Global and Local Deadlock Freedom in BIP*

Paul C Attie[1], Saddek Bensalem[2], Marius Bozga[3], Mohamad Jaber[4], Joseph Sifakis[5], and Fadi A Zaraket[6]

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

September 27, 2016

Abstract

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 (i.e., is sound for deadlock-freedom). If not satisfied, then we increase the size of the subsystems and re-evaluate, as this improves the accuracy of the check. In the limit, the subsystem being checked becomes the entire system, and then our criterion is also complete for deadlock-freedom. Our criterion can thus only fail to decide the deadlock-freedom of a system because of computational limitations: state-space explosion sets in when the subsystems being checked become too large, and cannot be model-checked in practice. Our method thus combines the possibility of fast response together with theoretical completeness. This is in contrast to other criteria for deadlock-freedom, which are incomplete in principle, and so may fail to decide deadlock-freedom even if unlimited computational resources are available.

In addition, our criterion certifies freedom from local deadlock, in which a subsystem is deadlocked while the rest of the system can execute.

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. These show that our method compares favorably with existing approaches.

*The research leading to these results has received funding from University Research Board (URB) at AUB.

Contents

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 [17, 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 in Antonio et. al. [2], 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, etc), 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 Fig. 5 below).

Our method, in contrast, characterizes deadlock by the occurrence of a *supercycle* [7, 6], 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 [2, 15, 18, 9, 12, 14, 13, 1] guarantee only global deadlock freedom.

This paper significantly extends a preliminary conference version [5] 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 [5] 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 present experimental results for dining philosophers and for a multi token-based resource allocation system, which generalizes Milner's token based scheduler [16]. These show that our method compares favorably with existing approaches.

2 BIP — Behavior Interaction Priority

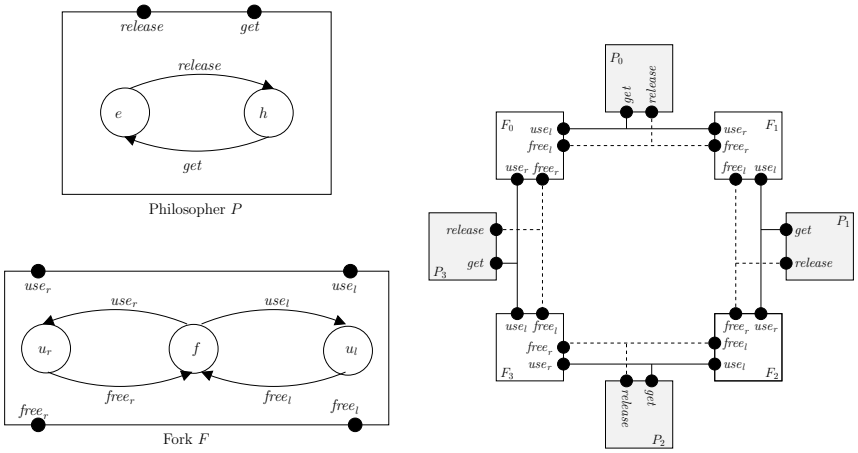
BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. A technical treatment of priority is beyond the scope of this paper. Adding priorities never introduces a deadlock, since priority enforces a choice between possible transitions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



(a) Philosopher P and fork F atomic components.

(b) Dining philosophers composite component with four philosophers.

Figure 1: Dining philosophers.

Definition 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*(a) the set of atomic components participating in a, formally: *components*(a) $= \{\mathsf{B}_i \mid p_i \in \mathsf{a}\}$.

Definition 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 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 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 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 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 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 8 (Reachable state, transition) *A state or transition that occurs in some execution is called* reachable.

Definition 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* $s{\restriction}\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\} \triangleq \langle s_{i_1}, \ldots, s_{i_k} \rangle$. *For a single* B_i, *we write* $s{\restriction}\mathsf{B}_i = s_i$. *We extend state projection to sets of states element-wise.*

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

Definition 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. *$Q_0' = Q_0{\restriction}\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$*

Definition 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, $\rho{\restriction}(\mathsf{B}', Q_0')$, the projection of ρ onto (B', Q_0'), is the sequence resulting from:*

1. *replacing each s_j by $s_j{\restriction}\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$, 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 1 (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, $\rho{\restriction}(\mathsf{B}', Q_0')$ is an execution of (B', Q_0').*

Proof. By Definitions 9, 11, and 12, we have $\rho{\restriction}(\mathsf{B}', Q_0') = s_0'\mathsf{b}_1 s_1'\mathsf{b}_2 s_2' \ldots$ for some $s_0', \mathsf{b}_1 s_1' \mathsf{b}_2 s_2' \ldots$, where $s_j' \in Q' = Q{\restriction}\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$ for $j \geq 0$. Also by Definitions 9, 11, and 12, we have $s_0' \in Q_0' = Q_0{\restriction}\{\mathsf{B}_{i_1}, \ldots, \mathsf{B}_{i_k}\}$, since $s_0' = s_0{\restriction}\mathsf{B}'$, and $s_0 \in Q_0$, by Definition 7.

Consider an arbitrary step $(s_{j-1}', \mathsf{b}_j, s_j')$ of $\rho{\restriction}(\mathsf{B}', Q_0')$. Since $\mathsf{b}_j s_j'$ was not removed in Clause 2 of Definition 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 3 and 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 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 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 7.

Since both clauses of Definition 7 are satisfied, we conclude that $\rho{\restriction}(\mathsf{B}', Q'_0)$ is an execution of (B', Q'_0). □

Corollary 2 *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 1. □

3 Characterizing Deadlock-freedom

We define our notion of *wait-for graph*, and characterize deadlock graph-theoretically as the occurrence of a *supercycle* within the wait-for graph. We show that this gives a sound *and* complete characterization of deadlock.

Definition 13 (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 14 (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 4 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 5 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 15 (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.*

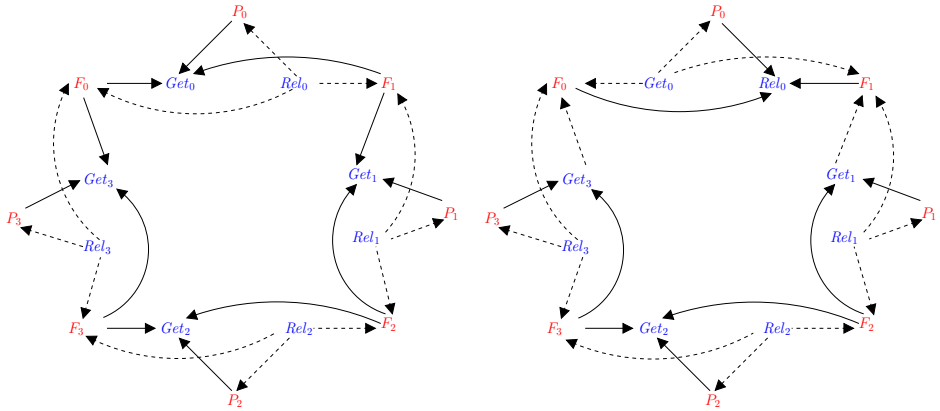
A component B_i is an and-node since all of its successor actions (or-nodes) must be disabled for B_i to be incapable of executing. An interaction a is an or-node since it is disabled if any of its participant components do not enable it. An edge (path) in a wait-for-graph is called a wait-for-edge (wait-for-path). 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 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 (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 $s(enb^j_b) = t(enb^j_b)$. It follows from Definition 15 that $e \in W_{\mathsf{B}}(s)$ iff $e \in W_{\mathsf{B}}(t)$. □



(a) Wait-for-graph in initial state.

(b) Wait-for-graph after execution of get_0.

Figure 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 16 (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 17 (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.*

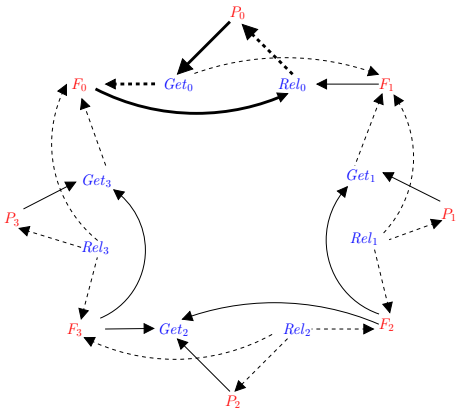


Figure 3: Example supercycle for dining philosophers system of Figure 1.

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.

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 there is a cycle of wait-for-edges, without there being a supercycle. This shows that a cycle does not necessarily imply a supercycle, and hence deadlock.

The existence of a supercycle is sufficient and necessary for the occurrence of a deadlock, and so checking for supercycles gives a sound and complete check for deadlocks. Proposition 4 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 4 *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 16, 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 5 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.

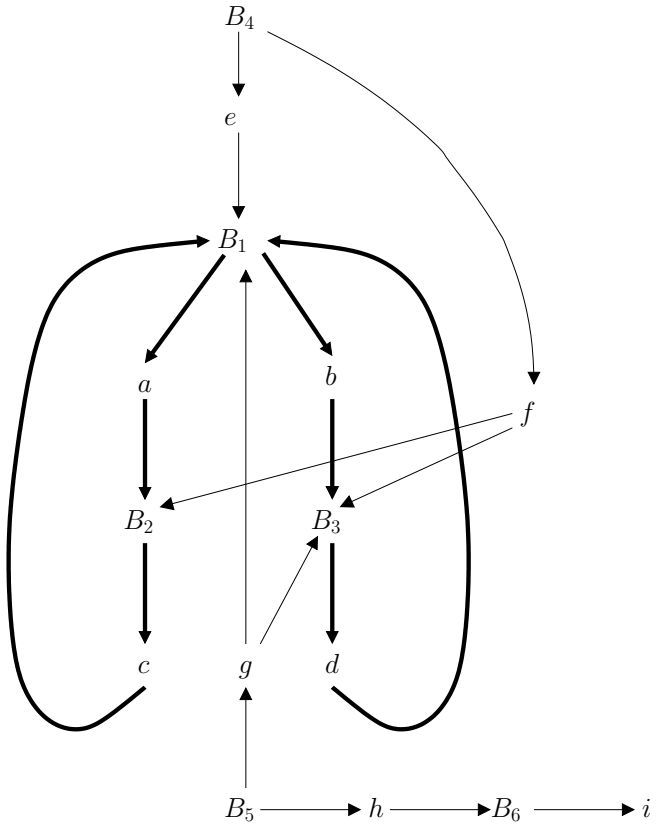


Figure 4: Example supercycle that is not a simple cycle

Proposition 5 *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* $s{\restriction}\mathsf{B}'$*. 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 16 and so is a supercycle. □

We consider subcomponent B' in isolation to avoid other phenomena that prevent interactions from executing, e.g., conspiracies [8]. Now the converse of Proposition 5 is that absence of supercycles in $W_{\mathsf{B}}(s)$ means there is no locally deadlocked subsystem.

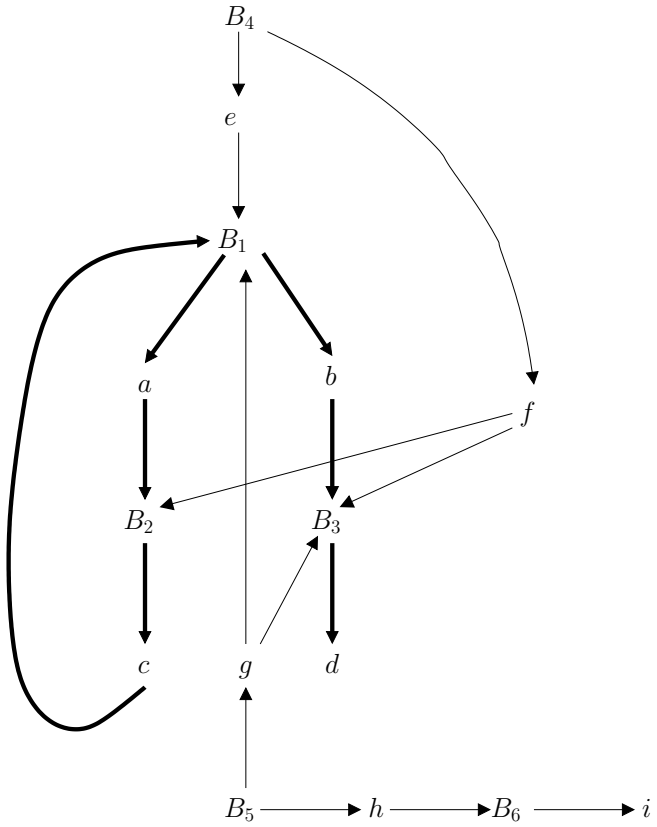


Figure 5: Example where a wait-for cycle does not imply deadlock

Corollary 6 (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 $s{\restriction}\mathsf{B}'$. By Proposition 5, $W_{\mathsf{B}}(s)$ 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 18 (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, $0 \leq i \leq n$, 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 19 (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 7 *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 19 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 16, contradicting the assumption that SC is a supercycle. □

Proposition 8 *Every supercycle SC contains at least two nodes.*

Proof. By Definition 16, SC is nonempty, and so contains at least one node v. If v is an interaction a, then by Definition 16, 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 16, every interaction that B_i enables must be in SC. Hence in both cases, SC contains at least two nodes. □

Proposition 9 *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 7, SC cannot be a supercycle. □

Proposition 10 *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 9, $SC' \neq \emptyset$. Thus SC' satisfies clause (1) of the supercycle definition (16). 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 19. 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 (16). It follows that every vertex v in SC' has enough successors in SC' to satisfy clauses (2) and (3) of the supercycle definition (16). □

Proposition 11 *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 (16). 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 (16). It follows that every vertex v in CC has enough successors in CC to satisfy clauses (2) and (3) of the supercycle definition (16).

Hence, by Definition 16, CC is itself a supercycle, and so Clause (1) of the Proposition is established. □

Note also that by Proposition 8, CC contains at least two nodes. Hence CC is not a trivial strongly connected component.

Proposition 12 *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. 16. □

4 Supercycle Formation and its Consequences

4.1 Supercycle Membership

Definition 20 (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. 16) imposes certain constraints on supercycle membership of a node w.r.t. its predecessors and successors in the wait-for-graph, as follows:

Proposition 13 (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. 16, 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. 12, $SC \subseteq W_B(s)$ is a supercycle. Let SC' be SC with $\mathsf{B}_i \to \mathsf{a}$ added, for all $\mathsf{a} \in succs_{\mathsf{B}}(s,\mathsf{B}_i)$. Then SC' is a supercycle by Def. 16, 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 $SC \subseteq W_B(s)$ si the supercycle containing B_i. Let $\mathsf{a} \in preds_{\mathsf{B}}(s,\mathsf{B}_i)$, and let SC' be SC with $\mathsf{a} \to \mathsf{B}_i$ added. Hence SC' is a supercycle by Definition 16, 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. 16, 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 $\mathsf{a} \to \mathsf{B}_i$ added. Then SC' is a supercycle by Def. 16, 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. 16, 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. 13 gives relationships between supercycle membership of a node and both its successors and predecessors, nevertheless Def. 16 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. 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 21 (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. 13, 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. 13, 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

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

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

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 14 (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., v is not a node of a supercycle in* $W_{\mathsf{B}}(t)$*.*

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. 13, 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. 13, Clause 3, $\neg scyc_{\mathsf{B}}(t, \mathsf{a})$. □

Proposition 15 (Completeness of supercycle violation w.r.t. supercycle non-membership) *If* $(\forall\, d \geq 1 : \neg\mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ *then* $scyc_{\mathsf{B}}(t, v)$*, i.e., v is a node of a supercycle in* $W_{\mathsf{B}}(t)$*.*

Proof. 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 21, 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 21, 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 16: 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 16 $\neg scyc_{\mathsf{B}}(t, v)$ *iff* $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$.

Proof. Immediate from Propositions 14 and 15. □

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) to define a condition that must hold whenever a supercycle is created. Negating this condition then implies the absence of supercycles.

Proposition 17 (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* $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ *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 11, 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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$. Then, no edge in CC is B_i-incident. Hence, by Proposition 3, 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, $\mathsf{B}_j \notin \mathit{components}(\mathsf{a})$, and so $s{\restriction}\mathsf{B}_j = t{\restriction}\mathsf{B}_j$ by Definition 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. 16). 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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, and so Clause 5 is established. □

4.3 General supercycle violation condition

We use Prop. 17 to formulate a condition that prevents the formation of supercycles. For transition $s \xrightarrow{\mathsf{a}} t$, we determine for every component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ 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., $\neg \mathit{scyc}_{\mathsf{B}}(s, \mathsf{B}_i)$.
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 22 (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 string connectedness violation conditions.

Definition 23 (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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, $\mathsf{genViolate}_{\mathsf{B}}(v,t)$ 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 24 (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 25 (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 18 (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 25. □

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 19 (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. 18, 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 17) 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 21) 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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ exhibits a supercycle-violation, as given by $\mathsf{scViolate}_{\mathsf{B}}(\mathsf{B}_i, d, t)$ (Definition 21).

Definition 26 ($\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$) *Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0). Then, in t, the following holds. For every component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, the formation violation condition holds. Formally,*

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

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

Theorem 20 *$\mathcal{GALT}$ 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 17 there exists a component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ 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 22, that $\neg\mathsf{sConnViolate}_{\mathsf{B}}(\mathsf{B}_i, t)$ holds.

Since CC is a supercycle, we have, by Proposition 16, that $\neg(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(\mathsf{B}_i, d, t))$ holds.

Hence, by Definition 23, $\neg\mathsf{genViolate}_{\mathsf{B}}(\mathsf{B}_i, t)$ But, by Definition 26, 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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$. 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 10. 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 7, 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 27 ($\mathcal{GLIN}(B, Q_0, \mathsf{a})$) *$\mathcal{GLIN}(B, Q_0, \mathsf{a})$ holds iff, for every reachable transition $s \xrightarrow{\mathsf{a}} t$ of BIP-system (B, Q_0), the following holds in state t:*

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

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

Proposition 21 *Assume that node v of $W_{\mathsf{B}}(t)$ has a finite in-depth of d in $W_{\mathsf{B}}(t)$, i.e., $\mathit{in\_depth}_{\mathsf{B}}(v, t) = d$. 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 22 *Assume that node v of $W_{\mathsf{B}}(t)$ has a finite out-depth of d in $W_{\mathsf{B}}(t)$, i.e., $\mathit{out\_depth}_{\mathsf{B}}(v, t) = d$. 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 18 and 19, v has no outgoing wait-for-edges in $W_{\mathsf{B}}(t)$. Hence by Definition 21, $\mathsf{scViolate}_{\mathsf{B}}(v,1,t)$.

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 18 and 19, 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 21, Clauses 1 and 2, $\mathsf{scViolate}_{\mathsf{B}}(v,d+1,t)$. □

Lemma 23 $\forall\, \mathsf{a} \in \gamma : \mathcal{GLIN}(\mathsf{B},Q_0,\mathsf{a}) \Rightarrow \mathcal{GALT}(\mathsf{B},Q_0,\mathsf{a})$.

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 21 and 22,

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 23,
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 24 *$\mathcal{GLIN}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 23 and Theorem 20. □

5.3 Deadlock freedom using global restrictions

Corollary 25 (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 20, 24 and Corollary 19. □

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 28 (Structure Graph G_{B}, G^{ℓ}_{a}) *The structure graph G_{B} of composite component $B = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ is a bipartite graph whose nodes are the $\mathsf{B}_1, \ldots, \mathsf{B}_n$ 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 29 (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 30 (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 26 (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 $s' = s{\restriction}\mathsf{B}'$. 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 15, $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(s)$ iff $s{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = false$. Since $s' = s{\restriction}\mathsf{B}'$, we have $s'{\restriction}i = s{\restriction}i$. Hence $s{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = s'{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}})$. By Definition 15, $a \to \mathsf{B}_i \in W_{\mathsf{B}'}(s')$ iff $s'{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = false$. Putting together these three equalities gives us clause (1).

By Definition 15, $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(s)$ iff $s{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = true$. Since $s' = s{\restriction}\mathsf{B}'$, we have $s'{\restriction}i = s{\restriction}i$. Hence $s{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = s'{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}})$. By Definition 15, $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}'}(s')$ iff $s'{\restriction}i(enb^{\mathsf{B}_i}_{\mathsf{a}}) = true$. 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 21 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 $\mathsf{scViolateLoc}(v, d, t, D_{\mathsf{a}}^{\ell})$ to hold iff node v in $W_B(t)$ has a level-d supercycle-violation *that can be confirmed within* D_{a}^{ℓ}.

Definition 31 (Local supercycle violation, $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$) *Let t_{a} be a state of D_{a}^{ℓ} and v be a node of D_{a}^{ℓ}. We define $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ by induction on d, as follows.*

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

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

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

Note that if v is an interaction aa and a border node, then $\mathsf{scViolateLoc}(\mathsf{aa}, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ is false, for all d. This is because aa has some component that is outside D_{a}^{ℓ}, 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 $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$. 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.

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

Figure 7: Formal definition of $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$.

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

Proposition 27 *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 $t_{\mathsf{a}} = t{\restriction}D_{\mathsf{a}}^{\ell}$. Then*

$$\forall\, d \ge 1 : \mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell}) \Rightarrow \mathsf{scViolate}_{\mathsf{B}}(v, d, t).$$

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 21, 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 21, 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 22.

Definition 32 (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 $WL = W_{D^\ell_a}(t_\mathsf{a})\restriction L$, 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 28 *Let t be an arbitrary reachable state of BIP-system (B, Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell > 0$, let $t_{\mathsf{a}} = t{\restriction}D_{\mathsf{a}}^{\ell}$. Then*
$\mathsf{sConnViolateLoc}(v, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell}) \Rightarrow \mathsf{sConnViolate}_{\mathsf{B}}(v, t)$.

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 22, 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 14, 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 27, 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 32. Then $x \in L$, and since x is an arbitrary node of SSC, we have $SSC \subseteq WL$. Thus Clause 1 of Definition 32 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 27, the nodes on π, π' cannot have a local supercycle violation. Hence Clauses 2a and 2b of Definition 32 are violated, since they require that at least one node along π, π' respectively, have a local supercycle violation.

In both cases, Definition 32 is violated. But Definition 32 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 33 (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 29 *Let t be an arbitrary reachable state of BIP-system (B, Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell > 0$, let $t_{\mathsf{a}} = t{\restriction}D_{\mathsf{a}}^{\ell}$. 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 23, $(\exists\, d \geq 1 : \mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})) \vee \mathsf{sConnViolateLoc}(v, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$. We proceed by cases:

1. $(\exists\, d \geq 1 : \mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}}))$: hence $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$ by Proposition 27.
2. $\mathsf{sConnViolateLoc}(v, t_{\mathsf{a}}, D^{\ell}_{\mathsf{a}})$: hence $\mathsf{sConnViolate}_{\mathsf{B}}(v, t)$ by Proposition 28.

By Definition 23, $\mathsf{genViolate}_{\mathsf{B}}(v, t) \triangleq (\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(u, d, t)) \vee \mathsf{sConnViolate}_{\mathsf{B}}(v, t)$. Hence we conclude that $\mathsf{genViolate}_{\mathsf{B}}(v, t)$ holds. □

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 34 ($\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$) *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 components(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 30 *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, $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ is a reachable transition of D^{ℓ}_{a}.

By Definition 34, 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 29,
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 26, $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$ holds. □

Theorem 31 *$\mathcal{LALT}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 30 and Theorem 20. □

6.3 A Local Linear Condition for Deadlock Freedom

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

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

Figure 8: Summary of predicates

Definition 35 ($\mathcal{LLIN}(\mathsf{B},Q_0,\mathsf{a},\ell)$) *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 components(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,*

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

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 32 (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 $G^{2\ell}_a$. Let s be a state of (B,Q_0), and $s'=s{\restriction}B'$. 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 15 and $s'=s{\restriction}B'$, $W_{\mathsf{B}'}(s')$ 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 $G^{2\ell}_a$, and since the distance from B_i to the border of $G^{2\ell}_a$ 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 33 *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 $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, $s_\mathsf{a} = s{\restriction}D^\ell_\mathsf{a}$, $t_\mathsf{a} = t{\restriction}D^\ell_\mathsf{a}$. Then $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ is a reachable transition of D^ℓ_a by Corollary 2. By $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$, $\mathit{in\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1 \vee \mathit{out\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1$. Hence by Proposition 32, $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) < 2\ell - 1 \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) < 2\ell - 1$. 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 34 *$\mathcal{LLIN}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 33 and Theorem 24. □

Proposition 35 (Finite out-depth implies local supercucle-violation) *For $d < \ell$: $(\mathit{out\_depth}_{D^\ell_\mathsf{a}}(v, t_\mathsf{a}) = d) \Rightarrow \mathsf{scViolateLoc}(v, d+1, t_\mathsf{a}, D^\ell_\mathsf{a})$.*

Proof. Proof is by induction on d.

Base case, $d = 0$. Then v has no outgoing wait-for edges. Hence $\mathsf{scViolateLoc}(v, 1, t_\mathsf{a}, D^\ell_\mathsf{a})$ by Definition 31.

Induction step, $d > 0$. Assume $(\mathit{out\_depth}_{D^\ell_\mathsf{a}}(v, t_\mathsf{a}) = d)$. Then, every outgoing wait-for edge of v is to some v' such that $(\mathit{out\_depth}_{D^\ell_\mathsf{a}}(v', t_\mathsf{a}) = d' < d)$. By the induction hypothesis, $\mathsf{scViolateLoc}(v', d'+1, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence, by Definition 31, $\mathsf{scViolateLoc}(v, d+1, t_\mathsf{a}, D^\ell_\mathsf{a})$. □

Lemma 36 *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 $\mathit{components}(\mathsf{a})$. Then, from Definition 35, we have:

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

The proof proceeds by two cases.

$\mathit{in\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1$: 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 32, Clause 1. Hence by Definition 33, $\mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$.

$\mathit{out\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) < 2\ell - 1$: Hence $\mathit{out\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) = d$ for some $d < 2\ell - 1$. By Proposition 35, $\mathsf{scViolateLoc}(\mathsf{B}_i, d+1, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence by Definition 33, $\mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$.

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 $\mathit{components}(\mathsf{a})$, we have $\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathsf{genViolateLoc}(\mathsf{B}_i, t_\mathsf{a}, D^\ell_\mathsf{a})$. Hence, by Definition 34, we conclude $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$. □

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.

6.4 Deadlock freedom using local and global restrictions

Theorem 37 (Deadlock-freedom via $\mathcal{LALT}$, $\mathcal{LLIN}$) *Assume that*

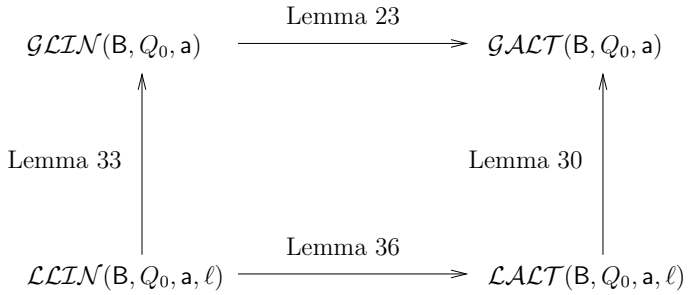


Figure 9: Implication relations between deadlock-freedom conditions

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 deadlock.*

Proof. Immediate from Theorems 20, 24, 31, 34 and Corollary 19. □

7 Implementation and Experiments

7.1 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
1. **forall** $s_0 \in Q_0$
2. compute $W_{\mathsf{B}}(s_0)$
3. let U be the result of removing from $W_{\mathsf{B}}(s_0)$ all nodes v such that $(\exists\, d \geq 1 : \mathsf{scViolate}_{\mathsf{B}}(v, d, t))$
4. **if** (U is nonempty) **then return**(`ff`) ▷ s_0 not supercycle-free, so return false
5. **else return**(`tt`)

Figure 10: Procedure to check that all initial states are supercycle-free

Proposition 38 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 15, 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 14, 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 11, $W_{\mathsf{B}}(s_0)$ does not contain a supercycle. □

7.2 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 35.

```
LLin(B, Q0), where B ≜ γ(B1, ..., Bn)
1.  forall interactions a ∈ γ
2.      if (LLinInt(B, Q0, a) = ff) return(ff) fi
3.  endfor;
4.  return(tt)                          ▷ return tt if check succeeds for all a ∈ γ

LLinInt(B, Q0, a), where B ≜ γ(B1, ..., Bn), a ∈ γ
▷ check (∃ ℓ > 0 : LLIN(B, Q0, a, ℓ))
1.  ℓ ← 1;                                                  ▷ start with ℓ = 1
2.  while (tt)
3.      if (LLinIntDist(a, ℓ) = tt) return(tt) fi;          ▷ success, so return true
4.      if (D^ℓ_a = γ(B1, ..., Bn)) return(ff) fi;          ▷ exhausted all subsystems, return false
5.      ℓ ← ℓ + 1                       ▷ increment ℓ until success or intractable or failure
6.  endwhile


LLinIntDist(B, Q0, a, ℓ)
1.  forall reachable transitions s_a -a-> t_a of D^ℓ_a
2.      if (¬(∀ Bi ∈ components(a) : in_depth_{D^ℓ_a}(Bi, t_a) < 2ℓ − 1 ∨ out_depth_{D^ℓ_a}(Bi, t_a) < 2ℓ − 1))
3.          return(ff)                                      ▷ check Definition 35
4.      fi
5.  endfor;
6.  return(tt)                          ▷ return tt if check succeeds for all transitions
```

Figure 11: Pseudocode for the implementation of the linear condition.

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

7.3 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_{a}^{ℓ}.

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_{a}^{ℓ}. We start with $\ell = 1$ since D_{a}^{1} 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.

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$(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$	true iff B_i has local sc-formation violation in state t_{a} of D_{a}^{ℓ}, i.e., genViolateLoc$(B_i, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ holds
LOCSCONNSCVIOL$(B_i, D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$	true iff B_i has local strong connectedness violation in t_{a}, i.e., sConnViolateLoc$(B_i, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ holds
LOCSCVIOL$(D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}})$	compute local supercycle violations in state t_{a} of D_{a}^{ℓ}, i.e., scViolateLoc$(v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$ for all v, d

Figure 12: Summary of procedures

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

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

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

LALTINTDIST($\mathsf{B}, Q_0, \mathsf{a}, \ell$)
▷ returns 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^ℓ_a
2. **forall** $\mathsf{B}_i \in$ *components*(a)
3. **if** ¬LOCFORMVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$) **then return**(ff) **fi** ▷return ff if no violation for B_i
4. **endfor**
5. **endfor**;
6. **return**(tt) ▷return tt if all $\mathsf{B}_i \in$ *components*(a) violate local supercycle formation

LOCFORMVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$)
▷returns true iff genViolateLoc($B_i, t_\mathsf{a}, D^\ell_\mathsf{a}$) holds (Definition 33)
▷i.e., B_i has a local supercycle formation violation in state t_a of subsystem D^ℓ_a
1. LOCSCVIOL($D^\ell_\mathsf{a}, t_\mathsf{a}$)
2. **return**($V_{D^\ell_\mathsf{a}, t_\mathsf{a}}[B_i]$ ∨ LOCSCONNSCVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$))

LOCSCONNSCVIOL($B_i, D^\ell_\mathsf{a}, t_\mathsf{a}$)
▷returns true iff sConnViolateLoc($B_i, t_\mathsf{a}, D^\ell_\mathsf{a}$) holds (Definition 32)
▷i.e., B_i has a local strong connectedness supercycle formation violation in state t_a of subsystem D^ℓ_a
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**(ff) **fi** ▷Definition 32, Clause 1 holds here
6. **forall** wait-for paths π from B_i to the border of D^ℓ_a
7. **if** some node of π has a local supercycle violation **then return**(tt) **fi** ▷Clause 2a holds
8. **forall** wait-for paths π' from the border of D^ℓ_a to B_i
9. **if** some node of π' has a local supercycle violation **then return**(tt) **fi** ▷Clause 2b holds
10. **return**(ff) ▷Definition 32, Clause 2 does not hold

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

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* ← `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* ← `tt`
6. **fi**
7. **endfor**

8. **while** (*foundScViolate*)
9. *foundScViolate* ← `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* ← `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* ← `tt`
17. **fi**
18. **endfor**
19. **endwhile**

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

7.4 Tool-set

We provide LALT-BIP, a suite of supporting tools that implement our method. LALT-BIP is around $\sim$ 2500 Java LOC. LALT-BIP is equipped with a command line interface (see Figure 1) that accepts a set of configuration options. It takes the name of the input BIP file and other optional flags.

```
> 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
```

Listing 1: LALT-BIP Command Line Interface

7.5 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 configurations 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) *i*7-6700, CPU @ 3.40GHZ, 32GB RAM, running a CentOS Linux distribution.

7.5.1 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 [9]. 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 [9].

Table 1 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 $\mathcal{LLIN}$.

7.5.2 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

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	–
$10,000$	$47s$	$62s$	–

Table 1: Benchmarks: Dining Philosopher

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 15, 16, and 17 show BIP atomic components for client, resource and manager components.

The client in Figure 15 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.

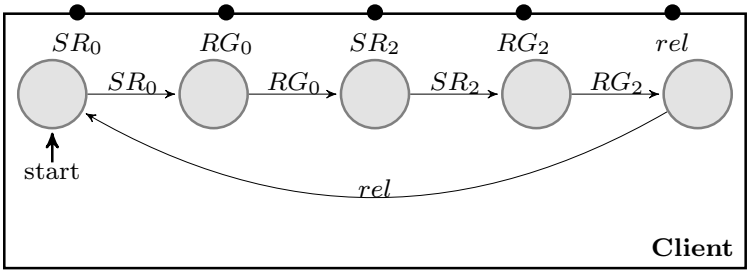


Figure 15: Client

Figure 16 shows a resource component. A resource component waits for a request from a connected client on port RR. Once a request is received, the resource component transitions to a state where it is ready to receive a token from the corresponding resource manager using port RTT. The resource transitions to a state where it grants the client request using port STC and waits until it is released on port *done*. There, it returns the token back to the resource manager and transitions to the start state.

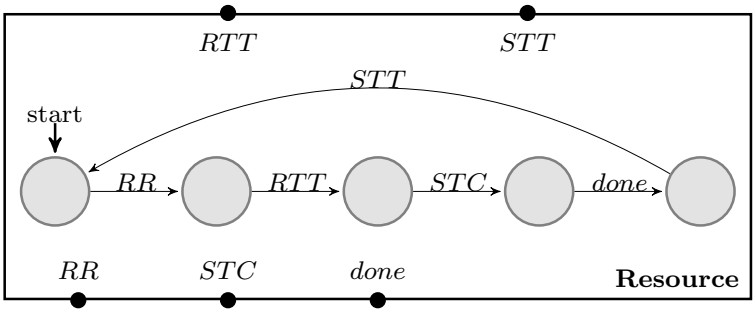


Figure 16: Resource

Figure 17 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.

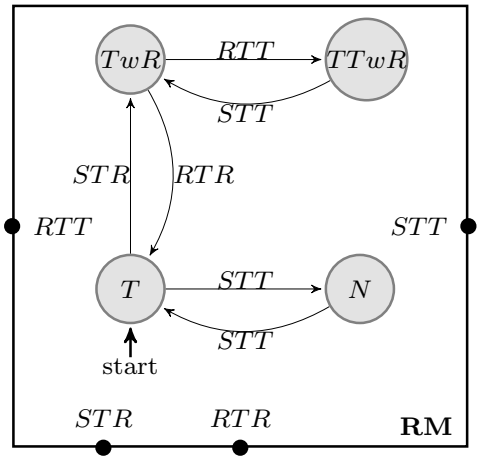


Figure 17: Token Resource Manager

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 18 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.

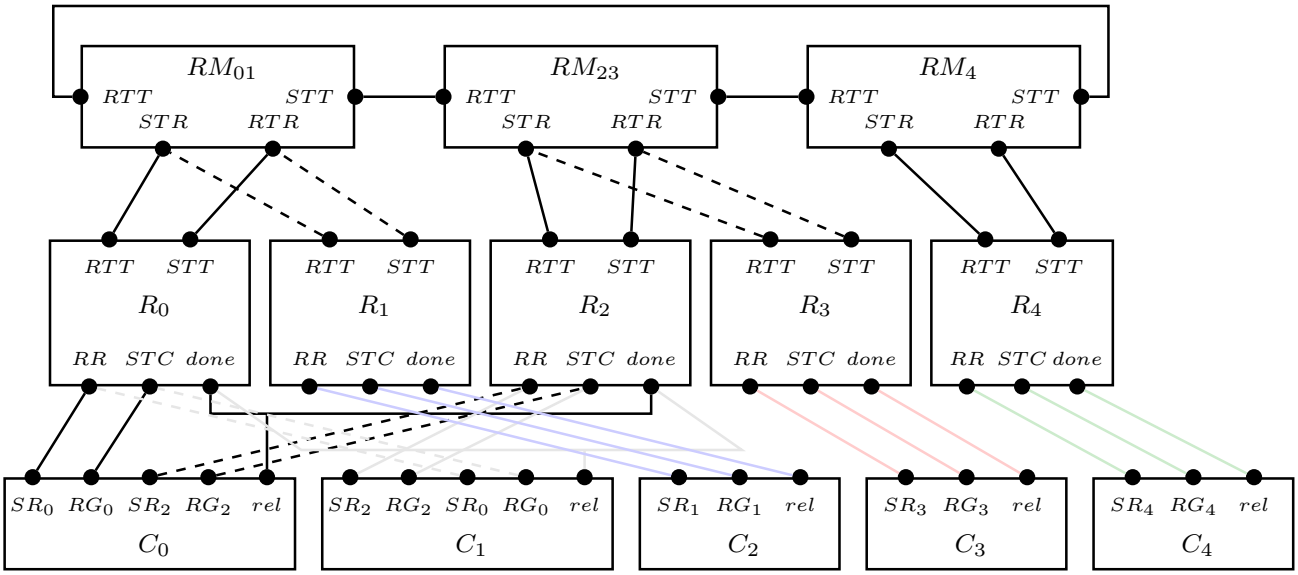


Figure 18: Conflict-Resource Allocation System

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.

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

Table 2: Benchmarks: Time required for $\mathcal{LALT}$ on the resource allocation system

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 [8]. 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.

- 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 2 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 [16] which is essentially a token ring with precisely one token present [2]. The technique presented in [2] fails to prove deadlock freedom for Milner Scheduler because it requires a large subset of the system, while $\mathcal{LALT}$ succeeds.

8 Discussion, Related Work, and Further Work

8.1 Related work.

The notions of wait-for-graph and supercycle [6, 7] were initially defined for a shared memory program $P = P_1 \| \cdots \| P_K$ in *pairwise normal form* [4, 3]: 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 [6] 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 $P_i \| P_j$, 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 [13] 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. Aldini and Bernardo [1] 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 [18] 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 [12] also provide criteria for deadlock freedom of triple-disjoint CSP programs, and use the same technical framework as [18]. 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 [18]. 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 [15] applies the results in [18] and [12] 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. Antonio et. al. [2] 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 Antonio et. al. [2] 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 [9]. 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 [6]) 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, and the $\mathcal{LALT}$ condition is complete w.r.t. the occurrence of a supercycle wholly within the subsystem being checked. 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. 16, Def. 23, and Def. 26).

8.2 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 [10].

Locality If a component B_i is modified, or is added to an existing system, then $\mathcal{LACLT}(B, Q_0, a, \ell)$ 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., [1, 9, 13] 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 [6] 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.

8.3 Further work.

Our implementation uses explicit state enumeration. Using BDD's may improve the running time when $\mathcal{LACLT}(B, Q_0, a, \ell)$ holds only for large ℓ. Another potential method for improving the running time is to use SAT solving, cf. Antonio et. al. [2]. 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 $\mathcal{LACLT}(B, Q_0, a, \ell)$. These can be used to manually modify the system to eliminate a possible deadlock. Also, when $\mathcal{LACLT}(B, Q_0, a, \ell)$ 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 $\mathcal{LACLT}(B, Q_0, a, \ell)$ 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 $\mathcal{LACLT}(B, Q_0, a, \ell)$ 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*, [11], where participants can add and remove interactions at run time.

References

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

[2] Pedro Antonio, Thomas Gibson-Robinson, and A.W. Roscoe. Efficient deadlock-freedom checking using local analysis and sat solving. In *Proceedings of IFM*, 2016.

[3] Paul C. Attie. Finite-state concurrent programs can be expressed in pairwise normal form. *Theoretical Computer Science*, 619:1 – 31, 2016.

[4] Paul C. Attie. Synthesis of large dynamic concurrent programs from dynamic specifications. *Formal Methods in System Design*, pages 1–54, 2016.

[5] Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. An abstract framework for deadlock prevention in BIP. In *Formal Techniques for Distributed Systems - Joint IFIP WG 6.1 International Conference, FMOOD-S/FORTE 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, pages 161–177, 2013.

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

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

[8] P.C. Attie, N. Francez, and O. Grumberg. Fairness and Hyperfairness in Multiparty Interactions. *Distributed Computing*, 6:245–254, 1993.

[9] Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. D-finder 2: Towards efficient correctness of incremental design. In *NASA Formal Methods*, pages 453–458, 2011.

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

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

[12] S.D. Brookes and A.W. Roscoe. Deadlock analysis in networks of communicating processes. *Distributed Computing*, 4:209–230, 1991.

[13] Gregor Gössler and Joseph Sifakis. Component-based construction of deadlock-free systems. In *FSTTCS*, pages 420–433. Springer, 2003.

[14] Moritz Martens and Mila Majster-Cederbaum. Deadlock-freedom in component systems with architectural constraints. *FMSD*, 41:129–177, 2012.

[15] Jeremy Malcolm Randolph Martin. *The Design and Construction of Deadlock-Free Concurrent Systems*. PhD thesis, The University of Buckingham, 1996.

[16] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[17] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[18] A.W. Roscoe and Naiem Dathi. The pursuit of deadlock freedom. *Information and Computation*, 75(3):289 – 327, 1987.