An Abstract Framework for Deadlock Prevention in BIP*

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

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

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

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

1 Introduction

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

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

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

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

2 BIP – Behavior Interaction Priority

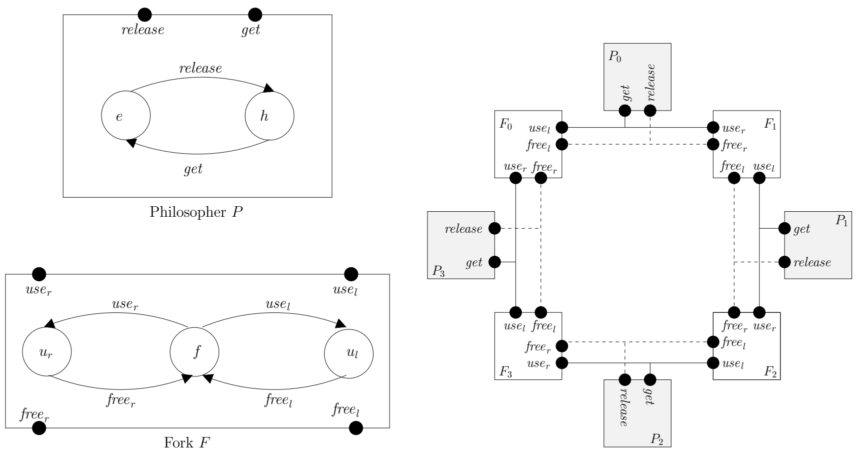
BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, Interaction, and Priority. A technical treatment of priority is beyond the scope of this paper. Adding priorities never introduces a deadlock, since priority enforces a choice between possible transitions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

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

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

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

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



(a) Philosopher P and fork F atomic components.

(b) Dining philosophers composite component with four philosophers.

Fig. 1. Dining philosophers

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

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

Definition 3 (Composite Component). *A* composite component *(or simply* component*) $B \triangleq \gamma(B_1, \ldots, B_n)$ is defined by a composition operator parameterized by a set of interactions $\gamma \subseteq 2^P$. B has a transition system $(Q, \gamma, \to)$, where $Q = Q_1 \times \cdots \times Q_n$ and $\to \subseteq Q \times \gamma \times Q$ is the least set of transitions satisfying the rule*

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

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

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

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

Definition 5 (BIP System). *Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component with transition system $(Q, \gamma, \rightarrow)$, and let $Q_0 \subseteq Q$ be a set of initial states. Then (B, Q_0) is a BIP system.*

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

Definition 6 (Execution). *Let (B, Q_0) be a BIP system with transition system $(Q, \gamma, \rightarrow)$. Let $\rho = s_0 a_1 s_1 \ldots s_{i-1} a_i s_i \ldots$ be an alternating sequence of states of B and interactions of B. Then ρ is an execution of (B, Q_0) iff (1) $s_0 \in Q_0$, and (2) $\forall i > 0 : s_{i-1} \xrightarrow{a_i} s_i$.*

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Characterizing Deadlock-Freedom

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

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

3.1 Wait-For Graphs

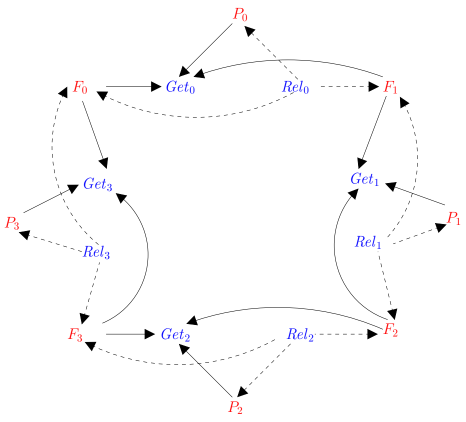
The wait-for-graph for a state s is a directed bipartite and-or graph which contains as nodes the atomic components $B_1, \dots, B_n$, and all the interactions γ. Edges in the wait-for-graph are from a B_i to all the interactions that B_i enables (in s), and from an interaction a to all the components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

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

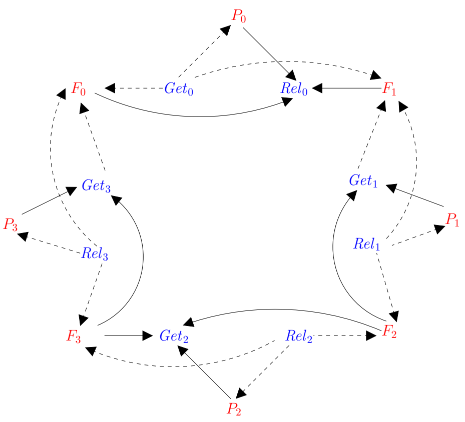
1. *the nodes of $W_B(s)$ are as follows:*
   (a) *the and-nodes are the atomic components B_i, $i \in \{1..n\}$,*
   (b) *the or-nodes are the interactions $a \in \gamma$,*
2. *there is an edge in $W_B(s)$ from B_i to every node a such that $B_i \in C_a$ and $s_i(enb_a^i) = true$, i.e., from B_i to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

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

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



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



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

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

3.2 Supercycles and Deadlock-Freedom

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

Definition 13 (Supercycle). *Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component and s be a state of B. A subgraph SC of $W_B(s)$ is a supercycle in $W_B(s)$ if and only if all of the following hold:*

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

where $a \in SC$ means that a is a node in SC, etc. $W_B(s)$ is *supercycle-free* iff there does not exist a supercycle SC in $W_B(s)$. In this case, say that state s is supercycle-free.

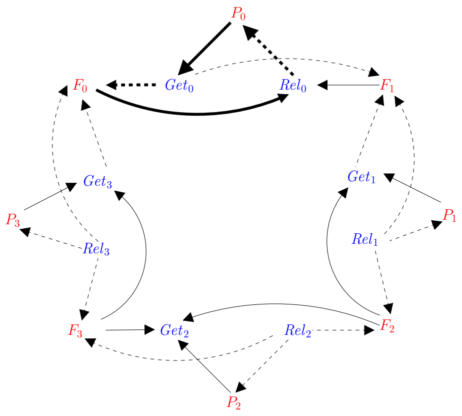


Fig. 3. Example supercycle for dining philosophers system of Figure 1

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

The existence of a supercycle is sufficient and necessary for the occurrence of a deadlock, and so checking for supercycles gives a sound and complete check for deadlocks. Write $SC \subseteq W_B(s)$ when SC is a subgraph of $W_B(s)$. Proposition 2 states that the existence of a supercycle implies a local deadlock: all components in the supercycle are blocked forever.

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

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

Proposition 3 states that the existence of a supercycle is necessary for a local deadlock to occur: if a set of components, *considered in isolation*, are blocked, then there exists a supercycle consisting of exactly those components, together with the interactions that each component enables.

Proposition 3. *Let B' be a subcomponent of B, and let s be an arbitrary state of B such that B', when considered in isolation, has no enabled interaction in state $s \restriction B'$. Then, $W_B(s)$ contains a supercycle.*

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

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

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

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

3.3 Structural Properties of Supercycles

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

Definition 14 (Path, path length). *Let G be a directed graph and v a vertex in G. A path π in G is a* finite *sequence $v_1, v_2, \ldots, v_n$ such that (v_i, v_{i+1}) is an edge in G for all $i \in \{1, \ldots, n-1\}$. Write $path_G(\pi)$ iff π is a path in G. Define $first(\pi) = v_1$ and $last(\pi) = v_n$. Let $|\pi|$ denote the length of π, which we define as follows:*

- *if π is simple, i.e., all v_i, $1 \leq i \leq n$, are distinct, then $|\pi| = n - 1$, i.e., the number of edges in π*
- *if π contains a cycle, i.e., there exist v_i, v_j such that $i \neq j$ and $v_i = v_j$, then $|\pi| = \omega$ (ω for "infinity").*

Definition 15 (In-depth, Out-depth). *Let G be a directed graph and v a vertex in G. Define the in-depth of v in G, notated as $in\_depth_G(v)$, as follows:*

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

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

Likewise define $out\_depth_G(v) = (\text{MAX } \pi : path_G(\pi) \wedge first(\pi) = v : |\pi|)$, the out-depth of v in G, i.e., we consider paths starting (rather than ending) in v.

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

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

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

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

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

Proposition 6. *Let $B = \gamma(B_1, \ldots, B_n)$ be a composite component and s a state of B. Let SC be a supercycle in $W_B(s)$, and let SC' be the graph obtained from SC by removing all vertices of finite in-depth and their incident edges. Then SC' is also a supercycle in $W_B(s)$.*

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

4 A Global Condition for Deadlock Freedom

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

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

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

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

To proceed, we show that wait-for-edges not involving some interaction a and its participants $B_i \in C_a$ are unaffected by the execution of a. Say that edge e in a wait-for-graph is *B_i-incident* iff B_i is one of the endpoints of e.

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

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

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

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

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

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

5 A Local Condition for Deadlock Freedom

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

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

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

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

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

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

Proposition 8 (Wait-for-edge projection). *Let (B', Q_0') be a subsystem of (B, Q_0). Let s be a state of (B, Q_0), and $s' = s\lceil B'$. Let a be an interaction of (B', Q_0'), and $B_i \in C_a$ an atomic component of B'. Then (1) $a \rightarrow B_i \in W_B(s)$ iff $a \rightarrow B_i \in W_{B'}(s')$, and (2) $B_i \rightarrow a \in W_B(s)$ iff $B_i \rightarrow a \in W_{B'}(s')$.*

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

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

Proposition 9 (In-projection, Out-projection). *Let $\ell \geq 0$, let B_i be an atomic component of B, and let (B', Q_0') be a subsystem of (B, Q_0) which is based on a superset of $G_i^{\ell+1}$. Let s be a state of (B, Q_0), and $s' = s{\upharpoonright}B'$. Then (1) $in\_depth_B(B_i, s) \leq \ell$ iff $in\_depth_{B'}(B_i, s') \leq \ell$, and (2) $out\_depth_B(B_i, s) \leq \ell$ iff $out\_depth_{B'}(B_i, s') \leq \ell$.*

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

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

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

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

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

Proof sketch. Immediate from Lemma 1 and Theorem 1.

6 Implementation and Experimentation

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

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

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

6.1 Experiment: Dining Philosophers

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

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

locLDFC$(a, \ell)$
1.  forall reachable transitions $s_a \xrightarrow{a} t_a$ of $D_a^\ell$
2.      if $(\neg(\forall B_i \in C_a : in\_depth_{D_a^\ell}(B_i, t_a) = \ell \vee out\_depth_{D_a^\ell}(B_i, t_a) = \ell))$
3.          return(false)                                     //check Definition 18
4.  endfor;
5.  return(true)                   //return true if check succeeds for all transitions
```

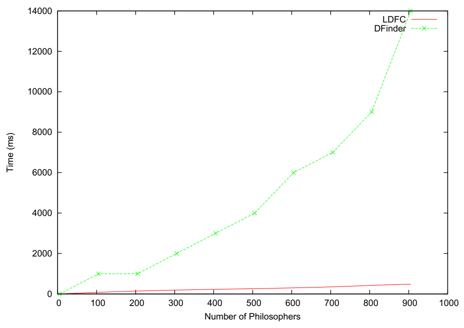
Fig. 4. Pseudocode for the implementation of our method

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

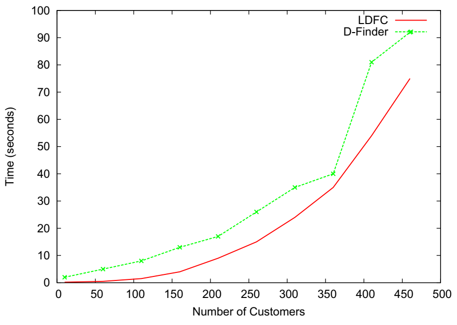
6.2 Experiment: Gas Station

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

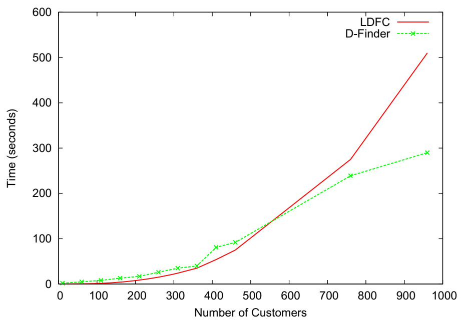
We verified $\mathcal{LDFC}(a, \ell)$ for $\ell = 2$ and all interactions a. Hence gas station is deadlock-free. Figures 5(b), 5(c), and 5(d) present the verification times using LDFC-BIP and D-Finder 2. We consider a system with 3 pumps and variable number of customers. In these figures, the x axis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



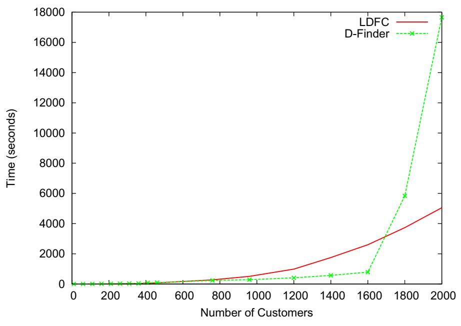
(a) Dining philosophers benchmark.



(b) Gas station benchmark 1.



(c) Gas station benchmark 2.



(d) Gas station benchmark 3.

Fig. 5. Benchmarks generated by our experiments

7 Discussion, Related Work, and Further Work

Related Work. The notions of wait-for-graph and supercycle [3, 4] were initially defined for a shared memory program $P = P_1 \parallel \cdots \parallel P_K$ 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 [12] use a BIP-like formalism, Interaction Models. They present a criterion for global deadlock freedom, based on an and-or graph with components and constraints as the two sets of nodes. A constraint gives the condition under which a component is blocked. Edges are labeled with conjuncts of the constraints. Deadlock freedom is checked by traversing every cycle, taking the conjunction of all the conditions labeling its edges, and verifying that this conjunction is always false, i.e., verifying the absence of cyclical blocking. No complexity bounds are given. Martens and Majster-Cederbaum [14] present a polynomial time checkable deadlock freedom condition based on structural restrictions: "the communication structure between the components is given by a tree." This restriction allows them to analyze only pair systems. Brookes and Roscoe [11] provide criteria for deadlock freedom of CSP programs based on structural and behavioral restrictions combined with analysis of pair systems. No implementation, or complexity bounds, are given. Aldini and Bernardo [1] use a formalism based on process algebra. They check deadlock by analysing cycles in the connections between software components, and claim scalability, but no complexity bounds are given.

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

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

Discussion. Our approach has the following advantages:

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

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

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

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

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

Further Work. Our implementation uses explicit state enumeration. Using BDD's may improve the running time when $\mathcal{LDFC}(a, \ell)$ holds only for large ℓ. An enabled port p enables all interactions containing p. Deadlock-freedom conditions based on ports could exploit this interdepence among interaction enablement. Our implementation should produce *counterexamples* when a system fails to satisfy $\mathcal{LDFC}(a, \ell)$. *Design rules* for ensuring $\mathcal{LDFC}(a, \ell)$ will help users to produce deadlock-free systems, and also to interpret counterexamples. A *fault* may create a deadlock, i.e., a supercycle, by creating wait-for-edges that would not normally arise. Tolerating a fault that creates up to f such spurious wait-for-edges requires that there do not arise during normal (fault-free) operation subgraphs of $W_B(s)$ that can be made into a supercycle by adding f edges. We will investigate criteria for preventing formation of such subgraphs. Methods for evaluating $\mathcal{LDFC}(a, \ell)$ on *infinite state* systems will be devised, e.g.,, by extracting proof obligations and verifying using SMT solvers. We will extend our method to *Dynamic BIP*, [10], where participants can add and remove interactions at run time.

References

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

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