An Abstract Framework for Deadlock Prevention in BIP*

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

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

June 24, 2014

Abstract

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

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

1 Introduction

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

Deciding deadlock freedom of finite-state concurrent programs is PSPACE-complete in general [16, chapter 19]. To achieve tractability, we can either make our deadlock freedom check

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

incomplete (sufficient but not necessary), or we can restrict the systems that we check to special cases. We choose the first option: a system meeting our condition is free of both local and global deadlocks, while a system which fails to meet our condition may or may not be deadlock free.

We generalize previous works [2, 3, 4] by removing the requirement that interaction between processes be expressed pairwise, and also by applying to BIP [6], a framework from which efficient distributed code can be generated. In contrast, the model of concurrency in [2, 3, 4] requires shared memory read-modify-write operations with a large grain of atomicity. The full paper, including proofs for all theorems, is available on-line, as is our implementation of the method.

2 BIP — Behavior Interaction Priority

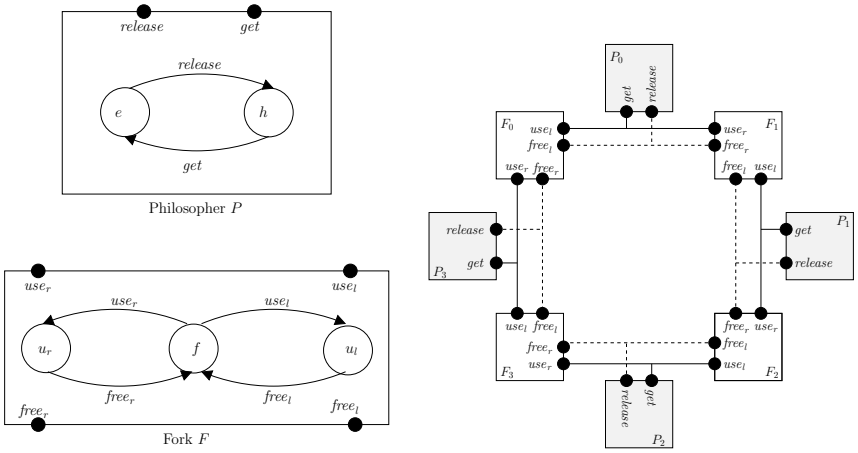
BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. A technical treatment of priority is beyond the scope of this paper. Adding priorities never introduces a deadlock, since priority enforces a choice between possible transitions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

Definition 1 (Atomic Component) *An* atomic component B_i *is a labeled transition system represented by a triple* $(Q_i, P_i, \rightarrow_i)$ *where* Q_i *is a set of* states*,* P_i *is a set of* communication ports*, and* $\rightarrow_i \subseteq Q_i \times P_i \times Q_i$ *is a set of* possible transitions*, each labeled by some port.*

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

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

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



(a) Philosopher P and fork F atomic components.

(b) Dining philosophers composite component with four philosophers.

Figure 1: Dining philosophers.

Definition 2 (Interaction) *For a given system built from a set of n atomic components $\{\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)\}_{i=1}^n$, we require that their respective sets of ports are pairwise disjoint, i.e., for all i, j such that $i, j \in \{1..n\} \wedge i \neq j$, we have $P_i \cap P_j = \emptyset$. An* interaction *is a set of ports not containing two or more ports from the same component. That is, for an interaction a we have $a \subseteq P \wedge (\forall\, i \in \{1..n\} : |\mathsf{a} \cap P_i| \leq 1)$, where $P = \bigcup_{i=1}^n P_i$ is the set of all ports in the system. When we write $a = \{p_i\}_{i \in I}$, we assume that $p_i \in P_i$ for all $i \in I$, where $I \subseteq \{1..n\}$.*

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

Definition 3 (Composite Component) *A* composite component *(or simply* component*)* $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *is defined by a composition operator parameterized by a set of interactions $\gamma \subseteq 2^P$. B has a transition system $(Q, \gamma, \rightarrow)$, where $Q = Q_1 \times \cdots \times Q_n$ and $\rightarrow \subseteq Q \times \gamma \times Q$ is the least set of transitions satisfying the rule*

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

This inference rule says that a composite component $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ can execute an interaction $\mathsf{a} \in \gamma$, iff for each port $p_i \in \mathsf{a}$, the corresponding atomic component B_i can execute a transition labeled with p_i; the states of components that do not participate in the interaction stay unchanged. Given an interaction $\mathsf{a} = \{p_i\}_{i \in I}$, we denote by *components*(a) the set of atomic components participating in a, formally: *components*$(\mathsf{a}) = \{\mathsf{B}_i \mid p_i \in \mathsf{a}\}$. Figure 1(b) shows a composite component consisting of four philosophers and the four forks between them. Each philosopher and its two neighboring forks share two interactions: $Get = \{get, use_l, use_r\}$ in which the philosopher obtains the forks, and $Rel = \{release, free_l, free_r\}$ in which the philosopher releases the forks.

Definition 4 (Interaction enablement) *An atomic component* $\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)$ *enables a port* $p_i \in P_i$ *in state* s_i *iff* $s_i \xrightarrow{p_i}_i$. B_i *enables interaction* a *in state* s_i *iff* $s_i \xrightarrow{p_i}_i$, *where* $p_i = P_i \cap a$ *is the port of* B_i *involved in* a. *That is,* B_i *enables* a *in state* s_i *iff* B_i *enables port* $\mathsf{a} \cap P_i$ *in state* s_i.

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

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

The definition of interaction enablement is a consequence of Definition 3. Interaction a being enabled in state s means that executing a is one of the possible transitions that can be taken from s.

To avoid pathological cases of deadlock due solely to a single component refusing to enable any interaction at all, we assume that every component always enables at least one interaction.

Definition 5 (Local Enablement Assumption) *For every component* $\mathsf{B}_i = (Q_i, P_i, \rightarrow_i)$, *the followingholds. In every* $s_i \in Q_i$, B_i *enables some interaction* a.

Definition 6 (BIP System) *Let* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *be a composite component with transition system* $(Q, \gamma, \rightarrow)$, *and let* $Q_0 \subseteq Q$ *be a set of initial states. Then* (B, Q_0) *is a BIP system.*

Figure 1(b) gives a BIP-system with philosophers initially in state h (hungry) and forks initially in state f (free).

To avoid tedious repetition, we fix, for the rest of the paper, an arbitrary BIP-system (B, Q_0), with $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and transition system $(Q, \gamma, \rightarrow)$.

Definition 7 (Execution) *Let* (B, Q_0) *be a BIP system with transition system* $(Q, \gamma, \rightarrow)$. *Let* $\rho = s_0 a_1 s_1 \ldots s_{i-1} a_i s_i \ldots$ *be an alternating sequence of states of* B *and interactions of* B. *Then* ρ *is an execution of* (B, Q_0) *iff (1)* $s_0 \in Q_0$, *and (2)* $\forall\, i > 0 : s_{i-1} \xrightarrow{a_i} s_i$.

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

Definition 9 (State Projection) *Let* (B, Q_0) *be a BIP system where* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *and let* $s = \langle s_1, \ldots, s_n \rangle$ *be a state of* (B, Q_0). *Let* $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\} \subseteq \{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$. *Then* $s{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\} \triangleq \langle s_{j_1}, \ldots, s_{j_k} \rangle$. *For a single* B_i, *we write* $s{\restriction}\mathsf{B}_i = s_i$. *We extend state projection to sets of states element-wise.*

Definition 10 (Subcomponent) *Let* $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *be a composite component, and let* $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ *be a subset of* $\{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$. *Let* $P' = P_{j_1} \cup \cdots \cup P_{j_k}$, *i.e., the union of the ports of* $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$. *Then the subcomponent* B' *of* B *based on* $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ *is as follows:*

1. $\gamma' \triangleq \{\mathsf{a} \cap P' \mid \mathsf{a} \in \gamma \wedge \mathsf{a} \cap P' \neq \emptyset\}$

2. $\mathsf{B}' \triangleq \gamma'(\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k})$

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

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

Definition 11 (Subsystem) *Let (B, Q_0) be a BIP system where $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and let $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ be a subset of $\{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$. Then the subsystem (B', Q_0') of (B, Q_0) based on $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ is as follows:*

1. *B' is the subcomponent of B based on $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$*
2. *$Q_0' = Q_0{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$*

We write $s{\restriction}\mathsf{B}'$ as an abbreviation for $s{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$.

Definition 12 (Execution Projection) *Let (B, Q_0) be a BIP system where $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and let (B', Q_0'), with $\mathsf{B}' = \gamma'(\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k})$ be the subsystem of (B, Q_0) based on $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$. Let $P' = P_{j_1} \cup \cdots \cup P_{j_k}$,i.e., P' is the set of ports of (B', Q_0'). Let $\rho = s_0\mathsf{a}_1 s_1 \ldots s_{i-1}\mathsf{a}_i s_i \ldots$ be an execution of (B, Q_0). Then, $\rho{\restriction}(\mathsf{B}', Q_0')$, the projection of ρ onto (B', Q_0'), is the sequence resulting from:*

1. *replacing each s_i by $s_i{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$, i.e., replacing each state by its projection onto $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$*
2. *removing all $\mathsf{a}_i s_i$ where $\mathsf{a}_i \notin \gamma'$*
3. *replacing each a_i by $\mathsf{a}_i \cap P'$, i.e., replacing each interaction by its projection onto the port set P'*

Proposition 1 (Execution Projection) *Let (B, Q_0) be a BIP system where $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and let (B', Q_0'), with $B' = \gamma'(\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k})$ be the subsystem of (B, Q_0) based on $\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$. Let $\rho = s_0\mathsf{a}_1 s_1 \ldots s_{i-1}\mathsf{a}_i s_i \ldots$ be an execution of (B, Q_0). Then, $\rho{\restriction}(\mathsf{B}', Q_0')$ is an execution of (B', Q_0').*

Proof. Let $\rho = s_0\mathsf{a}_1 s_1 \ldots s_{i-1}\mathsf{a}_i s_i \ldots$, be an execution of (B, Q_0), and let $s_0' = s_0{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$.

Then, by Definition 12, $s_0' \in Q_0'$ and $\rho{\restriction}(B', Q_0') = s_0' b_1 s_1' b_2 s_2' \ldots$ for some $b_1 s_1' b_2 s_2' \ldots$, where $s_j' \in Q' = Q{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ for $j \geq 1$.

Consider an arbitrary step (s_{j-1}', b_j, s_j') of $\rho{\restriction}(\mathsf{B}', Q_0')$. Since $b_j s_j'$ was not removed in Clause 2 of Definition 12, we have

(1) $s_j' = s_k{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ for some $k > 0$ and such that $\mathsf{a}_k \in \gamma'$
(2) $b_j = \mathsf{a}_k$, and
(3) $s_{j-1}' = s_\ell{\restriction}\{\mathsf{B}_{j_1}, \ldots, \mathsf{B}_{j_k}\}$ for the smallest ℓ such that
$\ell < k$ and $\forall\, m : \ell + 1 \leq m < k : \mathsf{a}_m \notin \gamma'$

From (3) and Definitions 3 and 12 $s_\ell{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\} = s_{k-1}{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\}$. Hence $s'_{j-1} = s_{k-1}{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\}$. From $s_{k-1} \xrightarrow{\mathsf{a}_k} s_k$, $\mathsf{a}_k \in \gamma'$, and Definition 3, we have $s_{k-1}{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\} \xrightarrow{\mathsf{a}_k} s_k{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\}$. Hence $s'_{j-1} \xrightarrow{b_j} s'_j$ from $s'_{j-1} = s_{k-1}{\restriction}\{\mathsf{B}_{j_1},\ldots,\mathsf{B}_{j_k}\}$ established above and (1), (2).

Since (s'_{j-1}, b_j, s'_j) was arbitrarily chosen, we conclude that every step of $\rho{\restriction}(\mathsf{B}', Q'_0)$ is a step of (B', Q'_0). Since the first state of $\rho{\restriction}(\mathsf{B}', Q'_0)$ is s_0, and $s_0 \in Q'_0$, we have established that $\rho{\restriction}(\mathsf{B}', Q'_0)$ is an execution of (B', Q'_0). □

Corollary 2 *Let (B', Q'_0) be a subsystem of (B, Q_0), and let P' be the port set of (B', Q'_0). Let s be a reachable state of (B, Q_0). Then $s{\restriction}\mathsf{B}'$ is a reachable state of (B', Q'_0). Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0), and let a be an interaction of (B', Q'_0). Then $s{\restriction}\mathsf{B}' \xrightarrow{\mathsf{a}\cap P'} t{\restriction}\mathsf{B}'$ is a reachable transition of (B', Q'_0).*

Proof. Immediate corollary of Proposition 1. □

3 Characterizing Deadlock-freedom

Definition 13 (Global Deadlock-freedom) *A BIP-system (B, Q_0) is* free of global deadlock *iff, in every reachable state s of (B, Q_0), some interaction a is enabled. Formally, $\forall\, s \in \mathit{rstates}(\mathsf{B}, Q_0), \exists\, \mathsf{a} : s \xrightarrow{\mathsf{a}}_{\mathsf{B}}$.*

Definition 14 (Local Deadlock-freedom) *A BIP-system (B, Q_0) is* free of local deadlock *iff, for every subsystem (B', Q'_0) of (B, Q_0), and every reachable state s of (B, Q_0), (B', Q'_0) has some interaction enabled in state $s{\restriction}\mathsf{B}'$. Formally:*

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

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

Proposition 3 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 4 states that the existence of a supercycle is necessary for a local deadlock to occur: if a set of components, *considered in isolation*, are blocked, then there exists a supercycle consisting of exactly those components, together with the interactions that each component enables.

3.1 Wait-for graphs

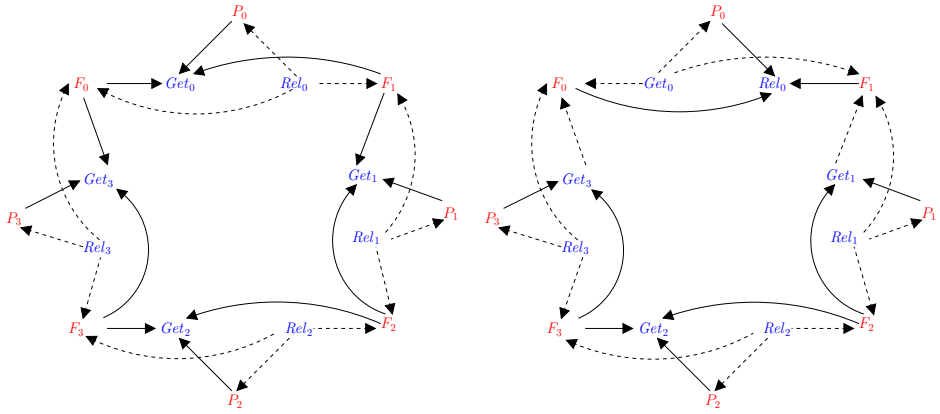
The wait-for-graph for a state s is a directed bipartite and-or graph which contains as nodes the atomic components $\mathsf{B}_1,\ldots,\mathsf{B}_n$, and all the interactions γ. Edges in the wait-for-graph are from a B_i to all the interactions that B_i enables (in s), and from an interaction a to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

Definition 15 (Wait-for-graph $W_{\mathsf{B}}(s)$) *Let $\mathsf{B} = \gamma(\mathsf{B}_1,\ldots,\mathsf{B}_n)$ be a BIP composite component, and let $s = \langle s_1,\ldots,s_n\rangle$ be an arbitrary state of B. The* wait-for-graph *$W_{\mathsf{B}}(s)$ of s is a directed bipartite and-or graph, where*

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

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

Consider the dining philosophers system given in Figure 1. Figure 2(a) shows its wait-for-graph in its sole initial state. Figure 2(b) shows the wait-for-graph after execution of get_0. Edges from components to interactions are shown solid, and edges from interactions to components are shown dashed.



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

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

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

3.2 Supercycles and deadlock-freedom

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

Definition 16 (Supercycle) *Let* $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ *be a composite component and* s *be a state of* B*. A subgraph* SC *of* $W_{\mathsf{B}}(s)$ *is a supercycle in* $W_{\mathsf{B}}(s)$ *if and only if all of the following hold:*

1. *SC is nonempty, i.e., contains at least one node,*
2. *if B_i is a node in SC, then for all interactions a such that there is an edge in $W_{\mathsf{B}}(s)$ from B_i to a:*
   (a) *a is a node in SC, and*
   (b) *there is an edge in SC from B_i to a,*

   that is, $\mathsf{B}_i \rightarrow \mathsf{a} \in W_{\mathsf{B}}(s)$ implies $\mathsf{B}_i \rightarrow \mathsf{a} \in SC$,
3. *if a is a node in SC, then there exists a B_j such that:*
   (a) *B_j is a node in SC, and*
   (b) *there is an edge from a to B_j in $W_{\mathsf{B}}(s)$, and*
   (c) *there is an edge from a to B_j in SC,*

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

where $\mathsf{a} \in SC$ means that a is a node in SC, etc.

Definition 17 (Supercycle-free) $W_{\mathsf{B}}(s)$ *is* supercycle-free *iff there does not exist a supercycle SC in $W_{\mathsf{B}}(s)$. In this case, say that state s is supercycle-free. Formally, we define the predicate $sc\_free_B(s) \triangleq \neg\exists\, SC : SC \subseteq W_{\mathsf{B}}(s)$ and SC is a supercycle.*

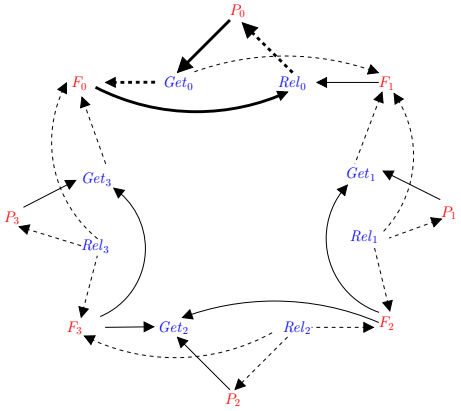


Figure 3: Example supercycle for dining philosophers system of Figure 1.

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

Figure 4 shows an "abstract" example, where these 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. Adding a wait-for-edge from d to B_1 would create a supercycle in Figure 4.

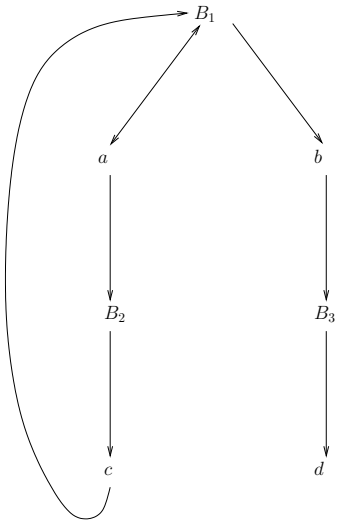


Figure 4: Example where a wait-for cycle does not imply deadlock

The existence of a supercycle is sufficient and necessary for the occurrence of a deadlock, and so checking for supercycles gives a sound and complete check for deadlocks. Write $SC \subseteq W_{\mathsf{B}}(s)$ when SC is a subgraph of $W_{\mathsf{B}}(s)$. Proposition 3 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

Proposition 3 *Let s be a state of B. If $SC \subseteq W_{\mathsf{B}}(s)$ is a supercycle, then all components B_i in SC cannot execute a transition in any state reachable from s, including s itself.*

Proof. Let B_i be an arbitrary component in SC. By Definition 16, every interaction that B_i enables has a wait-for-edge to some other component B_j in SC and so cannot be executed in state s. Hence in any transition from s to another global state t, all of the components B_i in SC remain in the same local state. Hence $SC \subseteq W_{\mathsf{B}}(t)$, i.e., the same supercycle SC remains in global state t. Repeating this argument from state t and onwards leads us to conclude that $SC \subseteq W_{\mathsf{B}}(u)$ for any state u reachable from s. □

Proposition 4 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 4 *Let B' be a subcomponent of B, and let s be an arbitrary state of B such that B', when considered in isolation, has no enabled interaction in state $s{\restriction}\mathsf{B}'$. Then, $W_{\mathsf{B}}(s)$ contains a supercycle.*

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

1. the atomic components of B_i of B',

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

SC satisfies Definition 16 and so is a supercycle. □

We consider subcomponent B' in isolation to avoid other phenomena that prevent interactions from executing, e.g., conspiracies [5]. Now the converse of Proposition 4 is that absence of supercycles in $W_{\mathsf{B}}(s)$ means there is no locally deadlocked subsystem.

Corollary 5 (Supercycle-free implies free of locak deadlock) *If, for every reachable state s of (B, Q_0), $W_{\mathsf{B}}(s)$ is supercycle-free, then (B, Q_0) is free of local deadlock.*

Proof. We establish the contrapositive. Suppose that (B, Q_0) is not free of local deadlock. Then there exists a subsystem (B', Q_0') of (B, Q_0), and a reachable state s of (B', Q_0'), such that B' enables no interaction in state $s{\restriction}\mathsf{B}'$. By Proposition 4, $W_{\mathsf{B}}(s)$ contains a supercycle. □

In the sequel, we wil say "deadlock-free" to mean "free of local deadlock".

3.3 Structural properties of supercycles

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

Definition 18 (Path, path length) *Let G be a directed graph and v a vertex in G. A path π in G is a* finite *sequence $v_0, v_1, \ldots, v_n$ such that (v_i, v_{i+1}) is an edge in G for all $i \in \{0, \ldots, n-1\}$. Write $path_G(\pi)$ iff π is a path in G. Define $first(\pi) = v_0$ and $last(\pi) = v_n$. Let $|\pi|$ denote the length of π, which we define as follows:*

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

Definition 19 (In-depth, Out-depth) *Let G be a directed graph and v a vertex in G. Define the in-depth of v in G, notated as $in\_depth_G(v)$, as follows:*

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

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

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

- *if there exists a path π in G that contains a cycle and starts in v, i.e., $|\pi| = \omega \wedge last(\pi) = v$, then $out\_depth_G(v) = \omega$,*

- *otherwise, let π be a longest (simple) path starting in v. Then $out\_depth_G(v) = |\pi|$.*

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

We use $in\_depth_{\mathsf{B}}(v, s)$ for $in\_depth_{W_{\mathsf{B}}(s)}(v)$, and also $out\_depth_{\mathsf{B}}(v, s)$ for $out\_depth_{W_{\mathsf{B}}(s)}(v)$.

Proposition 6 *A supercycle SC contains no nodes with finite out-depth.*

Proof. By contradiction. Let v be a node in SC with finite out-depth. Hence by Definition 19 all outgoing paths from v are simple (and finite), and end in a sink node w, so w has no outgoing wait-for-edges. By assumption, all atomic components are individually deadlock-free, i.e., they always enable at least one interaction. So if w is an atomic component B_i, we have a wait-for-edge $\mathsf{B}_i \rightarrow \mathsf{a}$ for some interaction a, contradicting the fact that w is a sink node. Hence w is some interaction a. Since a has no outgoing edges, it violates clause 3 in Definition 16, contradicting the assumption that SC is a supercycle. □

Proposition 7 *Every supercycle SC contains at least two nodes.*

Proof. By Definition 16, SC is nonempty, and so contains at least one node v. If v is an interaction a, then by Definition 16, SC also contains some component B_i such that $\mathsf{a} \rightarrow \mathsf{B}_i$. If v is a component B_i, then, by assumption, B_i enables at least one interaction a, and by Definition 16, every interaction that B_i enables must be in SC. Hence in both cases, SC contains at least two nodes. □

Proposition 8 *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 6, SC cannot be a supercycle. □

Proposition 9 *Let $\mathsf{B} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ be a composite component and s a state of B. Let SC be a supercycle in $W_{\mathsf{B}}(s)$, and let SC' be the graph obtained from SC by removing all vertices of finite in-depth and their incident edges. Then SC' is also a supercycle in $W_{\mathsf{B}}(s)$.*

Proof. A vertex with finite in-depth cannot lie on a cycle in SC. Hence by Proposition 8, $SC' \neq \emptyset$. Thus SC' satisfies clause (1) of the supercycle definition (16). Let v be an arbitrary vertex of SC'. Thus $v \in SC$ and $in\_depth_{SC}(v) = \omega$ by definition of SC'. Let w be an arbitrary successor of v in SC. $in\_depth_{SC}(w) = \omega$ by Definition 19. Hence $w \in SC'$. Furthermore, w is a successor of v in SC', by definition of SC'. Thus every vertex v of SC' is also a vertex of SC, and the successors of v in SC' are the same as the successors of v in SC Now since SC is a supercycle, every vertex v in SC has enough successors in SC to satisfy clauses (2) and (3) of the supercycle definition (16). It follows that every vertex v in SC' has enough successors in SC' to satisfy clauses (2) and (3) of the supercycle definition (16). □

Proposition 10 *Every supercycle SC contains a maximal strongly connected component that is itself a supercycle.*

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 acyclc, and so contains at least one node v with no outgoing edges. Let CC be the MSCC corresponding to v. It follows that no node in CC has an edge going to a node outside of CC. Hence, by Definition 16, CC is itself a supercycle. □

Note also that by Proposition 7, CC contains at least two nodes. Hence CC is not a trivial strongly connected component.

4 Superycle Formation and its Consequences

4.1 MSCC Condition

To proceed, we show that wait-for-edges not involving some interaction a and its participants $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ are unaffected by the execution of a. Say that edge e in a wait-for-graph is B_i*-incident* iff B_i is one of the endpoints of e.

Proposition 11 (Wait-for-edge preservation) *Let $s \xrightarrow{\mathsf{a}} t$ be a transition of composite component $B = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$, and let e be a wait-for edge that is not B_i-incident, for every $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$. Then $e \in W_{\mathsf{B}}(s)$ iff $e \in W_{\mathsf{B}}(t)$.*

Proof. Fix e to be an arbitrary wait-for-edge that is not B_i-incident. e is either $\mathsf{B}_j \to b$ or $b \to \mathsf{B}_j$, for some component B_j of B that is not in $\mathit{components}(\mathsf{a})$, and an interaction b (different from a) that B_j participates in. Now $s{\restriction}\mathsf{B}_j = t{\restriction}\mathsf{B}_j$, since $s \xrightarrow{\mathsf{a}} t$ and $\mathsf{B}_j \notin \mathit{components}(\mathsf{a})$. Hence $s(enb_b^j) = t(enb_b^j)$. It follows from Definition 15 that $e \in W_{\mathsf{B}}(s)$ iff $e \in W_{\mathsf{B}}(t)$. □

Proposition 12 (Supercycle Formation Condition) *Assume that $s \xrightarrow{\mathsf{a}} t$ is a transition of (B, Q_0), $W_{\mathsf{B}}(s)$ is supercycle-free, and that $W_{\mathsf{B}}(t)$ contains a supercycle. Then, there exists a CC such that*

1. *CC is a subgraph of $W_{\mathsf{B}}(t)$*
2. *CC is a supercycle*
3. *CC is strongly connected*
4. *there exists a component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ such that B_i is in CC*

Proof. By assumption, there is a supercycle SC that is a subgraph of $W_{\mathsf{B}}(t)$. By Proposition 10, SC contains a subgraph CC that is strongly connected, and is itself a supercycle. This establishes Clauses 1–3.

Now suppose $\mathsf{B}_i \notin CC$ for every $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$. Then, no edge in CC is B_i-incident. Hence, by Proposition 11, every edge in CC is an edge in $W_{\mathsf{B}}(s)$. Hence CC is a subgraph of $W_{\mathsf{B}}(s)$, which contadicts the assumption that $W_{\mathsf{B}}(s)$ is supercycle-free. Hence, $\mathsf{B}_i \in CC$ for some $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, and so Clause 4 is established. □

4.2 Violations of the Supercycle Definition

We wish to determine whether a node v can be part of a strongly-connected supercycle within $W_{\mathsf{B}}(t)$. For example, if a node v has no incoming wait-for-edges or no outgoing wait-for-edges then it cannot be part of a strongly-connected supercycle. We call this a *supercycle violation by node v*. Moreover, a node u may have a *supercycle violation* if its adjacent nodes (w.r.t. wait-for-edges) have supercycle violations. Referring to Definition 16, we see that this depends on the type of node (component or interaction):

1. If a component B_i has an interaction a such that $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(t)$ and a has a supercycle violation, then a cannot be part of a supercycle, and therefore neither can B_i, by Definition 16, Clause 2.
2. If, for an interaction a, all components B_i such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t)$ have a supercycle violation, then all such components B_i cannot be part of a supercycle, and therefore neither can a, by Definition 16, Clause 3.
3. If, for a node v (either component or interaction), all incoming wait-for-edges are from nodes with a supercycle violation, then v cannot be in a *strongly connected* supercycle, since all of its predecessors are not in a strongly connected supercycle.

We note that nodes with supercycle violations may still be connected in a cycle, since a cycle of wait-for-edges does not necesarily imply a supercycle. Hence, to compute supercycle violations properly, we introduce a notion of the *level* of a violation. A node with no incoming wait-for-edges or no outgoing wait-for-edges has a level-1 supercycle-violation. A node whose supercycle-violation is based on outgoing (or incoming) edges to (from) neighbors whose violation level is at most $d-1$, has itself a level-d supercycle-violation.

We formalize the notion of level-d supercycle-violation as the predicate $\mathsf{scViolate}(v, d, t)$, which holds exactly when node v has a level-d supercycle-violation.

Definition 20 (Supercycle violation, $\mathsf{scViolate}(v,d,t)$) *We define the predicate $\mathsf{scViolate}(v,d,t)$ by induction on d, as follows.*

Base case, $d = 1$. $\mathsf{scViolate}(v,1,t)$ *= true iff v has no incoming wait-for-edges or v has no outgoing wait-for-edges, otherwise* $\mathsf{scViolate}(v,1,t) =$ *false.*

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

1. *v is a component B_i and either*
   (a) *there exists interaction a such that $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(t)$ and $\mathsf{scViolate}(\mathsf{a}, d-1, t) =$ true, i.e., B_i enables an interaction a which has a level-$(d-1)$ supercycle-violation, or*
   (b) *for all interactions a such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t)$, $\mathsf{scViolate}(\mathsf{a}, d-1, t) =$ true, i.e., all interactions a that wait for B_i have a level-$(d-1)$ supercycle-violation.*
2. *v is an interaction a and either*
   (a) *for all components B_i such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t)$, $\mathsf{scViolate}(\mathsf{B}_i, d-1, t) =$ true, i.e., all components B_i that a waits for have a level-$(d-1)$ supercycle-violation.*

(b) for all components B_i *such that* $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(t)$, $\mathsf{scViolate}(\mathsf{B}_i, d-1, t) = \mathit{true}$, *i.e., all components* B_i *that enable* a *have a level-*$(d-1)$ *supercycle-violation.*

Figure 5 gives a formal definition of $\mathsf{scViolate}(v, d, t)$.

```
scViolate(v, d, t)
1.  if (d = 1)
2.      if (¬∃ u : u → v ∉ W_B(t)) return(tt) fi        v has no incoming wait-for-edges
3.      if (¬∃ u : v → u ∉ W_B(t)) return(tt) fi        v has no outgoing wait-for-edges
4.      return(ff)
5.  fi
6.  if (v is a component B_i)
7.      if (∃ a : B_i → a ∈ W_B(t) : scViolate(a, d − 1, t)) return(tt) fi
8.      if (∀ a : a → B_i ∈ W_B(t) : scViolate(a, d − 1, t)) return(tt) fi
9.      return(ff)
10. fi
11. if (v is an interaction a)
12.     if (∀ B_i : a → B_i ∈ W_B(t) : scViolate(B_i, d − 1, t)) return(tt) fi
13.     if (∀ B_i : B_i → a ∈ W_B(t) : scViolate(B_i, d − 1, t)) return(tt) fi
14.     return(ff)
15. fi
```

Figure 5: Formal definition of $\mathsf{scViolate}(v, d, t)$

Proposition 13 *If* v *has a level-*d *supercycle-violation, for some* $d \geq 1$, *then* v *cannot be a node of a strongly-connected supercycle.*

Proof. Proof is by induction in d.

Base case, $d = 1$. v has either no incoming edges or no outgoing edges. Hence v cannot be in a strongly connected component of $W_{\mathsf{B}}(t)$.

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.

Subcase 1.1: v has an out-violation. Hence there exists an interaction a such that $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(t)$ and a has a level-$(d-1)$ SC-violation. By the induction hypothesis, a cannot occur in a strongly connected supercycle in $W_{\mathsf{B}}(t)$. Hence by definition (16), B cannot be in a supercycle.

Subcase 1.2: v has an in-violation. Hence for all interactions a such that $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(t)$, a has a level-$(d-1)$ SC-violation. By the induction hypothesis, a cannot occur in a strongly connected supercycle in $W_{\mathsf{B}}(t)$. Hence by definition (16), B cannot be in a strongly connected supercycle, since B has no incoming edges that can be in a strongly connected supercycle.

Case 2: v is an interaction a.

Subcase 2.1: v has an out-violation. Hence for all components B_i such that $\mathsf{a} \rightarrow \mathsf{B}_i \in W_{\mathsf{B}}(t)$, B_i has a level-$(d-1)$ SC-violation. By the induction hypothesis, B_i cannot occur in a strongly connected supercycle in $W_{\mathsf{B}}(t)$. Hence by definition (16), a cannot be in supercycle.

Subcase 2.2: v has an in-violation. Hence for all components B_i such that $\mathsf{B}_i \rightarrow \mathsf{a} \in W_{\mathsf{B}}(t)$, B_i has a level-$(d-1)$ SC-violation. By the induction hypothesis, B_i cannot occur in a strongly connected supercycle in $W_{\mathsf{B}}(t)$. Hence by definition (16), a cannot be in a strongly connected supercycle, since B_i has no incoming edges that can be in a strongly connected supercycle. □

In the other direction, we have a slightly weaker result: if v has no level-d supercycle-violation in $W_{\mathsf{B}}(t)$, for all $d \geq 1$, then v is a node of a supercycle.

Proposition 14 *If* $(\forall\, d \geq 1 : \neg\mathsf{scViolate}(v, d, t))$ *then* v *is a node of a supercycle in* $W_{\mathsf{B}}(t)$.

Proof. Let U be the result of removing from $W_{\mathsf{B}}(t)$ all nodes w, with a supercycle-violation, i.e., all w such that $(\exists\, d : \mathsf{scViolate}(w, d, t))$.

Hence every node u in U has no supercyle-violation. By construction, u negates the supercycle-violation conditions, i.e., all conditions in Definition 20, based on having "enough" successors *and* "enough" predecessors, all of which are also in U, by construction of U.

Since each node $u \in U$ has enough successors in U to satisfy the supercycle definition (Def. 16), we conclude that U itself is a supercycle, by Definition 16. Since $v \in U$ by construction, we have that v is a node of a supercycle. □

5 Global Conditions for Deadlock Freedom

The supercycle formation condition (Definition 12) 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.

Our fundamental condition for the prevention of supercyles then, is to require that, for every reachable transition $s \rightarrow \mathsf{a}t$ resulting from execution of a, that every component B_i of a must exhibit a supercycle-violation (Definition 20) in global state t, i.e., the state resulting from the execution of a. For a given interaction a, we denote that condition $\mathcal{GALT}(\mathsf{a})$, and define it formally below. This condition is, in a sense, the "most general" condition for supercycle-freedom.

If $\mathcal{GALT}(\mathsf{a})$ holds, and global state s is supercycle-free, and $s \rightarrow \mathsf{a}t$, then it follows (as we establish below) that global state t is also supercycle-free. So, by requiring (1) that all initial states are supercycle-free, and (2) that $\mathcal{GALT}(\mathsf{a})$ holds for all interactions $\mathsf{a} \in \gamma$, we obtain, by straightforward induction on length of executions, that every reachable state is supercycle-free.

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

1. To provide a "linear" condition, $\mathcal{GLIN}$, that is easier to evaluate than $\mathcal{GALT}$, since it requires only the evaluation of lengths if wait-for-paths, i.e., it does not have the "alternating" character of $\mathcal{GALT}$.

2. TO provide "local variants of $\mathcal{GALT}$ and $GLin$, 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.

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. First, we define a condition for the prevention of supercyles as a "behavioral restriction condition". Moreover, since implication of $\mathcal{GALT}(\mathsf{a})$ can eb on a "per interaction" basis, with diffrent conditions being aplied to different interactions, we have:

Definition 21 *A behavioral restriction condition $\mathcal{BC}$ is a predicate $\mathcal{BC} : (\mathsf{B}, Q_0, \mathsf{a}) \to \{\mathtt{tt}, \mathtt{ff}\}$.*

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

Definition 22 (Supercycle-freedom preserving) *A behavioral restriction condition $\mathcal{BC}$ is supercycle-freedom preserving iff, for every system (B, Q_0) and $\mathsf{a} \in \gamma$ such that $\mathcal{BC}(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{tt}$, the following holds:*

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

Theorem 15 (Deadlock-freedom via supercycle-freedom preserving restriction) *Assume that*

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

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

Proof. Let u be an arbitrary reachable state. The proof is by induction on the length of the finite execution α that ends in u. Assumption 1 provides the base case, for α having length 0, and so $u \in Q_0$. For the induction step, we establish: for every reachable transition $s \xrightarrow{\mathsf{a}} t$, $W_{\mathsf{B}}(s)$ is supercycle-free implies that $W_{\mathsf{B}}(t)$ is supercycle-free. This is immediate from Assumption 2, and Definition 22. □

We notice that the above proof does not make any use of the requirement that there is a single restriction $\mathcal{BC}$ for all interactions. Hence we can (superficially) strengthen the result by allowing different restriction predicates for different interactions: This is convenient for later application.

Corollary 16 (Deadlock-freedom via several supercycle-freedom preserving restrictions) *Assume that*

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

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

Proof. As for Theorem 15. □

5.1 A Global AND-OR Condition for Deadlock Freedom

Our first global condition is the most general possible: simply assert that, after execution of interaction a, some $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ exhibits a supercycle-violation, as given by $\mathsf{scViolate}(\mathsf{B}_i, d, t)$ (Definition 20).

Definition 23 ($\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$) *Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0). Then, in t, the following holds. For every component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, B_i has a level-d SC-violation for some d. Formally,*

$$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}), \exists\, d \geq 1 : \mathsf{scViolate}(\mathsf{B}_i, d, t).$$

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

Theorem 17 *$\mathcal{GALT}$ is supercycle-freedom preserving*

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

By Proposition 12 there exists a component $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$ such that B_i is in CC, where CC is a strongly connected supercycle that is a subgraph of $W_{\mathsf{B}}(t)$. By Definition 23 and Proposition 13, B_i cannot be a node of a supercycle CC that is strongly connected. Hence we have the desired contradiction. □

5.2 A Global Linear Condition for Deadlock Freedom

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

Case 1 B_i has finite in-depth in $W_{\mathsf{B}}(t)$: then, if $\mathsf{B}_i \in SC$, it can be removed and still leave a supercycle SC', by Proposition 9. Hence SC' exists in $W_{\mathsf{B}}(s)$, and so B_i is not essential to the creation of a supercycle.

Case 2 B_i has finite out-depth in $W_{\mathsf{B}}(t)$: by Proposition 6, B_i cannot be part of a supercycle, and so $SC \subseteq W_{\mathsf{B}}(s)$.

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

We thus impose a condition which guarantees that only Case 2 or Case 3 occur.

Definition 24 ($\mathcal{GLIN}(\mathsf{a})$) *Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of BIP-system (B, Q_0). Then, in t, the following holds. For every component B_i of $\mathit{components}(\mathsf{a})$: either B_i has finite in-depth, or finite out-depth, in $W_{\mathsf{B}}(t)$. Formally,*

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

Proposition 18 *Assume that node v of $W_{\mathsf{B}}(t)$ has a finite in-depth of d in $W_{\mathsf{B}}(t)$, i.e., $in\_depth_{\mathsf{B}}(v,t) = d$. Then* $\mathsf{scViolate}(v, d, t)$

Proof. Proof is by induction on d.

Base case, $d = 1$. Hence by Definitions 18 and 19, v has no incoming wait-for-edges in $W_{\mathsf{B}}(t)$. Hence by Definition 20, $\mathsf{scViolate}(v, 1, t)$.

Inductive step, $d > 1$. By Definition 18 and 19, all predecessors (i.e., nodes u such that $u \rightarrow v \in W_{\mathsf{B}}(t)$) of v have an in-depth of at most $d - 1$. Hence, by the induction hypothesis applied to $d - 1$, we obtain $\mathsf{scViolate}(u, d - 1, t)$ for all predecessors u of v. Hence by Definition 20, Clauses 1b and 2b, $\mathsf{scViolate}(v, 1, t)$. □

Proposition 19 *Assume that node v of $W_{\mathsf{B}}(t)$ has a finite out-depth of d in $W_{\mathsf{B}}(t)$, i.e., $out\_depth_{\mathsf{B}}(v,t) = d$. Then* $\mathsf{scViolate}(v, d, t)$

Proof. Proof is by induction on d.

Base case, $d = 1$. Hence by Definitions 18 and 19, v has no outgoing wait-for-edges in $W_{\mathsf{B}}(t)$. Hence by Definition 20, $\mathsf{scViolate}(v, 1, t)$.

Inductive step, $d > 1$. By Definition 18 and 19, all successors (i.e., nodes u such that $v \rightarrow u \in W_{\mathsf{B}}(t)$) of v have an in-depth of at most $d - 1$. Hence, by the induction hypothesis applied to $d - 1$, we obtain $\mathsf{scViolate}(u, d - 1, t)$ for all predecessors u of v. Hence by Definition 20, Clauses 1a and 2a, $\mathsf{scViolate}(v, 1, t)$. □

Lemma 20 $\forall\, \mathsf{a} \in \gamma : \mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a}) \Rightarrow \mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$.

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

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

By Propositions 18 and 19,

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

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

Theorem 21 *$\mathcal{GLIN}$ is supercycle-freedom preserving*

Proof. Folows immediately from Lemma 20 and Theorem 17. □

5.3 Deadlock freedom using global restrictions

Corollary 22 (Deadlock-freedom via $\mathcal{GALT}$, $\mathcal{GLIN}$) *Assume that*

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

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

Proof. Immediate from Theorems 17, 21 and Corollary 16. □

6 Local Conditions for Deadlock Freedom

Evaluating the global restrictions $\mathcal{GACT}(B, Q_0, \mathsf{a})$, $\mathcal{GLIN}(B, Q_0, \mathsf{a})$ requires checking all reachable transitions of (B, Q_0), which is, in general, subject to state-explosion. We need restrictions which imply a global restriction, and 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 25 (Structure Graph G_{B}, G_i^ℓ, G_{a}^ℓ) *The structure graph G_{B} of composite component $B = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$ is a bipartite graph whose nodes are the $\mathsf{B}_1, \ldots, \mathsf{B}_n$ and all the $\mathsf{a} \in \gamma$. There is an edge between B_i and interaction a iff B_i participates in a, i.e., $\mathsf{B}_i \in components(\mathsf{a})$. Define the* distance *between two nodes to be the number of edges in a shortest path between them. Let G_i^ℓ (G_{a}^ℓ respectively) be the subgraph of G_{B} that contains B_i (a respectively) and all nodes of G_{B} that have a distance to B_i (a respectively) less than or equal to ℓ.*

Definition 26 (Deadlock-checking subsystem, D_{a}^ℓ) *Define D_{a}^ℓ, the* deadlock-checking subsystem for interaction a and depth ℓ, *to be the subsystem of (B, Q_0) based on G_{a}^ℓ.*

Definition 27 (Border node, interior node of D_{a}^ℓ) *A node v of D_{a}^ℓ is a* border-node *iff it has an edge in G_{B} to a node outside of D_{a}^ℓ. If node v of D_{a}^ℓ is not a border node, then it is an* internal node.

Proposition 23 (Wait-for-edge projection) *Let (B', Q_0') be a subsystem of (B, Q_0). Let s be a state of (B, Q_0), and $s' = s{\restriction}\mathsf{B}'$. Let a be an interaction of (B', Q_0'), and $\mathsf{B}_i \in components(\mathsf{a})$ an atomic component of B'. Then (1) $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(s)$ iff $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}'}(s')$, and (2) $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(s)$ iff $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}'}(s')$.*

Proof. By Definition 15, $\mathsf{a} \to \mathsf{B}_i \in W_{\mathsf{B}}(s)$ iff $s{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = \mathit{false}$. Since $s' = s{\restriction}\mathsf{B}'$, we have $s'{\restriction}i = s{\restriction}i$. Hence $s{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = s'{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i})$. By Definition 15, $a \to \mathsf{B}_i \in W_{\mathsf{B}'}(s')$ iff $s'{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = \mathit{false}$. Putting together these three equalities gives us clause (1).

By Definition 15, $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}}(s)$ iff $s{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = \mathit{true}$. Since $s' = s{\restriction}\mathsf{B}'$, we have $s'{\restriction}i = s{\restriction}i$. Hence $s{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = s'{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i})$. By Definition 15, $\mathsf{B}_i \to \mathsf{a} \in W_{\mathsf{B}'}(s')$ iff $s'{\restriction}i(enb_{\mathsf{a}}^{\mathsf{B}_i}) = \mathit{true}$. Putting the above three equalities together gives us clause (2). □

6.2 A Local AND-OR Condition for Deadlock Freedom

We now seek a local condition, which we evaluate in D^ℓ_{a}, and which implies $\mathcal{GALT}$. To achieve a conservative approximation, we make the "pessimistic" assumption that the violation status of border nodes of D^ℓ_{a} cannot be known, since it depends on nodes outside of D^ℓ_{a}. Now, if an internal node v of D^ℓ_{a} can be marked with a level d SC-violation, by applying Definition 20 only within D^ℓ_{a}, and with the border nodes marked as non-violating, then it is also the case, as we show below, that v has a level d SC-violation overall.

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

Definition 28 (Local supercycle violation, $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$) *We define* $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$ *by induction on* d*, as follows.*

Base case, $d = 1$. $\mathsf{scViolateLoc}(v, 1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ *true iff* v *is an interior node of* D^ℓ_{a} *that has either no incoming wait-for-edges or* v *has no outgoing wait-for-edges. A border node cannot be verified to have a level-1 supercycle-violation, since it may wait-for-edge to/from nodes outside of* D^ℓ_{a}*.*

Inductive step, $d > 1$. $\mathsf{scViolateLoc}(v, 1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ *true iff any of the following cases hold. Otherwise* $\mathsf{scViolateLoc}(v, 1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ *false.*

- *v is a component B_i and a border node of D^ℓ_{a}*
  - *there exists interaction aa such that $B_i \rightarrow \mathsf{aa} \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}})$ and $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ true, i.e., B_i enables an interaction aa which has a level-$(d-1)$ supercycle-violation in D^ℓ_{a}.*

  Since this conditions requires only some interaction aa, rather than all, it can be verified within D^ℓ_{a} if a suitable interaction aa within D^ℓ_{a} exists.
- *v is a component B_i and an internal node of D^ℓ_{a}, and either*
  - *there exists interaction aa such that $\mathsf{B}_i \rightarrow \mathsf{aa} \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}})$ and $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ true, i.e., B_i enables an interaction aa which has a level-$(d-1)$ supercycle-violation in D^ℓ_{a}, or*
  - *for all interactions aa such that $\mathsf{aa} \rightarrow B_i \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}})$, $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ true, i.e., all interactions aa that wait for B_i have a level-$(d-1)$ supercycle-violation in D^ℓ_{a}.*
- *v is an interaction aa and an internal node of D^ℓ_{a}, and either*
  - *for all components B_i such that $\mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}})$, $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ true, i.e., all components B_i that aa waits for have a level-$(d-1)$ supercycle-violation in D^ℓ_{a},*
  - *for all components B_i such that $\mathsf{B}_i \rightarrow \mathsf{aa} \in W_{D^\ell_{\mathsf{a}}}(t_{\mathsf{a}})$, $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_{\mathsf{a}}, D^\ell_{\mathsf{a}}) =$ true, i.e., all components B_i that enable aa have a level-$(d-1)$ supercycle-violation in D^ℓ_{a}.*

Note that if v is an interaction and a border node, then $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$ *is false,for all d. This is because v may have both incoming and outgoing wiat-for-edges from/to nodes outside of D^ℓ_a, and these edges may cause v to be in a strongly connected supercycle.*

Figure 6 gives a formal definition of $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$. *Border and interior nodes are handled by separate functions* $\mathsf{scViolateLocBorder}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$, $\mathsf{scViolateLocInterior}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$, *respectively, and the base case $d = 1$ is split across these. Note that* $\mathsf{scViolateLocBorder}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$ *always returns false for $d = 1$.*

$\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$
1. **if** (v is an interior node of D^ℓ_a)
2. **return**($\mathsf{scViolateLocInterior}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$) ▷use function for interior nodes
3. **else** ▷v is a border node
4. **return**($\mathsf{scViolateLocBorder}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$) ▷use function for border nodes
5. **fi**

$\mathsf{scViolateLocInterior}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$
▷ Precondition: v is a not border node of D^ℓ_a
1. **if** ($d = 1$)
2. **if** ($\neg\exists\, u : u \to v \notin W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$) **return**(`tt`) **fi** ▷v has no incoming wait-for-edges
3. **if** ($\neg\exists\, u : v \to u \notin W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$) **return**(`tt`) **fi** ▷v has no outgoing wait-for-edges
4. **return**(`ff`)
5. **fi**
6. **if** (v is a component B_i)
7. **if** ($\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : \mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$) **return**(`tt`) **fi**▷out-violation
8. **if** ($\forall\, \mathsf{aa} : \mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : \mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$) **return**(`tt`) **fi** ▷in-violation
9. **return**(`ff`)
10. **fi**
11. **if** (v is an interaction aa)
12. **if** ($\forall\, \mathsf{B}_i : \mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : \mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$) **return**(`tt`) **fi**▷out-violation
13. **if** ($\forall\, \mathsf{B}_i : \mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : \mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$) **return**(`tt`) **fi** ▷in-violation
14. **return**(`ff`)
15. **fi**

$\mathsf{scViolateLocBorder}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$
▷ Precondition: v is a border node of D^ℓ_a
1. **if** (v is an interaction aa) **return**(`ff`) **fi**
2. **if** (v is a component B_i and $d > 1$)
3. **if** ($\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) : \mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$) **return**(`tt`) **fi**
4. **fi**
5. **return**(`ff`)

Figure 6: Formal definition of $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D^\ell_\mathsf{a})$

Definition 29 ($\mathcal{LACT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$) *Let $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ be an arbitrary reachable transition of D^ℓ_a. Then, in t_a, the following holds. For every component B_i of components(a): B_i has a supercycle-violation that can be confirmed within D^ℓ_a. Formally,*

$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}), \exists\, d \geq 1 : \mathsf{scViolateLoc}(\mathsf{B}_i, d, t_\mathsf{a}, D^\ell_\mathsf{a})$.

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

Proposition 24 *Let t be an arbitrary reachable state of BIP-system (B, Q_0). For all interactions $\mathsf{a} \in \gamma$, and $\ell \geq 1$, let $t_\mathsf{a} = t\restriction D_\mathsf{a}^\ell$. Then*
$\forall\, d \geq 1 : \mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D_\mathsf{a}^\ell) \Rightarrow \mathsf{scViolate}(v, d, t)$.

Proof. Proof is by induction on d.

Base case, $d = 1$. Assume $\mathsf{scViolateLoc}(v, 1, t_\mathsf{a}, D_\mathsf{a}^\ell)$ for some node v. Then, by Definition 28, v cannot be a border node of D_a^ℓ. Hence v is an interior node of D_a^ℓ, and either has no incoming wait-for-edges or no outgoing wait-for-edges. Therefore, in $W_\mathsf{B}(t)$, it is still the case that v either has no incoming wait-for-edges or no outgoing wait-for-edges. Hence $\mathsf{scViolate}(v, 1, t)$ holds.

Inductive step, $d > 1$. Assume $\mathsf{scViolateLoc}(v, d, t_\mathsf{a}, D_\mathsf{a}^\ell)$ for some node v and some $d > 1$. We proceed by cases on Definition 28.

- v is a component B_i and a border node of D_a^ℓ
  Hence there exists interaction aa such that $\mathsf{B}_i \rightarrow \mathsf{aa} \in W_{D_\mathsf{a}^\ell}(t_\mathsf{a})$ and $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell) = \mathit{true}$. By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell)$, we have $\mathsf{scViolate}(\mathsf{aa}, d-1, t)$. Since $W_{D_\mathsf{a}^\ell}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction, we have $B_i \rightarrow \mathsf{aa} \in W_\mathsf{B}(t)$ and $\mathsf{scViolate}(\mathsf{aa}, d-1, t)$. Hence by Definition 20, Clause 1a, we have $\mathsf{scViolate}(v, d, t)$.
- v is a component B_i and an internal node of D_a^ℓ
  We have two subcases.
  - there exists interaction aa such that $B_i \rightarrow \mathsf{aa} \in W_{D_\mathsf{a}^\ell}(t)$ and $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell) = \mathit{true}$.
    Hence there exists interaction aa such that $B_i \rightarrow \mathsf{aa} \in W_{D_\mathsf{a}^\ell}(t_\mathsf{a})$ and $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell) = \mathit{true}$. By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell)$, we have $\mathsf{scViolate}(\mathsf{aa}, d-1, t)$. Since $W_{D_\mathsf{a}^\ell}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction, we have $B_i \rightarrow \mathsf{aa} \in W_\mathsf{B}(t)$ and $\mathsf{scViolate}(\mathsf{aa}, d-1, t)$. Hence by Definition 20, Clause 1a, we have $\mathsf{scViolate}(v, d, t)$.
  - for all interactions aa such that $\mathsf{aa} \rightarrow B_i \in W_{D_\mathsf{a}^\ell}(t_\mathsf{a})$, $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell) = \mathit{true}$.
    Hence for all interactions aa such that $\mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D_\mathsf{a}^\ell}(t_\mathsf{a})$, we have $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell)$. By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{aa}, d-1, t_\mathsf{a}, D_\mathsf{a}^\ell)$, we have for all interactions aa such that $\mathsf{aa} \rightarrow \mathsf{B}_i \in W_{D_\mathsf{a}^\ell}(t_\mathsf{a}) : \mathsf{scViolate}(\mathsf{aa}, d-1, t)$.
    Now $W_{D_\mathsf{a}^\ell}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction. Since B_i is an internal node of D_a^ℓ, all edges $\mathsf{aa} \rightarrow \mathsf{B}_i \in W_\mathsf{B}(t)$ are also edges in $W_{D_\mathsf{a}^\ell}(t_\mathsf{a})$. Hence, from the above, we have for all $\mathsf{aa} \rightarrow B_i \in W_\mathsf{B}(t) : \mathsf{scViolate}(\mathsf{aa}, d-1, t)$. Hence by Definition 20, Clause 1b, we have $\mathsf{scViolate}(v, d, t)$.
- v is an interaction aa and an internal node of D_a^ℓ.
  We have two subcases.

– for all components B_i such that $\mathsf{aa} \to \mathsf{B}_i \in W_{D^\ell_\mathsf{a}}(t)$, $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a}) = \mathit{true}$.

  By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$, we have $\mathsf{scViolate}(\mathsf{B}_i, d-1, t)$. Now $W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction. Since aa is an internal node of D^ℓ_a, all edges $\mathsf{aa} \to \mathsf{B}_i \in W_\mathsf{B}(t)$ are also edges in $W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$. Hence, for all components B_i such that $\mathsf{aa} \to \mathsf{B}_i \in W_\mathsf{B}(t) : \mathsf{scViolate}(\mathsf{B}_i, d-1, t)$. By Definition 20, Clause 2a, $\mathsf{scViolate}(v, d, t)$.

– for all components B_i such that $\mathsf{B}_i \to \mathsf{aa} \in W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$, $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a}) = \mathit{true}$.

  By the induction hypothesis applied to $\mathsf{scViolateLoc}(\mathsf{B}_i, d-1, t_\mathsf{a}, D^\ell_\mathsf{a})$, we have $\mathsf{scViolate}(\mathsf{B}_i, d-1, t)$. Now $W_{D^\ell_\mathsf{a}}(t_\mathsf{a}) \subseteq W_\mathsf{B}(t)$ by construction. Since aa is an internal node of D^ℓ_a, all edges $\mathsf{B}_i \to \mathsf{aa} \in W_\mathsf{B}(t)$ are also edges in $W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$. Hence, for all components B_i such that $\mathsf{B}_i \to \mathsf{aa} \in W_\mathsf{B}(t) : \mathsf{scViolate}(\mathsf{B}_i, d-1, t)$. By Definition 20, Clause 2b, $\mathsf{scViolate}(v, d, t)$.

□

Lemma 25 *Let* $\mathsf{a} \in \gamma$ *be an interaction of BIP-system* (B, Q_0). *Then*
$(\exists\, \ell \geq 1 : \mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell))$ *implies* $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$

Proof. Assume $\mathcal{LALT}(\mathsf{a}, \ell)$ for some $\ell \geq 1$. Let $s \xrightarrow{\mathsf{a}} t$ be an arbitrary reachable transition of BIP-system (B, Q_0), and let $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ be the projection of $s \xrightarrow{\mathsf{a}} t$ onto D^ℓ_a. By Corollary 2, $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ is a reachable transition of D^ℓ_a.

By $\mathcal{LALT}$, we have for some $\ell \geq 1$:
for every reachable transition $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ of D^ℓ_a:
$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}), \exists\, d \geq 1 : \mathsf{scViolateLoc}(\mathsf{B}_i, d, t_\mathsf{a}, D^\ell_\mathsf{a})$.

From this and Proposition 24, we have
for every reachable transition $s \xrightarrow{\mathsf{a}} t$ of (B, Q_0):
$\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}), \exists\, d \geq 1 : \mathsf{scViolate}(\mathsf{B}_i, d, t)$.

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

Theorem 26 $\mathcal{LALT}$ *is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 25 and Theorem 17. □

6.3 A Local Linear Condition for Deadlock Freedom

We now formulate a local version of $\mathcal{GLIN}$. Observe that if $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) < \omega \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) < \omega$, then there is some finite ℓ such that $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) = \ell \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) = \ell$. Then $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) = \ell \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) = \ell$ can be verified in the wait-for-graph of $G_i^{\ell+1}$, since we verify either that all wait-for-paths ending in B_i have length $\leq \ell$, or that all wait-for-paths starting in B_i have length $\leq \ell$. These conditions can be checked in $G_i^{\ell+1}$, since $G_i^{\ell+1}$ contains every node in a wait-for-path of length $\ell+1$ or less and which starts or ends in B_i. Since $G_i^{\ell+1} \subseteq G_\mathsf{a}^{\ell+2}$ for $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, we use $G_\mathsf{a}^{\ell+2}$ instead of the set of subsystems $\{G_i^{\ell+1} : \mathsf{B}_i \in \mathit{components}(\mathsf{a})\}$.

Definition 30 ($\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$) *Let $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ be a reachable transition of D^ℓ_a. Then, in t_a, the following holds. For every component B_i of components(a): either B_i has in-depth at most ℓ, or out-depth at most ℓ, in $W_{D^\ell_\mathsf{a}}(t_\mathsf{a})$. Formally,*

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

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

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

Proposition 27 (In-projection, Out-projection) *Let $\ell \geq 0$, let B_i be an atomic component of B, and let (B', Q'_0) be a subsystem of (B, Q_0) which is based on a superset of $G_i^{\ell+1}$. Let s be a state of (B, Q_0), and $s' = s{\restriction}B'$. Then (1) $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, s) \leq \ell$ iff $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \leq \ell$, and (2) $\mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, s) \leq \ell$ iff $\mathit{out\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \leq \ell$.*

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.

$\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, s) \leq \ell$ implies $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \leq \ell$: Let π be an arbitrary wait-for-path in $W_{\mathsf{B}'}(s')$ ending in B_i. Since (B', Q'_0) is a subsystem of (B, Q_0), by Definition 15 and $s' = s{\restriction}B'$, $W_{\mathsf{B}'}(s')$ is a subgraph of $W_\mathsf{B}(s)$. Hence π is a wait-for-path in $W_\mathsf{B}(s)$. By $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, s) \leq \ell$, we have $|\pi| \leq \ell$. Hence $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \leq \ell$ since π was arbitrarily chosen.

$\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, s) \leq \ell$ follows from $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') \leq \ell$: Let π be an arbitrary wait-for-path ending in B_i. Suppose that $|\pi| > \ell$. Since (B', Q'_0) is a subsystem of (B, Q_0), by Definition 15 and $s' = s{\restriction}\mathsf{B}'$, $W_{\mathsf{B}'}(s')$ is a subgraph of $W_\mathsf{B}(s)$. Hence every edge e in π that is within distance $\ell + 1$ from B_i is also en edge in $W_{\mathsf{B}'}(s')$, since (B', Q'_0) is based on a superset of $G_i^{\ell+1}$. Hence $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, s) > \ell$ implies $\mathit{in\_depth}_{\mathsf{B}'}(\mathsf{B}_i, s') > \ell$. The contrapositive is our desired result. □

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

Lemma 28 *Let a be an interaction of B, i.e., $\mathsf{a} \in \gamma$. If $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ holds for some finite $\ell \geq 0$, then $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$ holds.*

Proof. Let $s \xrightarrow{\mathsf{a}} t$ be a reachable transition of (B, Q_0) and let $\mathsf{B}_i \in \mathit{components}(\mathsf{a})$, $s_\mathsf{a} = s{\restriction}D^\ell_\mathsf{a}$, $t_\mathsf{a} = t{\restriction}D^\ell_\mathsf{a}$. Then $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ is a reachable transition of D^ℓ_a by Corollary 2. By $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$, $\mathit{in\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) \leq \ell \vee \mathit{out\_depth}_{D^\ell_\mathsf{a}}(\mathsf{B}_i, t_\mathsf{a}) \leq \ell$. Hence by Proposition 27, $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) \leq \ell \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) \leq \ell$. So $\mathit{in\_depth}_\mathsf{B}(\mathsf{B}_i, t) < \omega \vee \mathit{out\_depth}_\mathsf{B}(\mathsf{B}_i, t) < \omega$. Hence $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$. □

Theorem 29 *$\mathcal{LLIN}$ is supercycle-freedom preserving*

Proof. Follows immediately from Lemma 28 and Theorem 21. □

6.4 Deadlock freedom using local abd global restrictions

Theorem 30 (Deadlock-freedom via $\mathcal{LALT}$, $\mathcal{LLIN}$) *Assume that*

1. *for all $s_0 \in Q_0$, $W_{\mathsf{B}}(s_0)$ is supercycle-free, and*
2. *for all interactions a of B (i.e., $\mathsf{a} \in \gamma$), one of the following holds:*
   - *(a) $\mathcal{GALT}(\mathsf{B}, Q_0, \mathsf{a})$*
   - *(b) $\mathcal{GLIN}(\mathsf{B}, Q_0, \mathsf{a})$*
   - *(c) $\exists\, \ell \geq 2 : \mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$*
   - *(d) $\exists\, \ell \geq 2 : \mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$*

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

Proof. Immediate from Theorems 17, 21, 26, 29 and Corollary 16. □

7 Implementation and Experiments

7.1 Checking that initial states are supercycle-free

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

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

Figure 7: Procedure to check that all initial states are supercycle-free

Proposition 31 *checkInitSupercycleFree(Q_0) returns true iff all initial states are supercycle-free.*

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

Suppose that U is nonempty. By Proposition 14, U is a supercycle. Since $U \subseteq W_{\mathsf{B}}(s_0)$, we conclude that s_0 not supercycle-free, so false is the correct result in this case.

Now suppose that U is empty. Hence every node in $W_{\mathsf{B}}(s_0)$ has a supercycle violation, and so by Proposition 13, no node of $W_{\mathsf{B}}(s_0)$ can be in a strongly-connected supercycle. Hence $W_{\mathsf{B}}(s_0)$ does not contain a strongly-connected supercycle. So, by Proposition 10, $W_{\mathsf{B}}(s_0)$ does not contain a supercycle. □

7.2 Implementation and Experimentation for the AND-OR Condition

Our implementation evaluates $\mathcal{LALT}$. Figure 8 presents the pseudocode, and Figure 10 presents the pseudocode for computing supercycle violations based on D^ℓ_{a}.

$\mathsf{checkAlt}(\mathsf{B}, Q_0)$ verifies $\mathcal{LALT}$ by iterating over all $\mathsf{a} \in \gamma$. $\mathsf{checkAltInt}(\mathsf{B}, Q_0, \mathsf{a})$ checks $(\exists\, \ell \geq 2 : \mathcal{LALT}(B, Q_0, \mathsf{a}, \ell))$, i.e., if $\mathcal{LALT}$ for a can be verified in some D^ℓ_{a}. We start with $\ell = 2$ since D^2_{a} is the smallest system, in which a supercycle-violation can be confirmed. $\mathsf{checkAltIntDist}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ checks $\mathcal{LALT}(B, Q_0, \mathsf{a}, \ell)$ for a particular ℓ.

For a given D^ℓ_{a} and state t_{a} of D^ℓ_{a}, procedure $\mathsf{violations}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}})$ computes $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$ for $d = 1, \ldots, dd$ where dd is the diameter of D^ℓ_{a}. We use a "memoizing" strategy; we iterate over d, starting with $d = 1$ and incrementing. The iteration for d assumes that the iteration for $d - 1$ has already been computed. The results of each iteration are stored in an array $V_{D^\ell_{\mathsf{a}}, t_{\mathsf{a}}} : (\{B_1, \ldots, B_n\} \cup \gamma) \times \{1, \ldots, dd\} \to \{\mathtt{tt}, \mathtt{ff}\}$. $\mathsf{violationsDist}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d)$ computes the supercycle violations for a given distance d, and $\mathsf{violationsDistNode}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d, v)$ determines the level-d supercycle-violation of node v, if any, i.e., it computes $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$. It calls $\mathsf{violationsDistInteriorNode}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d, v)$ if v is an internal node of D^ℓ_{a}, and $\mathsf{violationsDistBorderNode}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d, v)$ v is a border node of D^ℓ_{a}. These last three procedures, $\mathsf{violationsDist}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d)$, $\mathsf{violationsDistInteriorNode}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d, v)$, and $\mathsf{violationsDistBorderNode}(D^\ell_{\mathsf{a}}, t_{\mathsf{a}}, d, v)$, essentially mimic the mutually recursive functions $\mathsf{scViolateLoc}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$, $\mathsf{scViolateLocInterior}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$, and $\mathsf{scViolateLocBorder}(v, d, t_{\mathsf{a}}, D^\ell_{\mathsf{a}})$.

Complexity. The running time of our implementation is ....TO BE DONE....

7.2.1 Experiments

TO BE DONE:

1. Implement above algorithm
2. Select case studies. We need examples that have "branching waiting"behavior, i.e., a process waits for alternatives. For example, dining philosopher is NOT an example of this.

   One possilbe source is the paper "Evaluating Deadlock Detection Methods for Concurrent Software", IEEE Transactions on Software Engineering, Vol. 22, No. 3, March 1996

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

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

checkAltIntDist($\mathsf{B}, Q_0, \mathsf{a}, \ell$)
$\triangleright$ returns tt iff $\mathcal{LALT}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
1. **forall** reachable transitions $s_\mathsf{a} \xrightarrow{\mathsf{a}} t_\mathsf{a}$ of D^ℓ_a
2. violations($D^\ell_\mathsf{a}, t_\mathsf{a}$);
3. **forall** $\mathsf{B}_i \in$ *components*(a)
4. **if** $(\neg\, V_{D^\ell_\mathsf{a}, t_\mathsf{a}}[v, \ell])$ **return**(ff) **fi** $\triangleright$return ff if no violation for a B_i
5. **endfor**
6. **endfor**;
7. **return**(tt) $\triangleright$return tt if check succeeds for all $\mathsf{B}_i \in$ *components*(a)

Figure 8: Pseudocode for the implementation of the local AND-OR condition.

violations($D^\ell_\mathsf{a}, t_\mathsf{a}$)
$\triangleright$ compute supercycle violations in state t_a of D^ℓ_a
$\triangleright$ Postcondition: $\forall\, v \in \{\mathsf{B}_1, \ldots, \mathsf{B}_n\} \cup \gamma, d' = 1, \ldots, dd : V_{D^\ell_\mathsf{a}, t_\mathsf{a}}[v, d'] = \mathsf{scViolateLoc}(v, d', t_\mathsf{a}, D^\ell_\mathsf{a})$
1. $dd \leftarrow$ the diameter of D^ℓ_a
2. **forall** $d \leftarrow 1$ *to* dd
3. violationsDist($D^\ell_\mathsf{a}, t_\mathsf{a}, d$)

violationsDist($D^\ell_\mathsf{a}, t_\mathsf{a}, d$)
$\triangleright$ compute level-d supercycle violations in state t_a of D^ℓ_a
$\triangleright$ Precondition: $\forall\, v \in \{B_1, \ldots, B_n\} \cup \gamma, d' = 1, \ldots, d-1 : V_{D^\ell_\mathsf{a}, t_\mathsf{a}}[v, d'] = \mathsf{scViolateLoc}(v, d', t_\mathsf{a}, D^\ell_\mathsf{a})$
$\triangleright$ Postcondition: $\forall\, v \in \{B_1, \ldots, B_n\} \cup \gamma, d' = 1, \ldots, d : V_{D^\ell_\mathsf{a}, t_\mathsf{a}}[v, d'] = \mathsf{scViolateLoc}(v, d', t_\mathsf{a}, D^\ell_\mathsf{a})$
1. **forall** $v \in \{B_1, \ldots, B_n\} \cup \gamma$
2. violationsDistNode($D^\ell_\mathsf{a}, t_\mathsf{a}, d, v$)

Figure 9: Procedure to compute all supercycle-violations in state t_a of D^ℓ_a

violationsDistNode($D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}, d, v$)
▷ compute level-d supercycle violation of node v in state t_{a} of D_{a}^{ℓ}, i.e., computes scViolateLoc($v, d, t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell}$)
▷ Precondition: $\forall\, w \in \{\mathsf{B}_1, \ldots, \mathsf{B}_n\} \cup \gamma, d' = 1, \ldots, d-1 : V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[w, d'] = \mathsf{scViolateLoc}(w, d', t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$
▷ Postcondition: $\forall\, w \in \{\mathsf{B}_1, \ldots, \mathsf{B}_n\} \cup \gamma, d' = 1, \ldots, d : V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[w, d'] = \mathsf{scViolateLoc}(w, d', t_{\mathsf{a}}, D_{\mathsf{a}}^{\ell})$
1. **forall** $v \in \{B_1, \ldots, B_n\} \cup \gamma$
2. **if** (v is an interior node of D_{a}^{ℓ})
3. violationsDistInteriorNode($D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}, d, v$)
4. **else**
5. violationsDistBorderNode($D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}, d, v$)

violationsDistInteriorNode($D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}, d, v$)
▷ Precondition: v is an internal node of D_{a}^{ℓ}, $d \geq 1$
1. **if** ($d = 1$)
2. **if** ($\neg\exists\, u : u \to v \notin W_{D_{\mathsf{a}}^{\ell}}(t_{\mathsf{a}})$) **return**(tt) **fi** ▷v has no incoming wait-for-edges
3. **if** ($\neg\exists\, u : v \to u \notin W_{D_{\mathsf{a}}^{\ell}}(t_{\mathsf{a}})$) **return**(tt) **fi** ▷v has no outgoing wait-for-edges
4. **return**(ff)
5. **fi**
6. **if** ($V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[w, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[w, d-1] \leftarrow$; **returnfi**▷return if violation already established
7. **if** (v is a component B_i)
8. **if** ($\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{D_{\mathsf{a}}^{\ell}}(t_{\mathsf{a}}) \wedge V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d] \leftarrow$ tt; **return fi**▷out-violation
9. **if** ($\forall\, \mathsf{aa} : \mathsf{aa} \to \mathsf{B}_i \in W_{D_{\mathsf{a}}^{\ell}}(t_{\mathsf{a}}) : V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d] \leftarrow$ tt; **return fi**▷in-violation
10. $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d] \leftarrow$ ff; **return**
11. **fi**
12. **if** (v is an interaction aa)
13. **if** ($\forall\, \mathsf{B}_i : \mathsf{aa} \to \mathsf{B}_i \in W_{D_{\mathsf{a}}^{\ell}}(t) : V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d] \leftarrow$ tt; **return fi**▷out-violation
14. **if** ($\forall\, \mathsf{B}_i : \mathsf{B}_i \to \mathsf{aa} \in W_{D_{\mathsf{a}}^{\ell}}(t) : V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d] \leftarrow$ tt; **return fi**▷in-violation
15. $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d] \leftarrow$ ff; **return**
16. **fi**

violationsDistBorderNode($D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}, d, v$)
▷ Precondition: v is a border node of D_{a}^{ℓ}
1. **if** (v is an interaction aa) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d] \leftarrow$ ff; **return fi**
2. **if** (v is a component B_i and $d > 1$)
3. **if** ($\exists\, \mathsf{aa} : \mathsf{B}_i \to \mathsf{aa} \in W_{\mathsf{B}}(t) \wedge V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{aa}, d-1]$) **then** $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[\mathsf{B}_i, d] \leftarrow$ tt; **return fi**
4. **fi**
5. $V_{D_{\mathsf{a}}^{\ell}, t_{\mathsf{a}}}[v, d] \leftarrow$ ff; **return**

Figure 10: Procedure to Compute scViolateLoc($v, d, t, D_{\mathsf{a}}^{\ell}$)

7.3 Implementation and Experimentation for the Linear Condition

LDFC-BIP, ($\sim$ 1500 LOC Java) implements our method for finite-state BIP-systems. Pseudocode for LDFC-BIP is shown in Figure 11. $\mathsf{checkLin}(\mathsf{B}, Q_0)$ iterates over each interaction a of (B, Q_0), and checks $(\exists\, \ell \geq 0 : \mathcal{LLIN}(\mathsf{a}, \ell))$ by starting with $\ell = 0$ and incrementing ℓ until either $\mathcal{LLIN}(\mathsf{a}, \ell)$ is found to hold, or D^{ℓ}_{a} has become the entire system and $\mathcal{LLIN}(\mathsf{a}, \ell)$ does not hold. In the latter case, $\mathcal{LLIN}(\mathsf{a}, \ell)$ does not hold for any finite ℓ, and, in practice, computation would halt before D^{ℓ}_{a} had become the entire system, due to exhaustion of resources.

$\mathsf{locLDFC}(a, \ell)$ checks $\mathcal{LLIN}(\mathsf{a}, \ell)$ by examining every reachable transition that executes a, and checking that the final state satisfies Definition 30.

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

$\mathsf{checkLin}(\mathsf{B}, Q_0)$, where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n)$
1. **forall** interactions $\mathsf{a} \in \gamma$
2. **if** $(\mathsf{checkLinInt}(\mathsf{B}, Q_0, \mathsf{a}) = \mathtt{ff})$ **return**$(\mathtt{ff})$
3. **endfor**;
4. **return**$(\mathtt{tt})$ ▷ return `tt` if check succeeds for all $\mathsf{a} \in \gamma$

$\mathsf{checkLinInt}(\mathsf{B}, Q_0, \mathsf{a})$, where $\mathsf{B} \triangleq \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n), \mathsf{a} \in \gamma$
▷ check $(\exists\, \ell \geq 2 : \mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell))$
1. $\ell \leftarrow 2$; ▷ start with $\ell = 2$
2. **while** $(\mathtt{tt})$
3. **if** $(\mathsf{checkLinIntDist}(\mathsf{a}, \ell) = \mathtt{tt})$ **return**$(\mathtt{tt})$ **endif**; ▷ success, so return true
4. **if** $(D^{\ell}_{\mathsf{a}} = \gamma(\mathsf{B}_1, \ldots, \mathsf{B}_n))$ **return**$(\mathtt{ff})$ **endif**; ▷ exhausted all subsystems, return false
5. $\ell \leftarrow \ell + 1$ ▷ increment ℓ until success or intractable or failure
6. **endwhile**

$\mathsf{checkLinIntDist}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$
1. **forall** reachable transitions $s_{\mathsf{a}} \xrightarrow{\mathsf{a}} t_{\mathsf{a}}$ of D^{ℓ}_{a}
2. **if** $(\neg(\forall\, \mathsf{B}_i \in \mathit{components}(\mathsf{a}) : \mathit{in\_depth}_{D^{\ell}_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) = \ell \vee \mathit{out\_depth}_{D^{\ell}_{\mathsf{a}}}(\mathsf{B}_i, t_{\mathsf{a}}) = \ell))$
3. **return**$(\mathtt{ff})$ ▷ check Definition 30
4. **endfor**;
5. **return**$(\mathtt{tt})$ ▷ return `tt` if check succeeds for all transitions

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

7.3.1 Experiment: Dining Philosophers

We consider n philosophers in a cycle, based on the components of Figure 1. Figure 13(a) provides experimental results. The x axis gives the number n of philosophers (and also the number of forks), and the y axis gives the verification time (in milliseconds). We verified that

$\mathcal{LLIN}(a, \ell)$ holds for $\ell = 1$ and all interactions a. Hence dining philosophers is deadlock-free. We increase n and plot the verification time for both LDFC-BIP and D-Finder 2 [8]. D-Finder 2 implements a compositional and incremental method for the verification of BIP-systems. D-Finder (the precursor of D-Finder 2) has been compared favorably with NuSmv and SPIN, outperforming both NuSmv and SPIN on dining philosophers, and outperforming NuSmv on the gas station example [7], treated next. Our results show that LDFC-BIP has a linear increase of computation time with the system size (n), and so outperforms D-Finder 2.

7.3.2 Experiment: Gas Station

A gas station [14] consists of an operator, a set of pumps, and a set of customers. Before using a pump, a customer has to prepay. Then the customer uses the pump, collects his change and starts a new transaction. Before being used by a customer, a pump has to be activated by the operator. When a pump is shut off, it can be re-activated for the next operation.

Figure 12 gives the model for a gas station system for one pump and two customers. The operator has two control locations and three ports. The transition labeled with *prepay* accepts a customer's prepay and activates the pump for the customer. When a customer is served, the transition labeled with *finish* synchronizes the pump and the customer. A pump has three control locations and three ports. Besides the synchronization between the operator and customer through activate and finish ports, a pump and a customer are synchronized through *start* ports.

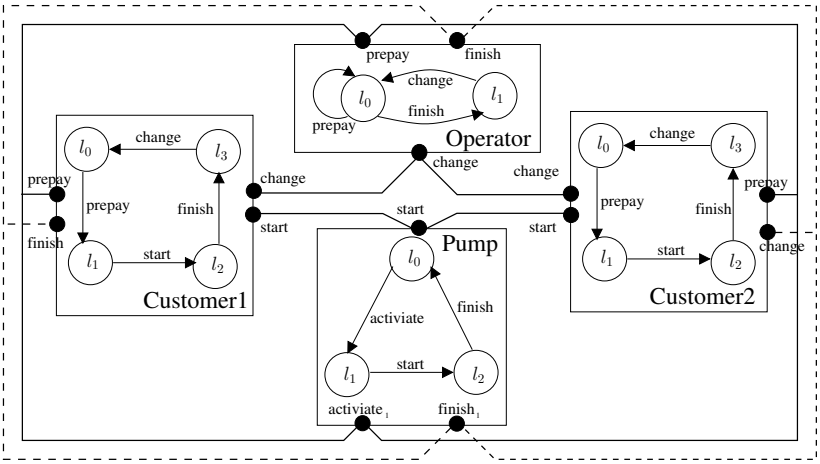
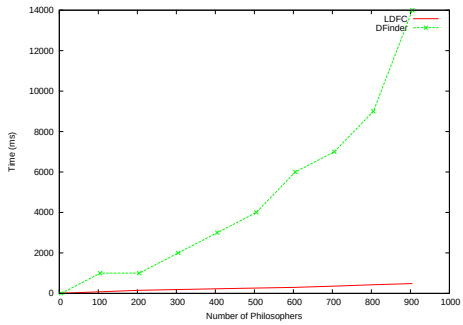
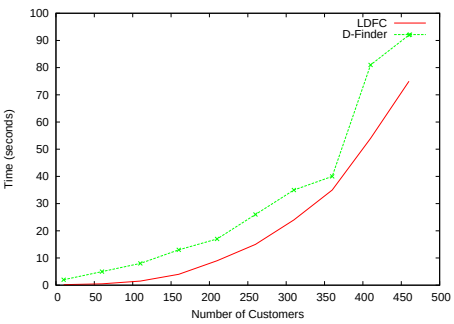


Figure 12: Sketch of Gas Station

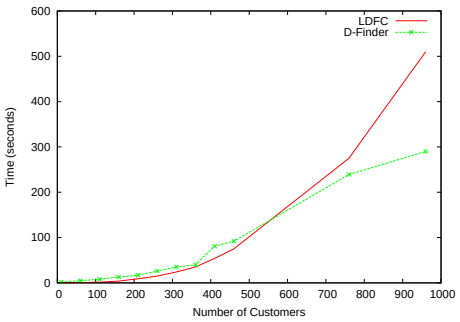
We verified $\mathcal{LLIN}(\mathsf{B}, Q_0, \mathsf{a}, \ell)$ for $\ell = 2$ and all interactions a. Hence gas station is deadlock-free. Figures 13(b), 13(c), and 13(d) present the verification times using LDFC-BIP and D-Finder 2. We consider a system with 3 pumps and variable number of customers. In these figures, the x axis gives the number n of customers, and the y axis gives the verification time (in seconds). D-Finder 2 suffers state-explosion at $n = 1800$, because we consider only three pumps, and so the incremental method used by D-Finder 2 deteriorates. LDFC-BIP outperforms D-Finder 2 as the number of customers increases.



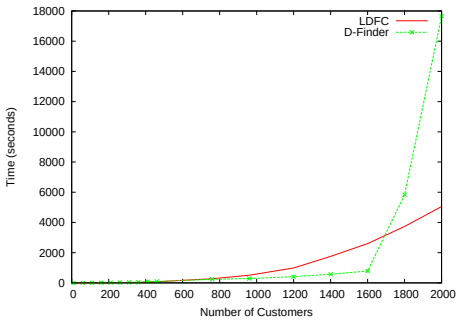
(a) Dining philosophers benchmark.



(b) Gas station benchmark 1.



(c) Gas station benchmark 2.



(d) Gas station benchmark 3.

Figure 13: Benchmarks generated by our experiments.

8 Discussion, Related Work, and Further Work

Related work. The notions of wait-for-graph and supercycle [3, 4] were initially defined for a shared memory program $P = P_1 \parallel \cdots \parallel P_K$ in *pairwise normal form*: a binary symmettric relation I specifies the directly interacting pairs ("neighbors") $\{P_i, P_j\}$. If P_i has neighbors P_j and P_k, then the code in P_i that interacts with P_j is expressed separately from the code in P_i that interacts with P_k. These synchronization codes are executed synchronously and atomically, so the grain of atomicity is proportional to the degree of I. Attie and Chockler [3] give two polynomial time methods for deadlock freedom. The first checks subsystems consisting of three processes. The second computes the wait-for-graphs of all pair subsystems $P_i \parallel P_j$, and takes their union, for all pairs and all reachable states of each pair. The first method considers only wait-for-paths of length ≤ 2. The second method is prone to false negatives, because wait-for edges generated by different states are all merged together, which can result in spurious supercycles

Gössler and Sifakis [13] use a BIP-like formalism, Interaction Models. They present a criterion for global deadlock freedom, based on an and-or graph with components and constraints as the two sets of nodes. A constraint gives the condition under which a component is blocked. Edges are labeled with conjuncts of the constraints. Deadlock freedom is checked by traversing every cycle, taking the conjunction of all the conditions labeling its edges, and verifying that this conjunction is always false, i.e., verifying the absence of cyclical blocking. No complexity bounds are given. Martens and Majster-Cederbaum [15] present a polynomial time checkable deadlock freedom condition based on structural restrictions: "the communication structure between the components is given by a tree." This restriction allows them to analyze only pair systems. Brookes and Roscoe [11] provide criteria for deadlock freedom of CSP programs based on structural and behavioral restrictions combined with analysis of pair systems. No implementation, or complexity bounds, are given. Aldini and Bernardo [1] use a formalism based on process algebra. They check deadlock by analysing cycles in the connections between software components, and claim scalability, but no complexity bounds are given.

We compared our implementation LDFC-BIP to D-Finder 2 [8]. D-Finder 2 computes a finite-state abstraction for each component, which it uses to compute a global invariant I. It then checks if I implies deadlock freedom. Unlike LDFC-BIP, D-Finder 2 handles infinite state systems. However, LDFC-BIP had superior running time for dining philosophers and gas station (both finite-state).

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

Discussion. Our approach has the following advantages:

Local and global deadlock Our method shows that no subset of processes can be deadlocked, i.e., absence of both local and global deadlock.

Check works for realistic formalism By applying the approach to BIP, we provide an efficient deadlock-freedom check within a formalism from which efficient distributed implementations can be generated [9].

Locality If a component B_i is modified, or is added to an existing system, then $\mathcal{LLIN}(a, \ell)$ only has to be re-checked for B_i and components within distance ℓ of B_i. A condition whose evaluation considers the entire system at once, e.g., [1, 8, 13] would have to be re-checked for the entire system.

Easily parallelizable Since the checking of each subsystem D_a^ℓ is independent of the others, the checks can be carried out in parallel. Hence our method can be easily parallelized and distributed, for speedup, if needed. Alternatively, performing the checks sequentially minimizes the amount of memory needed.

Framework aspect Supercycles and in/out-depth provide a *framework* for deadlock-freedom. Conditions more general and/or discriminating than the one presented here should be devisable in this framework. This is a topic for future work.

Further work. Our implementation uses explicit state enumeration. Using BDD's may improve the running time when $\mathcal{LLIN}(a, \ell)$ holds only for large ℓ. An enabled port p enables all interactions containing p. Deadlock-freedom conditions based on ports could exploit this interdepence among interaction enablement. Our implementation should produce *counterexamples* when a system fails to satisfy $\mathcal{LLIN}(a, \ell)$. *Design rules* for ensuring $\mathcal{LLIN}(a, \ell)$ 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 $\mathcal{LLIN}(a, \ell)$ on *infinite state* systems will be devised, e.g.,, by extracting proof obligations and verifying using SMT solvers. We will extend our method to *Dynamic BIP*, [10], where participants can add and remove interactions at run time.

References

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

[2] Paul C. Attie. Synthesis of large concurrent programs via pairwise composition. In *CONCUR'99*, number 1664 in LNCS. Springer-Verlag, August 1999.

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

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

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

[6] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *SEFM*, pages 3–12, September 2006.

[7] S. Bensalem, M. Bozga, T.H. Nguyen, and J. Sifakis. Compositional verification for component-based systems and application. *Software, IET*, 4(3):181–193, 2010.

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

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

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

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

[12] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975.

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

[14] David Heimbold and David Luckham. Debugging Ada tasking programs. *Software, IEEE*, 2(2):47 –57, March 1985.

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

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