DOI 10.1007/s00165-015-0349-8
The Author(s) © 2015. This article is published with open access at Springerlink.com
Formal Aspects of Computing

Formal Aspects of Computing

A general framework for architecture composability

Paul Attie[1], Eduard Baranov[2], Simon Bliudze[2], Mohamad Jaber[1] and Joseph Sifakis[2]

[1] American University of Beirut, Beirut, Lebanon
[2] École Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland

Abstract. Architectures depict design principles: paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes. They provide means for ensuring correctness by construction by enforcing global properties characterizing the coordination between components. An architecture can be considered as an operator A that, applied to a set of components $\mathcal{B}$, builds a composite component $A(\mathcal{B})$ meeting a characteristic property Φ. Architecture composability is a basic and common problem faced by system designers. In this paper, we propose a formal and general framework for architecture composability based on an associative, commutative and idempotent architecture composition operator $\oplus$. The main result is that if two architectures A_1 and A_2 enforce respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces the property $\Phi_1 \wedge \Phi_2$, that is both properties are preserved by architecture composition. We also establish preservation of liveness properties by architecture composition. The presented results are illustrated by a running example and a case study.

Keywords: Architecture composability, Component-based frameworks, Safety, Liveness, BIP

1. Introduction

Architectures depict design principles: paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes. They provide means for ensuring correctness by construction by enforcing global properties characterizing the coordination between components.

Using architectures largely accounts for our ability to master complexity and develop systems cost-effectively. System developers extensively use libraries of reference architectures ensuring both functional and non-functional properties, for example fault-tolerant architectures, architectures for resource management and QoS control, time-triggered architectures, security architectures and adaptive architectures. Nonetheless, we still lack theory and methods for combining architectures in principled and disciplined fully correct-by-construction design flows.

Correspondence and offprint requests to: S. Bliudze, E-mail: Simon.Bliudze@epfl.ch

Published online: 18 December 2015

Informally speaking, an architecture can be considered as an operator A that, applied to a set of components $\mathcal{B}$ builds a composite component $A(\mathcal{B})$ meeting a characteristic property Φ. In a design process, it is often necessary to combine more than one architectural solution on a set of components to achieve a global property. System engineers use libraries of solutions to specific problems and they need methods for combining them without jeopardizing their characteristic properties. For example, a fault-tolerant architecture combines a set of features building into the environment protections against trustworthiness violations. These include (1) triple modular redundancy mechanisms ensuring continuous operation in case of single component failure; (2) hardware checks to be sure that programs use data only in their defined regions of memory, so that there is no possibility of interference; (3) default to least privilege (least sharing) to enforce file protection. Is it possible to obtain a single fault-tolerant architecture consistently combining these features? The key issue here is *architecture composability* in the integrated solution, which can be formulated as follows:

> Consider two architectures A_1 and A_2, enforcing respectively properties Φ_1 and Φ_2 on a set of components $\mathcal{B}$. That is, $A_1(\mathcal{B})$ and $A_2(\mathcal{B})$ satisfy respectively the properties Φ_1 and Φ_2. Is it possible to find an architecture $A_1 \oplus A_2$ such that the composite component $(A_1 \oplus A_2)(\mathcal{B})$ meets $\Phi_1 \wedge \Phi_2$? For instance, if A_1 ensures mutual exclusion and A_2 enforces a scheduling policy is it possible to find an architecture on the same set of components that satisfies both properties?

Architecture composability is a very basic and common problem faced by system designers. Manifestations of lack of composability are also known as feature interaction in telecommunication systems [CKMRM03].

The development of a formal framework dealing with architecture composability implies a rigorous definition of the concept of architecture as well as of the underlying concepts of components and their interaction. The paper proposes such a framework based on results showing how architectures can be used for achieving correctness by construction in a rigorous component-based design flow [Sif12]. The underlying theory of components and their interaction is inspired from BIP [BS07]. BIP is a component framework rooted in well-defined operational semantics. It proposes an expressive and elegant notion of interaction models for component composition. Interaction models can be studied as sets of Boolean constraints expressing interactions between components. BIP has been fully implemented in a language and the supporting toolset, including compilers and code generators [BBB+11].

BIP allows the description of composite components as an expression $\gamma(\mathcal{B})$, where $\mathcal{B}$ is a set of atomic components and γ is an interaction model. Atomic components are characterized by their behaviour specified as transition systems.

An *interaction model* γ is a set of *interactions*. Each interaction is a set of actions of the composed components, executed synchronously. The meaning of γ can be specified by using operational semantics rules defining the transition relation of the composite component $\gamma(\mathcal{B})$ in terms of transition relations of the composed components $\mathcal{B}$. Intuitively, for each interaction $a \in \gamma$, $\gamma(\mathcal{B})$ can execute a transition labelled by a iff the components involved in a can execute the corresponding transitions labelled by the actions composing a, whereas other components do not move. A formal definition is given in Sect. 2 (Definition 2).

Given a set of components $\mathcal{B}$ an *architecture* is an operator A such that $A(\mathcal{B}) = \gamma(\mathcal{C}, \mathcal{B})$, where γ is an interaction model and $\mathcal{C}$ a set of coordinating components, and $A(\mathcal{B})$ satisfies a characteristic property Φ_A.

According to this definition, an architecture A is a solution to a specific coordination problem, specified by Φ_A, by using an interaction model specified by γ and $\mathcal{C}$. For instance, for distributed architectures, interactions are point-to-point by asynchronous message passing. Other architectures adopt a specific topology (e.g. ring architectures, hierarchically structured architectures). These restrictions entail reduced expressiveness of the interaction model γ that must be compensated by using the additional set of components $\mathcal{C}$ for coordination. The characteristic property Φ_A assigns a meaning to the architecture that can be informally understood without the need for explicit formalization (e.g. mutual exclusion, scheduling policy, clock synchronization).

Our contributions. We propose a general formal framework for architecture composability based on a composition operator '$\oplus$' which is associative, commutative and idempotent. We consider that characteristic properties are the conjunction of safety properties and liveness properties. We show that if two architectures A_1 and A_2 enforce respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces $\Phi_1 \wedge \Phi_2$, that is both properties are preserved by architecture composition. The concept of liveness for architectures derives from the Büchi-acceptance condition. We designate a subset of states of each coordinator as "idle", meaning that it is permissible for the coordinator to remain in such a state forever. Otherwise, the controller must execute infinitely often. The main result guaranteeing liveness preservation is based on a "pairwise non-interference" check of the composed architectures that can be performed algorithmically, in the finite-state case.

This paper extends our previous work [ABB$^+$14a] with the following additional contributions:

1. We introduce a notion of *partial application* of architectures and provide corresponding generalisations of the key results in [ABB$^+$14a];
2. We provide an additional example of hierarchical architecture application, enforcing *fair* mutual exclusion on an arbitrary number of tasks;
3. Finally, we provide full proofs of all the results in the paper.

The paper is structured as follows. Section 2 introduces the notions of component and architecture, as well as the corresponding composition operators. Section 3 presents the key results about the preservation of safety and liveness properties. Section 4 illustrates the application of our framework on an Elevator control use case. Some related work is discussed in Sect. 5. Finally, Sect. 6 concludes the paper.

2. The theory of architectures

2.1. Components and architectures

In order to consider component-based systems, we have to define the notions of *components* and *composition*. We use the BIP notions introduced in [BS07].

Definition 1 (*Components*) A *component* is a Labelled Transition System $B = (Q, q^0, P, \rightarrow)$, where Q is a set of *states*, $q^0 \in Q$ is the *initial state*, P is a set of *ports* and $\rightarrow \subseteq Q \times (2^P \setminus \{\emptyset\}) \times Q$ is a *transition relation*. Each transition is labelled by an *interaction* $\emptyset \neq a \subseteq P$. We call P the *interface* of B.

We use the notations $q \xrightarrow{a} q'$, $q \xrightarrow{a}$ and $q \not\xrightarrow{a}$ as usual. We also denote by Q_B, q_B^0, P_B and $\rightarrow_B$ the constituents of a component B.

In the rest of the paper, we will simplify the notation by writing pq instead of $\{p, q\}$ to denote an interaction, whenever the context allows doing so without introducing ambiguity.

Component composition in BIP is achieved through a synchronisation operator, parameterized by an *interaction model*, which is a set of interactions representing allowed synchronisations among the ports of the participating components.

Definition 2 (*Interaction model*) Let $\mathcal{B} = \{B_1, \dots, B_n\}$ be a finite set of components with $B_i = (Q_i, q_i^0, P_i, \rightarrow)$,[1] such that all P_i are pairwise disjoint, i.e. $\forall\, i \neq j,\ P_i \cap P_j = \emptyset$. Let $P = \bigcup_{i=1}^n P_i$. An *interaction model over* P is a set $\gamma \subseteq 2^P$. We call the set of ports P the *domain* of the interaction model.

The composition of $\mathcal{B}$ with the interaction model γ is given by the component $\gamma(\mathcal{B}) = (Q, q^0, P, \rightarrow)$, where $Q = \prod_{i=1}^n Q_i$, $q^0 = q_1^0 \dots q_n^0$ and $\rightarrow$ is the minimal transition relation inductively defined by the rule

$$\frac{\emptyset \neq a \in \gamma \qquad q_i \xrightarrow{a \cap P_i} q_i' \text{ (if } a \cap P_i \neq \emptyset) \qquad q_i = q_i' \text{ (if } a \cap P_i = \emptyset)}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}. \tag{1}$$

Notice that, although we require transition labels in components to be non-empty, we do allow the empty interaction to be part of an interaction model. According to (1), the empty interaction $\emptyset \in \gamma$ does not have any effect on the composed component $\gamma(\mathcal{B})$. However, it will be useful for the definitions of architecture application and composition as we will discuss below (Remark 1 on page 6).

In the sequel, when speaking of a set of components $\mathcal{B} = \{B_1, \dots, B_n\}$, we will always assume that it satisfies all the conditions of Definition 2.

We are now in position to define the notion of *architecture*. An architecture can be seen as an operator that transforms a set of components into a new composite component. It generalises BIP interaction models, by introducing stateful coordinating components. The interface of an architecture is a set of ports that comprises both the ports of the coordinating components and additional *dangling* ports that must belong to operand components, to which the architecture is applied.

[1] Here and below, we skip the index on the transition relation $\rightarrow$, since it is always clear from the context.

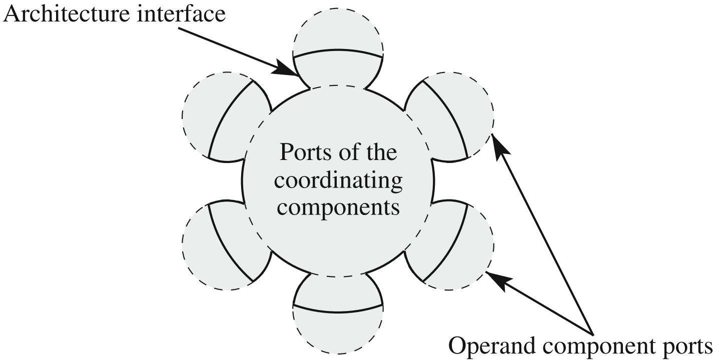


Fig. 1. A diagram illustrating the relation between ports of an architecture and of its operand components: the *inner circle* represents the ports of the coordinating components, the "ears" represent the ports of operand components, the representation of the architecture interface is delimited by the *solid line*

Definition 3 (*Architecture*) An *architecture* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite set of *coordinating components* with pairwise disjoint sets of ports, P_A is a set of ports, such that $\bigcup_{C\in\mathcal{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A}$ is an interaction model over P_A.

An architecture A can be applied to any set of components $\mathcal{B}$ that contains all the dangling ports of A. Intuitively, an architecture enforces coordination constraints on the components in $\mathcal{B}$. The interface P_A of an architecture A contains all ports of the coordinating components $\mathcal{C}$ and some additional ports, which must belong to the components in $\mathcal{B}$ as illustrated in Fig. 1. In the application $A(\mathcal{B})$, the ports belonging to P_A can only participate in the interactions defined by the interaction model γ of A. Ports that do not belong to P_A are not restricted and can participate in any interaction.

In particular, they can join the interactions in γ (see (2) below).

Definition 4 (*Application of an architecture*) Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and let $\mathcal{B}$ be a set of components, such that $\bigcup_{B\in\mathcal{B}} P_B \cap \bigcup_{C\in\mathcal{C}} P_C = \emptyset$ and $P_A \subseteq P \triangleq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}} P_B$. The *application of an architecture* A to the components $\mathcal{B}$ is the component

$$A(\mathcal{B}) \triangleq \left(\gamma \bowtie 2^{P\setminus P_A}\right)(\mathcal{C} \cup \mathcal{B}), \tag{2}$$

where, for interaction models γ' and γ'' over disjoint domains P' and P'' respectively,

$$\gamma' \bowtie \gamma'' \triangleq \{a' \cup a'' \mid a' \in \gamma', a'' \in \gamma''\}$$

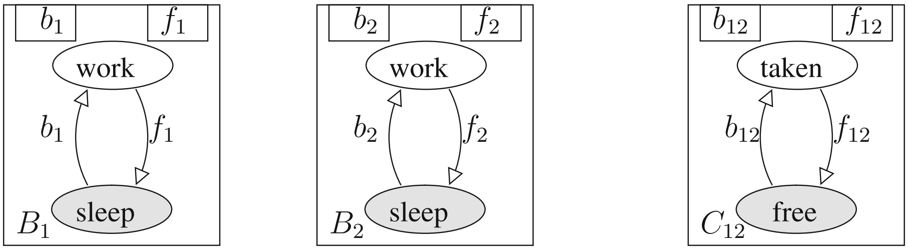
is an interaction model over $P' \cup P''$.

Notice that, when the interface of the architecture covers all ports of the system, i.e. $P = P_A$, we have $2^{P\setminus P_A} = \{\emptyset\}$ and the only interactions allowed in $A(\mathcal{B})$ are those belonging to γ.

Example 1 (Mutual exclusion) Consider the components B_1 and B_2 in Fig. 2a. In order to ensure mutual exclusion of their `work` states, we apply the architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$, where C_{12} is shown in Fig. 2b, $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \big\{\emptyset,\ b_1 b_{12},\ b_2 b_{12},\ f_1 f_{12},\ f_2 f_{12}\big\}$.

The interface P_{12} of A_{12} covers all ports of B_1, B_2 and C_{12}. Hence, the only possible interactions are those explicitly belonging to γ_{12}. Assuming that the initial states of B_1 and B_2 are `sleep`, and that of C_{12} is `free`, neither of the two states (`free`, `work`, `work`) and (`taken`, `work`, `work`) is reachable, i.e. the mutual exclusion property $(q_1 \neq \texttt{work}) \vee (q_2 \neq \texttt{work})$—where q_1 and q_2 are state variables of B_1 and B_2 respectively—holds in $A_{12}(B_1, B_2)$.

Let B_3 be a third component, similar to B_1 and B_2, with the interface $\{b_3, f_3\}$. Since $b_3, f_3 \notin P_{12}$, the interaction model of the application $A_{12}(B_1, B_2, B_3)$ is $\gamma_{12} \bowtie \big\{\emptyset, b_3, f_3\big\}$. (We omit the interaction $b_3 f_3$, since b_3 and f_3 are never enabled in the same state and, therefore, cannot be fired simultaneously.) Thus, the component $A_{12}(B_1, B_2, B_3)$ is the unrestricted product of the components $A_{12}(B_1, B_2)$ and B_3. The application of A_{12} enforces mutual exclusion between the `work` states of B_1 and B_2, but does not affect the behaviour of B_3.



(a) **(b)**

Fig. 2. Component (**a**) and coordinator (**b**) for Example 1 (the initial states of the components are *shaded*)

For the proofs of the results provided in the rest of this paper, it will be convenient to assume that an architecture has precisely one coordinating component, i.e. $\mathcal{C} = \{C\}$. In most cases, this can be done without loss of generality by noticing that the proof argument can be repeated for all coordinating components, since an architecture can have only a finite number of such. However, this assumption can be formalized explicitly by the following lemma.

Lemma 1 *Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and denote by $\gamma_{\mathcal{C}} \stackrel{\Delta}{=} \{a \cap P_{\mathcal{C}} \mid a \in \gamma\}$, with $P_{\mathcal{C}} = \bigcup_{C \in \mathcal{C}} P_C$, the projection of γ onto the ports of the coordinating components of A. Consider an architecture $A' = (\{C'\}, P_A, \gamma)$, where $C' = \gamma_{\mathcal{C}}(\mathcal{C})$. For any set of components $\mathcal{B}$, satisfying the conditions of Definition 4, we have $A(\mathcal{B}) = A'(\mathcal{B})$.*

Proof. First of all, notice that, by Definition 2, $P_{C'} = P_{\mathcal{C}}$. Hence, the conditions of Definition 3 are satisfied and A' is indeed an architecture. Furthermore, $\mathcal{B}$ satisfies the conditions of Definition 4 w.r.t. A'. Hence, the component $A'(\mathcal{B})$ is well defined.

Clearly the state spaces, initial states and interfaces of both components coincide. Thus we only have to prove that so do the transition relations. Let us assume that $\mathcal{C} = \{C_1, \ldots, C_m\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$. We will use $\tilde{q}_i$, $\tilde{q}'_i$ to denote the states of C_i and q_i, q'_i to denote the states of B_i.

By Definition 4, a transition $\tilde{q}_1 \ldots \tilde{q}_m q_1 \ldots q_n \xrightarrow{a} \tilde{q}'_1 \ldots \tilde{q}'_m q'_1 \ldots q'_n$ is possible in $A(\mathcal{B})$ iff $a \neq \emptyset$ and

1. for $i \in [1, m]$, $\tilde{q}_i \xrightarrow{a \cap P_{C_i}} \tilde{q}'_i$ is possible in C_i, or $a \cap P_{C_i} = \emptyset$ and $\tilde{q}_i = \tilde{q}'_i$;
2. for $i \in [1, n]$, $q_i \xrightarrow{a \cap P_{B_i}} q'_i$ is possible in B_i, or $a \cap P_{B_i} = \emptyset$ and $q_i = q'_i$;
3. $a \in \gamma \bowtie 2^{P \setminus P_A}$;

where $P = \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$.

Similarly, the above transition is possible in $A'(\mathcal{B})$ iff $a \neq \emptyset$ and

1. $\tilde{q}_1 \ldots \tilde{q}_m \xrightarrow{a \cap P_{C'}} \tilde{q}'_1 \ldots \tilde{q}'_m$ is possible in C', or $a \cap P_{C'} = \emptyset$ and $\tilde{q}_i = \tilde{q}'_i$, for all $i \in [1, m]$;
2. for $i \in [1, n]$, $q_i \xrightarrow{a \cap P_{B_i}} q'_i$ is possible in B_i, or $a \cap P_{B_i} = \emptyset$ and $q_i = q'_i$;
3. $a \in \gamma \bowtie 2^{P \setminus P_A}$.

If $a \cap P_{C'} \neq \emptyset$, the transition in condition 1 above is possible in C' iff

4. $a \cap P_{C'} \in \gamma_{\mathcal{C}}$ and,
5. for $i \in [1, m]$, $\tilde{q}_i \xrightarrow{a \cap P_{C_i}} \tilde{q}'_i$ is possible in C_i, or $a \cap P_{C_i} = \emptyset$ and $\tilde{q}_i = \tilde{q}'_i$.

Consider $a \in \gamma \bowtie 2^{P \setminus P_A}$. Since $P_{C'} = P_{\mathcal{C}} \subseteq P_A$, we have $a \cap P_{C'} = a \cap P_{\mathcal{C}} = (a \cap P_A) \cap P_{\mathcal{C}}$. Since $a \cap P_A \in \gamma$, we have $a \cap P_{C'} \in \gamma_{\mathcal{C}}$, which concludes the proof. □

2.2. Composition of architectures

As will be further illustrated in Sect. 3, architectures can be intuitively understood as enforcing constraints on the global state space of the system [BS11, Weg96]. More precisely, component coordination is realized by limiting the allowed interactions, thus enforcing constraints on the transitions components can take. From this perspective, architecture composition can be understood as the conjunction of their respective constraints. This intuitive notion is formalized by the two definitions below.

Definition 5 (*Characteristic predicates* [BS10]) Denote $\mathbb{B} = \{\text{true}, \text{false}\}$ and let $\gamma \subseteq 2^P$ be an interaction model over a set of ports P. Its *characteristic predicate* $(\varphi_\gamma : \mathbb{B}^P \to \mathbb{B}) \in \mathbb{B}[P]$ is defined by letting

$$\varphi_\gamma \overset{\Delta}{=} \bigvee_{a \in \gamma} \Big(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \Big).$$

For any valuation $v : P \to \mathbb{B}$, $\varphi_\gamma(v) = \text{true}$ if and only if $a_v \overset{\Delta}{=} \{p \in P \mid v(p) = \text{true}\} \in \gamma$. In such case, we say that the interaction a_v *satisfies* the predicate φ (denoted $a_v \models \varphi$). A predicate $\varphi \in \mathbb{B}[P]$ uniquely defines an interaction model γ_φ, such that $\varphi_{\gamma_\varphi} \equiv \varphi$.[2]

Example 2 (Mutual exclusion (contd.)) Consider the interaction model

$$\gamma_{12} = \big\{\emptyset,\ b_1 b_{12},\ b_2 b_{12},\ f_1 f_{12},\ f_2 f_{12}\big\}$$

from Example 1. The domain of γ_{12} is $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$. Hence, the characteristic predicate of γ_{12} is (omitting the conjunction operator):

$$\begin{aligned}
\varphi_{\gamma_{12}} = {} & \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \ \vee\ b_1\,\overline{b_2}\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \ \vee\ \overline{b_1}\,b_2\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \\
& \vee\ \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,f_1\,\overline{f_2}\,f_{12} \ \vee\ \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,f_2\,f_{12} \\
\equiv {} & (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \\
& \wedge\ (b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})\,.
\end{aligned} \tag{3}$$

Intuitively, the implication $b_1 \Rightarrow b_{12}$, for instance, means that, for the port b_1 to be fired, it is necessary that the port b_{12} be fired in the same interaction [BS10].

Definition 6 (*Architecture composition*) Let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$ be two architectures. The *composition* of A_1 and A_2 is an architecture $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_{A_1} \cup P_{A_2}, \gamma_\varphi)$, where $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

The following lemma states that the interaction model of the composed component consists precisely of the interactions a such that the projections of a onto the interfaces of the composed architectures (A_1 and A_2, resp.) belong to the corresponding interaction models (γ_1 and γ_2, resp.). In other words, these are precisely the interactions that satisfy the coordination constraints enforced by both composed architectures. In particular, as we will show in Theorem 1 (Sect. 3), this means that, for two architectures A_1, A_2 and a set of components $\mathcal{B}$, the execution traces allowed by $A_1 \oplus A_2$ on $\mathcal{B}$ are those that are allowed by both A_1 and A_2, which guarantees the preservation of safety properties by the composition of architectures.

Lemma 2 *Consider two interaction models $\gamma_i \subseteq 2^{P_i}$, for $i = 1, 2$, and let $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. For an interaction $a \subseteq P_1 \cup P_2$, $a \in \gamma_\varphi$ iff $a \cap P_i \in \gamma_i$, for $i = 1, 2$.*

Proof. Let $v(p) = (p \in a)$ be a valuation $P_1 \cup P_2 \to \mathbb{B}$ corresponding to a. We have $a \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$ iff $(\varphi_{\gamma_1} \wedge \varphi_{\gamma_2})(v) = \text{true}$, which is equivalent to $\varphi_{\gamma_1}(v) = \text{true}$ and $\varphi_{\gamma_2}(v) = \text{true}$. Consider a restriction $v' : P_1 \to \mathbb{B}$ of v to P_1, defined by putting $\forall\, p \in P_1$, $v'(p) = v(p)$. Since the variables $p \in P_2 \setminus P_1$ do not appear in φ_{γ_1}, we have $\varphi_{\gamma_1}(v) = \text{true}$ iff $\varphi_{\gamma_1}(v') = \text{true}$, i.e. $a \cap P_1 \in \gamma_1$. The same holds for $a \cap P_2 \in \gamma_2$. □

Remark 1 Every interaction allowed by $A_1 \oplus A_2$ must comprise both an interaction allowed by A_1 and an interaction allowed by A_2. To allow architecture A_1 to progress independently from A_2, one must have $\emptyset \in \gamma_2$ and vice-versa.

[2] Here and below, we use '$\equiv$' to denote logical equivalence of predicates.

Lemma 3 *Consider a set of components $\mathcal{B}$ and two architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$. Let $\tilde{q}_1 \tilde{q}_2 q \xrightarrow{a} \tilde{q}_1' \tilde{q}_2' q'$ be a transition in $(A_1 \oplus A_2)(\mathcal{B})$, where, for $i = 1, 2$, $\tilde{q}_i, \tilde{q}_i' \in \prod_{C \in \mathcal{C}_i} Q_C$ and $q, q' \in \prod_{B \in \mathcal{B}} Q_B$. Then, for $i = 1, 2$, if $a \cap (P_{A_i} \cup P) \neq \emptyset$, then $\tilde{q}_i q \xrightarrow{a \cap (P_{A_i} \cup P)} \tilde{q}_i' q'$ is a transition in $A_i(\mathcal{B})$, where $P = \bigcup_{B \in \mathcal{B}} P_B$.*

Proof. By Lemma 1, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$.

By Definition 6, $a \cap (P_{A_1} \cup P_{A_2}) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lemma 2, $a \cap P_{A_1} \in \gamma_1$. Hence,

$$\tilde{a} \overset{\Delta}{=} a \cap (P_{A_1} \cup P) = \big(a \cap P_{A_1}\big) \cup \big(a \cap (P \setminus P_{A_1})\big) \in \big(\gamma_1 \bowtie 2^{P \setminus P_{A_1}}\big).$$

By the assumption of the lemma, $\tilde{a} \neq \emptyset$. Furthermore, since $\tilde{q}_1 \tilde{q}_2 q \xrightarrow{a} \tilde{q}_1' \tilde{q}_2' q'$, we have by (1),

$$\begin{cases} \tilde{q}_1 \xrightarrow{a \cap P_{C_1}} \tilde{q}_1', & \text{if } a \cap P_{C_1} \neq \emptyset, \\ \tilde{q}_1 = \tilde{q}_1', & \text{if } a \cap P_{C_1} = \emptyset, \end{cases} \quad \text{and,} \quad \text{for } i \in [1, n], \begin{cases} q_i \xrightarrow{a \cap P_i} q_i', & \text{if } a \cap P_i \neq \emptyset, \\ q_i = q_i', & \text{if } a \cap P_i = \emptyset. \end{cases}$$

Since $P_{C_1} \subseteq P_{A_1}$, we have $\tilde{a} \cap P_{C_1} = a \cap P_{C_1}$. Similarly, for any $i \in [1, n]$, $P_i \subseteq P$, hence $\tilde{a} \cap P_i = a \cap P_i$.

Thus, all premises of the instance of the rule (1) for $\tilde{a}$ in $A_1(\mathcal{B})$ are satisfied and we have $\tilde{q}_1 q \xrightarrow{\tilde{a}} \tilde{q}_1' q'$ in $A_1(\mathcal{B})$. For $A_2(\mathcal{B})$, the result is obtained by a symmetrical argument. □

Proposition 1 (Properties of $\oplus$) *Architecture composition $\oplus$ is commutative and associative; it is idempotent if all coordinating components are deterministic; $A_{id} = \big(\emptyset, \emptyset, \{\emptyset\}\big)$ is its neutral element, i.e. for any architecture A, we have $A \oplus A_{id} = A$. Furthermore, for any component B, we have $A_{id}(B) = B$.*

Sketch of the proof Commutativity and associativity follow from the corresponding properties of set union and boolean conjunction. Suppose we have two architectures $A = A'$. As illustrated by Lemma 1, this does not necessarily mean that their sets of coordinating components coincide. However, if all the involved coordinating components are deterministic, then, in any state of $(A \oplus A')(\mathcal{B})$, both architectures will impose the same restrictions, enabling the same interactions between the coordinating and operand components. Hence, we have $(A \oplus A')(\mathcal{B}) = A(\mathcal{B}) = A'(\mathcal{B})$. Since this holds for any set of components $\mathcal{B}$, we conclude that $A \oplus A' = A = A'$. The properties of A_{id} follow immediately from the definitions of architecture application and composition. □

Notice that, by (2), for an arbitrary set of components $\mathcal{B}$ with $P = \bigcup_{B \in \mathcal{B}} P_B$, we have $A_{id}(\mathcal{B}) = \big(2^P\big)(\mathcal{B})$ (cf. Definition 2).

Example 3 (Mutual exclusion (contd.)) Building upon Example 1, let B_3 be a third component, similar to B_1 and B_2, with the interface $\{b_3, f_3\}$. We define two additional architectures A_{13} and A_{23} similar to A_{12}: for $i = 1, 2$, $A_{i3} = (\{C_{i3}\}, P_{i3}, \gamma_{i3})$, where, up to the renaming of ports, C_{i3} is the same as C_{12} in Fig. 2b, $P_{i3} = \{b_i, b_3, b_{i3}, f_i, f_3, f_{i3}\}$ and $\gamma_{i3} = \big\{\emptyset,\ b_i b_{i3},\ b_3 b_{i3},\ f_i f_{i3},\ f_3 f_{i3}\big\}$.

By considering, for $\varphi_{\gamma_{13}}$ and $\varphi_{\gamma_{23}}$, expressions similar to (3), it is easy to compute $\varphi_{\gamma_{12}} \wedge \varphi_{\gamma_{13}} \wedge \varphi_{\gamma_{23}}$ as the conjunction of the following implications:

$$\begin{array}{lllll} b_1 \Rightarrow b_{12} \wedge b_{13}, & f_1 \Rightarrow f_{12} \wedge f_{13}, & b_{12} \Rightarrow b_1 \,\mathtt{XOR}\, b_2, & f_{12} \Rightarrow f_1 \,\mathtt{XOR}\, f_2, & b_{12} \Rightarrow \overline{f_{12}}, \\ b_2 \Rightarrow b_{12} \wedge b_{23}, & f_2 \Rightarrow f_{12} \wedge f_{23}, & b_{13} \Rightarrow b_1 \,\mathtt{XOR}\, b_3, & f_{13} \Rightarrow f_1 \,\mathtt{XOR}\, f_3, & b_{13} \Rightarrow \overline{f_{13}}, \\ b_3 \Rightarrow b_{13} \wedge b_{23}, & f_3 \Rightarrow f_{13} \wedge f_{23}, & b_{23} \Rightarrow b_2 \,\mathtt{XOR}\, b_3, & f_{23} \Rightarrow f_2 \,\mathtt{XOR}\, f_3, & b_{23} \Rightarrow \overline{f_{23}}. \end{array}$$

Finally, it is straightforward to obtain the interaction model for $A_{12} \oplus A_{13} \oplus A_{23}$:

$$\big\{\emptyset,\ b_1 b_{12} b_{13},\ f_1 f_{12} f_{13},\ b_2 b_{12} b_{23},\ f_2 f_{12} f_{23},\ b_3 b_{13} b_{23},\ f_3 f_{13} f_{23}\big\}.$$

Notice that this interaction model is different from the union of the interaction models of the three architectures.

Assuming that the initial states of B_1, B_2 and B_3 are `sleep`, whereas those of C_{12}, C_{13} and C_{23} are `free`, one can observe that none of the states $(\cdot, \cdot, \cdot, \mathtt{work}, \mathtt{work}, \cdot)$, $(\cdot, \cdot, \cdot, \mathtt{work}, \cdot, \mathtt{work})$ and $(\cdot, \cdot, \cdot, \cdot, \mathtt{work}, \mathtt{work})$ are reachable in $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$. Thus, we conclude that the composition of the three architectures,

$(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$, enforces mutual exclusion among the `work` states of all three components. In Sect. 3.1, we provide a general result stating that architecture composition preserves the enforced state properties.

2.3. Hierarchical composition of architectures

The following proposition establishes a link between the architecture composition as defined in the previous section and the usual notion of functional composition.

Proposition 2 (Relation between notions of composition) *Let $\mathcal{B}$ be a set of components and let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that 1) $P_{A_1} \subseteq P_1 \overset{\Delta}{=} \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1} P_B$ and 2) $P_{A_2} \subseteq P_2 \overset{\Delta}{=} \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1\cup\mathcal{C}_2} P_B$. Then $A_2\big(A_1(\mathcal{B})\big)$ is defined and equal to $(A_1 \oplus A_2)(\mathcal{B})$.*

Proof. Clearly, the state spaces, initial states and interfaces of both components coincide. Thus we only have to prove that so do the transition relations. By Lemma 1, we assume $\mathcal{C}_1 = \{C_1\}$, $\mathcal{C}_2 = \{C_2\}$ and $\mathcal{B} = \{B_1, \dots, B_n\}$.

By Definition 4 a transition $q_{C_1} q_{C_2} q_1 \dots q_n \xrightarrow{a} q'_{C_1} q'_{C_2} q'_1 \dots q'_n$ is possible in $A_2\big(A_1(\mathcal{B})\big)$ iff $a \neq \emptyset$ and

1. $q_{C_2} \xrightarrow{a\cap P_{C_2}} q'_{C_2}$ is possible in C_2, or $a \cap P_{C_2} = \emptyset$ and $q_{C_2} = q'_{C_2}$;
2. $q_{C_1} q_1 \dots q_n \xrightarrow{a\cap P_1} q'_{C_1} q'_1 \dots q'_n$ is possible in $A_1(\mathcal{B})$, or $a \cap P_1 = \emptyset$ and $q_{C_1} q_1 \dots q_n = q'_{C_1} q'_1 \dots q'_n$;
3. $a \in \gamma_2 \bowtie 2^{P_2 \setminus P_{A_2}}$.

If $a \cap P_1 \neq \emptyset$, the transition in condition 1 above is possible in $A_1(\mathcal{B})$ iff

4. $q_{C_1} \xrightarrow{a\cap P_{C_1}} q'_{C_1}$ is possible in C_1, or $a \cap P_{C_1} = \emptyset$ and $q_{C_1} = q'_{C_1}$;
5. for $i \in [1, n]$, $q_i \xrightarrow{a\cap P_{B_i}} q'_i$ is possible in B_i, or $a \cap P_{B_i} = \emptyset$ and $q_i = q'_i$;
6. $a \cap P_1 \in \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}$.

Similarly, the above transition is possible in $(A_1 \oplus A_2)(\mathcal{B})$ iff $a \neq \emptyset$ and

1. for $i = 1, 2$, $q_{C_i} \xrightarrow{a\cap P_{C_i}} q'_{C_i}$ is possible in C_i, or $a \cap P_{C_i} = \emptyset$ and $q_{C_i} = q'_{C_i}$;
2. for $i \in [1, n]$, $q_i \xrightarrow{a\cap P_{B_i}} q'_i$ is possible in B_i, or $a \cap P_{B_i} = \emptyset$ and $q_i = q'_i$;
3. $a \in \gamma_{A_1\oplus A_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$.

Thus, to prove the proposition it is sufficient to show that $a \in \gamma_{A_1\oplus A_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$ iff $a \in \gamma_2 \bowtie 2^{P_2 \setminus P_{A_2}}$ and $a \cap P_1 \in \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}$.

For $a \subseteq P_2$, we have $a \in \gamma_{A_1\oplus A_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$ iff $a \cap (P_{A_1} \cup P_{A_2}) \in \gamma_{A_1\oplus A_2}$, i.e. $a \cap (P_{A_1} \cup P_{A_2}) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lemma 2, this is equivalent to $a \cap (P_{A_1} \cup P_{A_2}) \cap P_{A_1} = a \cap P_{A_1} \in \gamma_1$ and $a \cap (P_{A_1} \cup P_{A_2}) \cap P_{A_2} = a \cap P_{A_2} \in \gamma_2$. Since $a \subseteq P_2$, we have $a \cap P_{A_2} \in \gamma_2$ iff $a \in \gamma_2 \bowtie 2^{P_2 \setminus P_{A_2}}$. Finally, since $P_{A_1} \subseteq P_1$, we have $a \cap P_{A_1} \in \gamma_1$ iff $a \cap P_1 \in \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}$. □

The first condition in Proposition 2 states that A_1 can be applied to the behaviours in $\mathcal{B}$ (cf. Definition 4). Similarly, the second condition states that A_2 can be applied to $A_1(\mathcal{B})$. Note that, when $P_{A_i} \subseteq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_i} P_B$ holds for both $i \in \{1, 2\}$—for $i = 1$, this is the first condition of Proposition 2—and none of the architectures involves the ports of the other, i.e. $P_{A_i} \cap \bigcup_{C\in\mathcal{C}_j} P_C = \emptyset$, for $i \neq j \in \{1, 2\}$, then the two architectures are independent and their composition is commutative: $A_2\big(A_1(\mathcal{B})\big) = (A_1 \oplus A_2)(\mathcal{B}) = A_1\big(A_2(\mathcal{B})\big)$.

The following proposition shows that the application of an architecture only affects the components that have ports belonging to its interface. Components that do not involve such ports are not affected, even if they interact with the operand components of the architecture. In Proposition 3, such potential interactions are modelled by applying the architecture A_2, which also provides a context for the comparison of the resulting systems. In the special case, where such independent components do not interact with the architecture operands, one can consider $A_2 = A_{id}$.

Proposition 3 (Application of an architecture to independent components) *Let $\mathcal{B}_1, \mathcal{B}_2$ be two sets of components, such that $\bigcup_{B\in\mathcal{B}_1} P_B \cap \bigcup_{B\in\mathcal{B}_2} P_B = \emptyset$. Let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that $P_{A_1} \subseteq P_1 \overset{\Delta}{=} \bigcup_{B\in\mathcal{B}_1\cup\mathcal{C}_1} P_B$ and $P_{A_2} \subseteq P_2 \overset{\Delta}{=} \bigcup_{B\in\mathcal{B}_1\cup\mathcal{B}_2\cup\mathcal{C}_1\cup\mathcal{C}_2} P_B$. Then $A_2\big(A_1(\mathcal{B}_1, \mathcal{B}_2)\big) = A_2\big(A_1(\mathcal{B}_1), \mathcal{B}_2\big)$.*

Proof. As in the proof of Proposition 2, we notice that the sets of states are equal in both composed components, thus we only have to prove the equality of transition relations. By Lemma 1, we assume $\mathcal{C}_1 = \{C_1\}$ and $\mathcal{C}_2 = \{C_2\}$. Furthermore, let $\mathcal{B}_1 = \{B_1, \dots, B_k\}$ and $\mathcal{B}_2 = \{B_{k+1}, \dots, B_n\}$. We then have $P_1 = P_{C_1} \cup \bigcup_{i=1}^{k} P_{B_i}$ and $P_2 = P_{C_1} \cup P_{C_2} \cup \bigcup_{i=1}^{n} P_{B_i}$.

Assume that we have a transition $q_{C_1} q_{C_2} q_1 \dots q_n \xrightarrow{a} q'_{C_1} q'_{C_2} q'_1 \dots q'_n$ in $A_2\big(A_1(\mathcal{B}_1, \mathcal{B}_2)\big)$. All components can make their corresponding transitions and a can be represented as $a = a_{C_2} \cup a_{\gamma_1} \cup a_1$, where $a_{C_2} \subseteq P_{C_2}$, $a_{\gamma_1} \in \gamma_1$ and $a_1 \in 2^{P_1 \setminus P_{A_1}}$. As $P_{B_i} \cap P_{B_j} = \emptyset$, for all $i \neq j \in [1, n]$, all the ports of $\mathcal{B}_2$ that belong to a are in a_1. Let $a_1 = a_{\mathcal{B}_2} \cup a_2$, where $a_{\mathcal{B}_2} = a \cap \bigcup_{i=k+1}^{n} P_{B_i}$. Then either $a_{\gamma_1} \cup a_2 = \emptyset$ or it is enabled in $A_1(\mathcal{B}_1)$. Hence, interaction $a = a_{C_2} \cup a_{\gamma_1} \cup a_2 \cup a_{\mathcal{B}_2}$ is enabled in $A_2\big(A_1(\mathcal{B}_1), \mathcal{B}_2\big)$.

Assume interaction a is enabled in $A_2\big(A_1(\mathcal{B}_1), \mathcal{B}_2\big)$. It can be represented as $a = a_{C_2} \cup a_{\gamma_1} \cup a_2 \cup a_{\mathcal{B}_2}$. Then interaction $a_{\gamma_1} \cup a_2 \cup a_{\mathcal{B}_2}$ is enabled in $A_1(\mathcal{B}_1, \mathcal{B}_2)$ and consequently a is enabled in $A_2\big(A_1(\mathcal{B}_1, \mathcal{B}_2)\big)$ in the corresponding state. □

Intuitively, Proposition 3 states that one only has to apply the architecture A_1 to those components that have ports involved in its interface. Notice that, in order to compare the semantics of two sets of components, one has to compose them into compound components, by applying *some* architecture. Hence the need for A_2 in Proposition 3. As a special case, one can consider the "most liberal" identity architecture A_{id} (see Proposition 1). A_{id} does not impose any coordination constraints, allowing all possible interactions between the components it is applied to.

Example 4 (Mutual exclusion (contd.)) Example 3 can be generalized to an arbitrary number n of components by repeating the architecture application pairwise. However, this solution requires $n(n-1)/2$ architectures, and so does not scale well. Instead, we apply architectures hierarchically.

Let $n = 4$ and consider two architectures A_{12}, A_{34}, with the respective coordination components C_{12}, C_{34}, that respectively enforce mutual exclusion between B_1, B_2 and B_3, B_4 as in Example 3. Assume furthermore, that an architecture A enforces mutual exclusion between the `taken` states of C_{12} and C_{34}. It is clear that the system $A\big(A_{12}(B_1, B_2), A_{34}(B_3, B_4)\big)$ ensures mutual exclusion between all four components $(B_i)_{i=1}^4$. Furthermore, by the above propositions,

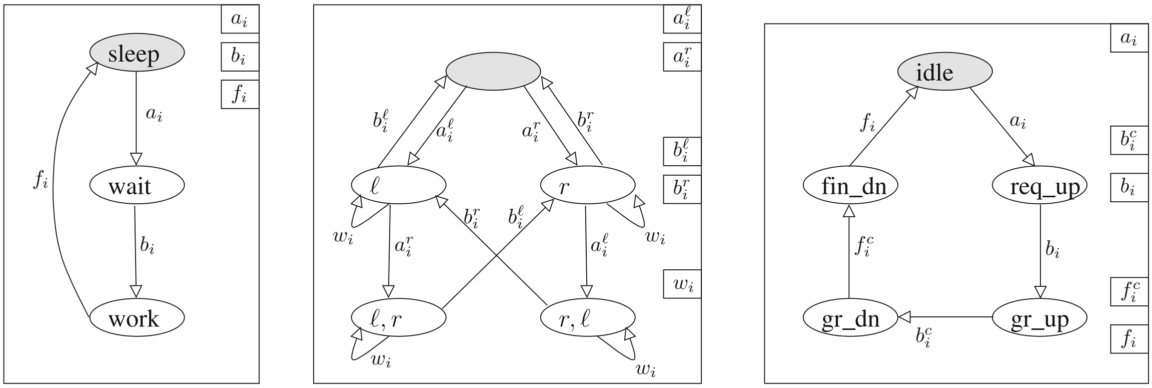
$$\begin{aligned} A\big(A_{12}(B_1, B_2), A_{34}(B_3, B_4)\big) &= A\big(A_{12}\big(B_1, B_2, A_{34}(B_3, B_4)\big)\big) \\ &= A\big(A_{12}\big(A_{34}(B_1, B_2, B_3, B_4)\big)\big) = (A \oplus A_{12} \oplus A_{34})(B_1, B_2, B_3, B_4)\,. \end{aligned}$$

Example 5 (Fair mutual exclusion) Examples 3 and 4 are not fair: a component can be forever denied access to its `work` state. We remedy this by adding requests, and modifying the architecture so that a component that has requested access is eventually granted access. Figure 3a shows a basic component B_i (with i ranging over some set of indices I_B) which *asks* for the critical section (a_i), waits to *begin* its work (b_i), then *finishes* its work (f_i).

We use a hierarchical scheme: a binary tree in which basic components B_i are the leaves, and architectures are the internal nodes. Each architecture receives requests from its left and right children and passes them to its parent. Likewise an architecture receives grants from its parent and passes them to its children.

The architecture at the root of the tree does not pass requests up and does not need to receive grants from above. The details of the root architecture are straightforward, and are omitted. Each architecture A_i (with i ranging over some set of indices I_A, such that $I_A \cap I_B = \emptyset$) has two coordinators:

1. The *priority coordinator*, C_i^{pr}, shown in Fig. 3b, stores requests from the left (a_i^ℓ) and right (a_i^r) subtrees and determines priority based on the order of receiving the requests (FIFO). Priority coordinators enforce the property:



(a) Component B_i **(b)** Priority Coordinator C_i^{pr} **(c)** Access Coordinator C_i^{acc}

Fig. 3. Hierarchical and fair mutual exclusion (the initial states of the components are *shaded*)

> If the left subtree submits a request to its parent before the right subtree, then the left subtree receives a grant from its parent before the right subtree does (and vice-versa).

This is achieved by maintaining a queue of length at most 2, which stores pending requests. In Fig. 3b, the queue is shown labeling each state, with the head of the queue to the left. When there are requests waiting in the queue, the priority coordinator indicates this by enabling the port w_i.

2. The *access coordinator*, C_i^{acc}, shown in Fig. 3c, sends ask requests to the parent architecture (a_i), keeps track and passes the begin and finish actions up (b_i and f_i) and down (b_i^c and f_i^c) the hierarchy tree (the superscript c stands for "child"). Access coordinators enforce the property:

> In total, the left and right subtrees have at most one grant at any time.

This is achieved as follows. The access coordinator proceeds in a cycle: when the corresponding priority coordinator signals that the request queue is not empty, it sends the request to its parent (a_i) and waits for the parent to return a grant (b_i). It then relays this grant to its children (b_i^c), with the priority coordinator ensuring that the grant goes to the higher-priority child in case both children have outstanding requests. Upon receiving a done notification from a child (f_i^c) the access coordinator relays it to its parent (f_i) and returns to the initial state.

Thus all architectures in the tree are instances of the same parameterized architecture $A_{(i,j_\ell,j_r)} = \big(\{C_i^{pr}, C_i^{acc}\}, P_{(i,j_\ell,j_r)}, \gamma_{(i,j_\ell,j_r)}\big)$, where $i \in I_A$ is the index of the architecture and $j_\ell, j_r \in I_A \cup I_B$ are, respectively, the indices of its left and right operands. The interface of $A_{(i,j_\ell,j_r)}$ is

$$P_{(i,j_\ell,j_r)} = \{a_i^\ell, a_i^r, b_i^\ell, b_i^r\} \cup \{a_i, b_i^c, b_i, f_i^c, f_i\} \cup \{a_{j_\ell}, b_{j_\ell}, f_{j_\ell}\} \cup \{a_{j_r}, b_{j_r}, f_{j_r}\},$$

it consists of the ports of the two coordinators and the "up" ports of the two children. The interaction model is

$$\gamma_{(i,j_\ell,j_r)} = \big\{a_i w_i,\ b_i,\ f_i,\ a_i^\ell a_{j_\ell},\ a_i^r a_{j_r},\ b_i^\ell b_i^c b_{j_\ell},\ b_i^r b_i^c b_{j_r},\ f_i^c f_{j_\ell},\ f_i^c f_{j_r}\big\},$$

where

- $a_i w_i$ synchronises the two coordinators, forcing the access coordinator to only send requests to the parent, when there is at least one request from a child waiting;
- b_i and f_i are singleton interactions for propagating begin and finish information to the parent architecture;
- $a_i^\ell a_{j_\ell}$ and $a_i^r a_{j_r}$ correspond, respectively, to asking by the left and right children,
- $b_i^\ell b_i^c b_{j_\ell}$ and $b_i^r b_i^c b_{j_r}$ correspond, respectively, to granting the access to the left or right child;
- $f_i^c f_{j_\ell}$ and $f_i^c f_{j_r}$ correspond, respectively, to the left or right child reporting that it has finished working.

We can apply this architecture to generate any binary tree that is desired.

Clearly, for any two components B_k and B_l, to which such an architecture $A_{(\cdot,k,l)}$ can be applied (see Definition 4), the mutual exclusion holds in $A_{(\cdot,k,l)}(B_k, B_l)$ and the access is granted to B_k and B_l in a fair manner, as discussed above. For more complex systems, correctness can be shown recursively.

Notice that having two coordinators for each architecture, as opposed to combining them into a single more complex coordinator, has the following benefits:

- each coordinator is simpler, hence easier to understand, modify and debug,
- each coordinator enforces a single primitive property, hence there is a one-to-one correspondence between properties and coordinators.

In particular, notice that the behaviour of the access coordinator does not depend on the number of its children. Hence, to change the structure of the overall architecture, we would only need to modify the behaviour of the priority coordinator and adapt accordingly the interaction model.

The ability to factor into simple coordinators is a key advantage of our method. In a typical distributed algorithm (e.g. distributed mutual exclusion) where responsibility for enforcement of properties is distributed amongst all the components, such factorization is very difficult.

2.4. Partial application of architectures

Notice that the main condition, limiting the application of Propositions 2 and 3, is that the architectures must be applicable, i.e. every port of the architecture interface must belong to some component. Below we lift this restriction by introducing the notion of *partial application*. We generalize Definition 4 for architectures $A = (\mathcal{C}, P_A, \gamma)$ applied to sets of components $\mathcal{B}$, such that $P_A \not\subseteq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}} P_B$. This means that the architecture enforces constraints on some ports which are not present in any of the coordinating or base components. In other words, the system obtained by applying the architecture to the set of components $\mathcal{B}$ is not *complete*. The result can then itself be considered as an architecture where the coordinating component is the one obtained by applying to $\mathcal{B} \cup \mathcal{C}$ the projection of interactions in γ.

Definition 7 (*Partial application*) Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and $\mathcal{B}$ be a set of components. Let $P = \bigcup_{B\in\mathcal{B}\cup\mathcal{C}} P_B$. A *partial application* of A to $\mathcal{B}$ is an architecture $A[\mathcal{B}] \triangleq (\{C'\}, P \cup P_A, \gamma \bowtie 2^{P\setminus P_A})$, where $C' \triangleq (\gamma^P \bowtie 2^{P\setminus P_A})(\mathcal{C} \cup \mathcal{B})$ with $\gamma^P = \{a \cap P \mid a \in \gamma\}$ and the operator $\bowtie$ as in Definition 4.

Notice that an architecture obtained by partial application has precisely one coordinating component C'. It is also important to notice that the interaction model in $A[\mathcal{B}]$ is not the same as in the definition of C'. On the other hand, if $P_A \subseteq P$ (as in Definition 4), we have $\gamma_P = \gamma$ and $A[\mathcal{B}] = \big(\{A(\mathcal{B})\}, P, \gamma \bowtie 2^{P\setminus P_A}\big)$.

Lemma 4 *Let $\mathcal{B}$ be a set of components and $A = (\mathcal{C}, P_A, \gamma)$ be an architecture, such that $P_A \subseteq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}} P_B$. Then $A(\mathcal{B}) = A[\mathcal{B}](\emptyset)$.*

Proof. Follows immediately from Definitions 4 and 7. □

Proposition 4 (Partial application to a subset of components) *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two sets of components, such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$, and let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture. Then $A[\mathcal{B}_1 \cup \mathcal{B}_2] = \big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$.*

Proof. Clearly the interfaces of both architectures coincide. Furthermore, since the two architectures are obtained by partial application, each has only one coordinating component (see Definition 7). Thus we have to show that the coordinating components and the interaction models of both architectures coincide.

Let $P_1 = \bigcup_{B\in\mathcal{C}\cup\mathcal{B}_1} P_B$ and $P_2 = \bigcup_{B\in\mathcal{C}\cup\mathcal{B}_2} P_B$. By Definition 7, the interaction models of $A[\mathcal{B}_1 \cup \mathcal{B}_2]$ and $\big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$ are, respectively $\gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}$ and $\big(\gamma \bowtie 2^{P_1\setminus P_A}\big) \bowtie 2^{P_2\setminus P_A}$. Since $2^{P_1\setminus P_A} \bowtie 2^{P_2\setminus P_A} = 2^{(P_1\cup P_2)\setminus P_A}$, we conclude that the interaction models coincide.

It is also clear that the state spaces, initial states and interfaces of both coordination components coincide. Thus, we only have to show that so do the transition relations. Let us consider the coordinating components of the two architectures. By Definition 7, we have[3]

$$\begin{aligned} &A[\mathcal{B}_1 \cup \mathcal{B}_2] = \big(\{C_{12}\}, P_A \cup P_1 \cup P_2, \gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)\,, \\ &\quad \text{with } C_{12} = \big(\gamma^{P_1\cup P_2} \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1 \cup \mathcal{B}_2)\,, \\ &\text{where } \gamma^{P_1\cup P_2} = \{a \cap (P_1 \cup P_2) \mid a \in \gamma\}\,. \end{aligned} \tag{4}$$

Similarly,

$$\begin{aligned} &A[\mathcal{B}_1] = \big(\{C_1\}, P_A \cup P_1, \gamma \bowtie 2^{P_1\setminus P_A}\big)\,, \\ &\text{with } C_1 = \big(\gamma^{P_1} \bowtie 2^{P_1\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1)\,,\ \text{where } \gamma^{P_1} = \{a \cap P_1 \mid a \in \gamma\}\,, \end{aligned} \tag{5}$$

and

$$\begin{aligned} &(A[\mathcal{B}_1])[\mathcal{B}_2] = \big(\{C_2\}, P_A \cup P_1 \cup P_2, \gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)\,, \\ &\quad \text{with } C_2 = \big(\gamma^{P_1\cup P_2} \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)(\{C_1\} \cup \mathcal{B}_2)\,. \end{aligned} \tag{6}$$

Since the interaction models and the constituent atomic components of C_{12} and C_2 coincide, any transition allowed in C_2 is also allowed in C_{12}. Hence, to prove that $C_{12} = C_2$, we have to show that any interaction allowed in C_{12}, after projection, is allowed in C_1. Notice, further, that the interface of C_1 is P_1, whereas those of C_2 and C_{12} are both $P_1 \cup P_2$.

Consider $a \in \gamma^{P_1\cup P_2} \bowtie 2^{(P_1\cup P_2)\setminus P_A}$. By definition of $\bowtie$, $a = a_1 \cup a_2$, with $a_1 \in \gamma^{P_1\cup P_2}$ and $a_2 \subseteq (P_1\cup P_2)\setminus P_A$. By (4), we have $a_1 = \tilde{a}_1 \cap (P_1 \cup P_2)$ with some $\tilde{a}_1 \in \gamma$. We deduce that $a_1 \cap P_1 = \tilde{a}_1 \cap (P_1 \cup P_2) \cap P_1 = \tilde{a}_1 \cap P_1$ and, therefore $a_1 \cap P_1 \in \gamma^{P_1}$. Since, $a \cap P_1 = (a_1 \cap P_1) \cup (a_2 \cap P_1)$ and $a_2 \cap P_1 \subseteq ((P_1 \cup P_2)\setminus P_A) \cap P_1 = P_1\setminus P_A$, we have $a \cap P_1 \in \gamma^{P_1} \bowtie 2^{P_1\setminus P_A}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (5). By (1), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

Proposition 4 generalises Proposition 3. In order to generalize Proposition 2, we first define the application of one architecture to another, by putting

$$A_1[A_2] \overset{\Delta}{=} (A_1 \oplus A_2)[\emptyset]\,. \tag{7}$$

Lemma 5 *For any set of components $\mathcal{B}$ and any architectures A_1 and A_2, we have $(A_1 \oplus A_2)[\mathcal{B}] = (A_1[\mathcal{B}] \oplus A_2)[\emptyset] = (A_1 \oplus A_2[\mathcal{B}])[\emptyset]$.*

Proof. We only prove $(A_1 \oplus A_2)[\mathcal{B}] = (A_1[\mathcal{B}] \oplus A_2)[\emptyset]$. The other equality is symmetrical.

Let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, and $A_1[\mathcal{B}] = (\{C_1'\}, P'_{A_1}, \gamma_1')$. Let $P_1 = \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1} P_B$ and $P_2 = \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1\cup\mathcal{C}_2} P_B$.

Clearly the interfaces of both architectures coincide. Furthermore, since the two architectures are obtained by partial application, each has only one coordinating component (see Definition 7). Thus we have to show that the coordinating components and the interaction models of both architectures coincide.

Let us consider the characteristic predicates of the interaction models. Notice, first, that for any two interaction models $\gamma' \subseteq 2^{P'}$ and $\gamma'' \subseteq 2^{P''}$, over disjoint sets of ports $P' \cap P'' = \emptyset$, one has (cf. Definition 4)

$$\varphi_{\gamma'} \wedge \varphi_{\gamma''} \equiv \varphi_{\gamma'\bowtie\gamma''}\,. \tag{8}$$

Denote the interaction model of $A_1[\mathcal{B}]$ by $\gamma_1' = \gamma_1 \bowtie 2^{P_1\setminus P_{A_1}}$. Clearly, $\varphi_{\left(2^{P_1\setminus P_{A_1}}\right)} \equiv \text{true}$. Hence, by (8), we have $\varphi_{\gamma_1'} \equiv \varphi_{\gamma_1} \wedge \varphi_{\left(2^{P_1\setminus P_{A_1}}\right)} \equiv \varphi_{\gamma_1}$ and, consequently, the characteristic predicate of the interaction model of $(A_1[\mathcal{B}] \oplus A_2)[\emptyset]$ is $\varphi_{\gamma_1'} \wedge \varphi_{\gamma_2} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By a similar argument, we can conclude that the characteristic predicate of the interaction model of $(A_1 \oplus A_2)[\mathcal{B}]$ is also $\varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. Since the interfaces of the two architectures coincide, this implies that so do their interaction models. We denote the interaction model in question by γ_{12}. Recall that $\varphi_{\gamma_{12}} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

[3] Keep in mind the difference between roman C, denoting a single coordinating component, and calligraphic $\mathcal{C}$, denoting a set of coordinating components.

Let us consider the coordinating components of the two architectures. By Definition 7, we have[4]

$$\begin{aligned}&(A_1 \oplus A_2)[\mathcal{B}] = \big(\{C_{12}\}, P_2 \cup P_{A_1} \cup P_{A_2}, \gamma_{12} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big),\\ &\qquad \text{with } C_{12} = \big(\gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)(\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{B}),\\ &\quad \text{where } \gamma_{12}^{P_2} = \{a \cap P_2 \mid a \in \gamma_{12}\}.\end{aligned} \tag{9}$$

Similarly,

$$\begin{aligned}&A_1[\mathcal{B}] = \big(\{C_1\}, P_1 \cup P_{A_1}, \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}\big),\\ &\text{with } C_1 = \big(\gamma_1^{P_1} \bowtie 2^{P_1 \setminus P_{A_1}}\big)(\mathcal{C}_1 \cup \mathcal{B}), \ \text{ where } \gamma_1^{P_1} = \{a \cap P_1 \mid a \in \gamma_1\},\end{aligned} \tag{10}$$

and

$$\begin{aligned}&(A_1[\mathcal{B}] \oplus A_2)[\emptyset] = \big(\{C_2\}, P_2 \cup P_{A_1} \cup P_{A_2}, \gamma_{12} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big),\\ &\qquad \text{with } C_2 = \big(\gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)(\{C_1\} \cup \mathcal{C}_2).\end{aligned} \tag{11}$$

Notice that the interaction models and the constituent atomic components in (9) and (11) coincide. Therefore, any transition allowed in C_2 is also allowed in C_{12}. Hence, to prove that $C_{12} = C_2$, we have to show that any interaction allowed in C_{12}, after projection, is allowed in C_1. Notice, further, that the interface of C_1 is P_1, whereas those of C_2 and C_{12} are both P_2.

Consider $a \in \gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$. By definition of $\bowtie$, $a = a_1 \cup a_2$ with $a_1 \in \gamma_{12}^{P_2}$ and $a_2 \subseteq P_2 \setminus (P_{A_1} \cup P_{A_2}) \subseteq P_1 \setminus P_{A_1}$. By (9), we have $a_1 = \tilde{a}_1 \cap P_2$ with some $\tilde{a}_1 \in \gamma_{12}$. Since $\varphi_{\gamma_{12}} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$, by Lemma 2, we have $\tilde{a}_1 \cap P_{A_1} \in \gamma_1$ and $\tilde{a}_1 \cap P_1 \in \gamma_1^{P_1}$. Notice that $P_1 \subseteq P_2$. Hence $a_1 \cap P_1 = \tilde{a}_1 \cap P_2 \cap P_1 = \tilde{a}_1 \cap P_1 \in \gamma_1^{P_1}$. We conclude that $a \cap P_1 = (a_1 \cap P_1) \cup (a_2 \cap P_1) = (a_1 \cap P_1) \cup a_2 \in \gamma_1^{P_1} \bowtie 2^{P_1 \setminus P_{A_1}}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (10). By (1), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

As a consequence of Lemma 5, we immediately obtain the following generalisation of Proposition 2.

Proposition 5 (Commutativity of the partial application) *For any set of components $\mathcal{B}$ and any architectures A_1 and A_2, we have $A_2\big[A_1[\mathcal{B}]\big] = A_1\big[A_2[\mathcal{B}]\big]$.*

Proof. By (7) and Lemma 5, we have

$$A_2\big[A_1[\mathcal{B}]\big] = \big(A_2 \oplus A_1[\mathcal{B}]\big)[\emptyset] = (A_1 \oplus A_2)[\mathcal{B}] = \big(A_1 \oplus A_2[\mathcal{B}]\big)[\emptyset] = A_1\big[A_2[\mathcal{B}]\big].$$

□

Notice, furthermore, that (7) generalises Definition 7. Indeed, to a given set of components $\mathcal{B}$, we can associate the architecture $A_{\mathcal{B}} \stackrel{\Delta}{=} A_{id}[\mathcal{B}]$ (cf. Proposition 1). By (7) and Lemma 5, we obtain, for any architecture A,

$$A[A_{\mathcal{B}}] = A\big[A_{id}[\mathcal{B}]\big] = (A \oplus A_{id}[\mathcal{B}])[\emptyset] = (A \oplus A_{id})[\mathcal{B}] = A[\mathcal{B}].$$

Thus, partial application of an architecture to a set of components can be considered a special case of the application of an architecture to another architecture.

The results of the last two subsections provide two ways for using architectures at early design stages, by partially applying them to other architectures or to components that are already defined. An architecture restricts the behaviour of its arguments, which can be both components and other architectures.

[4] Again, keep in mind the difference between roman C, denoting a single coordinating component, and calligraphic $\mathcal{C}$, denoting a set of coordinating components.

3. Property preservation

Throughout this section we use several classical notions, which we recall here.

Definition 8 *(Paths, path fragments and reachable states)* Let $B = (Q, q^0, P, \rightarrow)$ be a component. A finite or infinite sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k \cdots$ is a *path fragment* in B. If in addition $q_0 = q^0$, then it is also a *path*. A state $q \in Q$ is *reachable* iff there exists a finite path in B terminating in q. A path fragment is *reachable* iff its first state is reachable.

In the sequel, we use subscripts on states (as above) when we are discussing a path fragment, i.e. $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k \cdots$, where q_0 is not necessarily the initial state q^0. We use superscripts on the states when we are discussing a path (i.e. starting in the initial state q^0), so that a path is written as $q^0 \xrightarrow{a_1} q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q^k \cdots$.

In the rest of this section, unless explicitly stated otherwise, we consider a set of architectures $A_1, \ldots, A_m$ and a set of operand components $\mathcal{B}$. For a set of indices $I \subseteq [1, m]$, we will denote by $\bigoplus_{i \in I} A_i$ the composition of all architectures with indices in I. This is well-defined, since $\oplus$ is associative and commutative. In particular, $\bigoplus_{i=1}^{m} A_i \triangleq \bigoplus_{i \in [1,m]} A_i = A_1 \oplus \cdots \oplus A_m$.

We consider that any property can be decomposed as the conjunction of safety and liveness properties, and address both kinds separately in the following two subsections. We show that if two architectures A_1 and A_2 enforce respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces $\Phi_1 \wedge \Phi_2$, that is both properties are preserved by architecture composition. Since the application of an architecture restricts the behaviour of its arguments, liveness properties cannot be preserved in general, because liveness depends on the existence of live extensions for every finite execution. Thus, we have to make a special provision for liveness, by introducing the notion of *non-interference*.

3.1. Safety properties

Definition 9 Let $B = (Q, q^0, P, \rightarrow)$ be a component. A *safety property* (in the rest of this subsection, simply *property*) of B is a state predicate $\Phi : Q \rightarrow \mathbb{B}$. We write $q \models \Phi$ iff $\Phi(q) = \text{true}$. A property Φ is *initial* if $q^0 \models \Phi$; it is *reachable* iff there exists a possibly empty path $q^0 \xrightarrow{a_1} q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q^n$, such that $q^n \models \Phi$.

The main idea of our approach is that an architecture enforces its characteristic property on the set of its operand components. From this point of view, the set of coordinating components is not relevant, neither are their states. Thus, to talk about properties enforced by architectures, we consider properties on the unrestricted composition of the operand components as formalized by the following definition.

Definition 10 (*Enforcing properties*) Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture; let $\mathcal{B}$ be a set of components and Φ be an initial property of their parallel composition $A_{id}(\mathcal{B})$ (see Proposition 1). We say that *A enforces Φ on $\mathcal{B}$* iff, for every state $q = (q_b, q_c)$ reachable in $A(\mathcal{B})$, with $q_b \in \prod_{B \in \mathcal{B}} Q_B$ and $q_c \in \prod_{C \in \mathcal{C}} Q_C$, we have $q_b \models \Phi$.

According to the above definition, when we say that an architecture enforces some property Φ, it is implicitly assumed that Φ is initial for the coordinated components. Below, we omit mentioning this explicitly.

Example 6 Consider again the mutual exclusion in Example 1. Component $A_{12}(B_1, B_2)$ is shown in Fig. 4 (we abbreviate `sleep`, `work`, `free` and `taken` to `s`, `w`, `f` and `t` respectively).

Clearly A_{12} enforces on $\{B_1, B_2\}$ the mutual exclusion property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$, where q_1 and q_2 are state variables of B_1 and B_2 respectively.

Theorem 1 (Preserving enforced properties) *Let $\mathcal{B}$ be a set of components; let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two architectures enforcing on $\mathcal{B}$ the properties Φ_1 and Φ_2 respectively. The composition $A_1 \oplus A_2$ enforces on $\mathcal{B}$ the property $\Phi_1 \wedge \Phi_2$.*

Proof. Again, by Lemma 1, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$. We also denote, for $i = 1, 2$, $P_i = P_{C_i} \cup \bigcup_{B \in \mathcal{B}} P_B$.

The initiality of $\Phi_1 \wedge \Phi_2$, is trivial: both Φ_1 and Φ_2 are initial, hence $q^0 \models \Phi_1 \wedge \Phi_2$.

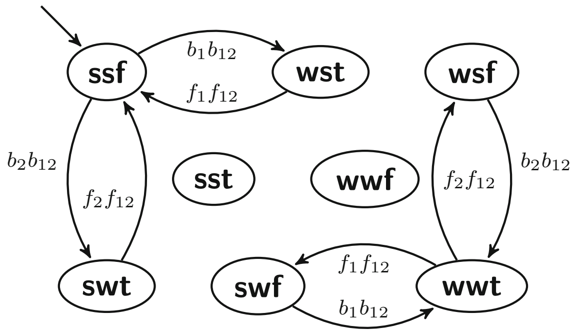
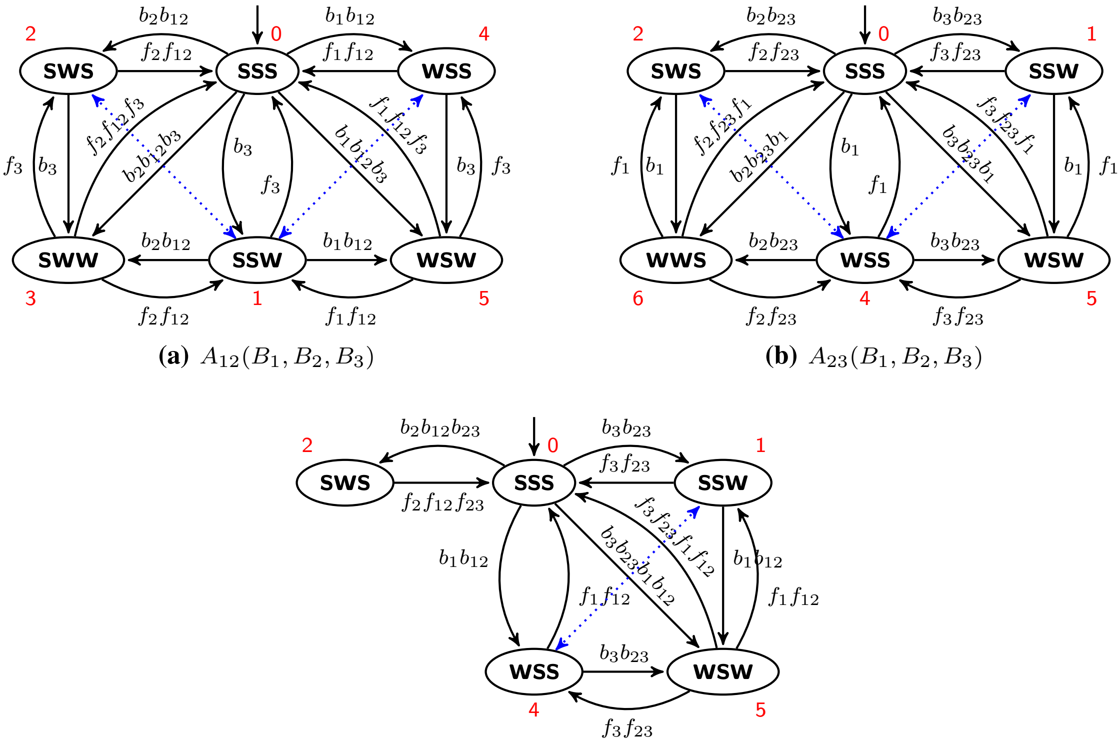


Fig. 4. Component $A_{12}(B_1, B_2)$ from Example 6



(a) $A_{12}(B_1, B_2, B_3)$

(b) $A_{23}(B_1, B_2, B_3)$

(c) $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$

Fig. 5. Projections of reachable states of Example 7 components onto $A_{id}(B_1, B_2, B_3)$ (for ease of reading, we omit the transitions indicated by *dotted blue arrows* and additionally label each state with a *red number*, whereof the main label is the binary representation with $\texttt{s} = 0$ and $\texttt{w} = 1$)

Consider a path $\tilde{q}_1^0\tilde{q}_2^0q^0 \xrightarrow{a_1} \tilde{q}_1^1\tilde{q}_2^1q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_1^k\tilde{q}_2^kq^k$ in $(A_1 \oplus A_2)(\mathcal{B})$, where $q^0, \ldots, q^k \in \prod_{B\in\mathcal{B}} Q_B$ and $\tilde{q}_i^0, \ldots, \tilde{q}_i^k \in Q_{C_i}$, for $i = 1, 2$.

By Lemma 3, $\tilde{q}_1^0q^0 \xrightarrow{a_1\cap P_1} \tilde{q}_1^1q^1 \xrightarrow{a_2\cap P_1} \cdots \xrightarrow{a_k\cap P_1} \tilde{q}_1^kq^k$ is a path in $A_1(\mathcal{B})$. (If, for some $i \in [1, k]$, $a_i \cap P_1 = \emptyset$, the corresponding transition can be omitted from the path.) Thus the state $\tilde{q}_1^kq^k$ is reachable in $A_1(\mathcal{B})$. Since A_1 enforces Φ_1 on $\mathcal{B}$, this implies that $q^k \models \Phi_1$. Symmetrically, $q^k \models \Phi_2$, which concludes the proof. □

Example 7 In the context of Example 3, consider the application of architectures A_{12} and A_{23} to the components B_1, B_2 and B_3. The former enforces the property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$ (the projections of reachable states of $A_{12}(B_1, B_2, B_3)$ onto the state-space of the atomic components are shown in Fig. 5a), whereas the latter enforces $\Phi_{23} = (q_2 \neq \mathtt{w}) \vee (q_3 \neq \mathtt{w})$ (the projections of reachable states of $A_{23}(B_1, B_2, B_3)$ onto the state-space of the atomic components are shown in Fig. 5b). By Theorem 1, the composition $A_{12} \oplus A_{23}$ enforces $\Phi_{12} \wedge \Phi_{23} = (q_2 \neq \mathtt{w}) \vee \big((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w})\big)$, i.e. mutual exclusion between, on one hand, the `work` state of B_2 and, on the other hand, the `work` states of B_1 and B_3 (see Fig. 5c). Mutual exclusion between the `work` states of B_1 and B_3 is not enforced. Furthermore, it is easy to check that $A_{12} \oplus A_{23} \oplus A_{13}$ enforces mutual exclusion between the `work` states of B_1, B_2 and B_3 as $\Phi_{12} \wedge \Phi_{13} \wedge \Phi_{23} = \big((q_1 \neq \mathtt{w}) \wedge (q_2 \neq \mathtt{w})\big) \vee \big((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w})\big) \vee \big((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w})\big)$.

In [ABB$^+$14b], we provide a similar result, showing that invariants are also preserved by the architecture composition.

3.2. Liveness properties

As mentioned in the previous subsection, the main idea of our approach is that an architecture enforces its characteristic property on the set of its operand components and that the states of the coordinating components are not relevant. Thus, to talk about properties enforced by architectures, we have to abstract away their coordinating components. To this end, below, we introduce the notion of path projection, which allows us to consider the states and paths within $A(\mathcal{B})$, limited to the corresponding states and paths of a "subsystem" that is obtained by removing some of the architectures.

For a state q of $\bigoplus_{i=1}^m A_i$, the projection of q onto $\bigoplus_{i\in I} A_i$, where $I \subseteq [1, m]$, is denoted $q \upharpoonright I$ and obtained by removing the state components of all $\mathcal{C}_i$ with $i \notin I$.

Definition 11 (*Path projection,* $\upharpoonright$) Let $\alpha = \tilde{q}^0 q^0 \xrightarrow{a_1} \tilde{q}^1 q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}^k q^k \cdots$ be a path in $\big(\bigoplus_{i=1}^m A_i\big)(\mathcal{B})$, where $q^0, q^1 \dots, q^k, \dots \in \prod_{B\in\mathcal{B}} Q_B$ and $\tilde{q}^0, \tilde{q}^1, \dots, \tilde{q}^k, \dots \in \prod_{i=1}^m \prod_{C\in\mathcal{C}_i} Q_C$. Also let $I \subseteq [1, m]$ and $P_I = (\bigcup_{B\in\mathcal{B}} P_B) \cup (\bigcup_{i\in I} \bigcup_{C\in\mathcal{C}_i} P_C)$. Then, the projection of α onto the "subsystem" $\big(\bigoplus_{i\in I} A_i\big)(\mathcal{B})$, denoted $\alpha \upharpoonright I$, is obtained as follows. Start with the path

$$\tilde{r}^0 q^0 \xrightarrow{a_1 \cap P_I} \tilde{r}^1 q^1 \xrightarrow{a_2 \cap P_I} \cdots \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k \cdots,$$

where $\tilde{r}^k = \tilde{q}^k \upharpoonright I$ for all $k \geq 0$. Then, replace all transitions $\tilde{r}^{k-1} q^{k-1} \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k$, such that $a_k \cap P_I = \emptyset$, by $\tilde{r}^k q^k$, i.e. remove transitions with empty labels (notice that, if $a_k \cap P_I = \emptyset$, we necessarily have $\tilde{r}^{k-1} q^{k-1} = \tilde{r}^k q^k$).

Proposition 6 (Path projection) *Let α be a path in $\big(\bigoplus_{i=1}^m A_i\big)(\mathcal{B})$. Then, for any $I \subseteq [1, m]$, $\alpha \upharpoonright I$ is a path in $\big(\bigoplus_{i\in I} A_i\big)(\mathcal{B})$.*

Proof. Let $\alpha = \tilde{q}^0 q^0 \xrightarrow{a_1} \tilde{q}^1 q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}^k q^k \cdots$. By Definition 11, $\alpha \upharpoonright I = \tilde{r}^0 q^0 \xrightarrow{a_1 \cap P_I} \tilde{r}^1 q^1 \xrightarrow{a_2 \cap P_I} \cdots \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k \cdots$. Consider an arbitrary transition $\tilde{r}^{k-1} q^{k-1} \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k$ along $\alpha \upharpoonright I$. By Lemma 3, $\tilde{r}^{k-1} q^{k-1} \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k$ is a transition in $\big(\bigoplus_{i\in I} A_i\big)(\mathcal{B})$. Also, $\tilde{r}^0 q^0$ is the initial state of $\big(\bigoplus_{i\in I} A_i\big)(\mathcal{B})$, by Definition 11. Since $\tilde{r}^{k-1} q^{k-1} \xrightarrow{a_k \cap P_I} \tilde{r}^k q^k$ was chosen arbitrarily, we conclude that $\alpha \upharpoonright I$ is a path in $\big(\bigoplus_{i\in I} A_i\big)(\mathcal{B})$. □

Our treatment of liveness properties is based on the idea that each coordinator C must be "invoked sufficiently often", so that the liveness properties inherent in C are imposed on the system as a whole. So, what does sufficiently often mean? A reasonable initial idea is to require that each coordinator is executed infinitely often (along an infinite path). But that turns out to be too strong. For example, a mutual exclusion coordinator should not be invoked infinitely often if no process that it coordinates requests the critical resource. So, we add "idle states", so that it is permitted for a coordinator to remain forever in an idle state. A coordinator not in an idle state must eventually be executed.

Hence, we consider an infinite path to be live with respect to a coordinator C iff either C is executed infinitely often, or is in an idle state continuously from some point onwards. An equivalent formulation is that an infinite path is live with respect to a coordinator C iff either C is executed infinitely often, or is in an idle state infinitely often. A live path is one that (1) is live with respect to all coordinators, and (2) executes some component $B \in \mathcal{B}$ infinitely often. This latter condition prevents "divergence", i.e. an infinite path fragment where none of the operand components $B \in \mathcal{B}$ is ever executed. An architecture is live with respect to a set of components iff every finite path can be extended to an infinite live one.

Definition 12 (*Architecture with liveness conditions*) An *architecture with liveness conditions* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a set of *coordinating components with liveness condition*, P_A is a set of ports such that $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$ and $\gamma \subseteq 2^{P_A}$ is an interaction model. A coordinating component with liveness condition is $C = (Q_C, q_C^0, Q_C^{idle}, P_C, \rightarrow)$, where $(Q_C, q_C^0, P_C, \rightarrow)$ is a component (Definition 1) and $Q_C^{idle} \subseteq Q_C$.

Thus, we augment each coordinator with a *liveness condition*: a subset Q_C^{idle} of its states Q_C, which are considered "idle", and in which it can remain forever without violating liveness.

We use the following definitions in discussing liveness. A transition $q \xrightarrow{a} q'$ *executes* interaction a. An infinite path fragment α *executes* a *infinitely often* iff α contains an infinite number of transitions that execute a. A transition $q \xrightarrow{a} q'$ *executes a coordinator* C iff $a \cap P_C \neq \emptyset$, where P_C is the set of ports of C. An infinite path fragment α *executes* C *infinitely often* iff α contains an infinite number of transitions that execute C. An infinite path fragment $\tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_k q_k \cdots$ *visits an idle state* of coordinator C infinitely often iff, for infinitely many $k \geq 0$, $\tilde{q}_k \upharpoonright C$ (the state component of C in $\tilde{q}_k$) is an idle state of C, i.e. $\tilde{q}_k \upharpoonright C \in Q_C^{idle}$. A state q *enables an interaction* a iff $q \xrightarrow{a}$. An infinite path fragment $\tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_k q_k \cdots$ *enables an interaction* a *continuously* iff, for all $k \geq 0$, $\tilde{q}_k q_k$ enables a. An infinite path fragment α *enables interaction* a *continuously from some point onwards* iff some infinite suffix of α enables a continuously. A state $\tilde{q}q$ *enables coordinator* C iff $\tilde{q}q$ enables every interaction a that C is ready to execute, i.e. for every a such that $\tilde{q} \upharpoonright C \xrightarrow{a \cap P_C}$, we also have $\tilde{q}q \xrightarrow{a}$. An infinite path fragment α *enables* C *continuously* iff every state of α enables C. An infinite path fragment α *enables* C *continuously from some point onwards* iff some infinite suffix of α enables C continuously.

Definition 13 (*Live path*) Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture with liveness conditions and $\mathcal{B}$ a set of components. A path α in $A(\mathcal{B})$ is *live* iff α is infinite, and, for every $C \in \mathcal{C}$, whenever C is in a non-idle state, then C must be subsequently executed. Formally, if $\alpha = \tilde{q}^0 q^0 \xrightarrow{a_1} \tilde{q}^1 q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}^k q^k \cdots$, then, for every $C \in \mathcal{C}$:

$$\forall\, k \geq 0,\ \left(\tilde{q}^k \upharpoonright C \notin Q_C^{idle} \Rightarrow \exists\, j > k : a_j \cap P_C \neq \emptyset\right),$$

where $C = (Q_C, q_C^0, Q_C^{idle}, P_C, \rightarrow)$, and $\tilde{q}^k \upharpoonright C$ denotes the local state of C in $\tilde{q}^k$.

Equivalent formulations of Definition 13 are:

1. For every $C \in \mathcal{C}$,

   $$\forall\, k \geq 0, \exists\, j > k : \left(a_j \cap P_C \neq \emptyset\ \vee\ \tilde{q}^j \upharpoonright C \in Q_C^{idle}\right).$$

   That is, α executes C infinitely often or α visits an idle state of C infinitely often;
2. For every $C \in \mathcal{C}$,

   $$\left(\forall\, k \geq 0, \exists\, j > k : a_j \cap P_C \neq \emptyset\right)\ \vee\ \left(\exists\, k \geq 0, q_C \in Q_C^{idle} : \forall\, j > k,\ \tilde{q}^j \upharpoonright C = q_C\right).$$

   That is, either α executes C infinitely often or, after some state along α, C remains forever in some idle state.

The intuition behind this definition is that each liveness condition guarantees that its coordinator executes "sufficiently often", i.e. infinitely often unless it remains forever in some idle state. When architectures are composed, we take the union of all the coordinators. Since each coordinator carries its liveness condition with it, we obtain that each coordinator is also executed sufficiently often in the composed architecture. We also obtain that architecture composition is as before, i.e. we use Definition 6, with the understanding that we compose two architectures with liveness conditions. For the rest of this section, we use "architecture" to mean "architecture with liveness conditions".

When we apply an architecture with liveness conditions to a set of components, thereby obtaining a system, we need the notion of *machine closure* [AL93]: every finite path can be extended to a live one.

Definition 14 (*Live w.r.t. a set of components*) Let A be an architecture with liveness conditions and $\mathcal{B}$ be a set of components. A is *live* w.r.t. $\mathcal{B}$ iff every finite path in $A(\mathcal{B})$ can be extended to a live path.

Even if $A_1, \ldots, A_m$ are each live w.r.t. $\mathcal{B}$, it is still possible for $(A_1 \oplus \cdots \oplus A_m)(\mathcal{B})$ to be not live w.r.t. $\mathcal{B}$, due to "interference" between the coordinators of the A_i. For example, consider two architectures A_1 and A_2, as follows. A_1 enforces safe and fair mutual exclusion between two components (say B_1 and B_2), so that every component that requests the critical section eventually receives it. A_2 enforces a priority queue over n components $B_1, \ldots, B_n$, including B_1 and B_2, and dynamically determines the queue ordering at run time. If B_1 and B_2 repeatedly contend for the critical section, then it is possible to reach a state where (1) both B_1 and B_2 are in a wait state, waiting for the critical section, (2) A_1, to ensure fairness of critical section access, has decided to grant access to B_1, and (3) A_2, for other reasons, has decided to order B_2 higher than B_1 in the queue. Then, the coordinator(s) of A_1 cannot execute an interaction, since this would require the involvement of B_1, which is prohibited by A_2.

This example shows that liveness, unlike safety, is affected by the interaction of coordinators. We must therefore impose additional conditions sufficient to guarantee that $(A_1 \oplus \cdots \oplus A_m)(\mathcal{B})$ is live w.r.t. $\mathcal{B}$. These conditions are (1) *deadlock-freedom*, discussed in the next section, and (2) *non-interference*, which we introduce and define in Sect. 3.2.2 below. To avoid state-explosion w.r.t. the number of architectures, we define non-interference as a condition on the interaction of all *pairs of architectures*, in the context of a given set of components. This also makes non-interference compositional.

3.2.1. Deadlock-freedom

A system that deadlocks cannot be live, in general, since it has a finite execution which cannot be extended to a live one. A system is free of *global deadlock* iff, in every reachable state, there is at least one enabled interaction. We define, in [ABB+13], a characterization of deadlock in BIP called a *supercycle*: roughly, a supercycle SC is a subset of the components such that every interaction a that some $B_i \in SC$ enables (i.e. $q_i \xrightarrow{a \cap P_i}$, where q_i is the current state and P_i is the set of ports of the component B_i) is blocked by some other component B_j in SC, i.e. B_j is a participant in a and $q_j \not\xrightarrow{a \cap P_j}$. These blocking relations are clearly determined by the current global state.

Let $s \xrightarrow{a} t$ be an arbitrary reachable transition such that there is no supercycle in state s, and there is a supercycle SC in state t. This transition "created" SC in some sense, and we show that some component B_i that participates in a must be in SC. This enables us to formulate a *local deadlock-freedom condition* (denoted $\mathcal{LDFC}$ in [ABB+13]) that implies the impossibility of creating a supercycle, i.e. $\mathcal{LDFC}$ is sufficient, but not necessary, for deadlock-freedom. Furthermore, $\mathcal{LDFC}$ can often be evaluated in a "small subsystem", which contains all the components that participate in a, and maybe some others. Actually, we start with the subsystem consisting of just the components that participate in a, and, if the check fails, we add more components (to obtain a better over-approximation of the reachable states) and try again. For finite-state systems, the check can be automated, and the running time of the check is linear in the number of reachable states and transitions of the smallest subsystem in which the check succeeds. For example, for n dining philosophers in a cycle, the check runs in time linear with the number of philosophers n. See [ABB+13] for details and experimental results.

3.2.2. Non-interference condition for ensuring liveness

We now give a criterion for liveness that can be evaluated without state-explosion w.r.t. the number of architectures. Avoidance of state-explosion is achieved by analyzing the interaction between architectures two at a time, rather than all at once.

Definition 15 (*Non-interference of architectures*) Let architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$ be live with respect to a set of components $\mathcal{B}$. Then A_1 is *non-interfering* with respect to A_2 and components $\mathcal{B}$ iff, for every reachable infinite path fragment α in $(A_1 \oplus A_2)(\mathcal{B})$, the following hold: (1) for every $C_2 \in \mathcal{C}_2$, either α executes

C_2 infinitely often, or α visits an idle state of C_2 infinitely often, or α enables C_2 continuously from some point onwards, and (2) some $B \in \mathcal{B}$ is executed infinitely often along α. Formally, for all infinite $\alpha = \tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_k q_k \cdots$, we have, for every $C_2 \in \mathcal{C}_2$, the following:

$$\big(\forall\, k \geq 0, \exists\, j > k : a_j \cap P_{C_2} \neq \emptyset\big) \;\vee\; \big(\forall\, k \geq 0, \exists\, j > k : \tilde{q}_j \upharpoonright C_2 \in Q^{idle}_{C_2}\big)$$
$$\vee\; \Big(\exists\, k \geq 0 : \forall\, j > k, \forall\, a, \big(\tilde{q}_j \upharpoonright C_2 \xrightarrow{a \cap P_{C_2}} \Rightarrow \tilde{q}_j q_j \xrightarrow{a}\big)\Big),$$

and, for some $B \in \mathcal{B}$, the following:

$$\big(\forall\, k \geq 0, \exists\, j > k : a_j \cap P_B \neq \emptyset\big).$$

A set of architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i \in [1, m]$ is *pairwise-noninterfering* w.r.t. components $\mathcal{B}$, iff for all $j, k \in [1, m], j \neq k$: A_j is non-interfering w.r.t. A_k and components $\mathcal{B}$.

Definition 16 (*Fair paths, fair path fragments*) Let $B = (Q, q^0, P, \rightarrow)$ be a component and let α be an infinite path fragment of B. Then, α is a *fair path fragment* iff, for every interaction a, if α enables a continuously from some point onwards, then α executes a infinitely often. If α is in addition a path, we say it is a *fair path*.

That is, we use here *weak interaction fairness* [FF96].

Theorem 2 (Live architectures using pairwise non-interference) *Let architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i \in [1, m]$ be live w.r.t. a set of components $\mathcal{B}$. Assume that:*

(a) *for all $j, k \in [1, m], j \neq k$, A_j is non-interfering w.r.t. A_k and components $\mathcal{B}$, according to Definition 15, and*

(b) *$\big(\bigoplus_{i=1}^{m} A_i\big)(\mathcal{B})$ is free of global deadlock.*

Then $\big(\bigoplus_{i=1}^{m} A_i\big)$ is live w.r.t. $\mathcal{B}$.

Proof. Let α_{fin} be an arbitrary finite path in $\big(\bigoplus_{i=1}^{m} A_i\big)(\mathcal{B})$. By assumption (b), there exists at least one extension of α_{fin} to an infinite path, which we call α. Furthermore, choose α to be fair. This can always be done, since weak interaction fairness is *feasible* [FF96], i.e., in a system free of global deadlock, any finite path can be extended to an infinite path that satisfies weak interaction fairness.

Since there are a finite number of coordinators and components, α must either execute some coordinator $C_k \in \mathcal{C}_k$ (for some $k \in [1, m]$) infinitely often, or it must execute some component $B \in \mathcal{B}$ infinitely often. Assume that α does not execute some component $B \in \mathcal{B}$ infinitely often. Then α executes infinitely often some coordinator $C_k \in \mathcal{C}_k$ for some $k \in [1, m]$. Consider arbitrary $j \in [1, m]$, $k \neq j$, and let $\alpha_{jk} = \alpha \upharpoonright \{j, k\}$, i.e. α_{jk} is the projection of α onto $(A_j \oplus A_k)(\mathcal{B})$, as given by Definition 11. By Proposition 6, α_{jk} is a path in $(A_j \oplus A_k)(\mathcal{B})$. Since α executes C_k infinitely often, we have that α_{jk} is infinite, by Definition 11. Hence α_{jk} is an infinite path in $(A_j \oplus A_k)(\mathcal{B})$, and so by (a), we can apply Definition 15 to α_{jk}. Hence α_{jk} executes some component $B \in \mathcal{B}$ infinitely often. Hence, by Definition 11, α executes B infinitely often. We conclude that the initial assumption is false, and that α does indeed execute some component $B \in \mathcal{B}$ infinitely often.

Now consider for arbitrary $\ell \in [1, m]$, an arbitrary $C_\ell \in \mathcal{C}_\ell$. Also choose an arbitrary $j \in [1, m]$, $\ell \neq j$. Let $\alpha_{j\ell} = \alpha \upharpoonright \{j, \ell\}$, i.e. $\alpha_{j\ell}$ is the projection of α onto $(A_j \oplus A_\ell)(\mathcal{B})$, as given by Definition 11. By Proposition 6, $\alpha_{j\ell}$ is a path in $(A_j \oplus A_\ell)(\mathcal{B})$. Since α executes B infinitely often, it follows by Definition 11 that $\alpha_{j\ell}$ executes B infinitely often, and so $\alpha_{j\ell}$ is infinite. Hence $\alpha_{j\ell}$ is an infinite path in $(A_j \oplus A_\ell)(\mathcal{B})$, and so by (a), we can apply Definition 15 to $\alpha_{j\ell}$. Hence, we conclude that, along $\alpha_{j\ell}$, C_ℓ is either executed infinitely often, or visits an idle state infinitely often, or is enabled continuously from some point onwards. If C_ℓ is executed infinitely often or visits an idle state infinitely often along $\alpha_{j\ell}$, then the same holds along α. Otherwise, C_ℓ is enabled continuously from some point onwards in $\alpha_{j\ell}$. Since j was chosen arbitrarily, we conclude that, in this case, C_ℓ is enabled continuously from some point onwards in all $\alpha_{j\ell}$, as j ranges over $[1, m] - \{\ell\}$. Hence by Definition 2 (semantics of interaction models) C_ℓ is enabled continuously from some point onwards in α. Hence C_ℓ is executed infinitely often since α is fair. We have thus established that, along α, C_ℓ is executed infinitely often or visits an idle state infinitely often. Therefore, α is live w.r.t. C_ℓ. Since C_ℓ was chosen arbitrarily, we have that α is live w.r.t. every coordinator. We also showed above that α executes some component $B \in \mathcal{B}$ infinitely often. By Definition 13, we conclude that α is a live path.

Now α_{fin} was chosen arbitrarily, and so we conclude that every finite path can be extended to a live path. So by Definition 14, $\bigoplus_{i=1}^{m} A_i$ is live w.r.t. $\mathcal{B}$. □

The above proof also shows that weak fairness can always be used as a scheduling strategy to ensure liveness.

Example 8 (Non-interference in mutual exclusion) Consider the system $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$, as in Example 3. Let each coordinator have a single idle state, namely the `free` state. Consider the applications of each pair of architectures, i.e. $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$, $(A_{23} \oplus A_{13})(B_1, B_2, B_3)$ and $(A_{12} \oplus A_{13})(B_1, B_2, B_3)$. For $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$, we observe that along any infinite path, either C_{12} executes infinitely often, or remains forever in its idle state after some point, or is continuously enabled after some point. Hence A_{23} is non-interfering w.r.t. A_{12} and $\{B_1, B_2, B_3\}$. Likewise for the five other ordered pairs of coordinators. We verify that $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$ is free from global deadlock using the method of [ABB^{+}13]. Hence by Theorem 2, we conclude that $(A_{12} \oplus A_{23} \oplus A_{13})$ is live w.r.t. $\{B_1, B_2, B_3\}$. Hence, it is possible to schedule the system (e.g. by using weak interaction fairness) so that each coordinator A_{12}, A_{23} and A_{13} is executed infinitely often or remains forever in its free state after some point. This means, in particular, that no process remains in its critical section forever, which is a usual liveness property of mutual exclusion systems.

Example 9 (Non-interference in hierarchical fair mutual exclusion) Consider now Example 5, fair mutual exclusion. Coordinator C_{pr} enforces an event-ordering property: if some request and grant events occur, then they must occur in a certain order. This is a pure safety property, and so we designate all of the states of C_{pr} as idle. Coordinator C_{acc}, on the other hand, enforces a conjunction of safety and liveness properties. In Example 5 we presented only the safety property, namely mutual exclusion between the left and right subtrees. In addition, there is a liveness property: if a request is received from a subtree, then this request is eventually granted. Hence we designate only the initial state of C_{acc} as idle. Thus, once a request (a) is received, it must eventually be granted (b). As in the previous example, we verify that non-interference holds. Hence by Theorem 2, we conclude that any system (binary tree) formed by applying the fair mutual exclusion architecture is live w.r.t. the basic components B_i, which are the leaves of the tree.

3.2.3. Algorithm to check non-interference in finite-state systems

We have implemented an algorithm to check (for finite-state systems) that A_j is non-interfering with respect to A_k and behaviors $\mathcal{B}$. We generate the state-transition diagram of $(A_j \oplus A_k)(\mathcal{B})$, which is of course a directed graph. We then remove all transitions of C_k and all states whose C_k-component is an idle state of C_k. We then check for the existence of a non-trivial strongly connected component of the resulting directed graph (note overloading of the word "component" here) containing a state in which C_k is not enabled. We consider a strongly connected component to be nontrivial if it is either a single state with a self-loop, or it contains at least two states. The existence of such a nontrivial strongly connected component certifies the existence of a reachable infinite path fragment along which C_k does not execute, is not in an idle state, and is not continuously enabled. Hence non-interference is violated. We next check for the existence of a cycle in which no operand component $B \in \mathcal{B}$ is executed (a single state with a self loop is considered to be a cycle). This again violates non-interference (we use $P_{\mathcal{B}} \triangleq \bigcup_{B \in \mathcal{B}} P_B$ in the code for this check). If both checks pass, we return true, otherwise we return false. Figure 6 gives pseudocode for our algorithm.

The two propositions below state the correctness of the checkNonIntrf$(\cdot)$ algorithm, when applied to two architectures A_1, A_2 and a set of components $\mathcal{B}$, and give its complexity.

Proposition 7 (Correctness) *The algorithm* checkNonIntrf$(A_1, A_2, \mathcal{B})$ *returns true iff* A_1 *is non-interfering with respect to* A_2 *and* $\mathcal{B}$*, according to Definition 15.*

Proof. Call a maximal strongly connected component κ of M'' *non-trivial* iff either κ contains more than one state, or κ consists of a single state with a self-loop. Proof is by double implication.

```
checkNonIntrf($A_1, A_2, \mathcal{B}$)
// check that $A_1$ is non-interfering with respect to $A_2$ and $\mathcal{B}$
1.  compute the state transition diagram $M$ of $(A_1 \oplus A_2)(\mathcal{B})$
2.  forall $C_2 \in \mathcal{C}_2$
3.        $M' := M - \{q \xrightarrow{a} q' \mid a \cap P_{C_2} \neq \emptyset\}$
4.              // that is, $M'$ is obtained by removing from $M$ all transitions that execute $C_2$
5.        $M'' := M' - \{q \mid q \upharpoonright C_2 \in Q^{idle}_{C_2}\}$
6.              // that is, $M''$ is obtained by removing from $M'$ all states that project onto
7.              // an idle state of $C_2$
8.        compute the set of maximal strongly connected components of $M''$
9.        if there exists a maximal strongly connected component $\kappa$ of $M''$ such that
10.             ($\kappa$ contains more than one state, or $\kappa$ consists of a single state with a self-loop), and
11.              $\kappa$ contains a state in which $C_2$ is not enabled
12.       then return(false)                                    // non-interference violated for $C_2$
13.       fi
14. endfor
15. $M''' := M - \{q \xrightarrow{a} q' \mid a \cap P_{\mathcal{B}} \neq \emptyset\}$                     //$P_{\mathcal{B}} \triangleq \bigcup_{B \in \mathcal{B}} P_B$
16.       // that is, $M'''$ is obtained by removing from $M$ all transitions that execute a component $B \in \mathcal{B}$
17. if $M'''$ contains a cycle return(false)          //non-interference violated for the components $B \in \mathcal{B}$
18. return(true)                                  // return true if $A_1$ does not interfere with $A_2$
```

Fig. 6. Pseudo-code for checking non-interference according to Definition 15

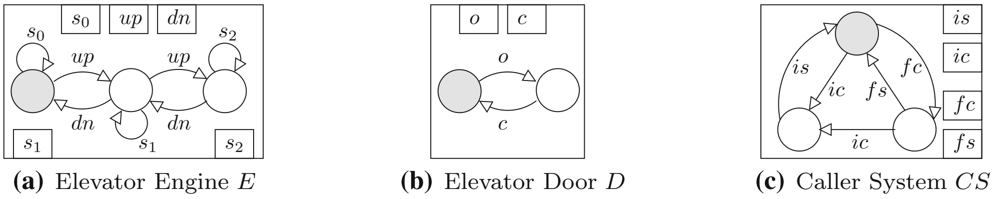
Suppose first that checkNonIntrf(A_1, A_2, $\mathcal{B}$) returns true. Then, for every $C_2 \in \mathcal{C}_2$, the following argument applies. If M'' contains a non-trivial maximal strongly connected component κ, then C_2 is enabled in every state of κ, since otherwise checkNonIntrf(A_1, A_2, $\mathcal{B}$) would return false. Hence, every cycle in M must either contain a transition involving C_2, or a global state that projects onto an idle state of C_2, or must consist entirely of global states that enable C_2. Otherwise this cycle would remain in M'' as part of a non-trivial maximal strongly connected component that does not enable C_2 in every state. Hence, there is no reachable infinite path fragment in M along which C_2 never executes, is never in an idle state, and is not continuously enabled. Hence, along every infinite path in M, either C_2 executes infinitely often, or it is in an idle state infinitely often, or it is enabled continuously from some point on. Also, for every $B \in \mathcal{B}$, the following argument applies. There is no cycle in M'''. Hence there is no cycle in M that never executes a component $B \in \mathcal{B}$. Hence there is no reachable infinite path fragment in M along which no component $B \in \mathcal{B}$ ever executes. Hence A_1 is non-interfering with respect to A_2.

Now suppose that A_1 is non-interfering with respect to A_2. Then, for every $C_2 \in \mathcal{C}_2$, the following argument must apply. Along every infinite path in M, either C_2 executes infinitely often, or it is in an idle state infinitely often, or it is enabled continuously from some point onwards. Let σ be an arbitrary cycle in M, where we consider a state with a self-loop to be a cycle. Hence, σ either contains a transition by C_2, or it contains some state that projects onto an idle state of C_2, or it consists entirely of states that enable C_2. Hence, by construction of checkNonIntrf(A_1, A_2, $\mathcal{B}$), either σ cannot remain in M'' as a cycle (some states/transitions of σ are deleted), or it remains in M'' as a cycle because it consists entirely of states that enable C_2. Hence, the only non-trivial maximal strongly connected components in M'' are those in which C_2 is enabled in every state. Hence checkNonIntrf(A_1, A_2, $\mathcal{B}$) does not return false in line 12. Furthermore, along every infinite path in M, some component $B \in \mathcal{B}$ executes infinitely often. Hence there is no cycle in M which never executes some $B \in \mathcal{B}$. Hence M''' contains no cycles. Hence checkNonIntrf(A_1, A_2, $\mathcal{B}$) does not return false in line 17. Hence checkNonIntrf(A_1, A_2, $\mathcal{B}$) returns true. □

Let M be the state-transition graph of $(A_1 \oplus A_2)(\mathcal{B})$, and let $\mid M \mid$ denote the number of nodes (states) plus the number of edges (transitions) in M. Let $\mid (A_1 \oplus A_2)(\mathcal{B}) \mid$ denote the number of interactions (syntactically) in $\mid (A_1 \oplus A_2)(\mathcal{B}) \mid$.

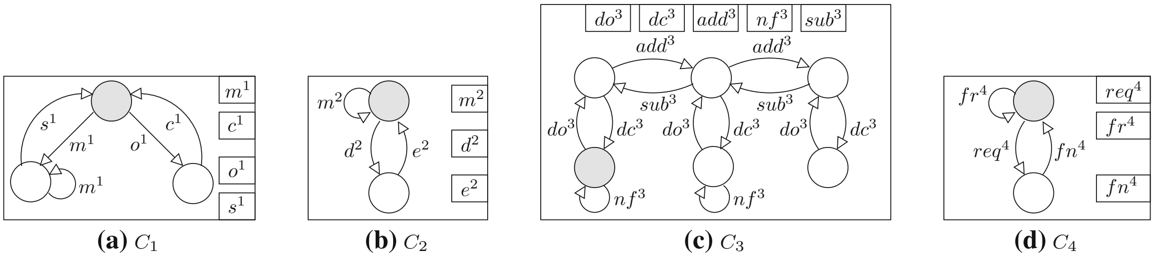
Proposition 8 (Complexity) *The time complexity of algorithm* checkNonIntrf(A_1, A_2, $\mathcal{B}$) *is in* $O(\mid \mathcal{C}_2 \mid * \mid M \mid * \mid (A_1 \oplus A_2)(\mathcal{B}) \mid)$.

Proof. The outer for-loop is repeated $\mid \mathcal{C}_2 \mid$ times. Line 3 takes time linear in $\mid M \mid * \mid (A_1 \oplus A_2)(\mathcal{B}) \mid$, since execution of every interaction must be checked in every transition. Line 5 takes time linear in $\mid M \mid$, since it can be implemented using a depth-first (or breadth-first) search of M (we assume that checking that a state is idle can be



(a) Elevator Engine E **(b)** Elevator Door D **(c)** Caller System CS

Fig. 7. Elevator atomic components (the initial states of the components are *shaded*)



(a) C_1 **(b)** C_2 **(c)** C_3 **(d)** C_4

Fig. 8. Coordinating components for the elevator example (the initial states of the components are *shaded*)

done in constant time, e.g. by encoding the idle attribute into the state's name). Lines 8 takes time linear in $| M |$, using e.g. Tarjan's algorithm [Tar72]. Lines 9–12 take time linear in $| M | * | (A_1 \oplus A_2)(\mathcal{B}) |$, since enablement of every interaction must be checked in every state. Line 15 takes time linear in $| M | * | (A_1 \oplus A_2)(\mathcal{B}) |$, since execution of every interaction must be checked in every transition. Line 17 takes time linear in $| M |$, using e.g. depth-first graph search. Hence the overall time complexity of checkNonIntrf$(A_1, A_2, \mathcal{B})$ is in $O(| \mathcal{C}_2 | * | M | * | (A_1 \oplus A_2)(\mathcal{B}) |)$. □

4. Case study: control of an elevator cabin

We illustrate our results with the case study adapted from the literature [DG08, PR01], which models an elevator in a building with three floors. Control of the elevator cabin is modelled as a set of coordinated atomic components shown in Fig. 7. Each floor of the building has a separate caller system, which allows floor selection inside the elevator and calling from the floor. Ports ic and fc respectively represent calls made within the elevator and calls from a floor. Ports is and fs represent cabin stops in response to these calls. Furthermore, in port names, m, c, o, do, dc, s, dn, up and nf stand respectively for "move", "close", "open", "door open", "door close", "stop", "move down", "move up" and "not full". For the coordinating components of the architectures in the case study, we will use super-indices to show explicitly which port belongs to which coordinating component, as in s^1 for the port "stop" of coordinator C_1 (see Fig. 8a). Caller system components and their ports are indexed by floor numbers. We denote $\mathcal{B} = \{E, D, CS_0, CS_1, CS_2\}$ the set of atomic components. To enforce required properties, we successively apply and compose architectures.

In order to provide the basic functionality of the elevator we apply to $\mathcal{B}$ the architecture $A_1 = \big(\{C_1\}, P_1, \gamma_1\big)$. Component C_1 is shown in Fig. 8a. P_1 contains all ports of C_1 and all ports of $\mathcal{B}$. γ_1 comprises the empty interaction $\emptyset$ and the following interactions (for $i \in [0, 2]$):

- Door control: $o\, o^1$, $c\, c^1$,
- Floor selection control: fc_i, ic_i,
- Moving control: $s_i\, s^1\, fs_i$, $s_i\, s^1\, is_i$, $up\, m^1$, $dn\, m^1$.

The system $A_1(\mathcal{B})$ provides the basic elevator functionality, i.e. moving up and down, stopping only at the requested floors and door control. Architecture A_1 enforces the safety property: *the elevator does not move with open doors*.

Nonetheless, $A_1(\mathcal{B})$ allows the elevator to stop at a floor, and then to leave without having opened the door. To prevent this, we apply architecture $A_2 = (\{C_2\}, P_2, \gamma_2)$ where C_2 is shown in Fig. 8b, $P_2 = \{e^2, d^2, m^2, c^1, m^1, s_0, s_1, s_2\}$, and $\gamma_2 = \{\emptyset,\ c^1 e^2,\ m^1 m^2\} \cup \{s_i d^2 \mid i \in [0, 2]\}$. This grants priority to the door controller after a "stop" action. By Proposition 2, $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$. $(A_2 \oplus A_1)(\mathcal{B})$ provides the same functionality as $A_1(\mathcal{B})$ and also this additional property.

The property *"if the elevator is full, it must stop only at floors selected from the cabin and ignore outside calls"* [DG08, PR01], is enforced by applying architecture $A_3 = (\{C_3\}, P_3, \gamma_3)$ with C_3 shown in Fig. 8c, $P_3 = \{add^3, sub^3, nf^3, do^3, dc^3, o, c\} \cup \{s_i, fs_i \mid i \in [0, 2]\}$ and $\gamma_3 = \{\emptyset,\ add^3,\ sub^3,\ do^3 o,\ dc^3 c\} \cup \{s_i\, fs_i\, nf^3 \mid i \in [0, 2]\}$. An elevator is full in our example if it has two passengers on board, i.e. C_3 has reached the bottom right-most state (see Fig. 8) by twice firing the port add^3 without firing the port sub^3 in the meantime. By Proposition 2, $A_3\big((A_1 \oplus A_2)(\mathcal{B})\big) = (A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$. By Theorem 1, $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ satisfies all three properties.

We specify liveness properties for $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ by choosing idle states for the coordinators. C_1 and C_2 have only their initial states idle, since a moving elevator must eventually stop, and an open door must eventually close. C_3 has all of its states idle, since C_3 enforces a pure safety property. We implemented our algorithm for checking non-interference, and used the implementation to verify that A_1, A_2, and A_3 are pairwise mutually non-interfering w.r.t. $\mathcal{B}$. Also $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ is deadlock-free. Hence by Theorem 2, $(A_1 \oplus A_2 \oplus A_3)$ is live w.r.t. $\mathcal{B}$. Furthermore, liveness can be enforced by using weak interaction fairness.

Finally, we consider the additional property: *"requests from the second floor have priority over all other requests"* [DG08, PR01]. This is enforced by the architecture $A_4 = (\{C_4\}, P_4, \gamma_4)$ with C_4 shown in Fig. 8. P_4 consists of all ports of C_4 and CS_2, and ports o and dn of E, whereas $\gamma_4 = \{\emptyset,\ fc_2\, req^4,\ ic_2\, req^4,\ o\, fr^4,\ dn\, fr^4,\ fs_2\, fn^4,\ is_2\, fn^4\}$. The system obtained by application of A_4 to $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ has a local deadlock, which was detected by using the deadlock analysis method and tool presented in [ABB+13]. This deadlock occurs when a full elevator is called from the second floor. Once the second floor is reached, A_4 enforces the constraint of not going down, while A_3 forbids stopping at this floor. Thus, the only choice is to move upward, which is impossible. Hence, the system is in a local deadlock state involving the elevator engine.

The system $(A_1 \oplus A_2 \oplus A_4)(\mathcal{B})$, obtained by applying A_4 to $(A_1 \oplus A_2)(\mathcal{B})$, is verified to be deadlock-free, using [ABB+13]. $\{A_1, A_2, A_4\}$ are pairwise-noninterfering w.r.t. $\mathcal{B}$, using our implementation. Hence, by Theorem 2, $(A_1 \oplus A_2 \oplus A_4)$ is live w.r.t. $\mathcal{B}$, and again, liveness can be enforced by using weak interaction fairness.

5. Related work

A number of paradigms for unifying component composition have been studied in [BWH+03, BGK+06, EJL+03]. These achieve unification by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behaviour. This is also true for the numerous compositional and algebraic frameworks [Arb04, Fia04, RC03, SG03, BCD00, BLM06, Hoa85, Mil89, LRL10]. Most of these frameworks are based on a single operator for concurrent composition. This entails poor expressiveness, which results in overly complex architectural designs. In contrast, BIP allows expression of general multiparty interaction and strictly respects separation of concerns. Coordination can be studied as a separate entity that admits a simple Boolean characterization that is instrumental for expressing composability.

BIP has some similarities with CSP, which can directly express multiparty interaction by using composition operators parameterized by channel names. For example, $B \mid \{a\} \mid B'$ is the system that enforces synchronisation of a-actions of components B and B'. Nonetheless, CSP is not adequate for architecture composition as the components must be modified when additional architecture constraints are applied. Consider for example the components $B_i = a_i \to STOP$ for $i = 1, 2, 3$. To model the system described in BIP by $\{a_1 a_2,\ a_2 a_3\}\{B_1, B_2, B_3\}$, two channels α and β must be defined representing respectively interactions $a_1 a_2$ and $a_2 a_3$ and the components modified as follows: $B_1 = \alpha \to STOP$, $B_2 = \alpha \to STOP \,\square\, \beta \to STOP$, $B_3 = \beta \to STOP$. That is, in addition to renaming, B_2 must be modified to show explicitly the conflict between α and β.

Existing research on architecture composability deals mainly with resource composability for particular types of architectures, e.g. [LRL10]. The feature interaction problem is how to rapidly develop and deploy new features without disrupting the functionality of existing features. It can be considered as an architecture composability problem to the extent that features can be modelled as architectural constraints. A survey on feature interaction research is provided in [CKMRM03]. Existing results focus mainly on modelling aspects and checking feature interaction by using algorithmic verification techniques with well-known complexity limitations. Our work takes a constructive approach. It has some similarities to [HA00] which presents a formal framework for detecting and avoiding feature interactions by using priorities. Nonetheless, these results do not deal with property preservation through composition. Similarly, existing work on service interaction mainly focuses on modelling and verification aspects, e.g. [DPW06, LJH06].

6. Conclusion

Our work makes two novel contributions towards correct-by-construction system design. First, it proposes a general concept of architecture. Architectures are operators restricting the behaviour of their arguments by enforcing a characteristic property. They can be composed and studied independently. Composition of architectures can be naturally expressed as the conjunction of the induced synchronisation constraints. This implies nice properties such as associativity, commutativity and idempotence. Nonetheless, it is not easy to understand it as an operation on interaction models. Using BIP to describe architectures proves to be instrumental for achieving this. In contrast to other formalisms, BIP is expressive enough and keeps a strict separation between behaviour and coordination aspects. *Application of architectures does not require any modification of the atomic components*. Furthermore, if we wish to modify the property enforced by an architecture, then only the coordinators of the architecture need be modified. The base components, and other architectures, need not be changed. This provides *locality* and *modifiability*, key properties for good software engineering [LG00, chapter 5].

The second contribution is preservation of safety properties enforced by architectures. The preservation of state predicates is guaranteed by the very nature of architecture composition. This result is different from existing results stipulating the preservation of invariants of components when composed by using parallel composition operators, e.g. an invariant of B_1 is also an invariant of $B_1 \;||\; B_2$, for some parallel composition operator $||$. Our result is about preservation of properties over the *same* state-space, which is the Cartesian product of the atomic components. That is, a property of $A_1(\mathcal{B})$ is also a property of $(A_1 \oplus A_2)(\mathcal{B})$, and so the state-space of the components $\mathcal{B}$ is unchanged.

Architecture composition also preserves liveness properties, subject to a requirement of non-interference amongst the architectures that are applied. We consider liveness properties expressed implicitly by requiring each coordinator to be executed infinitely often or to remain forever in some idle state after some point. This can be expressed in linear temporal logic as $\Box\Diamond\, exec(C) \vee \bigvee_{idle(s)} \Diamond\Box\, at(s)$, where $\Box$, $\Diamond$ are the *always* and *eventually* modalities of linear temporal logic, $exec(C)$ holds when C has just been executed, $idle(s)$ holds iff s is an idle state, and $at(s)$ holds iff the current state of C is s. It remains to determine the exact fragment (possibly all) of linear temporal logic that can be handled by our framework (both safety and liveness). This is a topic for future work.

Our work pursues similar objectives as the research on interaction of features or services, insofar as they can be modelled as architectural constraints. Nonetheless, it adopts a radically different approach. It privileges constructive techniques to avoid costly and intractable verification. It proposes a concept of composability focusing on property preservation.

Our work is part of a broader research program investigating correct-by-construction approaches. These are at the root of any mature engineering discipline. They are scalable and do not suffer limitations of correctness-by-checking. Our vision is that systems can be built incrementally by composing architectural solutions ensuring elementary properties, e.g. mutual exclusion, schedulability, fault-tolerance and timeliness. The desired global properties can be established as the conjunction of elementary properties. To put this vision into practice, we need to develop a repository of reference architectures with their characteristic properties.

We also need to generalize our approach to liveness: in some cases, we can ensure liveness by checking for the continuous *enablement* of interactions, and ensuring their execution by using weak fairness. This would enlarge the set of systems whose liveness we can enforce.

There exists a plethora of results on solving coordination problems including distributed algorithms, protocols, and scheduling algorithms, hardware architectures. Most of these results focus on principles of solutions and discard essential operational details. Their formalization as architectures will make explicit the underlying concrete coordination mechanisms based on operational semantics. Is it possible to find a taxonomy induced by a hierarchy of characteristic properties? Moreover, is it possible to determine a minimal set of basic properties and corresponding architectural solutions from which more general properties and their corresponding architectures can be obtained? Bringing answers to these questions would greatly enhance our capability to design systems that are correct-by-construction and minimal.

References

[ABB$^+$13] Attie PC, Bensalem S, Bozga M, Jaber M, Sifakis J, Zaraket FA (2013) An abstract framework for deadlock prevention in BIP. In: FMOODS/FORTE, pp 161–177

[ABB$^+$14a] Attie P, Baranov E, Bliudze S, Jaber M, Sifakis J (2014) A general framework for architecture composability. In: Giannakopoulou D, Salaün G (eds) 12th international conference on software engineering and formal methods (SEFM 2014), LNCS, vol 8702, Switzerland. Springer International Publishing, pp 128–143

[ABB$^+$14b] Attie PC, Baranov E, Bliudze S, Jaber M, Sifakis J (2014) A general framework for architecture composability. Technical Report EPFL-REPORT-196536, EPFL IC IIF RiSD. http://infoscience.epfl.ch/record/196536

[AL93] Abadi M, Leslie L (1993) Composing specifications. ACM Trans Program Lang Syst 15(1):73–132

[Arb04] Arbab F (2004) Reo: a channel-based coordination model for component composition. Math Struct Comput Sci 14(3):329–366

[BBB$^+$11] Basu A, Bensalem S, Bozga M, Combaz J, Jaber M, Nguyen T-H, Sifakis J (2011) Rigorous component-based system design using the BIP framework. Softw IEEE 28(3):41–48

[BCD00] Bernardo M, Ciancarini P, Donatiello L (2000) On the formalization of architectural types with process algebras. In: SIGSOFT FSE, pp 140–148

[BGK$^+$06] Balasubramanian K, Gokhale AS, Karsai G, Sztipanovits J, Neema S (2006) Developing applications using model-driven design environments. IEEE Comput 39(2):33–40

[BLM06] Bruni R, Lanese I, Montanari U (2006) A basic algebra of stateless connectors. Theor Comput Sci 366(1):98–120

[BS07] Bliudze S, Sifakis J (2007) The algebra of connectors—structuring interaction in BIP. In: Proceedings of the EMSOFT'07, Salzburg . ACM SigBED, pp 11–20

[BS10] Bliudze S, Sifakis J (2010) Causal semantics for the algebra of connectors. Formal Methods Syst Des 36(2):167–194

[BS11] Bliudze S, Sifakis J (2011) Synthesizing glue operators from glue constraints for the construction of component-based systems. In: Sven A, Ethan J (eds) Software composition, vol 6708. LNCSSpringer, Berlin/Heidelberg, pp 51–67

[BWH$^+$03] Balarin F, Watanabe Y, Hsieh H, Lavagno L, Passerone C, Sangiovanni-Vincentelli A (2003) Metropolis: an integrated electronic system design environment. IEEE Comput 36(4):45–52

[CKMRM03] Calder M, Kolberg M, Magill EH, Reiff-Marganiec S (2003) Feature interaction: a critical review and considered forecast. Comput Netw 41(1):115–141

[DG08] D'Souza D, Gopinathan M (2008) Conflict-tolerant features. In: CAV, LNCS, vol 5123. Springer, pp 227–239

[DPW06] Decker G, Puhlmann F, Weske M (2006) Formalizing service interactions. In: Business process management, pp 414–419

[EJL$^+$03] Eker J, Janneck JW, Lee EA, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S, Xiong Y (2003) Taming heterogeneity: the ptolemy approach. Proc IEEE 91(1):127–144

[FF96] Francez N, Forman IR (1996) Interacting processes: a multiparty approach to coordinated distributed programming. ACM Press/Addison-Wesley Publishing Co., New York

[Fia04] Fiadeiro JL (2004) Categories for Software Engineering. Springer-Verlag

[HA00] Hay JD, Atlee JM (2000) Composing features and resolving interactions. SIGSOFT Softw Eng Notes 25(6):110–119

[Hoa85] Hoare CAR (1985) Communicating sequential processes. Prentice Hall International Series in Computer Science. Prentice Hall

[LG00] Liskov B, Guttag J (2000) Program development in java. Addison Wesley

[LJH06] Li Z, Jin Y, Han J (2006) A runtime monitoring and validation framework for web service interactions. In: ASWEC, pp 70–79

[LRL10] Liu I, Reineke J, Lee EA (2010) A PRET architecture supporting concurrent programs with composable timing properties. In: Signals, systems and computers (ASILOMAR), 2010 Conference record of the forty fourth asilomar conference, pp 2111–2115

[Mil89] Milner R (1989) Communication and concurrency. Prentice Hall International Series in Computer Science, Prentice Hall

[PR01] Plath M, Ryan M (2001) Feature integration using a feature construct. Sci Comput Programm 41(1):53–84

[RC03] Ray A, Cleaveland R (2003) Architectural interaction diagrams: AIDs for system modeling. In: ICSE'03: Proceedings of the 25th international conference on software engineering, Washington, DC. IEEE Computer Society, pp 396–406

[SG03] Spitznagel B, Garlan D (2003) A compositional formalization of connector wrappers. In: ICSE. IEEE Computer Society, pp 374–384

[Sif12] Sifakis J (2012) Rigorous system design. Found Trends Electron Des Autom 6(4):293–362, 2012

[Tar72] Tarjan R (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2):146–160

[Weg96] Wegner P (1996) Coordination as constrained interaction (extended abstract). In: Coordination languages and models, LNCS, vol 1061. Springer, pp 28–33

Received 30 January 2015

Revised 17 August 2015

Accepted 21 October 2015 by Dimitra Giannakopoulou, Gwen Salaün, and Michael Butler