

# Global and Local Deadlock Freedom in BIP<sup>1</sup>

PAUL C ATTIE, Department of Computer Science, American University of Beirut, Beirut, Lebanon

SADDEK BENSALAM, UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France

MARIUS BOZGA, UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France

MOHAMAD JABER, Department of Computer Science, American University of Beirut, Beirut, Lebanon

JOSEPH SIFAKIS, Rigorous System Design Laboratory, EPFL, Lausanne, Switzerland

FADI A ZARAKET, Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon

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

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

Additional Key Words and Phrases: Nondeterminism, Completeness

## ACM Reference Format:

Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, Fadi A Zaraket, 2016. Global and Local Deadlock Freedom in BIP. *ACM Trans. Softw. Eng. Methodol.* V, N, Article A (January YYYY), 42 pages.  
DOI: <http://dx.doi.org/10.1145/0000000.0000000>

## 1. INTRODUCTION

Deadlock freedom is a crucial property of concurrent and distributed systems. With increasing system complexity, the challenge of assuring deadlock freedom and other cor-

---

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© YYYY ACM. 1049-331X/YYYY/01-ARTA \$15.00

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

rectness properties becomes even greater. In contrast to the alternatives of (1) deadlock detection and recovery, and (2) deadlock avoidance, we advocate deadlock prevention: design the system so that deadlocks do not occur.

Deciding deadlock freedom of finite-state concurrent programs is PSPACE-complete, in general [Papadimitriou 1994, chapter 19]. To achieve tractability, we present a criterion for deadlock-freedom that is evaluated by model-checking a set of subsystems of the overall system. If the subsystems are small, the criterion can be checked quickly, and is sound (if true, it implies deadlock-freedom) but not complete (if false, then it yields no information about deadlock). If the subsystems are larger, then our criterion becomes more “accurate”: roughly speaking, there is less possibility for the criterion to evaluate to false when the system is actually deadlock-free. In the limit, when the set of subsystems includes the entire system itself, our criterion is complete, so that evaluation to false implies that the system is actually deadlock-prone. Hence, our criterion only fails to resolve the question of deadlock-freedom when its evaluation exhausts available computational resources, because the subsystems being checked have become too large, and state-explosion has set in.

Our method thus combines the possibility of fast response together with theoretical completeness. All deadlock-freedom checks given in the literature to date are, to our knowledge, incomplete in principle, and so remain incomplete even if unlimited computational resources are available. Hence these criteria could fail to resolve deadlock freedom for theoretical reasons, as well as for lack of computational resources. The reason for this incompleteness is that existing criteria all characterize deadlock by the occurrence of a wait-for cycle, e.g., as stated by Antonino et al. [2016], discussion of related work:

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

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

Our method, in contrast, characterizes deadlock by the occurrence of a *supercycle* [Attie and Emerson 1998; Attie and Chockler 2005], which, very roughly, is the AND-OR analogue of a wait-for cycle: a subset of processes constitutes a supercycle  $SC$  iff every possible action of every process in  $SC$  is blocked by another process in  $SC$ . We show that supercycles are a sound and complete characterization of deadlock: a system is deadlock-prone iff a supercycle can arise in some reachable state. We then present our criterion, which prevents the occurrence of supercycles in reachable states of the system. We first present a “global” version of our criterion, which is both sound and complete w.r.t. absence of supercycles, and then a “local” version, which is sound w.r.t. absence of supercycles, and can be evaluated over small subsystems.

In addition our criterion guarantees freedom from local (and therefore global) deadlock. A local deadlock occurs when a subsystem is deadlocked while the rest of the system can execute. Other criteria in the literature [Antonino et al. 2016; Martin 1996; Roscoe and Dathi 1987; Bensalem et al. 2011; Brookes and Roscoe 1991; Martens and Majster-Cederbaum 2012; Gössler and Sifakis 2003; Aldini and Bernardo 2003] guarantee only global deadlock freedom.

This paper significantly extends a preliminary conference version [Attie et al. 2013] as follows: (1) we present an “AND-OR” criterion for deadlock-freedom, which exploits the AND-OR structure of supercycles, and is therefore complete for deadlock-freedom in the limit, while our preliminary work [Attie et al. 2013] gives a “linear” criterion, which is a special case in which the AND-OR structure is ignored, and (2) experimental results show that the new criterion is more efficient in practice, and also succeeds in cases where the linear criterion fails. We therefore have the best of both worlds: early stopping, and therefore efficient verification of deadlock-freedom, in many cases, together with theoretical completeness. Our criterion is, to the best of our knowledge, the first criterion that is sound *and complete* for local and global deadlock-freedom in concurrent programs with nondeterministic local choice, i.e., a process can nondeterministically choose among enabled actions.

We present experimental results for dining philosophers and for a multi token-based resource allocation system, which generalizes Milner’s token based scheduler [Milner 1989]. These show that our method compares favorably with existing approaches.

## 2. BIP — BEHAVIOR INTERACTION PRIORITY

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

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

For states  $s_i, t_i \in Q_i$  and port  $p_i \in P_i$ , write  $s_i \xrightarrow{p_i} t_i$ , iff  $(s_i, p_i, t_i) \in \rightarrow_i$ . When  $p_i$  is irrelevant, write  $s_i \rightarrow_i t_i$ . Similarly,  $s_i \xrightarrow{p_i}$  means that there exists  $t_i \in Q_i$  such that  $s_i \xrightarrow{p_i} t_i$ . 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  $enb_{p_i}^i$  that holds in state  $s_i$  of component  $B_i$  iff port  $p_i$  is enabled in  $s_i$ , i.e.,  $s_i(enb_{p_i}^i) = \text{true}$  iff  $s_i \xrightarrow{p_i}$ .

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<sub>l</sub>*) and so moves to state  $u_l$  (used by left philosopher), or (2) the right philosopher (transition *get<sub>r</sub>*) and so moves

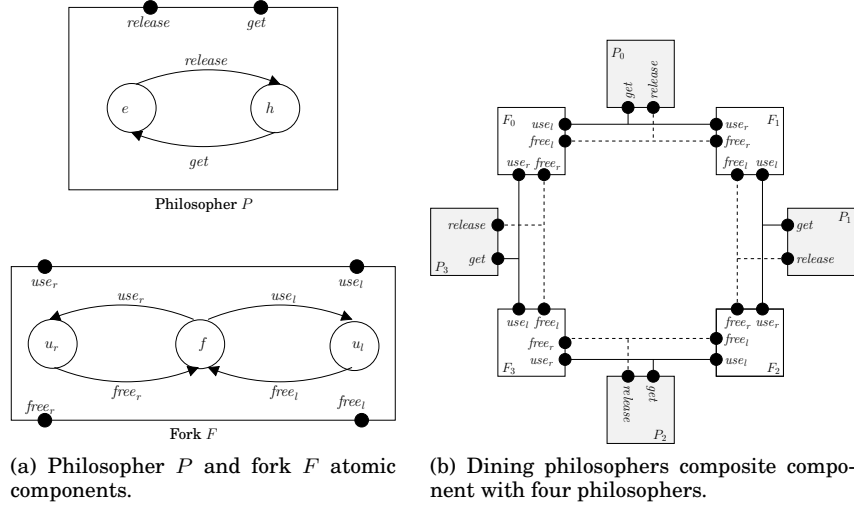


Fig. 1. Dining philosophers.

to state  $u_r$  (used by right philosopher). From state  $u_r$  (resp.  $u_l$ ),  $F$  is released by the right philosopher (resp. left philosopher) and so moves back to state  $f$  (free).

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

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

**Definition 2.3 (Composite Component)** A composite component (or simply component)  $B \triangleq \gamma(B_1, \dots, B_n)$  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  $Q = Q_1 \times \dots \times Q_n$  and  $\rightarrow \subseteq Q \times \gamma \times Q$  is the least set of transitions satisfying the rule

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

This inference rule says that a composite component  $B = \gamma(B_1, \dots, B_n)$  can execute an interaction  $a \in \gamma$ , iff for each port  $p_i \in a$ , the corresponding atomic component  $B_i$  can execute a transition labeled with  $p_i$ ; the states of components that do not participate in the interaction stay unchanged. Figure 1(b) shows a composite component consisting of four philosophers and the four forks between them. Each philosopher and its two neighboring forks share two interactions:  $Get = \{get, use_l, use_r\}$  in which the philosopher obtains the forks, and  $Rel = \{release, free_l, free_r\}$  in which the philosopher releases the forks.

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

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

Let  $B = \gamma(B_1, \dots, B_n)$  be a composite component, and let  $s = \langle s_1, \dots, s_n \rangle$  be a state of  $B$ . Then  $B$  enables  $a$  in  $s$  iff every  $B_i \in \text{components}(a)$  enables  $a$  in  $s_i$ .

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

To avoid pathological cases of deadlock due solely to a single component refusing to enable any interaction at all, we assume that every component always enables at least one interaction. Structurally, this means that there is no local state zero transitions, and every port labeling a transition is part of at least one interaction.

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

**Definition 2.6 (BIP System)** Let  $B = \gamma(B_1, \dots, B_n)$  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  $B \triangleq \gamma(B_1, \dots, B_n)$ , and transition system  $(Q, \gamma, \rightarrow)$ .

**Definition 2.7 (Execution)** Let  $(B, Q_0)$  be a BIP system with transition system  $(Q, \gamma, \rightarrow)$ . Let  $\rho = s_0 a_1 s_1 \dots s_{j-1} a_j s_j \dots$  be an alternating sequence of states of  $B$  and interactions of  $B$ . Then  $\rho$  is an execution of  $(B, Q_0)$  iff (1)  $s_0 \in Q_0$ , and (2)  $\forall j > 0 : s_{j-1} \xrightarrow{a_j} s_j$ .

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

**Definition 2.9 (State Projection)** Let  $(B, Q_0)$  be a BIP system where  $B = \gamma(B_1, \dots, B_n)$  and let  $s = \langle s_1, \dots, s_n \rangle$  be a state of  $(B, Q_0)$ . Let  $\{B_{i_1}, \dots, B_{i_k}\} \subseteq \{B_1, \dots, B_n\}$ . Then  $s \upharpoonright \{B_{i_1}, \dots, B_{i_k}\} \triangleq \langle s_{i_1}, \dots, s_{i_k} \rangle$ . For a single  $B_i$ , we write  $s \upharpoonright B_i = s_i$ . We extend state projection to sets of states element-wise.

**Definition 2.10 (Subcomponent)** Let  $B \triangleq \gamma(B_1, \dots, B_n)$  be a composite component, and let  $\{B_{i_1}, \dots, B_{i_k}\}$  be a subset of  $\{B_1, \dots, B_n\}$ . Let  $P' = P_{i_1} \cup \dots \cup P_{i_k}$ , i.e., the union of the ports of  $\{B_{i_1}, \dots, B_{i_k}\}$ . Then the subcomponent  $B'$  of  $B$  based on  $\{B_{i_1}, \dots, B_{i_k}\}$  is as follows:

- (1)  $\gamma' \triangleq \{a \cap P' \mid a \in \gamma \wedge a \cap P' \neq \emptyset\}$
- (2)  $B' \triangleq \gamma'(B_{i_1}, \dots, B_{i_k})$

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

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

**Definition 2.11 (Subsystem)** Let  $(B, Q_0)$  be a BIP system where  $B = \gamma(B_1, \dots, B_n)$ , and let  $\{B_{i_1}, \dots, B_{i_k}\}$  be a subset of  $\{B_1, \dots, B_n\}$ . Then the subsystem  $(B', Q'_0)$  of  $(B, Q_0)$  based on  $\{B_{i_1}, \dots, B_{i_k}\}$  is as follows:

- (1)  $B'$  is the subcomponent of  $B$  based on  $\{B_{i_1}, \dots, B_{i_k}\}$
- (2)  $Q'_0 = Q_0 \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$

**Definition 2.12 (Execution Projection)** Let  $(B, Q_0)$  be a BIP system where  $B = \gamma(B_1, \dots, B_n)$ , and let  $(B', Q'_0)$ , with  $B' = \gamma'(B_{i_1}, \dots, B_{i_k})$  be the subsystem of  $(B, Q_0)$  based on  $\{B_{i_1}, \dots, B_{i_k}\}$ . Let  $P' = P_{i_1} \cup \dots \cup P_{i_k}$ , i.e.,  $P'$  is the set of ports of  $(B', Q'_0)$ . Let  $\rho = s_0 a_1 s_1 \dots s_{j-1} a_j s_j \dots$  be an execution of  $(B, Q_0)$ . Then,  $\rho \upharpoonright (B', Q'_0)$ , the projection of  $\rho$  onto  $(B', Q'_0)$ , is the sequence resulting from:

- (1) replacing each  $s_j$  by  $s_j \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$ , i.e., replacing each state by its projection onto  $\{B_{i_1}, \dots, B_{i_k}\}$
- (2) removing all  $a_j s_j$  where  $a_j \cap P' = \emptyset$
- (3) replacing each  $a_j$  by  $a_j \cap P'$ , i.e., replacing each interaction by its projection onto the port set  $P'$

**Proposition 2.13 (Execution Projection)** Let  $(B, Q_0)$  be a BIP system where  $B = \gamma(B_1, \dots, B_n)$ , and let  $(B', Q'_0)$ , with  $B' = \gamma'(B_{i_1}, \dots, B_{i_k})$  be the subsystem of  $(B, Q_0)$  based on  $\{B_{i_1}, \dots, B_{i_k}\}$ . Let  $P' = P_{i_1} \cup \dots \cup P_{i_k}$ , i.e., the union of the ports of  $\{B_{i_1}, \dots, B_{i_k}\}$ . Let  $\rho = s_0 a_1 s_1 \dots s_{j-1} a_j s_j \dots$  be an execution of  $(B, Q_0)$ . Then,  $\rho \upharpoonright (B', Q'_0)$  is an execution of  $(B', Q'_0)$ .

*Proof.* By Definitions 2.9, 2.11, and 2.12, we have  $\rho \upharpoonright (B', Q'_0) = s'_0 b_1 s'_1 b_2 s'_2 \dots$  for some  $s'_0, b_1 s'_1 b_2 s'_2 \dots$ , where  $s'_j \in Q' = Q \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$  for  $j \geq 0$ . Also by Definitions 2.9, 2.11, and 2.12, we have  $s'_0 \in Q'_0 = Q_0 \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$ , since  $s'_0 = s_0 \upharpoonright B'$ , and  $s_0 \in Q_0$ , by Definition 2.7.

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

- (1)  $s'_j = s_\ell \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$  for some  $\ell > 0$  and such that  $a_\ell \cap P' \neq \emptyset$
- (2)  $b_j = a_\ell \cap P'$
- (3)  $s'_{j-1} = s_m \upharpoonright \{B_{i_1}, \dots, B_{i_k}\}$  for the smallest  $m$  such that  $m < \ell$  and  $\forall m' : m + 1 \leq m' < \ell : a_{m'} \cap P' = \emptyset$

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

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

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

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

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

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

### 3. CHARACTERIZING DEADLOCK-FREEDOM

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

**Definition 3.2 (Local Deadlock-freedom)** *A BIP-system  $(B, Q_0)$  is free of local deadlock iff, for every subsystem  $(B', Q'_0)$  of  $(B, Q_0)$ , and every reachable state  $s$  of  $(B, Q_0)$ ,  $(B', Q'_0)$  has some interaction enabled in state  $s \upharpoonright B'$ . Formally:*  
*for every subsystem  $(B', Q'_0)$  of  $(B, Q_0)$ :*  
 $\forall s \in rstates(B, Q_0), \exists a : s \upharpoonright B' \xrightarrow{a} B'$ .

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

Proposition 3.8 states that the existence of a supercycle is necessary for a local deadlock to occur: if a set of components, *considered in isolation*, are blocked, then there exists a supercycle consisting of exactly those components, together with the interactions that each component enables.

#### 3.1. Wait-for graphs

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

**Definition 3.3 (Wait-for-graph  $W_B(s)$ )** *Let  $B = \gamma(B_1, \dots, B_n)$  be a BIP composite component, and let  $s = \langle s_1, \dots, s_n \rangle$  be an arbitrary state of  $B$ . The wait-for-graph  $W_B(s)$  of  $s$  is a directed bipartite and-or graph, where*

1. *the nodes of  $W_B(s)$  are as follows:*
  - (a) *the and-nodes are the atomic components  $B_i, i \in \{1..n\}$ ,*
  - (b) *the or-nodes are the interactions  $a \in \gamma$ ,*
2. *there is an edge in  $W_B(s)$  from  $B_i$  to every node  $a$  such that  $B_i \in components(a)$  and  $s_i(enb_a^i) = true$ , i.e., from  $B_i$  to every interaction which  $B_i$  enables in  $s_i$ ,*

3. *there is an edge in  $W_B(s)$  from  $a$  to every  $B_i$  such that  $B_i \in \text{components}(a)$  and  $s_i(\text{enb}_a^i) = \text{false}$ , i.e., from  $a$  to every component  $B_i$  which participates in  $a$  but does not enable it, in state  $s_i$ .*

A component  $B_i$  is an and-node since all of its successor actions (or-nodes) must be disabled for  $B_i$  to be incapable of executing. An interaction  $a$  is an or-node since it is disabled if any of its participant components do not enable it. An edge (path) in a wait-for-graph is called a wait-for-edge (wait-for-path). Write  $a \rightarrow B_i$  ( $B_i \rightarrow a$  respectively) for a wait-for-edge from  $a$  to  $B_i$  ( $B_i$  to  $a$  respectively). We abuse notation by writing  $e \in W_B(s)$  to indicate that  $e$  (either  $a \rightarrow B_i$  or  $B_i \rightarrow a$ ) is an edge in  $W_B(s)$ . Also  $B_i \rightarrow a \rightarrow B'_i \in W_B(s)$  for  $B_i \rightarrow a \in W_B(s) \wedge a \rightarrow B'_i \in W_B(s)$ , i.e., for a wait-for-path of length 2, and similarly for longer wait-for-paths.

Consider the dining philosophers system given in Figure 1. Figure 2(a) shows its wait-for-graph in its sole initial state. Figure 2(b) shows the wait-for-graph after execution of  $\text{get}_0$ . Edges from components to interactions are shown solid, and edges from interactions to components are shown dashed.

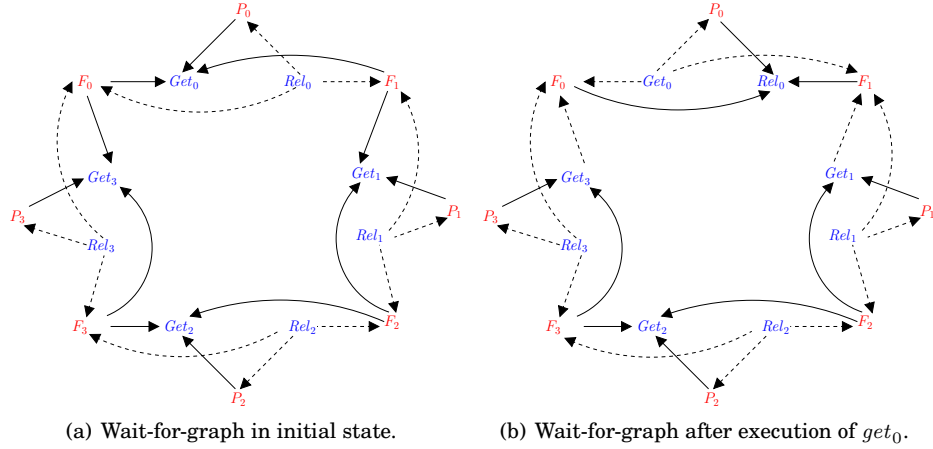


Fig. 2. Example wait-for-graphs for dining philosophers system of Figure 1.

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

**Proposition 3.4 (Wait-for edge preservation)** *Let  $s \xrightarrow{a} t$  be a transition of composite component  $B = \gamma(B_1, \dots, B_n)$ , and let  $e$  be a wait-for edge in  $W_B(s)$  that is not  $B_i$ -incident, for every  $B_i \in \text{components}(a)$ . Then  $e \in W_B(s)$  iff  $e \in W_B(t)$ .*

*Proof.* Fix  $e$  to be an arbitrary wait-for-edge that is not  $B_i$ -incident.  $e$  is either  $B_j \rightarrow b$  or  $b \rightarrow B_j$ , for some component  $B_j$  of  $B$  that is not in  $\text{components}(a)$ , and an interaction  $b$  (different from  $a$ ) that  $B_j$  participates in. Now  $s|B_j = t|B_j$ , since  $s \xrightarrow{a} t$  and  $B_j \notin \text{components}(a)$ . Hence  $s(\text{enb}_b^j) = t(\text{enb}_b^j)$ . It follows from Definition 3.3 that  $e \in W_B(s)$  iff  $e \in W_B(t)$ .  $\square$



### 3.2. Supercycles and deadlock-freedom

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

**Definition 3.5 (Supercycle)** Let  $B = \gamma(B_1, \dots, B_n)$  be a composite component and  $s$  be a state of  $B$ . A subgraph  $SC$  of  $W_B(s)$  is a supercycle in  $W_B(s)$  if and only if all of the following hold:

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

where  $a \in SC$  means that  $a$  is a node in  $SC$ , etc. Also, write  $SC \subseteq W_B(s)$  when  $SC$  is a subgraph of  $W_B(s)$ .

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

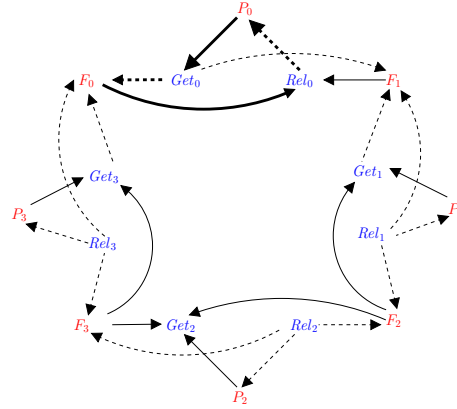


Fig. 3. Example supercycle for dining philosophers system of Figure 1.

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

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 neither  $B_5$  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

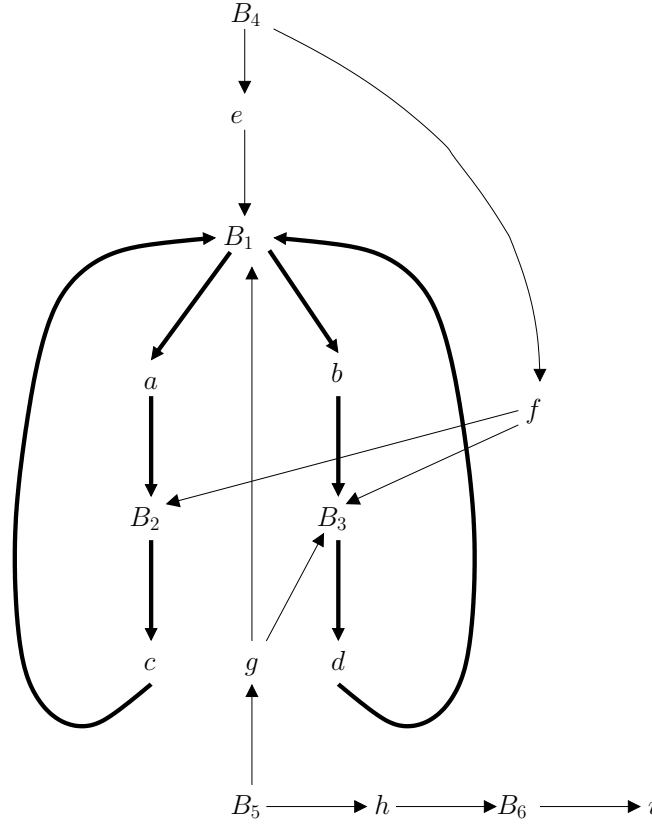


Fig. 4. Example supercycle that is not a simple cycle

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

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

*Proof.* Let  $B_i$  be an arbitrary component in  $SC$ . By Definition 3.5, every interaction that  $B_i$  enables has a wait-for-edge to some other component  $B_j$  in  $SC$  and so cannot be executed in state  $s$ . Hence in any transition from  $s$  to another global state  $t$ , all of

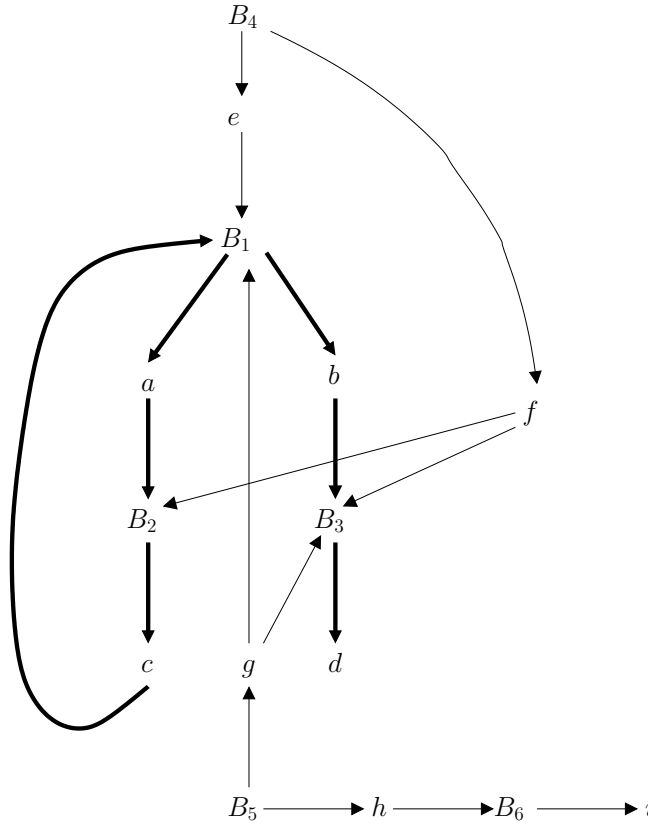


Fig. 5. Example where a wait-for cycle does not imply deadlock

the components  $B_i$  in  $SC$  remain in the same local state. Hence  $SC \subseteq W_B(t)$ , i.e., the same supercycle  $SC$  remains in global state  $t$ . Repeating this argument from state  $t$  and onwards leads us to conclude that  $SC \subseteq W_B(u)$  for any state  $u$  reachable from  $s$ .  $\square$

Proposition 3.8 states that the existence of a supercycle is necessary for a local deadlock to occur: if a set of components, *considered in isolation*, are blocked, then there exists a supercycle consisting of exactly those components, together with the interactions that each component enables.

**PROPOSITION 3.8.** *Let  $B'$  be a subcomponent of  $B$ , and let  $s$  be an arbitrary state of  $B$  such that  $B'$ , when considered in isolation, has no enabled interaction in state  $s|B'$ . Then,  $W_B(s)$  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  $W_B(s)$  induced by:

- (1) the atomic components of  $B'$ ,
- (2) the interactions  $a$  that each atomic component  $B_i$  enables, and the edges  $B_i \rightarrow a$ , and

- (3) the edges  $a \rightarrow B_j$  from each interaction to some atomic component  $B_j$  in  $B'$  that does not enable  $B_j$ .

$SC$  satisfies Definition 3.5 and so is a supercycle.  $\square$

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

**Corollary 3.9 (Supercycle-free implies free of local deadlock)** *If, for every reachable state  $s$  of  $(B, Q_0)$ ,  $W_B(s)$  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  $s|B'$ . By Proposition 3.8,  $W_B(s)$  contains a supercycle.  $\square$

In the sequel, we say “deadlock-free” to mean “free of local deadlock”.

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

### 3.3. Structural properties of supercycles

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

**Definition 3.10 (Path, path length)** *Let  $G$  be a directed graph and  $v$  a vertex in  $G$ . A path  $\pi$  in  $G$  is a finite sequence  $v_0, v_1, \dots, v_n$  such that  $(v_i, v_{i+1})$  is an edge in  $G$  for all  $i \in \{0, \dots, n-1\}$ . Write  $path_G(\pi)$  iff  $\pi$  is a path in  $G$ . Define  $first(\pi) = v_0$  and  $last(\pi) = v_n$ . Let  $|\pi|$  denote the length of  $\pi$ , which we define as follows:*

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

**Definition 3.11 (In-depth, Out-depth)** *Let  $G$  be a directed graph and  $v$  a vertex in  $G$ . Define the in-depth of  $v$  in  $G$ , notated as  $in\_depth_G(v)$ , as follows:*

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

Formally,  $\text{out\_depth}_G(v) = (\text{MAX } \pi : \text{path}_G(\pi) \wedge \text{first}(\pi) = v : |\pi|)$ .

We use  $\text{in\_depth}_B(v, s)$  for  $\text{in\_depth}_{W_B(s)}(v)$ , and also  $\text{out\_depth}_B(v, s)$  for  $\text{out\_depth}_{W_B(s)}(v)$ .

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

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

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

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

**PROPOSITION 3.14.** *Every supercycle  $SC$  contains at least one cycle.*

*Proof.* By contradiction. Suppose that  $SC$  is a supercycle and is also acyclic. Then every path in  $SC$  is simple, and therefore finite. Hence every node in  $SC$  has finite out-depth. By Proposition 3.12,  $SC$  cannot be a supercycle.  $\square$

**PROPOSITION 3.15.** *Let  $B = \gamma(B_1, \dots, B_n)$  be a composite component and  $s$  a state of  $B$ . Let  $SC$  be a supercycle in  $W_B(s)$ , and let  $SC'$  be the graph obtained from  $SC$  by removing all vertices of finite in-depth and their incident edges. Then  $SC'$  is also a supercycle in  $W_B(s)$ .*

*Proof.* A vertex with finite in-depth cannot lie on a cycle in  $SC$ . Hence by Proposition 3.14,  $SC' \neq \emptyset$ . Thus  $SC'$  satisfies clause (1) of the supercycle definition (3.5). Let  $v$  be an arbitrary vertex of  $SC'$ . Thus  $v \in SC$  and  $\text{in\_depth}_{SC}(v) = \omega$  by definition of  $SC'$ . Let  $w$  be an arbitrary successor of  $v$  in  $SC$ .  $\text{in\_depth}_{SC}(w) = \omega$  by Definition 3.11. Hence  $w \in SC'$ , by definition of  $SC'$ . Furthermore,  $w$  is a successor of  $v$  in  $SC'$ , since  $SC'$  consists of all nodes of  $SC$  with infinite in-depth. Hence the successors of  $v$  in  $SC'$  are the same as the successors of  $v$  in  $SC$ . Now since  $SC$  is a supercycle, every vertex  $v$  in  $SC$  has enough successors in  $SC$  to satisfy clauses (2) and (3) of the supercycle definition (3.5). It follows that every vertex  $v$  in  $SC'$  has enough successors in  $SC'$  to satisfy clauses (2) and (3) of the supercycle definition (3.5).  $\square$

**PROPOSITION 3.16.** *Every supercycle  $SC$  contains a maximal strongly connected component  $CC$  such that (1)  $CC$  is itself a supercycle, and (2) there is no wait-for-edge from a node in  $CC$  to a node outside of  $CC$ .*

*Proof.*  $SC$  is a directed graph, and so consider the decomposition of  $SC$  into its maximal strongly connected components (MSCC). Let  $\overline{SC}$  be the graph resulting from replacing each MSCC by a single node. By its construction,  $\overline{SC}$  is acyclic, and so contains at least one node  $x$  with no outgoing edges. Let  $CC$  be the MSCC corresponding to  $x$ . It follows that  $CC$  is nonempty, and hence  $CC$  satisfies clause (1) of the supercycle definition (3.5). It also follows from the construction of  $CC$  that no node in  $CC$  has a wait-for-edge going to a node outside of  $CC$ , and so Clause (2) of the Proposition is established.

Let  $v$  be an arbitrary node in  $CC$ . Since  $CC \subseteq SC$ ,  $v$  is a node of  $SC$ . Let  $w$  be an arbitrary successor of  $v$  in  $SC$ . Since no node in  $CC$  has an edge going to a node outside of  $CC$ , it follows that  $w$  is a node of  $CC$ . Hence  $v$  has the same successors in  $CC$  as in  $SC$ . Now since  $SC$  is a supercycle, every vertex  $v$  in  $SC$  has enough successors in  $SC$  to satisfy clauses (2) and (3) of the supercycle definition (3.5). It follows that every vertex  $v$  in  $CC$  has enough successors in  $CC$  to satisfy clauses (2) and (3) of the supercycle definition (3.5).

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

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

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

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

## 4. SUPERCYCLE FORMATION AND ITS CONSEQUENCES

### 4.1. Supercycle Membership

**Definition 4.1 (Supercycle membership,  $scyc_B(s, v)$ )** *Let  $v$  be a node of  $W_B(s)$ . Then  $scyc_B(s, v)$  holds iff there exists a supercycle  $SC \subseteq W_B(s)$  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  $preds_B(s, v) = \{w \mid w \rightarrow v \in W_B(s)\}$  and  $succs_B(s, v) = \{w \mid v \rightarrow w \in W_B(s)\}$ . The definition of a supercycle (Def. 3.5) imposes certain constraints on supercycle membership of a node w.r.t. its predecessors and successors in the wait-for-graph, as follows:

**Proposition 4.2 (Supercycle-membership constraints)** *Let  $a, B_i$  be nodes of  $W_B(s)$ . Then*

- (1)  $scyc_B(s, B_i) \Leftrightarrow (\forall a \in succs_B(s, B_i) : scyc_B(s, a))$ .
- (2)  $scyc_B(s, B_i) \Rightarrow (\forall a \in preds_B(s, B_i) : scyc_B(s, a))$ .
- (3)  $scyc_B(s, a) \Leftrightarrow (\exists B_i \in succs_B(s, a) : scyc_B(s, B_i))$ .
- (4)  $scyc_B(s, a) \Leftarrow (\exists B_i \in preds_B(s, a) : scyc_B(s, B_i))$ .

**PROOF.** We deal with each clause in turn.

*Proof of Clause 1.* Assume  $scyc_B(s, B_i)$ , and let  $SC \subseteq W_B(s)$  be the supercycle containing  $B_i$ . Let  $aa \in succs_B(s, B_i)$ . By Def. 3.5, Clause 2,  $aa \in SC$ . Hence  $(\forall a \in succs_B(s, B_i) : scyc_B(s, a))$ . We conclude  $scyc_B(s, B_i) \Rightarrow (\forall a \in succs_B(s, B_i) : scyc_B(s, a))$ .

Now assume  $(\forall a \in \text{succs}_B(s, B_i) : \text{scyc}_B(s, a))$ , and let  $SC$  be the union of all the supercycles containing all the  $a \in \text{succs}_B(s, B_i)$ . By Prop. 3.17,  $SC \subseteq W_B(s)$  is a supercycle. Let  $SC'$  be  $SC$  with  $B_i \rightarrow a$  added, for all  $a \in \text{succs}_B(s, B_i)$ . Then  $SC'$  is a supercycle by Def. 3.5, and also  $SC' \subseteq W_B(s)$ . Hence  $\text{scyc}_B(s, a)$ . We conclude  $\text{scyc}_B(s, B_i) \Leftarrow (\forall a \in \text{succs}_B(s, B_i) : \text{scyc}_B(s, a))$ .

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

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

*Proof of Clause 4.* Assume  $\neg \text{scyc}_B(s, a)$ , so that  $a$  is not in any supercycle of  $W_B(s)$ . Let  $B_i \in \text{preds}_B(s, a)$ . By Def. 3.5, Clause 2,  $B_i$  cannot be in any supercycle of  $W_B(s)$ , since all  $a \in \text{succs}_B(s, B_i)$  must also be in the supercycle. Hence  $\neg \text{scyc}_B(s, B_i)$ . Since  $B_i$  was chosen arbitrarily, we conclude  $\neg \text{scyc}_B(s, a) \Rightarrow (\forall B_i \in \text{preds}_B(s, a) : \neg \text{scyc}_B(s, B_i))$ , the contrapositive of Clause 4.  $\square$

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

While Prop. 4.2 gives relationships between supercycle membership of a node and both its successors and predecessors, nevertheless Def. 3.5 implies that the “causality” of supercycle-membership of a node  $v$  is from the successors of  $v$  to  $v$ , i.e., membership of  $v$  in a supercycle is caused only by membership of  $v$ ’s successors in a supercycle. Repeating this step, we infer that  $v$ ’s supercycle-membership is caused by the subgraph of the wait-for graph that is reachable from  $v$ .

Hence, we follow outgoing wait-for edges in computing supercycle-membership. Actually, it turns out to be easier to compute the negation of supercycle membership, which we call *supercycle violation*. This is because supercycle-violation has a base case: when a node has no outgoing wait-for edges. We need a base case, and an inductive definition, because a node that is not in any supercycle may nevertheless be a node of a wait-for cycle, since a cycle of wait-for-edges does not necessarily imply a supercycle. 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  $\text{scViolate}_B(v, d, t)$ , defined by induction on  $d$ .

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

---

```

scViolateB(v, d, t)
0. if (d = 1 ∧ v = a ∧ ¬(∃ Bi : a → Bi ∈ WB(t))) return(tt) fi    ▷base case for tt result
1. if (v = Bi ∧ (∃ a : Bi → a ∈ WB(t) : (∃ d' : 1 ≤ d' < d : scViolateB(a, d', t)))) return(tt) fi
2. if (v = a ∧ (∀ Bi : a → Bi ∈ WB(t) : (∃ d' : 1 ≤ d' < d : scViolateB(Bi, d', t)))) return(tt) fi
3. return(ff)    ▷no case for tt result, so result is ff

```

---

Fig. 6. Formal definition of  $\text{scViolate}_B(v, d, t)$ 

Base case,  $d = 1$ .  $\text{scViolate}_B(v, 1, t)$  iff  $v$  is an interaction  $a$  and it has no outgoing wait-for-edges, otherwise  $\neg \text{scViolate}_B(v, 1, t)$ . *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,  $d > 1$ .  $\text{scViolate}_B(v, d, t)$  iff any of the following cases hold. Otherwise  $\neg \text{scViolate}_B(v, d, t)$ .

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

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

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

**Proposition 4.4 (Soundness of supercycle violation w.r.t. supercycle non-membership)** *If  $(\exists d \geq 1 : \text{scViolate}_B(v, d, t))$  then  $\neg \text{scyc}_B(t, v)$ , i.e.,  $v$  is not a node of a supercycle in  $W_B(t)$ .*

*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  $B_i \rightarrow a \in W_B(t)$  and  $a$  has a level- $(d - 1)$  SC-violation. By the induction hypothesis,  $\neg \text{scyc}_B(t, a)$ . By Prop. 4.2, Clause 1,  $\neg \text{scyc}_B(t, B_i)$ .

*Case 2:*  $v$  is an interaction  $a$ . Hence for all components  $B_i$  such that  $a \rightarrow B_i \in W_B(t)$ ,  $B_i$  has a level- $(d - 1)$  SC-violation. By the induction hypothesis,  $(\forall B_i : a \rightarrow B_i \in W_B(t) : \neg \text{scyc}_B(t, B_i))$ . By Prop. 4.2, Clause 3,  $\neg \text{scyc}_B(t, a)$ .  $\square$



**Proposition 4.5 (Completeness of supercycle violation w.r.t. supercycle non-membership)** *If  $(\forall d \geq 1 : \neg \text{scViolate}_B(v, d, t))$  then  $\text{scyc}_B(t, v)$ , i.e.,  $v$  is a node of a supercycle in  $W_B(t)$ .*

*Proof.* Let  $V$  be the set of nodes in  $W_B(t)$  with a supercycle-violation, i.e.,  $V = \{w \mid w \in W_B(t) \wedge (\exists d : \text{scViolate}_B(w, d, t))\}$ . Let  $\bar{V}$  be the remaining nodes, i.e., all nodes in  $W_B(t)$  that do not have a supercycle-violation, so  $\bar{V} = \{w \mid w \in W_B(t) \wedge (\forall d \geq 1 : \neg \text{scViolate}_B(w, d, t))\}$ .

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

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

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

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

**PROPOSITION 4.6.**  $\neg \text{scyc}_B(t, v)$  iff  $(\exists d \geq 1 : \text{scViolate}_B(v, d, t))$ .

*Proof.* Immediate from Propositions 4.4 and 4.5.  $\square$

#### 4.2. The supercycle formation condition

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

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

- (1)  $CC$  is a subgraph of  $W_B(t)$
- (2)  $CC$  is strongly connected
- (3)  $CC$  is a supercycle
- (4) in  $W_B(t)$ , there is no wait-for edge from a node in  $CC$  to a node outside of  $CC$
- (5) there exists a component  $B_i \in \text{components}(a)$  such that  $B_i$  is in  $CC$

**PROOF.** By assumption, there is a supercycle  $SC$  that is a subgraph of  $W_B(t)$ . By Proposition 3.16,  $SC$  contains a subgraph  $CC$  that is strongly connected, is itself a supercycle, and such that there is no wait-for-edge from a node in  $CC$  to a node outside of  $CC$ . This establishes Clauses 1–4.

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

### 4.3. General supercycle violation condition

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

We formalize the second condition as follows.

**Definition 4.8 (Strong connectedness violation,  $\text{sConnViolate}_B(v, t)$ )** *Let  $v$  be a node of  $W_B(t)$ . Then  $\text{sConnViolate}_B(v, t)$  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 4.9 (General supercycle violation,  $\text{genViolate}_B(v, t)$ )** *Let  $v$  be a node of  $W_B(t)$ . Then  $\text{genViolate}_B(v, t) \triangleq (\exists d \geq 1 : \text{scViolate}_B(v, d, t)) \vee \text{sConnViolate}_B(v, t)$ .*

Let  $s \xrightarrow{a} t$  be a reachable transition. If, for every  $B_i \in \text{components}(a)$ ,  $\text{genViolate}_B(v, t)$  holds, then, as we show below,  $s \xrightarrow{a} t$  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  $(\exists d \geq 1 : \text{scViolate}_B(v, d, t))$  implies that  $v$  cannot be in a supercycle. Hence,  $v$  cannot be in a strongly-connected supercycle. Hence  $(\exists d \geq 1 : \text{scViolate}_B(v, d, t))$  implies  $\text{sConnViolate}_B(v, t)$ . It is however convenient to state the formation violation condition in this manner, since we will formulate a local version for each of  $(\exists d \geq 1 : \text{scViolate}_B(v, d, t))$  and  $\text{sConnViolate}_B(v, t)$ , and the implication does not necessarily hold for the local versions.

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

#### 4.4. Abstract supercycle freedom conditions

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

**Definition 4.10 (Behavioral restriction condition)** A behavioral restriction condition  $BC$  is a predicate  $BC : (B, Q_0, a) \rightarrow \{\text{tt}, \text{ff}\}$ .

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

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

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

**Theorem 4.12 (Deadlock-freedom via supercycle-freedom preserving restriction)** Assume that

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

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

**PROOF.** Let  $u$  be an arbitrary reachable state. The proof is by induction on the length of the finite execution  $\alpha$  that ends in  $u$ . Assumption 1 provides the base case, for  $\alpha$  having length 0, and so  $u \in Q_0$ . For the induction step, we establish: for every reachable transition  $s \xrightarrow{a} t$ ,  $W_B(s)$  is supercycle-free implies that  $W_B(t)$  is supercycle-free. This is immediate from Assumption 2, and Definition 4.11.  $\square$

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

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

Assume that

- (1) for all  $s_0 \in Q_0$ ,  $W_B(s_0)$  is supercycle-free, and

- (2) for all  $a \in \gamma$ , there exists a supercycle-freedom preserving restriction  $BC$ :  $BC(B, Q_0, a) = \text{tt}$

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

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

#### 4.5. Overview of the four supercycle-freedom preserving restrictions

The supercycle formation condition (Proposition 4.7) tells us that, when a supercycle  $SC$  is created, some component  $B_i$  that participates in the interaction  $a$  whose execution created  $SC$ , must be a node of a strongly connected component  $CC$  of  $SC$ , and moreover  $CC$  is itself a supercycle in its own right. In a sense,  $CC$  is the “essential” part of  $SC$ .

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

If  $\mathcal{GACT}(B, Q_0, a)$  holds, and global state  $s$  is supercycle-free, and  $s \xrightarrow{a} t$ , 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  $\mathcal{GACT}(B, Q_0, a)$  holds for all interactions  $a \in \gamma$ , we obtain, by straightforward induction on length of executions, that every reachable state is supercycle-free.

It also follows that any condition which implies  $\mathcal{GACT}(B, Q_0, a)$  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  $\mathcal{GACT}$ , since it requires only the evaluation of lengths of wait-for-paths, i.e., it does not have the “alternating” character of  $\mathcal{GACT}$ .
- (2) To provide “local variants” of  $\mathcal{GACT}$  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  $B_i \in \text{components}(a)$  exhibits a supercycle-violation, as given by  $\text{scViolate}_B(B_i, d, t)$  (Definition 4.3).

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

$$\forall B_i \in \text{components}(a), \text{genViolate}_B(B_i, t).$$

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

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

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

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

Since  $CC$  is a strongly connected supercycle, we have, by Definition 4.8, that  $\neg \text{sConnViolate}_B(B_i, t)$  holds.

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

Hence, by Definition 4.9,  $\neg \text{genViolate}_B(B_i, t)$ . But, by Definition 5.1, we have  $\text{genViolate}_B(B_i, t)$ . Hence, we have the desired contradiction, and so the theorem holds.  $\square$

## 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  $\mathcal{GAL}\mathcal{T}$ . After execution of a reachable transition  $s \xrightarrow{a} t$  of  $(B, Q_0)$ , we consider the in-depth and out-depth of the components  $B_i \in \text{components}(a)$ . There are three cases:

*Case 1.*  $B_i$  has finite in-depth in  $W_B(t)$ : then, if  $B_i \in SC$ , it can be removed and still leave a supercycle  $SC'$ , by Proposition 3.15. Hence  $SC'$  exists in  $W_B(s)$ , and so  $B_i$  is not essential to the creation of a supercycle.

*Case 2.*  $B_i$  has finite out-depth in  $W_B(t)$ : by Proposition 3.12,  $B_i$  cannot be part of a supercycle, and so  $SC \subseteq W_B(s)$ .

*Case 3.*  $B_i$  has infinite in-depth and infinite out-depth in  $W_B(t)$ : in this case,  $B_i$  is possibly an essential part of  $SC$ , i.e.,  $SC$  was created in going from  $s$  to  $t$ .

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

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

$$\forall B_i \in \text{components}(a) : \text{in\_depth}_B(B_i, t) < \omega \vee \text{out\_depth}_B(B_i, t) < \omega.$$

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

**PROPOSITION 5.4.** *Assume that node  $v$  of  $W_B(t)$  has a finite in-depth of  $d$  in  $W_B(t)$ , i.e.,  $\text{in\_depth}_B(v, t) = d$ . Then  $\text{sConnViolate}_B(v, t)$ .*

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

**PROPOSITION 5.5.** *Assume that node  $v$  of  $W_B(t)$  has a finite out-depth of  $d$  in  $W_B(t)$ , i.e.,  $\text{out\_depth}_B(v, t) = d$ . Then  $\text{scViolate}_B(v, d + 1, t)$ .*

*Proof.* Proof is by induction on  $d$ .

**Base case,  $d = 0$ .** Hence by  $out\_depth_B(v, t) = 0$  and Definitions 3.10 and 3.11,  $v$  has no outgoing wait-for-edges in  $W_B(t)$ . Hence by Definition 4.3,  $scViolate_B(v, 1, t)$ .

**Inductive step,  $d > 0$ .** Let  $u$  be an arbitrary successor of  $v$ , i.e., a node  $u$  such that  $v \rightarrow u \in W_B(t)$ . By Definitions 3.10 and 3.11,  $u$  has an out-depth  $d'$  that is less than  $d$ . That is,  $out\_depth_B(u, t) = d' < d$ . By the induction hypothesis applied to  $d'$ , we obtain  $scViolate_B(u, d' + 1, t)$ . Hence by Definition 4.3, Clauses 1 and 2,  $scViolate_B(v, d + 1, t)$ .  $\square$

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

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

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

By Propositions 5.4 and 5.5,

For every reachable transition  $s \xrightarrow{a} t$  of  $(B, Q_0)$ ,  
 $\forall B_i \in components(a) : sConnViolate_B(B_i, t) \vee (\exists d \geq 1 : scViolate_B(B_i, d, t))$ .

Hence by Definition 4.9,

For every reachable transition  $s \xrightarrow{a} t$  of  $(B, Q_0)$ ,  
 $\forall B_i \in components(a) : genViolate_B(B_i, t)$

Hence  $\mathcal{GACT}(B, Q_0, a)$  holds.  $\square$

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

*Proof.* Follows immediately from Lemma 5.6 and Theorem 5.2.  $\square$

### 5.3. Deadlock freedom using global restrictions

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

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

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

*Proof.* Immediate from Theorems 5.2, 5.7 and Corollary 4.13.  $\square$

## 6. LOCAL CONDITIONS FOR DEADLOCK FREEDOM

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

### 6.1. Projection onto Subsystems

**Definition 6.1 (Structure Graph  $G_B$ ,  $G_a^\ell$ )** The structure graph  $G_B$  of composite component  $B = \gamma(B_1, \dots, B_n)$  is a bipartite graph whose nodes are the  $B_1, \dots, B_n$  and all the  $a \in \gamma$ . There is an edge between  $B_i$  and interaction  $a$  iff  $B_i$  participates in  $a$ , i.e.,  $B_i \in components(a)$ . Define the distance between two nodes to be the number of edges

in a shortest path between them. Let  $G_a^\ell$  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  $\ell$ .

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

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

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

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

*Proof.* By Definition 3.3,  $a \rightarrow B_i \in W_B(s)$  iff  $s|i(\text{enb}_a^{B_i}) = \text{false}$ . Since  $s' = s|B'$ , we have  $s'|i = s|i$ . Hence  $s|i(\text{enb}_a^{B_i}) = s'|i(\text{enb}_a^{B_i})$ . By Definition 3.3,  $a \rightarrow B_i \in W_{B'}(s')$  iff  $s'|i(\text{enb}_a^{B_i}) = \text{false}$ . Putting together these three equalities gives us clause (1).

By Definition 3.3,  $B_i \rightarrow a \in W_B(s)$  iff  $s|i(\text{enb}_a^{B_i}) = \text{true}$ . Since  $s' = s|B'$ , we have  $s'|i = s|i$ . Hence  $s|i(\text{enb}_a^{B_i}) = s'|i(\text{enb}_a^{B_i})$ . By Definition 3.3,  $B_i \rightarrow a \in W_{B'}(s')$  iff  $s'|i(\text{enb}_a^{B_i}) = \text{true}$ . Putting the above three equalities together gives us clause (2).  $\square$

## 6.2. A Local AND-OR Condition for Deadlock Freedom

We now seek a local condition, which we evaluate in  $D_a^\ell$ , and which implies  $\mathcal{GAL}$ . We define local versions of both  $\text{scViolate}_B(v, d, t)$  and  $\text{sConnViolate}_B(v, t)$ .

To achieve a local and conservative approximation of  $\text{scViolate}_B(v, d, t)$ , we make the “pessimistic” assumption that the violation status of border nodes of  $D_a^\ell$  cannot be known, since it depends on nodes outside of  $D_a^\ell$ . Now, if an internal node  $v$  of  $D_a^\ell$  can be marked with a level  $d$  sc-violation, by applying Definition 4.3 only within  $D_a^\ell$ , 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  $\text{sConnViolate}_B(v, t)$ , we project onto a subsystem.

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

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

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

Inductive step,  $d > 1$ .  $\text{scViolateLoc}(v, d, t_a, D_a^\ell)$  iff either of the following two cases hold. Otherwise  $\neg \text{scViolateLoc}(v, d, t_a, D_a^\ell)$ .

- (1)  $v$  is a component  $B_i$  and there exists an interaction  $aa$  such that  $B_i \rightarrow aa \in W_{D_a^\ell}(t_a)$  and  $(\exists d' : 1 \leq d' < d : \text{scViolateLoc}(aa, d', t_a, D_a^\ell))$ . That is,  $B_i$  enables an interaction  $aa$  which has a level- $d'$  supercycle-violation in  $D_a^\ell$ , for some  $d' < d$ .
- (2)  $v$  is an interaction  $aa$  and an internal node of  $D_a^\ell$  and for all components  $B_i$  such that  $aa \rightarrow B_i \in W_{D_a^\ell}(t_a)$ , we have  $(\exists d' : 1 \leq d' < d : \text{scViolateLoc}(B_i, d', t_a, D_a^\ell))$ . That is, each component  $B_i$  that  $aa$  waits for has a level- $d'$  supercycle-violation in  $D_a^\ell$ , for some  $d' < d$

Note that if  $v$  is an interaction  $aa$  and a border node, then  $\text{scViolateLoc}(aa, d, t_a, D_a^\ell)$  is false, for all  $d$ . This is because  $aa$  has some component that is outside  $D_a^\ell$ , 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  $\text{scViolateLoc}(v, d, t_a, D_a^\ell)$ . The notation  $v = B_i$  means that  $v$  is some component  $B_i$ . Likewise,  $v = aa$  means that  $v$  is some interaction  $a$ , and “ $v = aa$  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.

$\text{scViolateLoc}(v, d, t_a, D_a^\ell)$

▷ Precondition:  $v$  is a node of  $D_a^\ell$  and  $d \geq 1$

0. if  $(d = 1 \wedge v = aa \text{ is interior} \wedge \neg(\exists B_i : aa \rightarrow B_i \in W_{D_a^\ell}(t_a)))$  **return**(tt);

1. if  $(v = aa \text{ is interior} \wedge (\forall B_i : aa \rightarrow B_i \in W_{D_a^\ell}(t_a) : (\exists d' : 1 \leq d' < d : \text{scViolateLoc}(B_i, d', t_a, D_a^\ell))))$  **return**(tt);

2. if  $(v = B_i \wedge (\exists aa : B_i \rightarrow aa \in W_{D_a^\ell}(t_a) : (\exists d' : 1 \leq d' < d : \text{scViolateLoc}(aa, d', t_a, D_a^\ell))))$  **return**(tt);

3. **return**(ff)

Fig. 7. Formal definition of  $\text{scViolateLoc}(v, d, t_a, D_a^\ell)$ .

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

**PROPOSITION 6.6.** *Let  $t$  be an arbitrary reachable state of BIP-system  $(B, Q_0)$ . For all interactions  $a \in \gamma$ , and  $\ell \geq 1$ , let  $t_a = t \downarrow D_a^\ell$ . Then*

$$\forall d \geq 1 : \text{scViolateLoc}(v, d, t_a, D_a^\ell) \Rightarrow \text{scViolate}_B(v, d, t).$$

*Proof.* Proof is by induction on  $d$ .

Base case,  $d = 1$ . Assume  $\text{scViolateLoc}(v, 1, t_a, D_a^\ell)$  for some node  $v$ . Then, by Figure 7,  $v$  is an interior node and an interaction  $aa$  of  $D_a^\ell$ , and has no outgoing wait-for edges. Therefore, in  $W_B(t)$ , it is still the case that  $v$  has no outgoing wait-for edges. Hence  $\text{scViolate}_B(v, 1, t)$  holds.

Inductive step,  $d > 1$ . Assume  $\text{scViolateLoc}(v, d, t_a, D_a^\ell)$  for some node  $v$  and some  $d > 1$ . We proceed by cases on Figure 7.

- (1)  $v$  is an interior interaction  $aa$  and  $(\forall B_i : aa \rightarrow B_i \in W_{D_a^\ell}(t_a) : (\exists d' : 1 \leq d' < d : \text{scViolateLoc}(B_i, d', t_a, D_a^\ell)))$ . Choose an arbitrary  $B_i$  such that  $aa \rightarrow B_i \in W_{D_a^\ell}(t_a)$ . By the induction hypothesis applied to  $\text{scViolateLoc}(B_i, d', t_a, D_a^\ell)$ , we have  $\text{scViolate}_B(B_i, d', t)$  for some



- $d' < d$ . Since  $W_{D_a^\ell}(t_a) \subseteq W_B(t)$  by construction, we have  $aa \rightarrow B_i \in W_B(t)$  and  $\text{scViolate}_B(B_i, d', t)$ . Hence by Definition 4.3, Clause 1, we have  $\text{scViolate}_B(v, d, t)$ .
- (2)  $v$  is a component  $B_i$  and

$$(\exists aa : B_i \rightarrow aa \in W_{D_a^\ell}(t_a) : (\exists d' : 1 \leq d' < d : \text{scViolateLoc}(aa, d', t_a, D_a^\ell))).$$

By the induction hypothesis applied to  $\text{scViolateLoc}(aa, d', t_a, D_a^\ell)$ , we have  $\text{scViolate}_B(aa, d', t)$  for some  $d' < d$ . Since  $W_{D_a^\ell}(t_a) \subseteq W_B(t)$  by construction, we have  $B_i \rightarrow aa \in W_B(t)$  and  $\text{scViolate}_B(aa, d', t)$ . Hence by Definition 4.3, Clause 1, we have  $\text{scViolate}_B(v, d, t)$ .

□

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

**Definition 6.7 (Local strong connectedness violation,  $\text{sConnViolateLoc}(v, t_a, D_a^\ell)$ )**

Let  $L$  be the nodes of  $W_{D_a^\ell}(t_a)$  that have no local supercycle violation, i.e.,  $L = \{v \mid v \in D_a^\ell \wedge \neg(\exists d \geq 1 : \text{scViolateLoc}(v, d, t_a, D_a^\ell))\}$ . Let  $v$  be an arbitrary node in  $L$ . Let  $WL = W_{D_a^\ell}(t_a) \upharpoonright L$ , i.e.,  $WL$  is the subgraph of  $W_{D_a^\ell}(t_a)$  consisting of the nodes in  $L$ , and the edges between those nodes that are also edges in  $W_{D_a^\ell}(t_a)$ .

Then,  $\text{sConnViolateLoc}(v, t_a, D_a^\ell)$  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  $\pi$  from  $v$  to a border node of  $D_a^\ell$  contains at least one node with a local supercycle violation
  - or
  - (b) every wait-for path  $\pi'$  from a border node of  $D_a^\ell$  to  $v$  contains at least one node with a local supercycle violation

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

**PROPOSITION 6.8.** Let  $t$  be an arbitrary reachable state of BIP-system  $(B, Q_0)$ . For all interactions  $a \in \gamma$ , and  $\ell > 0$ , let  $t_a = t \upharpoonright D_a^\ell$ . Then

$$\text{sConnViolateLoc}(v, t_a, D_a^\ell) \Rightarrow \text{sConnViolate}_B(v, t).$$

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

- (1)  $SSC \subseteq W_{D_a^\ell}(t_a)$ : let  $x$  be any node in  $SSC$ . Since  $x$  is a node in a supercycle, we have by Proposition 4.4, that  $\neg(\exists d \geq 1 : \text{scViolate}_B(x, d, t))$ . Hence  $(\forall d \geq 1 : \neg \text{scViolate}_B(x, d, t))$ . Hence by Proposition 6.6, we have  $(\forall d \geq 1 : \neg \text{scViolateLoc}(x, d, t_a, D_a^\ell))$ . Let  $L, WL$  be as given in Definition 6.7. Then  $x \in L$ , and since  $x$  is an arbitrary node of  $SSC$ , we have  $SSC \subseteq WL$ . Thus Clause 1 of Definition 6.7 is violated.
- (2)  $SSC \not\subseteq W_{D_a^\ell}(t_a)$ : then there exists a node  $x \in SSC - D_a^\ell$ . Since  $v \in SSC$ , there must exist a wait-for path  $\pi$  from  $v$  to  $x$  and a wait-for path  $\pi'$  from  $x$  to  $v$ . Since  $v \in D_a^\ell$

and  $x \notin D_a^\ell$ , it follows that both  $\pi, \pi'$  cross a border node of  $D_a^\ell$ . Furthermore, since  $\pi, \pi'$  are part of  $SSC$ , every node along  $\pi, \pi'$  is in a supercycle, and so cannot have a supercycle violation. By Proposition 6.6, the nodes on  $\pi, \pi'$  cannot have a local supercycle violation. Hence Clauses 2a and 2b of Definition 6.7 are violated, since they require that at least one node along  $\pi, \pi'$  respectively, have a local supercycle violation.

In both cases, Definition 6.7 is violated. But Definition 6.7 must hold, since we have  $sConnViolateLoc(v, t_a, D_a^\ell)$ . Hence the desired contradiction.  $\square$

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

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

**PROPOSITION 6.10.** Let  $t$  be an arbitrary reachable state of BIP-system  $(B, Q_0)$ . For all interactions  $a \in \gamma$ , and  $\ell > 0$ , let  $t_a = t \downarrow D_a^\ell$ . Then  $genViolateLoc(v, t_a, D_a^\ell) \Rightarrow genViolate_B(v, t)$ .

**PROOF.** Assume that  $genViolateLoc(v, t_a, D_a^\ell)$  holds. Then, by Definition 4.9,  $(\exists d \geq 1 : scViolateLoc(v, d, t_a, D_a^\ell)) \vee sConnViolateLoc(v, t_a, D_a^\ell)$ . We proceed by cases:

- (1)  $(\exists d \geq 1 : scViolateLoc(v, d, t_a, D_a^\ell))$ : hence  $(\exists d \geq 1 : scViolate_B(v, d, t))$  by Proposition 6.6.
- (2)  $sConnViolateLoc(v, t_a, D_a^\ell)$ : hence  $sConnViolate_B(v, t)$  by Proposition 6.8.

By Definition 4.9,  $genViolate_B(v, t) \triangleq (\exists d \geq 1 : scViolate_B(v, d, t)) \vee sConnViolate_B(v, t)$ . Hence we conclude that  $genViolate_B(v, t)$  holds.  $\square$

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

**Definition 6.11 ( $\mathcal{LACT}(B, Q_0, a, \ell)$ )** Let  $\ell > 0$ , and let  $s_a \xrightarrow{a} t_a$  be an arbitrary reachable transition of  $D_a^\ell$ . 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  $D_a^\ell$ . Formally,  $\forall B_i \in components(a) : genViolateLoc(B_i, t_a, D_a^\ell)$ .

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

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

$\text{scViolate}_B(v, d, t)$	$v$ confirmed at depth $d$ to not be in supercycle
$\text{scViolateLoc}(v, d, t_a, D_a^\ell)$	$v$ locally determined to not be in a supercycle
$\text{sConnViolate}_B(v, t)$	$v$ not in a strongly connected supercycle
$\text{sConnViolateLoc}(v, t_a, D_a^\ell)$	$v$ locally determined to not be in a strongly connected supercycle
$\text{genViolate}_B(v, t)$	$v$ does not contribute to a supercycle
$\text{genViolateLoc}(v, t_a, D_a^\ell)$	$v$ locally determined to not contribute to a supercycle

Fig. 8. Summary of predicates

PROOF. Assume  $\mathcal{LACT}(B, Q_0, a, \ell)$  for some  $\ell > 0$ . Let  $s \xrightarrow{a} t$  be an arbitrary reachable transition of BIP-system  $(B, Q_0)$ , and let  $s_a \xrightarrow{a} t_a$  be the projection of  $s \xrightarrow{a} t$  onto  $D_a^\ell$ . By Corollary 2.14,  $s_a \xrightarrow{a} t_a$  is a reachable transition of  $D_a^\ell$ .

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

for every reachable transition  $s_a \xrightarrow{a} t_a$  of  $D_a^\ell$ :  
 $\forall B_i \in \text{components}(a) : \text{genViolateLoc}(B_i, t_a, D_a^\ell)$ .

From this and Proposition 6.10,

for every reachable transition  $s \xrightarrow{a} t$  of  $(B, Q_0)$ :  
 $\forall B_i \in \text{components}(a) : \text{genViolate}_B(B_i, t)$

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

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

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

### 6.3. A Local Linear Condition for Deadlock Freedom

We now formulate a local version of  $\mathcal{GLIN}$ . Observe that if  $\text{in\_depth}_B(B_i, t) < \omega \vee \text{out\_depth}_B(B_i, t) < \omega$ , then there is some finite  $\ell$  such that  $\text{in\_depth}_B(B_i, t) = \ell \vee \text{out\_depth}_B(B_i, t) = \ell$ .

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

$$\forall B_i \in \text{components}(a) : \text{in\_depth}_{D_a^\ell}(B_i, t_a) < 2\ell - 1 \vee \text{out\_depth}_{D_a^\ell}(B_i, t_a) < 2\ell - 1.$$

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

Since wait-for-edges project up and down, it follows that wait-for-paths project up and down, provided that the subsystem contains the entire wait-for-path.

**Proposition 6.15 (In-projection, Out-projection)** *Let  $\ell > 0$ , let  $B_i$  be an atomic component of  $B$ , and let  $(B', Q'_0)$  be a subsystem of  $(B, Q_0)$  which is based on a superset of  $G_a^{2\ell}$ . Let  $s$  be a state of  $(B, Q_0)$ , and  $s' = s|_{B'}$ . Then (1)  $\text{in\_depth}_B(B_i, s) < 2\ell - 1$  iff  $\text{in\_depth}_{B'}(B_i, s') < 2\ell - 1$ , and (2)  $\text{out\_depth}_B(B_i, s) < 2\ell - 1$  iff  $\text{out\_depth}_{B'}(B_i, s') < 2\ell - 1$ .*

*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.

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

$in\_depth_{B'}(B_i, s') < 2\ell - 1$  implies  $in\_depth_B(B_i, s) < 2\ell - 1$ : Assume  $in\_depth_{B'}(B_i, s') < 2\ell - 1$ . Then there exists a wait-for path  $\pi$  in  $W_{B'}(s')$  such that  $|\pi| \geq 2\ell - 1$ . Let  $\rho$  be the prefix of  $\pi$  with length  $2\ell - 1$ . Since  $(B', Q'_0)$  is based on a superset of  $G_a^{2\ell}$ , and since the distance from  $B_i$  to the border of  $G_a^{2\ell}$  is  $2\ell - 1$ , we conclude that  $\rho$  is a wait-for path that is wholly contained in  $W_{B'}(s')$ . Hence we have  $in\_depth_{B'}(B_i, s') \geq 2\ell - 1$ . We have thus established  $in\_depth_{B'}(B_i, s') < 2\ell - 1$  implies  $in\_depth_B(B_i, s) < 2\ell - 1$ . The contrapositive is the desired result.  $\square$

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

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

*Proof.* Let  $s \xrightarrow{a} t$  be a reachable transition of  $(B, Q_0)$  and let  $B_i \in components(a)$ ,  $s_a = s \upharpoonright D_a^\ell$ ,  $t_a = t \upharpoonright D_a^\ell$ . Then  $s_a \xrightarrow{a} t_a$  is a reachable transition of  $D_a^\ell$  by Corollary 2.14. By  $\mathcal{LLIN}(B, Q_0, a, \ell)$ ,  $in\_depth_{D_a^\ell}(B_i, t_a) < 2\ell - 1 \vee out\_depth_{D_a^\ell}(B_i, t_a) < 2\ell - 1$ . Hence by Proposition 6.15,  $in\_depth_B(B_i, t) < 2\ell - 1 \vee out\_depth_B(B_i, t) < 2\ell - 1$ . So  $in\_depth_B(B_i, t) < \omega \vee out\_depth_B(B_i, t) < \omega$ . Hence  $\mathcal{GLIN}(B, Q_0, a)$ .  $\square$

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

*Proof.* Follows immediately from Lemma 6.16 and Theorem 5.7.  $\square$

**Proposition 6.18 (Finite out-depth implies local supercycle-violation)** *For  $d < \ell$ :  $(out\_depth_{D_a^\ell}(v, t_a) = d) \Rightarrow scViolateLoc(v, d + 1, t_a, D_a^\ell)$ .*

*Proof.* Proof is by induction on  $d$ .

**Base case,  $d = 0$ .** Then  $v$  has no outgoing wait-for edges. Hence  $scViolateLoc(v, 1, t_a, D_a^\ell)$  by Definition 6.5.

**Induction step,  $d > 0$ .** Assume  $(out\_depth_{D_a^\ell}(v, t_a) = d)$ . Then, every outgoing wait-for edge of  $v$  is to some  $v'$  such that  $(out\_depth_{D_a^\ell}(v', t_a) = d' < d)$ . By the induction hypothesis,  $scViolateLoc(v', d' + 1, t_a, D_a^\ell)$ . Hence, by Definition 6.5,  $scViolateLoc(v, d + 1, t_a, D_a^\ell)$ .  $\square$

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

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

$$in\_depth_{D_a^\ell}(B_i, t_a) < 2\ell - 1 \vee out\_depth_{D_a^\ell}(B_i, t_a) < 2\ell - 1.$$

The proof proceeds by two cases.

$in\_depth_{D_a^\ell}(B_i, t_a) < 2\ell - 1$ : 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  $\text{sConnViolateLoc}(B_i, t_a, D_a^\ell)$  by Definition 6.7, Clause 1. Hence by Definition 6.9,  $\text{genViolateLoc}(B_i, t_a, D_a^\ell)$ .

$\text{out\_depth}_{D_a^\ell}(B_i, t_a) < 2\ell - 1$ : Hence  $\text{out\_depth}_{D_a^\ell}(B_i, t_a) = d$  for some  $d < 2\ell - 1$ . By Proposition 6.18,  $\text{scViolateLoc}(B_i, d + 1, t_a, D_a^\ell)$ . Hence by Definition 6.9,  $\text{genViolateLoc}(B_i, t_a, D_a^\ell)$ .

In both cases, we have  $\text{genViolateLoc}(B_i, t_a, D_a^\ell)$ . Since  $B_i$  is an arbitrarily chosen component of  $\text{components}(a)$ , we have  $\forall B_i \in \text{components}(a) : \text{genViolateLoc}(B_i, t_a, D_a^\ell)$ . Hence, by Definition 6.11, we conclude  $\mathcal{LACT}(B, Q_0, a, \ell)$ .  $\square$

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.

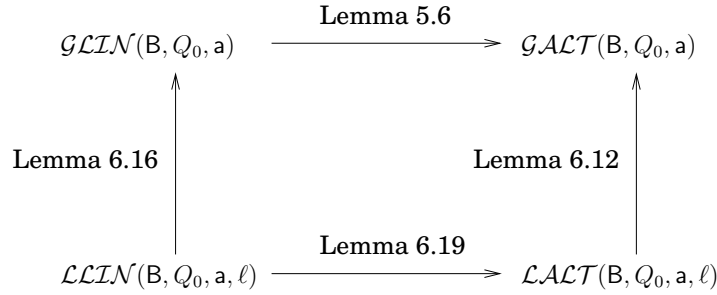


Fig. 9. Implication relations between deadlock-freedom conditions

#### 6.4. Deadlock freedom using local and global restrictions

**Theorem 6.20 (Deadlock-freedom via  $\mathcal{LACT}$ ,  $\mathcal{LLLIN}$ )** Assume that

- (1) for all  $s_0 \in Q_0$ ,  $W_B(s_0)$  is supercycle-free, and
- (2) for all interactions  $a$  of  $B$  (i.e.,  $a \in \gamma$ ), one of the following holds:
  - (a)  $\mathcal{GACT}(B, Q_0, a)$
  - (b)  $\mathcal{GLLIN}(B, Q_0, a)$
  - (c)  $\exists \ell > 0 : \mathcal{LACT}(B, Q_0, a, \ell)$
  - (d)  $\exists \ell > 0 : \mathcal{LLLIN}(B, Q_0, a, \ell)$

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

*Proof.* Immediate from Theorems 5.2, 5.7, 6.13, 6.17 and Corollary 4.13.  $\square$

## 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.

**PROPOSITION 7.1.**  $\text{CHECKINITSUPERCYCLEFREE}(Q_0)$  returns true iff all initial states are supercycle-free.

CHECKINITSUPERCYCLEFREE( $Q_0$ )

▷ returns true iff all initial states are supercycle-free

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

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

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

Suppose that  $U$  is nonempty. By Proposition 4.5,  $U$  is a supercycle. Since  $U \subseteq W_B(s_0)$ , 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  $W_B(s_0)$  has a supercycle violation, and so by Proposition 4.4, no node of  $W_B(s_0)$  can be in a strongly-connected supercycle. Hence  $W_B(s_0)$  does not contain a strongly-connected supercycle. So, by Proposition 3.16,  $W_B(s_0)$  does not contain a supercycle.  $\square$

## 7.2. Implementation of the Linear Condition

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

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

LLIN( $B, Q_0$ ), where  $B \triangleq \gamma(B_1, \dots, B_n)$

1. forall interactions  $a \in \gamma$
2.     if (LLININT( $B, Q_0, a$ ) = ff) return(ff) fi
3. endfor;
4. return(tt) ▷ return tt if check succeeds for all  $a \in \gamma$

LLININT( $B, Q_0, a$ ), where  $B \triangleq \gamma(B_1, \dots, B_n), a \in \gamma$

▷ check ( $\exists \ell > 0 : \mathcal{LLIN}(B, Q_0, a, \ell)$ )

1.  $\ell \leftarrow 1$ ; ▷ start with  $\ell = 1$
2. while (tt)
3.     if (LLININTDIST( $a, \ell$ ) = tt) return(tt) fi; ▷ success, so return true
4.     if ( $D_a^\ell = \gamma(B_1, \dots, B_n)$ ) return(ff) fi; ▷ exhausted all subsystems, return false
5.      $\ell \leftarrow \ell + 1$  ▷ increment  $\ell$  until success or intractable or failure
6. endwhile

LLININTDIST( $B, Q_0, a, \ell$ )

1. forall reachable transitions  $s_a \xrightarrow{a} t_a$  of  $D_a^\ell$
2.     if ( $\neg(\forall B_i \in \text{components}(a) : \text{in\_depth}_{D_a^\ell}(B_i, t_a) < 2\ell - 1 \vee \text{out\_depth}_{D_a^\ell}(B_i, t_a) < 2\ell - 1)$ )
3.         return(ff) ▷ check Definition 6.14
4.     fi
5. endfor;
6. return(tt) ▷ return tt if check succeeds for all transitions

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

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

### 7.3. Implementation of the AND-OR Condition

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

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

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

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

Fig. 12. Summary of procedures

$\text{LOCSCVIOL}(D_a^\ell, t_a)$

▷ compute supercycle violations in state  $t_a$  of  $D_a^\ell$

▷ Postcondition:  $\forall v \in D_a^\ell : V_{D_a^\ell, t_a}[v] = \begin{cases} \text{tt} & \text{if } \exists d \geq 1 : \text{scViolateLoc}(v, d, t_a, D_a^\ell) \\ \text{ff} & \text{otherwise} \end{cases}$

1.  $\text{foundScViolate} \leftarrow \text{ff}$

2. **forall**  $v \in D_a^\ell$

3.   **if** ( $v$  is an interior interaction  $aa$  and  $\neg(\exists B_i : aa \rightarrow B_i \in W_{D_a^\ell}(t_a))$ )

4.      $V_{D_a^\ell, t_a}[v] \leftarrow \text{tt}$                    ▷ Base case: interaction with no outgoing wait-for-edges

5.      $\text{foundScViolate} \leftarrow \text{tt}$

6.   **fi**

7. **endfor**

8. **while** ( $\text{foundScViolate}$ )

9.    $\text{foundScViolate} \leftarrow \text{ff}$

10. **forall**  $v \in D_a^\ell : \neg V_{D_a^\ell, t_a}[v]$

11.   **if** ( $v$  is an interior interaction  $aa$  and  $(\forall B_i : aa \rightarrow B_i \in W_{D_a^\ell}(t_a) : V_{D_a^\ell, t_a}[B_i])$ )

12.      $V_{D_a^\ell, t_a}[v] \leftarrow \text{tt}$

13.      $\text{foundScViolate} \leftarrow \text{tt}$

14.   **else if** ( $v$  is a component  $B_i$  and  $(\exists aa : B_i \rightarrow aa \in W_{D_a^\ell}(t_a) : V_{D_a^\ell, t_a}[aa])$ )

15.      $V_{D_a^\ell, t_a}[vd] \leftarrow \text{tt}$

16.      $\text{foundScViolate} \leftarrow \text{tt}$

17.   **fi**

18. **endfor**

19. **endwhile**

Fig. 14. Procedure to compute all supercycle-violations in state  $t_a$  of  $D_a^\ell$ 

#### 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 15) that accepts a set of configuration options. It takes the name of the input BIP file and other optional flags.

#### 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 Sys-*



$LALT(B, Q_0)$ , where  $B \triangleq \gamma(B_1, \dots, B_n)$

$\triangleright$  returns tt iff  $(\forall a \in \gamma, \exists \ell > 0 : \mathcal{LACT}(a, \ell))$

1. forall interactions  $a \in \gamma$
2.   if  $(LALTINT(B, Q_0, a) = ff)$  return(ff) fi
3. endfor;
4. return(tt)  $\triangleright$  return tt if check succeeds for all  $a \in \gamma$

$LALTINT(B, Q_0, a)$ , where  $B \triangleq \gamma(B_1, \dots, B_n)$ ,  $a \in \gamma$

$\triangleright$  returns tt iff  $(\exists \ell > 0 : \mathcal{LACT}(B, Q_0, a, \ell))$

1.  $\ell \leftarrow 1$ ;  $\triangleright$  start with  $\ell = 1$
2. while (tt)
3.   if  $(LALTINTDIST(a, \ell) = tt)$  return(tt) fi;  $\triangleright$  success, so return true
4.   if  $(D_a^\ell = \gamma(B_1, \dots, B_n))$  return(ff) fi;  $\triangleright$  exhausted all subsystems, return false
5.    $\ell \leftarrow \ell + 1$   $\triangleright$  increment  $\ell$  until success or intractable or failure
6. endwhile

$LALTINTDIST(B, Q_0, a, \ell)$

$\triangleright$  returns tt iff  $\mathcal{LACT}(B, Q_0, a, \ell)$

1. forall reachable transitions  $s_a \xrightarrow{a} t_a$  of  $D_a^\ell$
2.   forall  $B_i \in components(a)$
3.     if  $\neg LOCFORMVIOL(B_i, D_a^\ell, t_a)$  then return(ff) fi  $\triangleright$  return ff if no violation for  $B_i$
4.   endfor
5. endfor;
6. return(tt)  $\triangleright$  return tt if all  $B_i \in components(a)$  violate local supercycle formation

$LOCFORMVIOL(B_i, D_a^\ell, t_a)$

$\triangleright$  returns true iff  $genViolateLoc(B_i, t_a, D_a^\ell)$  holds (Definition 6.9)

$\triangleright$  i.e.,  $B_i$  has a local supercycle formation violation in state  $t_a$  of subsystem  $D_a^\ell$

1.  $LOCSCVIOL(D_a^\ell, t_a)$
2. return( $V_{D_a^\ell, t_a}[B_i] \vee LOCSCONNSCVIOL(B_i, D_a^\ell, t_a)$ )

$LOCSCONNSCVIOL(B_i, D_a^\ell, t_a)$

$\triangleright$  returns true iff  $sConnViolateLoc(B_i, t_a, D_a^\ell)$  holds (Definition 6.7)

$\triangleright$  i.e.,  $B_i$  has a local strong connectedness supercycle formation violation in state  $t_a$  of subsystem  $D_a^\ell$

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  $\triangleright$  Definition 6.7, Clause 1 holds here
6. forall wait-for paths  $\pi$  from  $B_i$  to the border of  $D_a^\ell$
7.   if some node of  $\pi$  has a local supercycle violation then return(tt) fi  $\triangleright$  Clause 2a holds
8. forall wait-for paths  $\pi'$  from the border of  $D_a^\ell$  to  $B_i$
9.   if some node of  $\pi'$  has a local supercycle violation then return(tt) fi  $\triangleright$  Clause 2b holds
10. return(ff)  $\triangleright$  Definition 6.7, Clause 2 does not hold

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

```

> java -jar ldfe.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 ldfe.jar -debug input.bip # deadlock freedom using default LALT
  java -jar ldfe.jar -condition=LLIN -debug input.bip # deadlock freedom using LLIN

```

Fig. 15. LALT-BIP Command Line Interface

tem 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) i7-6700, 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 [Bensalem et al. 2011]. IPM requires a manual partitioning of the system to exploit its efficiency. We applied IPM on all structural partitions and we report on the best result which is consistent with the results reported in Bensalem et al. [2011].

Table I shows the results. Both  $\mathcal{LALT}$  and  $\mathcal{LLIN}$  outperform the best performance of DFinder2 by several orders of magnitude for  $n \leq 3,000$ . Both  $\mathcal{LALT}$  successfully completed the deadlock freedom check for  $3,000 \leq n \leq 10,000$  in less than one minute, where DFinder2 timed out ( 1 Hour).  $\mathcal{LLIN}$  required 62 seconds for  $n = 10,000$ .

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

- The largest subsystem that  $\mathcal{LALT}$  had to consider was with depth  $\ell = 1$ . This corresponds to  $18 = 2^1 \times 3^2$  states regardless of  $n$ , the number of philosophers.
- The largest subsystem that  $\mathcal{LLIN}$  had to consider was with depth  $\ell = 2$ . This corresponds to  $648 = 2^3 \times 3^4$  states regardless of  $n$ .
- For a given depth  $\ell$ ,  $\mathcal{LLIN}$  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

Table I. Benchmarks: Dining Philosopher

Size	$\mathcal{LACT}$	$\mathcal{LLIN}$	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	—

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

Given configuration specifying  $n$ ,  $m$ ,  $k$ , a map of requests between clients and resources, and a set of sets of conflicting resources, we automatically generate a corresponding BIP model.

Figures 16, 17, and 18 show BIP atomic components for client, resource and manager components.

The client in Figure 16 requests resources  $R_0$  and  $R_2$  in sequence. It has 5 ports. Ports  $SR_0$  and  $SR_2$  send requests for resources  $R_0$  and  $R_2$ , respectively. Ports  $RG_0$  and  $RG_2$  receive grants for resources  $R_0$  and  $R_2$ , respectively. Port  $rel$  releases all resources. The behavior of the client depends on its request sequence.

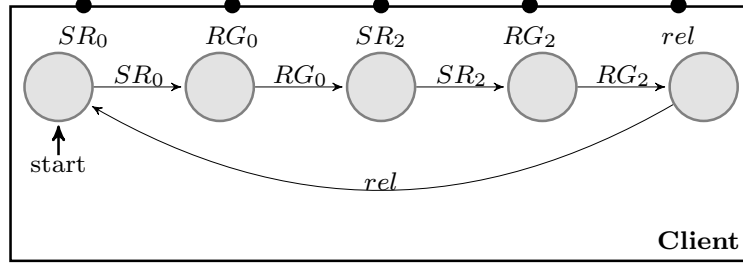


Fig. 16. Client

Figure 17 shows a resource component. A resource component waits for a request from a connected client on port  $RR$ . Once a request is received, the resource component transitions to a state where it is ready to receive a token from the corresponding resource manager using port  $RTT$ . The resource transitions to a state where it grants the client request using port  $STC$  and waits until it is released on port  $done$ . There, it returns the token back to the resource manager and transitions to the start state.

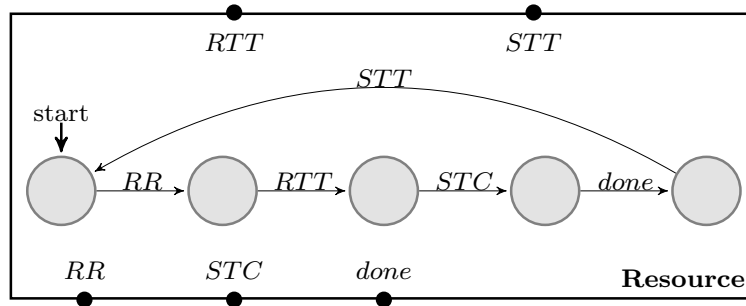


Fig. 17. Resource

Figure 18 shows a resource manager. A resource manager  $M$  has four states.

- State  $T$  denotes that  $M$  has a token.  $M$  may send the token to either (1) a resource on port  $STR$  and transition to state  $TwR$  (token with resource), or (2) the next resource manager on port  $STT$  and transition to state  $N$  (no token).
- State  $N$  denotes that  $N$  has no token. It may receive a token from a neighboring resource manager in the ring on port  $RTT$  and transition to state  $T$ .
- State  $TwR$  denotes that  $M$  has already passed a token to one of its resources.  $M$  may either receive (1) the assigned token back from the resource using port  $RTR$  and transition to state  $T$ , or (2) another token from a neighboring manager using port  $RTT$  and transition to state  $TTwR$  (token and token with resource).
- State  $TTwR$  denotes that  $M$  has a token and has already passed a token to one of its resources. In this state  $M$  can not send the token it has to a resource it manages to respect the conflict constraint.  $M$  may send the token to the next manager on port  $STT$  and transition back to state  $TwR$ .

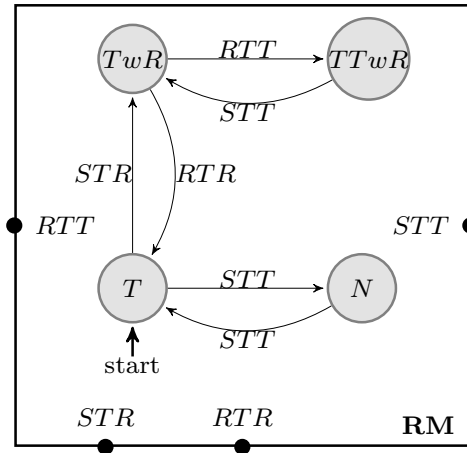


Fig. 18. Token Resource Manager

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

Figure 19 shows a configuration system with 5 clients and 5 resources where:

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

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

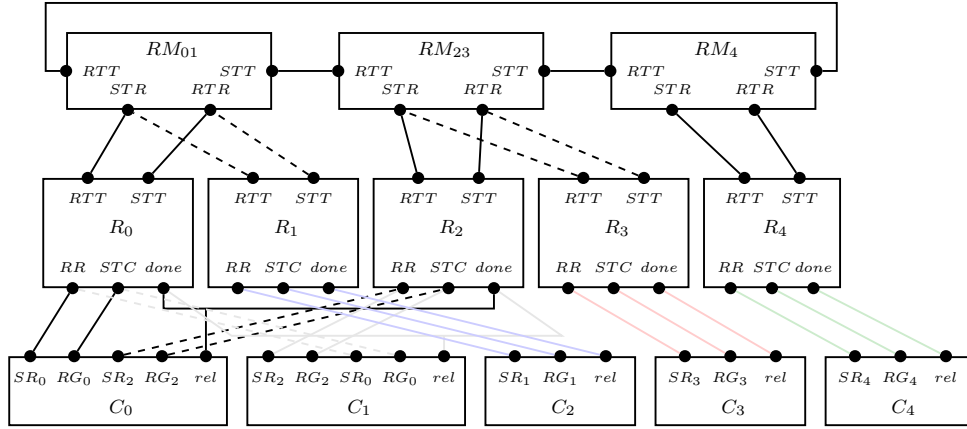


Fig. 19. Conflict-Resource Allocation System

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

*Lesson 1:*  $\mathcal{LACT}$  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  $\mathcal{LACT}$  successfully does.

*Lesson 2:*  $\mathcal{LACT}$  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

Table II. Benchmarks: Time required for  $\mathcal{LACT}$  on the resource allocation system

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

the system might contain a deadlock.  $\mathcal{LACT}$  successfully reports that the system is both global and local deadlock free.

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

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

*Benchmarking:* We evaluated the performance of  $\mathcal{LACT}$  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  $n = 10$ .  $\mathcal{LACT}$  was able to verify deadlock freedom. It has to check subsystems with 12 components out of  $3 \times n$  components regardless of  $n$ . This resulted from inspecting subsystems corresponding to a depth  $\ell = 2$  with  $\leq 23,040,000 = 4^6 \times 3^2 \times 5^4$  states regardless of  $n$ . The numbers in Table II show a linear increase in time required to check deadlock freedom using  $\mathcal{LACT}$  with respect to  $n$ . This indicates that the number of subsystems to check is proportional to  $n$ .

Our resource allocation system subsumes the token based Milner scheduler [Milner 1989] which is essentially a token ring with precisely one token present [Antonino et al. 2016]. [Antonino et al. 2016] present a technique that fails to prove deadlock freedom for Milner Scheduler because it requires a large subset of the system, while  $\mathcal{LACT}$  succeeds.

## 8. DISCUSSION, RELATED WORK, AND FURTHER WORK

### 8.1. Related work.

The notions of wait-for-graph and supercycle [Attie and Chockler 2005; Attie and Emerson 1998] were initially defined for a shared memory program  $P = P_1 \parallel \dots \parallel P_K$  in *pairwise normal form* [Attie 2016b; Attie 2016a]: a binary symmetric relation  $I$  specifies the directly interacting pairs (“neighbors”)  $\{P_i, P_j\}$ . If  $P_i$  has neighbors  $P_j$  and  $P_k$ , then the code in  $P_i$  that interacts with  $P_j$  is expressed separately from the code in  $P_i$  that interacts with  $P_k$ . These synchronization codes are executed synchronously and atomically, so the grain of atomicity is proportional to the degree of  $I$ . Attie and Chockler [2005] give two polynomial time methods for (local and global) deadlock free-

dom. The first checks subsystems consisting of three processes. The second computes the wait-for-graphs of all pair subsystems  $P_i \parallel P_j$ , and takes their union, for all pairs and all reachable states of each pair. The first method considers only wait-for-paths of length  $\leq 2$ . The second method is prone to false negatives, because wait-for edges generated by different states are all merged together, which can result in spurious supercycles.

Gössler and Sifakis [2003] use a BIP-like formalism, Interaction Models. They present a criterion for global deadlock freedom, based on an and-or graph with components and constraints as the two sets of nodes. A constraint gives the condition under which a component is blocked. Edges are labeled with conjuncts of the constraints. Deadlock freedom is checked by traversing every cycle, taking the conjunction of all the conditions labeling its edges, and verifying that this conjunction is always false, i.e., verifying the absence of cyclical blocking. No complexity bounds are given. Martens and Majster-Cederbaum [2012] present a polynomial time checkable deadlock freedom condition based on structural restrictions: “the communication structure between the components is given by a tree.” This restriction allows them to analyze only pair systems. Aldini and Bernardo [2003] use a formalism based on process algebra. They check deadlock by analyzing cycles in the connections between software components, and claim scalability, but no complexity bounds are given.

Roscoe and Dathi [1987] present several rules for freedom of global deadlock of “triple disjoint” (no action involves  $> 2$  processes) CSP concurrent programs. The basis for these rules is to first check that each individual process is deadlock free (i.e., the network is “busy”), and then to define a “variant function” that maps the state of each process to a partially ordered set. The first rule requires to establish that, if  $P_i$  waits for  $P_j$ , then the value of  $P_i$ ’s state is greater than the value of  $P_j$ ’s state. Since every process is blocked in a global deadlock, one can then construct an infinite sequence of processes with strictly decreasing values, which are therefore all distinct. This cannot happen in a finite network, and hence some process is not blocked. They treat several examples, including a self-timed systolic array (in 2 and 3 dimensions), dining philosophers, and a message switching network. They generalize the first rule to exploit “disconnecting edges” (whose removal partitions the network into disconnected components) to decompose the proof of deadlock freedom into showing that each disconnected component is deadlock-free, and also to weaken the restriction on the variant function so that it only has to decrease for at least one edge on each wait-for cycle. Brookes and Roscoe [1991] also provide criteria for deadlock freedom of triple-disjoint CSP programs, and use the same technical framework as Roscoe and Dathi [1987]. However, they do not use variant functions, but show that, in a busy network, a deadlock implies the existence of a wait-for cycle. They give many examples, and demonstrate the absence of wait-for cycles in each example, by ad-hoc reasoning. Finally, they give a deadlock freedom rule that exploits disconnecting edges, similar to that of Roscoe and Dathi [1987]. In both of these papers, the wait-for relations are defined by examining a pair of processes at a time:  $P_i$  waits for  $P_j$  iff  $P_i$  offers an action to  $P_j$  which  $P_j$  is not willing to participate in.

Martin [1996] applies the results of Roscoe and Dathi [1987] and Brookes and Roscoe [1991] to formulate deadlock-freedom design rules for several classes of CSP concurrent programs: cyclic processes, client-server protocols, and resource allocation protocols. He also introduces the notion of “state dependence digraph” (SDD), whose nodes are local states of individual processes, and whose edges are wait-for relations between processes in particular local states. An acyclic SDD implies deadlock-freedom. A cyclic SDD does not imply deadlock, however, since the cycle may be “spurious”: the

local states along the cycle may not be reachable at the same time, and so the cycle cannot give rise to an actual deadlock during execution. Hence the SDD approach cannot deal with “non-hereditary” deadlock freedom, i.e., a deadlock free system that contains a deadlock prone subsystem. Consider, e.g., the dining philosophers with a butler solution; removing the butler leaves a deadlock prone subsystem. Antonino et al. [2016] takes the SDD approach and improves its accuracy by checking for mutual reachability of pairs of local states, and also eliminating local states and pairs of local states, where action enablement can be verified locally. These checks are formulated as a Boolean formula which is then sent to a SAT solver. Their method is able to verify deadlock freedom of dining philosophers with a butler, whereas our method timed out, since the subsystems on which  $\mathcal{LALT}(B, Q_0, a, \ell)$  is evaluated becomes the entire system. On the other hand, our approach succeeded in quickly verifying deadlock-freedom of the resource allocation example, whereas the method of Antonino et al. [2016] failed for Milner’s token based scheduler, which is a special case of our resource allocation example. An intriguing topic for future work is to attempt to combine the two methods, to obtain the advantages of both.

We compared our implementation LALT-BIP to D-Finder 2 [Bensalem et al. 2011]. D-Finder 2 computes a finite-state abstraction for each component, which it uses to compute a global invariant  $I$ . It then checks if  $I$  implies deadlock freedom. Unlike LALT-BIP, D-Finder 2 handles infinite state systems. However, LALT-BIP had superior running time for dining philosophers and resource controller (both finite-state).

All the above methods (except Attie and Chockler [2005]) verify global (and not local) deadlock-freedom. Our method verifies local deadlock-freedom, which subsumes global deadlock-freedom as a special case. Also, our approach makes no structural restriction at all on the system being checked for deadlock. Our method checks for the absence of supercycles, which are a sound and complete characterization of deadlock, 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. 4.6, Def. 4.9, and Def. 5.1).

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



parallelized and distributed, for speedup, if needed. Alternatively, performing the checks sequentially minimizes the amount of memory needed.

- *Framework aspect*: supercycles and in/out-depth provide a *framework* for deadlock-freedom. Conditions more general and/or discriminating than the one presented here should be devisable in this framework. This is a topic for future work. In addition, our approach is applicable to any model of concurrency in which our notions of wait-for graph and supercycle can be defined. For example, Attie and Chockler [2005] give two methods for verifying global and local deadlock freedom of shared-memory concurrent programs in pairwise normal form, as noted above. Hence, our methods are applicable to other formalisms such as CSP, CCS, I/O Automata, etc.

### 8.3. Further work.

Our implementation uses explicit state enumeration. Using BDD's may improve the running time when  $\mathcal{LACT}(B, Q_0, a, \ell)$  holds only for large  $\ell$ . Another potential method for improving the running time is to use SAT solving, cf. Antonino et al. [2016]. An enabled port  $p$  enables all interactions containing  $p$ . Deadlock-freedom conditions based on ports could exploit this interdependence among interaction enablement. Our implementation should produce *counterexamples* when a system fails to satisfy  $\mathcal{LACT}(B, Q_0, a, \ell)$ . These can be used to manually modify the system to eliminate a possible deadlock. Also, when  $\mathcal{LACT}(B, Q_0, a, \ell)$  fails to verify deadlock-freedom, we increment  $\ell$ , 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  $\mathcal{LACT}(B, Q_0, a, \ell)$  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  $\mathcal{LACT}(B, Q_0, a, \ell)$  on *infinite state* systems will be devised, e.g., by extracting proof obligations and verifying using SMT solvers. We will extend our method to *Dynamic BIP*, [Bozga et al. 2012], where participants can add and remove interactions at run time.

## REFERENCES

- Alessandro Aldini and Marco Bernardo. 2003. A General Approach to Deadlock Freedom Verification for Software Architectures. *FME* 2805 (2003), 658–677.
- Pedro Antonino, Thomas Gibson-Robinson, and A. W. Roscoe. 2016. Efficient Deadlock-Freedom Checking Using Local Analysis and SAT Solving. In *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*. 345–360. DOI: [http://dx.doi.org/10.1007/978-3-319-33693-0\\_22](http://dx.doi.org/10.1007/978-3-319-33693-0_22)
- P.C. Attie, N. Francez, and O. Grumberg. 1993. Fairness and Hyperfairness in Multiparty Interactions. *Distributed Computing* 6 (1993), 245–254.
- Paul C. Attie. 2016a. Finite-state concurrent programs can be expressed in pairwise normal form. *Theoretical Computer Science* 619 (2016), 1 – 31. DOI: <http://dx.doi.org/10.1016/j.tcs.2015.11.032>
- Paul C. Attie. 2016b. Synthesis of large dynamic concurrent programs from dynamic specifications. *Formal Methods in System Design* (2016), 1–54. DOI: <http://dx.doi.org/10.1007/s10703-016-0252-9>
- Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. 2013. An Abstract Framework for Deadlock Prevention in BIP. In *Formal Techniques for Distributed Systems - Joint IFIP WG 6.1 International Conference, FMOODS / FORTE 2013, Held as Part of the 8th Interna-*

- tional Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings.* 161–177. DOI: [http://dx.doi.org/10.1007/978-3-642-38592-6\\_12](http://dx.doi.org/10.1007/978-3-642-38592-6_12)
- Paul C. Attie and H. Chockler. 2005. Efficiently Verifiable Conditions for Deadlock-freedom of Large Concurrent Programs. In *VMCAI*. France.
- Paul C. Attie and E. Allen Emerson. 1998. Synthesis of Concurrent Systems with Many Similar Processes. *TOPLAS* 20, 1 (Jan. 1998), 51–115.
- Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. 2011. D-Finder 2: Towards Efficient Correctness of Incremental Design. In *NASA Formal Methods*. 453–458.
- Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. 2010. From High-level Component-based Models to Distributed Implementations. In *EMSOFT*. 209–218.
- Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. 2012. Modeling Dynamic Architectures Using Dy-BIP. In *Software Composition*. 1–16.
- S.D. Brookes and A.W. Roscoe. 1991. Deadlock analysis in networks of communicating processes. *Distributed Computing* 4 (1991), 209–230. Issue 4.
- Gregor Gössler and Joseph Sifakis. 2003. Component-based Construction of Deadlock-free Systems. In *FSTTCS*. Springer, 420–433.
- Moritz Martens and Mila Majster-Cederbaum. 2012. Deadlock-freedom in component systems with architectural constraints. *FMSD* 41 (2012), 129–177. Issue 2. DOI: <http://dx.doi.org/10.1007/s10703-012-0160-6>
- Jeremy Malcolm Randolph Martin. 1996. *The Design and Construction of Deadlock-Free Concurrent Systems*. Ph.D. Dissertation. The University of Buckingham.
- Robin Milner. 1989. *Communication and concurrency*. Prentice Hall.
- Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley.
- A.W. Roscoe and Naiem Dathi. 1987. The pursuit of deadlock freedom. *Information and Computation* 75, 3 (1987), 289 – 327. DOI: [http://dx.doi.org/10.1016/0890-5401\(87\)90004-6](http://dx.doi.org/10.1016/0890-5401(87)90004-6)