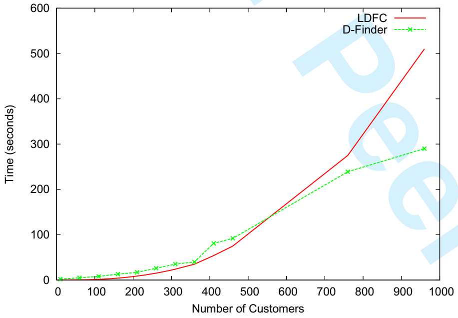
We verified $\mathcal{LDFC}(a, \ell)$ for $\ell = 2$ and all interactions a. Hence gas station is deadlock-free. Figures 5(b), 5(c), and 5(d) present the verification times using LDFC-BIP and D-Finder 2. We consider a system with 3 pumps and variable number of customers. In these figures, the x axis gives the number n of customers, and the y axis gives the verification time (in seconds). D-Finder 2 suffers state-explosion at $n = 1800$, because we consider only three pumps, and so the incremental method used by D-Finder 2 deteriorates. LDFC-BIP outperforms D-Finder 2 as the number of customers increases.



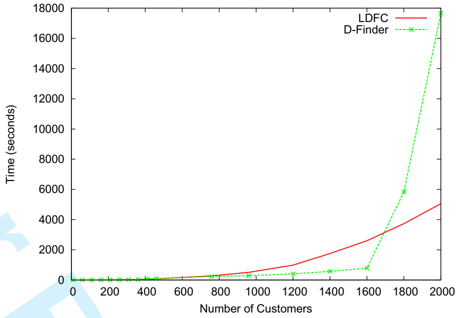
(a) Dining philosophers benchmark.



(b) Gas station benchmark 1.



(c) Gas station benchmark 2.



(d) Gas station benchmark 3.

Fig. 5. Benchmarks generated by our experiments

7 Discussion, Related Work, and Further Work

Related Work. The notions of wait-for-graph and supercycle [3, 4] were initially defined for a shared memory program $P = P_1 \| \cdots \| P_K$ in *pairwise normal form*: a binary symmettric 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 [3] give two polynomial time methods for deadlock freedom. The first checks subsystems consisting of three processes. The second computes the wait-for-graphs of all pair subsystems $P_i \| P_j$, 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 [12] 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 [14] 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. Brookes and Roscoe [11] provide criteria for deadlock freedom of CSP programs based on structural and behavioral restrictions combined with analysis of pair systems. No implementation, or complexity bounds, are given. Aldini and Bernardo [1] use a formalism based on process algebra. They check deadlock by analysing cycles in the connections between software components, and claim scalability, but no complexity bounds are given.

We compared our implementation LDFC-BIP to D-Finder 2 [8]. 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 LDFC-BIP, D-Finder 2 handles infinite state systems. However, LDFC-BIP had superior running time for dining philosophers and gas station (both finite-state).

All the above methods verify global (and not local) deadlock-freedom. Our method verifies both. Also, our approach makes no structural restriction at all on the system being checked for deadlock.

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 [9].
- **Locality.** If a component B_i is modified, or is added to an existing system, then $\mathcal{LDFC}(a, \ell)$ only has to be re-checked for B_i and components within distance ℓ of B_i. A condition whose evaluation considers the entire system at once, e.g., [1, 8, 12] would have to be re-checked for the entire system.
- **Easily Parallelizable.** Since the checking of each subsystem D_a^ℓ 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.

Further Work. Our implementation uses explicit state enumeration. Using BDD's may improve the running time when $\mathcal{LDFC}(a, \ell)$ holds only for large ℓ. An enabled port p enables all interactions containing p. Deadlock-freedom conditions based on ports could exploit this interdepence among interaction enablement. Our implementation should produce *counterexamples* when a system fails to satisfy $\mathcal{LDFC}(a, \ell)$. *Design rules* for ensuring $\mathcal{LDFC}(a, \ell)$ 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 $W_B(s)$ that can be made into a supercycle by adding f edges. We will investigate criteria for preventing formation of such subgraphs. Methods for evaluating $\mathcal{LDFC}(a, \ell)$ 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*, [10], where participants can add and remove interactions at run time.

References

1. Aldini, A., Bernardo, M.: A General Approach to Deadlock Freedom Verification for Software Architectures. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 658–677. Springer, Heidelberg (2003)
2. Attie, P.C.: Synthesis of large concurrent programs via pairwise composition. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 130–145. Springer, Heidelberg (1999)
3. Attie, P.C., Chockler, H.: Efficiently verifiable conditions for deadlock-freedom of large concurrent programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 465–481. Springer, Heidelberg (2005)
4. Attie, P.C., Allen Emerson, E.: Synthesis of Concurrent Systems with Many Similar Processes. TOPLAS 20(1), 51–115 (1998)
5. Attie, P.C., Francez, N., Grumberg, O.: Fairness and Hyperfairness in Multiparty Interactions. Distributed Computing 6, 245–254 (1993)
6. Basu, A., Bozga, M., Sifakis, J.: Modeling Heterogeneous Real-time Components in BIP. In: SEFM, pp. 3–12 (September 2006)
7. Bensalem, S., Bozga, M., Nguyen, T.H., Sifakis, J.: Compositional verification for component-based systems and application. IET Software 4(3), 181–193 (2010)
8. Bensalem, S., Griesmayer, A., Legay, A., Nguyen, T.-H., Sifakis, J., Yan, R.: D-finder 2: Towards efficient correctness of incremental design. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 453–458. Springer, Heidelberg (2011)
9. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: From High-level Component-based Models to Distributed Implementations. In: EMSOFT, pp. 209–218 (2010)
10. Bozga, M., Jaber, M., Maris, N., Sifakis, J.: Modeling Dynamic Architectures Using Dy-BIP. In: Gschwind, T., De Paoli, F., Gruhn, V., Book, M. (eds.) SC 2012. LNCS, vol. 7306, pp. 1–16. Springer, Heidelberg (2012)

11. Brookes, S.D., Roscoe, A.W.: Deadlock analysis in networks of communicating processes. Distributed Computing 4, 209–230 (1991)
12. Göler, G., Sifakis, J.: Component-based construction of deadlock-free systems. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 420–433. Springer, Heidelberg (2003)
13. Heimbold, D., Luckham, D.: Debugging Ada tasking programs. IEEE Software 2(2), 47–57 (1985)
14. Martens, M., Majster-Cederbaum, M.: Deadlock-freedom in component systems with architectural constraints. FMSD 41, 129–177 (2012)
15. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)