ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

1866 AUB American University of Beirut
الجامعة الأميركية في بيروت

A General Framework for Architecture Composability

EPFL IC IIF RiSD Technical Report
№ EPFL-REPORT-196536
http://infoscience.epfl.ch/record/196536

Paul Attie, Eduard Baranov, Simon Bliudze,
Mohamad Jaber and Joseph Sifakis

February 20, 2014

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 state 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 discuss preservation of liveness properties by architecture composition. The presented results are illustrated by a running example and a case study.

```
@TechReport{ABBJS14-Architectures-TR,
  author      = {Attie, Paul C. and
                 Baranov, Eduard and
                 Bliudze, Simon and
                 Jaber, Mohamad and
                 Sifakis, Joseph},
  title       = {A General Framework for Architecture Composability},
  institution = {EPFL IC IIF RiSD},
  month       = feb,
  year        = 2014,
  number      = {EPFL-REPORT-196536},
  note        = {Available at: \texttt{http://infoscience.epfl.ch/record/196536}}
}
```

A General Framework for Architecture Composability

Paul Attie * Eduard Baranov † Simon Bliudze † Mohamad Jaber *
Joseph Sifakis †

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 state 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 discuss preservation of liveness properties by architecture composition. The presented results are illustrated by a running example and a case study.

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.

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

*American University of Beirut, Lebanon; {pa07,mj54}@aub.edu.lb
†École Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland; firstname.lastname@epfl.ch

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 architectures 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 [10].

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 [22]. The underlying theory about components and their interaction is inspired from BIP [8]. BIP is a component framework rooted in well-defined operational semantics. It proposes an expressive and elegant notion of glue for component composition. Glue operators can be studied as sets of Boolean constraints expressing interactions between components. BIP has been fully implemented in a language and supporting tools, e.g. compilers and code generators [6].

We consider that a *component framework* consists of a set $\mathcal{B}$ of *atomic components* and a set $\Gamma = \{\gamma_k\}_{k \in K}$ of *glue* operators on these components. Atomic components are characterized by their behaviour specified as a transition system. The glue Γ includes general composition operators (behaviour transformers).

In this context, a glue operator γ is given by an *interaction model*, which 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 (Def. 2.2).

A component framework can be considered as a term algebra equipped with a congruence relation compatible with strong bisimulation on transition systems. A composite component is a well-formed expression built from atomic components.

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

An architecture A adequately restricts the behaviour of a set of components so that the resulting behaviour meets a characteristic property Φ. It is a solution to a specific coordination problem specified by Φ by using a particular interaction model specified by γ and 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 glue operator γ that must be compensated by using the additional set of components $\mathcal{C}$ for coordination. The characteristic property 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).

In this paper, we propose a general formal framework for architecture composability based on an architecture composition operator '$\oplus$' which is associative, commutative and idempotent. The

main result is that, if two architectures A_1 and A_2 enforce respectively state 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. Another family of results deals with preservation of liveness.

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

2 The Theory of Architectures

2.1 Behaviours and Architectures

Definition 2.1 (Behaviour)**.** A *behaviour* 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 $\rightarrow \subseteq Q \times 2^P \times Q$ is a *transition relation*. Each transition is labelled by an *interaction* $a \subseteq P$. 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 Q_B, P_B and $\rightarrow_B$ the constituents of a behaviour B.

Definition 2.2 (Interaction model)**.** Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be a finite set of behaviours 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 subset $\gamma \subseteq 2^P$. 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 behaviour $\gamma(\mathcal{B}) = (Q, q^0, P, \rightarrow)$, where $Q = \prod_{i=1}^n Q_i$, $q^0 = q_1^0 \ldots q_n^0$ and $\rightarrow$ is the minimal transition relation inductively defined by the following rules:

$$\frac{q_i \xrightarrow{\emptyset} q_i'}{q_1 \ldots q_i \ldots q_n \xrightarrow{\emptyset} q_1 \ldots q_i' \ldots q_n}, \tag{1}$$

$$\frac{a \in \gamma \quad q_i \xrightarrow{a \cap P_i} q_i' \text{ (if } a \cap P_i \neq \emptyset) \quad q_i = q_i' \text{ (if } a \cap P_i = \emptyset)}{q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'}. \tag{2}$$

In the sequel, when speaking of a set of behaviours $\mathcal{B} = \{B_1, \ldots, B_n\}$, we will always assume that it satisfies all assumptions of Def. 2.2.

Definition 2.3 (Architecture)**.** An *architecture* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite set of *coordinating behaviours* 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.

Definition 2.4 (Application of an architecture)**.** Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and let $\mathcal{B}$ be a set of behaviours, 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 behaviours $\mathcal{B}$ is the behaviour

$$A(\mathcal{B}) \triangleq \left(\gamma \parallel 2^{P \setminus P_A}\right)(\mathcal{C} \cup \mathcal{B}), \tag{3}$$

where, for interaction models γ' and γ'' over disjoint domains P' and P'' respectively, $\gamma' \parallel \gamma'' \triangleq \{a' \cup a'' \mid a' \in \gamma', a'' \in \gamma''\}$ is an interaction model over $P' \cup P''$.

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

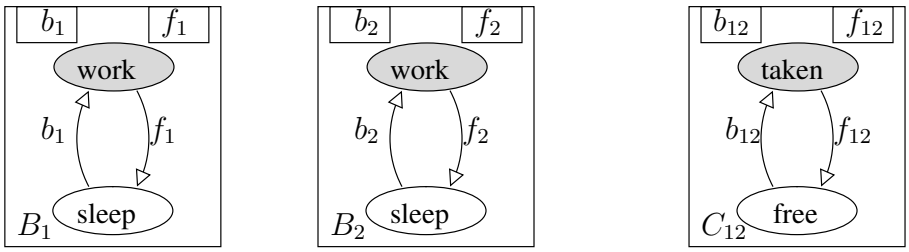


Figure 1: Behaviour (a) and coordinator (b) for Ex. 2.5.

Intuitively, an architecture A enforces coordination constraints on the behaviours in $\mathcal{B}$. The interface P_A of an architecture A contains all ports of the coordinating behaviours $\mathcal{C}$ and some additional ports, which must belong to the behaviours in $\mathcal{B}$. 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, which 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 (3)). If 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. Finally, the definition of $\gamma' \parallel \gamma''$, above, requires that an interaction from each of γ' and γ'' be involved in every interaction belonging to $\gamma' \parallel \gamma''$. To enable independent progress in (3), one must have $\emptyset \in \gamma$. (Notice that $\emptyset \in 2^{P \setminus P_A}$ holds always.)

Example 2.5 (Mutual exclusion)**.** Consider the behaviours B_1 and B_2 in Fig. 1(a). 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. 1(b), $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \{\emptyset, b_1 b_{12}, b_2 b_{12}, f_1 f_{12}, f_2 f_{12}\}$.

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 $(\texttt{free}, \texttt{work}, \texttt{work})$ and $(\texttt{taken}, \texttt{work}, \texttt{work})$ is reachable in $A_{12}(B_1, B_2)$.

Let B_3 be a third behaviour, 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 $\gamma_{12} \parallel \{\emptyset, b_3, f_3\}$. (We exclude 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 behaviour of $A_{12}(B_1, B_2, B_3)$ is the unrestricted product of the behaviours $A_{12}(B_1, B_2)$ 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.

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 [8, 24]. More precisely, behaviour coordination is realised by limiting the allowed synchronisation possibilities, thus enforcing constraints on the transitions behaviours can take. From this perspective, architecture composition can be understood as the conjunction of their respective constraints. This intuitive notion is formalised by the two definitions below.

Definition 2.6 (Characteristic predicates)**.** 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 putting

$$\varphi_\gamma \triangleq \bigvee_{a \in \gamma} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \right) .$$

For any valuation $v : P \to \mathbb{B}$, $\varphi_\gamma(v) = \mathtt{tt}$ if and only if $\{p \in P \,|\, v(p) = \mathtt{tt}\} \in \gamma$. A predicate $\varphi \in \mathbb{B}[P]$ uniquely defines an interaction model γ_φ, such that $\varphi_{\gamma_\varphi} = \varphi$.

Example 2.7 (Mutual exclusion (contd.))**.** Consider the interaction model

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

from Ex. 2.5. 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):

$$\begin{aligned} \varphi_{\gamma_{12}} &= \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \;\vee\; b_1\,\overline{b_2}\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \;\vee\; \overline{b_1}\,b_2\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \\ &\quad \vee\; \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,f_1\,\overline{f_2}\,f_{12} \;\vee\; \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,f_2\,f_{12} \\ &= (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \\ &\quad \wedge\; (b_{12} \Rightarrow b_1 \,\mathtt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\mathtt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}}) . \end{aligned} \tag{4}$$

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.

Definition 2.8 (Architecture composition)**.** Let $A_j = (\mathcal{C}_j, P_j, \gamma_j)$, for $j = 1, 2$ be two architectures. The *composition* of A_1 and A_2 is an architecture $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi)$, where $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

The following lemma states that the interaction model of the composed behaviour consists precisely of the interactions, such that both their projections on the interfaces of the composed architectures belong to the corresponding interaction models. In other words, these are precisely the interactions that satisfy the coordination constraints enforced by both composed architectures.

Lemma 2.9. *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. As observed above, $a \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$ iff $(\varphi_{\gamma_1} \wedge \varphi_{\gamma_2})(v) = \mathtt{tt}$, which is equivalent to $\varphi_{\gamma_1}(v) = \mathtt{tt}$ and $\varphi_{\gamma_2}(v) = \mathtt{tt}$. 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 $\varphi_{\gamma_1}(v) = \mathtt{tt}$ iff $\varphi_{\gamma_1}(v') = \mathtt{tt}$, i.e. $a \cap P_1 \in \gamma_1$. The same holds for $a \cap P_2 \in \gamma_2$. □

Lemma 2.10. *Consider a set of behaviours $\mathcal{B}$ 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 $i = 1, 2$, $\tilde{q}_i q \xrightarrow{a \cap (P_{A_i} \cup P)} \tilde{q}'_i q'$ is a transition in $A_i(\mathcal{B})$, where $P = \bigcup_{B \in \mathcal{B}} P_B$.*

Proof. Without loss of generality, we can assume that each of the two architectures has only one coordinating behaviour, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$.

By Def. 2.8, $a \cap (P_{A_1} \cup P_{A_2}) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lem. 2.9, $a \cap P_{A_1} \in \gamma_1$. Hence,

$$\tilde{a} \;\triangleq\; a \cap (P_{A_1} \cup P) \;=\; \big(a \cap P_{A_1}\big) \cup \big(a \cap (P \setminus P_{A_1})\big) \in \big(\gamma_1 \parallel 2^{P \setminus P_{A_1}}\big) .$$

Furthermore, since $\tilde{q}_1\tilde{q}_2 q \xrightarrow{a} \tilde{q}'_1\tilde{q}'_2 q'$, we have by (2),

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

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 (2) 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 2.11. *Architecture composition '$\oplus$' is commutative, associative and idempotent; $A_{id} = \big(\emptyset, \emptyset, \{\emptyset\}\big)$ is its neutral element, i.e. for any architecture A, holds $A \oplus A_{id} = A$. Furthermore, for any behaviour B, holds $A_{id}(B) = B$.*

Proof. Follows from the corresponding properties of set union and boolean conjunction. The properties of A_{id} follow immediately from the definitions of architecture application and composition. □

Notice that, for an arbitrary set of behaviours $\mathcal{B}$ with $P = \bigcup_{B\in\mathcal{B}} P_B$, we have, by (3), $A_{id}(\mathcal{B}) = \big(2^P\big)(\mathcal{B})$ (cf. Def. 2.2).

Example 2.12 (Mutual exclusion (contd.))**.** Building upon Ex. 2.5, let B_3 be a third behaviour, 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 the behaviour C_{12} in Fig. 1(*b*), $P_{i3} = \{b_i, b_3, b_{i3}, f_i, f_3, f_{i3}\}$ and $\gamma_{i3} = \{\emptyset, b_i b_{i3}, b_3 b_{i3}, f_i f_{i3}, f_3 f_{i3}\}$.

By considering, for $\varphi_{\gamma_{13}}$ and $\varphi_{\gamma_{23}}$, expressions similar to (4), it is easy to compute $\varphi_{\gamma_{12}} \wedge \varphi_{\gamma_{13}} \wedge \varphi_{\gamma_{23}}$ as the conjunction of the following implications:

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

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

$$\{\emptyset\,,\ b_1b_{12}b_{13}\,,\ f_1f_{12}f_{13}\,,\ b_2b_{12}b_{23}\,,\ f_2f_{12}f_{23}\,,\ b_3b_{13}b_{23}\,,\ f_3f_{13}f_{23}\}\,.$$

Again, 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, neither of the states $(\cdot,\cdot,\cdot,\texttt{work},\texttt{work},\cdot)$, $(\cdot,\cdot,\cdot,\texttt{work},\cdot,\texttt{work})$ and $(\cdot,\cdot,\cdot,\cdot,\texttt{work},\texttt{work})$ is 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 behaviours. In Sect. 3.1, we provide a general result stating that architecture composition preserves the enforced properties.

2.3 Hierarchical Composition of Architectures

Proposition 2.13. *Let $\mathcal{B}$ be a set of behaviours 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) $P_{A_1} \subseteq P_1 \triangleq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1} P_B$ and 2) $P_{A_2} \subseteq P_2 \triangleq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}_1\cup\mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$.*

Proof. Clearly, the state spaces, initial states and interfaces of both behaviours coincide. Thus we only have to prove that so do the transition relations. Without loss of generality, we assume $\mathcal{C}_1 = \{C_1\}$, $\mathcal{C}_2 = \{C_2\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$.

By Def. 2.4 a transition $q_{C_1} q_{C_2} q_1 \ldots q_n \xrightarrow{a} q'_{C_1} q'_{C_2} q'_1 \ldots q'_n$ is possible in $A_2(A_1(\mathcal{B}))$ iff

1. $q_{C_2} \xrightarrow{a \cap P_{C_2}} q'_{C_2}$ is possible in 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 $A_1(\mathcal{B})$ and
3. $a \in \gamma_2 || 2^{P_2 \setminus P_{A_2}}$.

The transition in condition 2 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,
5. for $i \in [1, n]$, $q_i \xrightarrow{a \cap P_{B_i}} q'_i$ is possible in B_i and
6. $a \cap P_1 \in \gamma_1 || 2^{P_1 \setminus P_{A_1}}$.

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

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

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

For $a \subseteq P_2$, we have $a \in \gamma_{A_1 \oplus A_2} \parallel 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$ 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 Lem. 2.9, 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 $a \in \gamma_2 \parallel 2^{P_2 \setminus P_{A_2}}$. Finally, since $P_{A_1} \subseteq P_1$, we have $a \cap P_{A_1} \in \gamma_1$ iff $a \cap P_1 \in \gamma_1 \parallel 2^{P_1 \setminus P_{A_1}}$. □

The first condition in Prop. 2.13 states that A_1 can be applied to the behaviours in $\mathcal{B}$ (cf. Def. 2.4). Similarly, the second condition states that A_2 can be applied to $A_1(\mathcal{B})$. Clearly, when condition 1) of Prop. 2.13 holds for both A_1 and A_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 $i \neq j \in \{1, 2\}$, the two are independent and their composition is commutative: $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B}) = A_1(A_2(\mathcal{B}))$. The restrictions on the architectures are lifted in Prop. 2.20, generalising this proposition.

Proposition 2.14. *Let $\mathcal{B}_1, \mathcal{B}_2$ be two sets of behaviours, 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 $P_{A_1} \subseteq \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{C}_1} P_B$ and $P_{A_2} \subseteq \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$.*

Proof. Similarly, as in Prop. 2.13 the set of states is equal in both composed behaviours, thus we only have to prove the equality of transitions relations. Without loss of generality, we assume, for $i = 1, 2$, $\mathcal{B}_i = \{B_i\}$ and $\mathcal{C}_i = \{C_i\}$.

Let $P = P_{C_1} \cup P_{C_2} \cup P_{B_1} \cup P_{B_2}$ be a union of ports from all behaviours including the coordinating behaviours of the architectures and let $P' = P_{C_1} \cup P_{B_1} \cup P_{B_2}$.

Assume, we have a transition $q_{C_1} q_{C_2} q_{B_1} q_{B_2} \xrightarrow{a} q'_{C_1} q'_{C_2} q'_{B_1} q'_{B_2}$ in the composed system $A_2(A_1(B_1, B_2))$. All behaviours can make a corresponding transition and a can be represented as $a = a_{c_2} \cup a_{\gamma_1} \cup a'$, where $a_{C_2} \subseteq P_{C_2}$, $a_{\gamma_1} \in \gamma_1$ and $a' \in 2^{P' \setminus P_{A_1}}$. As $P_{a_1} \cap P_2 = \emptyset$, all the ports of B_2 that belong to a are in a'. Let $a' = a_{B_2} \cup a''$, where $a_{B_2} = a \cap P_2$. Then interaction $a_{\gamma_1} \cup a''$ is enabled in $A_1(B_1)$, and interaction $a_{C_2} \cup a_{\gamma_1} \cup a'' \cup a_{B_2}$ is enabled in $A_2(A_1(B_1), B_2)$.

Assume interaction a is enabled in $A_2(A_1(B_1), B_2)$. It can be represented as $a = a_{C_2} \cup a_{\gamma_1} \cup a'' \cup a_{B_2}$. Then interaction $a_{\gamma_1} \cup a'' \cup a_{B_2}$ is enabled in $A_1(B_1, B_2)$ and consequently a is enabled in $A_2(A_1(B_1, B_2))$ in the corresponding state. □

Intuitively, Prop. 2.14 states that one only has to apply the architecture A_1 to those behaviours that have ports involved in its interface. Notice that, in order to compare the semantics of two sets of behaviours, one has to compose them into compound behaviours, by applying *some* architecture. Hence the need for A_2 in Prop. 2.14. As a special case, one can consider the "most liberal" identity architecture A_{id} (see Prop. 2.11). A_{id} does not impose any coordination constraints, allowing all possible interactions between the behaviours it is applied to.

Example 2.15 (Mutual exclusion (contd.))**.** It is clear that Ex. 2.12 can be generalised to an arbitrary number n of behaviours. However, such solution would require $n(n-1)/2$ architectures, and so may not scale up well. Instead, one can apply architectures hierarchically.

Let $n = 4$ and consider two architectures A_{12} and A_{34}, with the respective coordination behaviours C_{12} and C_{34}, that enforce mutual exclusion between B_1, B_2 and B_3, B_4 respectively in a similar manner to Ex. 2.12. 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 $A(A_{12}(B_1, B_2), A_{34}(B_3, B_4))$ ensures mutual exclusion between all four behaviours $(B_i)_{i=1}^4$. Furthermore, by the above propositions,

$$A(A_{12}(B_1, B_2), A_{34}(B_3, B_4)) = A(A_{12}(B_1, B_2, A_{34}(B_3, B_4))) = \\ A(A_{12}(A_{34}(B_1, B_2, B_3, B_4))) = (A \oplus A_{12} \oplus A_{34})(B_1, B_2, B_3, B_4)\,.$$

2.4 Partial Application of Architectures

Notice that the main condition, limiting the application of Prop. 2.13 and Prop. 2.14, is that the architectures must be applicable, i.e. every port of the architecture interface must belong to some behaviour. Below we lift this restriction by introducing the notion of *partial application*. We generalise Def. 2.4 for architectures $A = (\mathcal{C}, P_A, \gamma)$ applied to sets of behaviours $\mathcal{B}$, 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 behaviours. In other words, the system obtained by applying the architecture to the set of behaviours $\mathcal{B}$ is not *complete*. The result can then itself be considered as an architecture where the coordinating behaviour is the one obtained by applying to $\mathcal{B} \cup \mathcal{C}$ the projection of interactions in γ.

Definition 2.16. Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and $\mathcal{B}$ be a set of behaviours. Let $P = \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. A *partial application* of A to $\mathcal{B}$ is an architecture $A[\mathcal{B}] \triangleq (\{C'\}, P \cup P_A, \gamma \parallel 2^{P \setminus P_A})$, where $C' \triangleq (\gamma^P \parallel 2^{P \setminus P_A})(\mathcal{C} \cup \mathcal{B})$ with $\gamma^P = \{a \cap P \mid a \in \gamma\}$ and the operator '$\parallel$' as in Def. 2.4.

Notice that an architecture obtained by partial application has precisely one coordinating behaviour 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 Def. 2.4), we have $\gamma_P = \gamma$ and $A[\mathcal{B}] = \big(\{A(\mathcal{B})\}, P, \gamma \parallel 2^{P \setminus P_A}\big)$.

Lemma 2.17. *Let $\mathcal{B}$ be set of behaviours 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 $A(\mathcal{B}) = A[\mathcal{B}](\emptyset)$.*

Proposition 2.18. *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two sets of behaviours, such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$, and let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture. Then $A[\mathcal{B}_1, \mathcal{B}_2] = \big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$.*

Proof. Clearly the interfaces of both architectures coincide. Furthermore, since the two architectures are obtained by partial application, each has only one coordinating behaviour (see Def. 2.16). Thus we have to show that the coordinating behaviours 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 Def. 2.16, the interaction models of $A[\mathcal{B}_1, \mathcal{B}_2]$ and $\big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$ are, respectively $\gamma \parallel 2^{(P_1\cup P_2)\setminus P_A}$ and $\big(\gamma \parallel 2^{P_1\setminus P_A}\big) \parallel 2^{P_2\setminus P_A}$. Since $2^{P_1\setminus P_A} \parallel 2^{P_2\setminus P_A} = 2^{(P_1\cup P_2)\setminus P_A}$, we conclude that the interaction models coincide.

It is also clear that the state spaces, initial states and interfaces of both coordination behaviours coincide. Thus, we only have to show that so do the transition relations. Let us consider the coordinating behaviours of the two architectures. By Def. 2.16, we have[2]

$$\begin{aligned} A[\mathcal{B}_1, \mathcal{B}_2] &= \big(\{C_{12}\}, P_A \cup P_1 \cup P_2, \gamma \parallel 2^{(P_1\cup P_2)\setminus P_A}\big)\,, \\ \text{with } C_{12} &= \big(\gamma^{P_1\cup P_2} \parallel 2^{(P_1\cup P_2)\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1 \cup \mathcal{B}_2)\,, \\ \text{where } \gamma^{P_1\cup P_2} &= \{a \cap (P_1 \cup P_2) \mid a \in \gamma\}\,. \end{aligned} \tag{5}$$

Similarly,

$$\begin{aligned} A[\mathcal{B}_1] &= \big(\{C_1\}, P_A \cup P_1, \gamma \parallel 2^{P_1\setminus P_A}\big)\,, \\ \text{with } C_1 &= \big(\gamma^{P_1} \parallel 2^{P_1\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1)\,, \text{ where } \gamma^{P_1} = \{a \cap P_1 \mid a \in \gamma\}\,. \end{aligned} \tag{6}$$

and

$$\begin{aligned} (A[\mathcal{B}_1])[\mathcal{B}_2] &= \big(\{C_2\}, P_A \cup P_1 \cup P_2, \gamma \parallel 2^{(P_1\cup P_2)\setminus P_A}\big)\,, \\ \text{with } C_2 &= \big(\gamma^{P_1\cup P_2} \parallel 2^{(P_1\cup P_2)\setminus P_A}\big)(\{C_1\} \cup \mathcal{B}_2)\,. \end{aligned} \tag{7}$$

Since the interaction models and the constituent atomic behaviours 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 $a \in \gamma^{P_1\cup P_2} \parallel 2^{(P_1\cup P_2)\setminus P_A}$. By definition of '$\parallel$', $a = a_1 \cup a_2$, with $a_1 \in \gamma^{P_1\cup P_2}$ and $a_2 \subseteq (P_1 \cup P_2) \setminus P_A$. By (5), 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 $a \cap P_1 \in \gamma^{P_1} \parallel 2^{P_1\setminus P_A}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (6). By (2), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

Prop. 2.18 generalises Prop. 2.14. In order to generalise Prop. 2.13, we first define the application of one architecture to another, by putting

$$A_1[A_2] \triangleq (A_1 \oplus A_2)[\emptyset]\,. \tag{8}$$

Lemma 2.19. *For any set of behaviours $\mathcal{B}$ and any architectures A_1 and A_2, holds $(A_1 \oplus A_2)[\mathcal{B}] = (A_1[\mathcal{B}] \oplus A_2)[\emptyset] = (A_1 \oplus A_2[\mathcal{B}])[\emptyset]$.*

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 $i = 1, 2$, $A_1[\mathcal{B}] = (\{C_1'\}, P_{A_1}', \gamma_1')$. 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 behaviour (see Def. 2.16). Thus we have to show that the coordinating behaviours and the interaction models of both architectures coincide.

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

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. Def. 2.4)

$$\varphi_{\gamma'} \wedge \varphi_{\gamma''} = \varphi_{\gamma' || \gamma''}. \tag{9}$$

Denote the interaction model of $A_1[\mathcal{B}]$ by $\gamma_1' = \gamma_1 || 2^{P_1 \setminus P_{A_1}}$. Clearly, $\varphi_{\left(2^{P_1 \setminus P_{A_1}}\right)} = \mathtt{tt}$. Hence, by (9), we have $\varphi_{\gamma_1'} = \varphi_{\gamma_1} \wedge \varphi_{\left(2^{P_1 \setminus P_{A_1}}\right)} = \varphi_{\gamma_1}$ and, consequently, the characteristic predicate of the interaction model of $(A_1[\mathcal{B}] \oplus A_2)[\emptyset]$ is $\varphi_{\gamma_1'} \wedge \varphi_{\gamma_2} = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. 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 $\varphi_{\gamma_{12}} = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

Let us consider the coordinating behaviours of the two architectures. By Def. 2.16, we have[3]

$$\begin{aligned}(A_1 \oplus A_2)[\mathcal{B}] &= \left(\{C_{12}\}, P_2 \cup P_{A_1} \cup P_{A_2}, \gamma_{12} \parallel 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\right), \\ \text{with } C_{12} &= \left(\gamma_{12}^{P_2} \parallel 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\right)(\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{B}), \text{ where } \gamma_{12}^{P_2} = \{a \cap P_2 \mid a \in \gamma_{12}\}.\end{aligned} \tag{10}$$

Similarly,

$$\begin{aligned}A_1[\mathcal{B}] &= \left(\{C_1\}, P_1 \cup P_{A_1}, \gamma_1 \parallel 2^{P_1 \setminus P_{A_1}}\right), \\ \text{with } C_1 &= \left(\gamma_1^{P_1} \parallel 2^{P_1 \setminus P_{A_1}}\right)(\mathcal{C}_1 \cup \mathcal{B}), \text{ where } \gamma_1^{P_1} = \{a \cap P_1 \mid a \in \gamma_1\}.\end{aligned} \tag{11}$$

and

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

Notice that the interaction models and the constituent atomic behaviours in (10) and (12) 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 $a \in \gamma_{12}^{P_2} \parallel 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$. By definition of '$\parallel$', $a = a_1 \cup a_2$ 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 (10), we have $a_1 = \tilde{a}_1 \cap P_2$ with some $\tilde{a}_1 \in \gamma_{12}$. Since $\varphi_{\gamma_{12}} = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$, by Lem. 2.9, 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 $a \cap P_1 = (a_1 \cap P_1) \cup (a_2 \cap P_1) = (a_1 \cap P_1) \cup a_2 \in \gamma_1^{P_1} \parallel 2^{P_1 \setminus P_{A_1}}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (11). By (2), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

As a consequence of Lem. 2.19, we immediately obtain the following generalisation of Prop. 2.13.

Proposition 2.20. *For any set of behaviours $\mathcal{B}$ and any architectures A_1 and A_2, holds $A_2\big[A_1[\mathcal{B}]\big] = A_1\big[A_2[\mathcal{B}]\big]$.*

Proof. By (8) and Lem. 2.19, we have

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

□

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

Notice, furthermore, that (8) generalises Def. 2.16. Indeed, to a given set of behaviours $\mathcal{B}$, we can associate the architecture $A_{\mathcal{B}} \triangleq A_{id}[\mathcal{B}]$ (cf. Prop. 2.11). By (8) and Lem. 2.19, 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 behaviours 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 behaviours that are already defined. An architecture restricts the behaviour of it's arguments, which can be both behaviours and other architectures.

3 Property Preservation

3.1 Safety Properties

Definition 3.1 (Paths and path fragments)**.** Let $B = (Q, q^0, P, \to)$ be a behaviour. A finite or infinite sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{i-1}} q_{i-1} \xrightarrow{a_i} q_i \cdots$ is a *path* in B if $q_0 = q^0$, otherwise it is a *path fragment*.

Definition 3.2 (Properties and invariants)**.** Let $B = (Q, q^0, P, \to)$ be a behaviour. A *property* of B is a subset of states $\Phi \subseteq Q$. A property Φ is an *invariant* of B iff $\forall q \in \Phi, \forall a \in 2^P, (q \xrightarrow{a} q' \implies q' \in \Phi)$. Φ 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 $q^n \in \Phi$. If $q^0 \in \Phi$, then Φ is called *initial*.

Definition 3.3 (Projection)**.** Consider a set of behaviours $\mathcal{B}$ and an architecture $A = (\mathcal{C}, P_a, \gamma)$ with $P_a \subseteq P = \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. The *projection* of $A(\mathcal{B})$ onto the behaviours $\mathcal{B}$ is the behaviour $\overline{A(B)} \triangleq (Q, q^0, P, \to)$, where $Q = \prod_{B \in \mathcal{B}} Q_B$, $q^0 = (q^0_B)_{B \in \mathcal{B}}$ and, for any states $q, q' \in Q$, $q \xrightarrow{a} q'$ iff there exist $\tilde{q}, \tilde{q}' \in \prod_{C \in \mathcal{C}} Q_C$, such that $\tilde{q}q \xrightarrow{a} \tilde{q}'q'$ in $A(\mathcal{B})$.

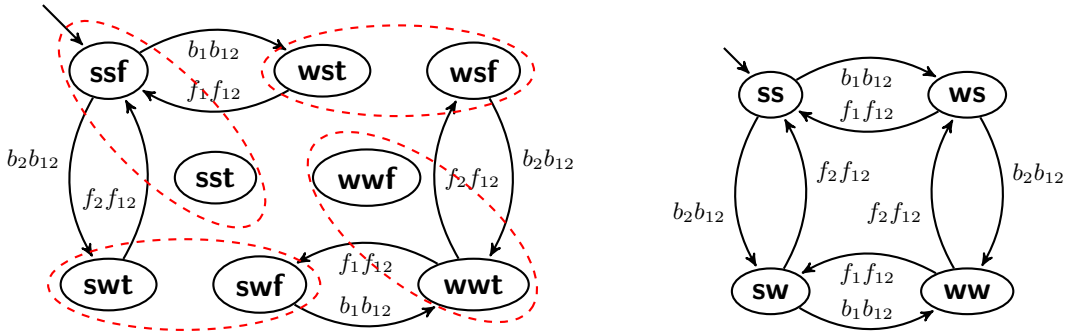
Notice that, since we are interested in enforcing properties, which are defined in terms of states, it is sufficient, in Def. 3.3, to project the states of $A(\mathcal{B})$, keeping the actions of coordinating behaviours in the transition labels. This choice does not affect the properties of the system, but simplifies the presentation and proofs.

Theorem 3.4 (Invariant preservation)**.** *Let $\mathcal{B}$ be a set of behaviours; let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two architectures, such that Φ_1 and Φ_2 are respectively invariants of $\overline{A_1(\mathcal{B})}$ and $\overline{A_2(\mathcal{B})}$. Then $\Phi_1 \cap \Phi_2$ is an invariant of $\overline{(A_1 \oplus A_2)(\mathcal{B})}$.*

Proof. Without loss of generality, we can assume that each of the two architectures has only one coordinating behaviour, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$. We also denote, for $i = 1, 2$, $P_i = P_{C_i} \cup \bigcup_{B \in \mathcal{B}} P_B$.

Consider a state $q \in \Phi_1 \cap \Phi_2$ and a transition $q \xrightarrow{a} q'$ in $\overline{(A_1 \oplus A_2)(\mathcal{B})}$. We have to show that $q' \in \Phi_1 \cap \Phi_2$.

By Def. 3.3, there exist some states $\tilde{q}_1, \tilde{q}'_1 \in Q_{C_1}$ and $\tilde{q}_2, \tilde{q}'_2 \in Q_{C_2}$, such that $\tilde{q}_1\tilde{q}_2 q \xrightarrow{a} \tilde{q}'_1\tilde{q}'_2 q'$ in $(A_1 \oplus A_2)(\mathcal{B})$. Thus, by Lem. 2.10, we have $\tilde{q}_1 q \xrightarrow{a \cap P_1} \tilde{q}'_1 q'$ in $A_1(\mathcal{B})$ and, consequently, by Def. 3.3, $q \xrightarrow{a \cap P_1} q'$ in $\overline{A_1(\mathcal{B})}$. As observed above $q \in \Phi_1$ and, since Φ_1 is an invariant for $\overline{A_1(B)}$, we deduce that $q' \in \Phi_1$. By symmetry, we also have $q' \in \Phi_2$, which concludes the proof. □



(a) $A_{12}(B_1, B_2)$ (b) $\overline{A_{12}(B_1, B_2)}$

Figure 2: Behaviours from Ex. 3.5 (states merged at projection shown by red ellipses).

Th. 3.4 can be further strengthened. Indeed, the definition of the projection does not exclude the existence of a path fragment of the form $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$ in $\overline{A(\mathcal{B})}$ generated by some transitions $\tilde{q}_1 q_1 \xrightarrow{a} \tilde{q}_2 q_2$ and $\tilde{q}'_2 q_2 \xrightarrow{b} \tilde{q}_3 q_3$ in $A(\mathcal{B})$ with $\tilde{q}_2 \neq \tilde{q}'_2$. It is possible furthermore, for some Φ, that $q_3 \notin \Phi$, but $\tilde{q}'_2 q_2$ is unreachable in $A(\mathcal{B})$. Thus, even though Φ is not an invariant of $\overline{A(\mathcal{B})}$, it can still represent a property enforced by A on $\mathcal{B}$.

Example 3.5. Consider again the mutual exclusion in Ex. 2.5. The behaviours of $A_{12}(B_1, B_2)$ and its projection $\overline{A_{12}(B_1, B_2)}$ are shown in Fig. 2(a) and Fig. 2(b) respectively (we abbreviate `sleep`, `work`, `free` and `taken` to `s`, `w`, `f` and `t` respectively). Although mutual exclusion property $\Phi = \{\texttt{ss}, \texttt{ws}, \texttt{sw}\}$ is not an invariant of $\overline{A_{12}(B_1, B_2)}$, it is clearly enforced by A_{12} on $\{B_1, B_2\}$.

Definition 3.6 (Enforcing properties)**.** Let $A = (\mathcal{C}, P_a, \gamma)$ be an architecture; let $\mathcal{B}$ be behaviours; let Φ be an initial property of their parallel composition $A_{id}(\mathcal{B})$ (see Prop. 2.11). We say that A *enforces* Φ *on* $\mathcal{B}$ iff, for every state q reachable in $A(\mathcal{B})$, the projection of q in $\overline{A(\mathcal{B})}$ belongs to Φ.

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 considered behaviours. Below, we omit mentioning this explicitly.

Theorem 3.7 (Combining enforced properties)**.** *Let $\mathcal{B}$ be a set of behaviours; 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 $\Phi_1 \cap \Phi_2$.*

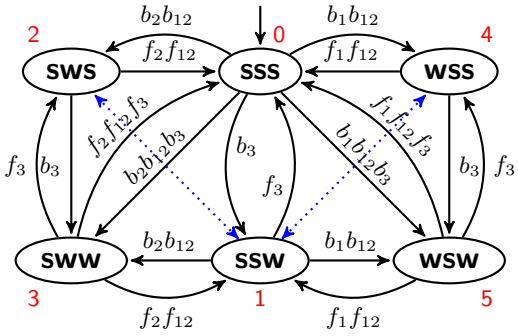
Proof. Again, without loss of generality, we can assume that each of the two architectures has only one coordinating behaviour, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$. We also denote, for $i = 1, 2$, $P_i = P_{C_i} \cup \bigcup_{B \in \mathcal{B}} P_B$.

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

Consider a path $\tilde{q}_1^0 \tilde{q}_2^0 q^0 \xrightarrow{a_1} \tilde{q}_1^1 \tilde{q}_2^1 q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_1^k \tilde{q}_2^k q^k$ 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 $i = 1, 2$.

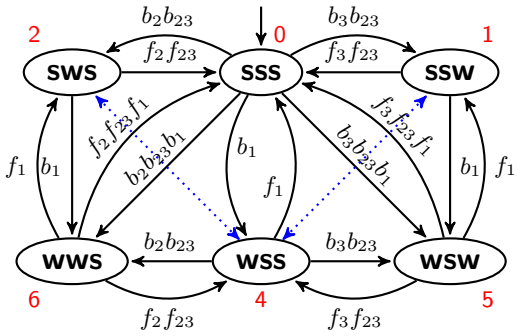
By Lem. 2.10, $\tilde{q}_1^0 q^0 \xrightarrow{a_1 \cap P_1} \tilde{q}_1^1 q^1 \xrightarrow{a_2 \cap P_1} \cdots \xrightarrow{a_k \cap P_1} \tilde{q}_1^k q^k$ is a path in $A_1(\mathcal{B})$. Thus the state $\tilde{q}_1^k q^k$ is reachable in $A_1(\mathcal{B})$. Since A_1 imposes Φ_1 on $\mathcal{B}$, this implies that $q^k \in \Phi_1$. Symmetrically, $q^k \in \Phi_2$, which concludes the proof. □

Example 3.8. In the context of Ex. 2.12, consider the application of architectures A_{12} and A_{23} to the behaviours B_1, B_2 and B_3. The former enforces the property $\Phi_{12} = \{\texttt{ss}*, \texttt{ws}*, \texttt{sw}*\}$

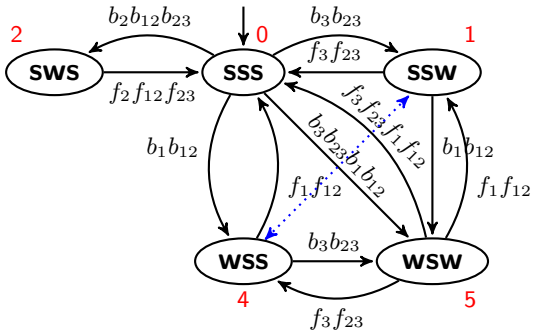


For ease of reading, we omit the transitions indicated in blue and additionally label each state with a red number, whereof the main label is the binary representation with $\mathtt{s} = 0$ and $\mathtt{w} = 1$.

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



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



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

Figure 3: Projections of reachable states of Ex. 3.8 behaviours onto $A_{id}(B_1, B_2, B_3)$.

(the projections of reachable states of $A_{12}(B_1, B_2, B_3)$ are shown in Fig. 3(a)), whereas the latter enforces $\Phi_{23} = \{*\texttt{ss}, *\texttt{ws}, *\texttt{sw}\}$ (the projections of reachable states of $A_{23}(B_1, B_2, B_3)$ are shown in Fig. 3(b)). Notice that the composition $A_{12} \oplus A_{23}$ enforces $\Phi_{12} \cap \Phi_{23} = \{\texttt{sss}, \texttt{wss}, \texttt{sws}, \texttt{ssw}, \texttt{wsw}\}$, i.e. mutual exclusion between the `work` states of B_1 and B_2, and between those of B_2 and B_3 (see Fig. 3(c)). Mutual exclusion between the `work` states of B_1 and B_3 is not enforced.

3.2 Liveness Properties

Our treatment of liveness properties derives from the Büchi-acceptance condition: we designate a subset of the states of each coordinator as "idle", meaning that it is permissible (w.r.t. liveness) for the coordinator to remain in such a state forever. However, we depart from Büchi as follows: if the coordinator executes infinitely often, then we consider it to be live regardless of whether or not it visits an idle state infinitely often.

Definition 3.9 (Architecture with liveness conditions)**.** An *architecture with liveness conditions* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is the set of *coordinating behaviours with liveness condition*, P_A is a set of ports, such that $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$, $\gamma \subseteq 2^{P_A}$ is an interaction model. A coordinating behaviour with liveness condition is a behavior $C = (Q, q^0, Q_{idle}, P_C, \rightarrow)$, where $Q_{idle} \subseteq Q$, and the other components are as in Def. 2.1.

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

Definition 3.10 (Live path)**.** Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture with liveness conditions and $\mathcal{B}$ a set of behaviours. An infinite path α in $A(\mathcal{B})$ is *live* iff, for every $C \in \mathcal{C}$, α contains infinitely many occurrences of interactions containing some port from C, or α contains infinitely many states whose projection onto C is an idle state of C.

That is, if $\alpha \triangleq \tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} \tilde{q}_i q_i \cdots$ then, for every $C \in \mathcal{C}$, for infinitely many i: $a_i \cap P \neq \emptyset$ or $\tilde{q}_i \restriction C \in Q_{idle}$, where $C = (Q, q^0, Q_{idle}, P, \rightarrow)$, and $\tilde{q}_i \restriction C$ denotes the local state of C in $\tilde{q}_i$.

The intuition behind this definition is that each liveness condition guarantees that its coordinator executes "sufficiently often", i.e. infinitely often unless it is in an 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 Def. 2.8, 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 behaviors, thereby obtaining a system, we need the notion of machine closure [1]: every finite path can be extended to a live one.

Definition 3.11 (Live w.r.t. a set of behaviors)**.** Let A be an architecture with liveness conditions and $\mathcal{B}$ be a set of behaviours. 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 that enforce contradictory scheduling policies. Hence, we define a notion of "non-interference" which guarantees that $(A_1 \oplus \cdots \oplus A_m)(\mathcal{B})$ is live w.r.t. $\mathcal{B}$.

We use the following definitions in discussing liveness. 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 α executes C infinitely

often iff α contains an infinite number of transitions that execute C. An infinite path fragment $\tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \cdots$ *visits an idle state* of coordinator C infinitely often iff, for infinitely many $i \geq 0$, $\tilde{q}_i \restriction C$ (the state component of C in $\tilde{q}_i$) is an idle state of C.

A system is free of *global deadlock* iff, in every reachable global state, there is at least one enabled interaction. We show in [3] how to verify that a system is free of global deadlock, using a sufficient but not necessary condition that, in many cases, can be evaluated quickly, without state-explosion. Essentially, we check, for every interaction a in the system, that the execution of a cannot possibly lead to a deadlock state. The check can often be discharged within a "small subsystem," which contains all of the components that participate in a.

We now give a criterion for liveness that can be evaluated without state-explosion. For simplicity, we assume in the sequel that each architecture A_i has exactly one coordinating component C_i.

Definition 3.12 (Noninterfering live architectures)**.** Let architectures $A_i = (\{C_i\}, P_{A_i}, \gamma_i)$, for $i = 1, 2$ be live w.r.t. a set of behaviours $\mathcal{B}$. Then A_1 is *non-interfering* with respect to A_2 and behaviors $\mathcal{B}$ iff, for every infinite path fragment α in $(A_1 \oplus A_2)(\mathcal{B})$ which executes C_1 infinitely often: either α executes C_2 infinitely often or α visits an idle state of C_2 infinitely often.

Theorem 3.13 (Pairwise noninterfering live architectures)**.** *Let architectures $A_i = (\{C_i\}, P_{A_i}, \gamma_i)$, for $i \in \{1, \ldots, m\}$ be live w.r.t. a set of behaviours $\mathcal{B}$. Assume that (a) for all $j, k \in \{1, \ldots, m\}, j \neq k$: A_j is non-interfering w.r.t A_k and behaviors $\mathcal{B}$, (b) $(\bigoplus_{i=1}^m A_i)(\mathcal{B})$ is free of global deadlock. Then $(\bigoplus_{i=1}^m A_i)$ is live w.r.t $\mathcal{B}$.*

Proof. Let α_{fin} be a finite path in $(\bigoplus_{i=1}^m A_i)(\mathcal{B})$. By assumption (b), there exists at least one extension of α_{fin} to an infinite path, which we call α. Since there are a finite number of controllers, and since, by construction, every interaction of $(\bigoplus_{i=1}^m A_i)(\mathcal{B})$ involves at least one controller, it follows that at least one controller executes infinitely often along α. Hence for some $j \in \{1, \ldots, m\}$, C_j executes infinitely often along α. Now consider C_k for arbitrary $k \in \{1, \ldots, m\} \setminus \{j\}$. Let α_{jk} be the projection of α onto $(A_j \oplus A_k)(\mathcal{B})$, which is constructed as follows.

A transition that involves either or both of C_j, C_k is retained in α_{jk}.

A transition that involves neither C_j nor C_k is dealt with as follows. Let a be the interaction executed along this transition. Hence a is supplied by some architecture A_ℓ, where $\ell \neq j, \ell \neq k$. There are two cases. First, suppose that a contains some port p which is also contained in some a' which involves at least one of C_j, C_k. Then, a and a' will be "fused" in $(\bigoplus_{i=1}^m A_i)(\mathcal{B})$, and so the projection of this transition onto $(A_j \oplus A_k)(\mathcal{B})$ is a valid transition of $(A_j \oplus A_k)(\mathcal{B})$. Onthe other hand, suppose that is is not the case. Then a includes only ports that are not in the port set of $A_j \oplus A_k$). We emulate the effect of a on $\mathcal{B}$ by a sequence of local transitions of the involved components. Since the behavior in $(A_j \oplus A_k)(\mathcal{B})$ of ports in $\mathcal{B}$ which are not in the port set of $A_j \oplus A_k$) is unrestricted, this is always possible. We then "project" the transition onto $(A_j \oplus A_k)(\mathcal{B})$ by replacing it by such a sequence of local transitions.

We conclude that α_{jk} is a path in $(A_j \oplus A_k)(\mathcal{B})$. Furthermore, α_{jk} is an infinite path in $(A_j \oplus A_k)(\mathcal{B})$, since it contains an infintie number of transiitons by C_j. By assumption (a), we conclude that, along α_{jk}, C_k is either executed infinitely often, or visits an idle state infintiely often. HEnce, the same holds along α. Since C_k was chosen arbitrarily, we conclude that α is a live path. □

Example 3.14 (Noninterference in mutual exclusion)**.** Consider the system $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$, as in Ex. 2.12. Let each coordinator have a single idle state, namely the `free` state. Consider the applications of each pair of coordinators, 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.

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 local deadlock using the method of [3]. Hence by Th. 3.13, we conclude that $(A_{12} \oplus A_{23} \oplus A_{13})$ is live w.r.t. (B_1, B_2, B_3),

We have implemented an algorithm to check (for finite-state systems) that A_1 is non-interfering with respect to A_2 and behaviors $\mathcal{B}$. We generate the state-transition diagram of $(A_1 \oplus A_2)(\mathcal{B})$, remove all transitions of C_2 and all global states whose C_2-component is an idle state of C_2. We then check for the existence of a non-trivial strongly connected components. 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 an infinite path fragment along which C_1 executes forever, while C_2 does not execute and is not in an idle state. Hence non-interference is violated. Figure 4 gives pseudocode for our algorithm.

checkStrongNonIntrf$(A_1, A_2, \mathcal{B})$
//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. let M' be the result of removing from M all transitions that involve C_2
3. let M'' be the result of removing from M' all global states that project onto an idle state of C_2
4. compute the set of maximal strongly connected components of M''
5. **if** there exists a maximal strongly connected component φ of M'' such that
6. $\quad$ φ contains more than one state, or
7. $\quad$ φ consists of a single state with a self-loop,
8. **then return**(ff)
9. **else return**(tt) //return tt if A_1 does not interfere with A_2

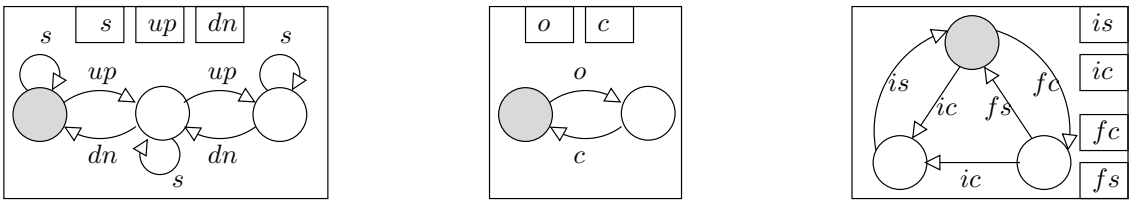
Figure 4: Pseudo-code for checking strong non-interference

Proposition 3.15. checkStrongNonIntrf$(A_1, A_2, \mathcal{B})$ *returns true iff* A_1 *is non-interfering with respect to* A_2 *and* $\mathcal{B}$.

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

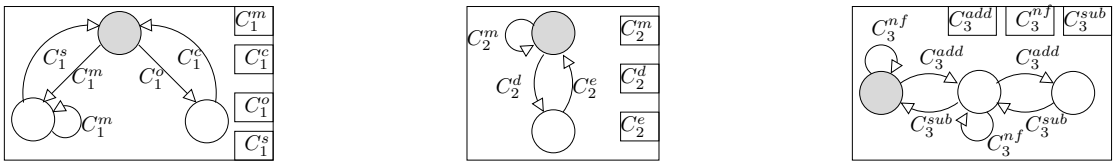
Suppose first that checkStrongNonIntrf$(A_1, A_2, \mathcal{B})$ returns tt. Then, M'' contains non non-trivial maximal strongly connected components. Hence, every cycle in M must contain either a transition involving C_2, or a global state that projects onto an idle state of C_2, since otherwise this cycle would project onto a non-trivial maximal strongly connected component in M''. Hence, there is no infinite path fragment in M along which C_2 never executes and is never in an idle state. Hence, along every infinite path in M, either C_2 executes infinitely often, or it is in an idle state infinitely often. Hence A_1 is non-interfering with respect to A_2.

Now suppose that A_1 is non-interfering with respect to A_2. Hence along every infinite path in M, either C_2 executes infinitely often, or it is in an idle state infinitely often. Let Cy be an arbitrary cycle in M, where we consider a state with a self-loop to be a cycle. Hence, Cy either contains a transition by C_2, or it contains some state that projects onto an idle state of C_2. Hence Cy cannot project onto M'' as a cycle, by construction of checkStrongNonIntrf$(A_1, A_2, \mathcal{B})$. Hence, M'' contains no cycles , and so no non-trivial maximal strongly connected components. Hence checkStrongNonIntrf$(A_1, A_2, \mathcal{B})$ returns tt. □



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

Figure 5: Elevator behaviours.



(a) C_1 (b) C_2 (c) C_3

Figure 6: Coordinating behaviours for the elevator example.

4 Case Study: Control of an Elevator Cabin

We illustrate our approach with the Elevator case study adapted from the literature [11, 20]. Control of the elevator cabin is modelled as a set of coordinated atomic components shown in Fig. 5. 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* represents cabin stops in response to these calls. Furthermore, in port names, m, c, o, s, dn, up and nf stand respectively for "move", "call", "open", "stop", "move down", "move up" and "not full".

Caller system components and their ports are indexed by floor numbers. In this case study, we consider a building with three floors. We denote $\mathcal{B} = \{E, D, CS_0, CS_1, CS_2\}$ the set of base behaviours.

Possible behaviours of a system are constrained by architectures. Each architecture ensures some properties of the system. Composing these architectures together we obtain the system, which satisfies all properties.

In order to provide the basic functionality of the elevator we apply to $\mathcal{B}$ the architecture $A_1 = (\{C_1\}, P_1, \gamma_1)$. C_1 is shown in Fig. 6(a), P_1 contains all ports of C_1,and all ports of base behaviours. γ_1 compraises the empty interaction $\emptyset$ and the following interactions (for $i \in [0, 2]$)

- Door control: $o\, C_1^o$, $c\, C_1^c$
- Floor selection control: fc_i , ic_i
- Moving control: $s\, C_1^s\, fs_i$, $s\, C_1^s\, is_i$, $up\, C_1^m$, $dn\, C_1^m$

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

Nonetheless, this system allows the elevator to stop at a floor and then continue moving without having opened the doors. To disable this behaviour, we apply the architecture $A_2 = (\{C_2\}, P_2, \gamma_2)$ with C_2 shown in Fig. 6(b), $P_2 = \{C_2^e, C_2^d, C_2^m, C_1^c, C_1^s, C_1^m\}$, and $\gamma_2 = \{\emptyset, C_1^s\, C_2^d, C_1^c\, C_2^e, C_1^m\, C_2^m\}$,

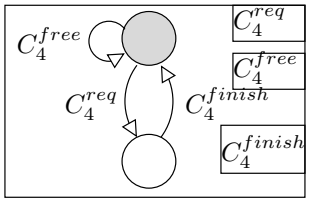


Figure 7: Executive floor coordinator.

which grants priority to the door controller after a C_1^s action. The $A_2(A_1(\mathcal{B}))$ provides the same basic functionality, as A_1 does, but enforces the additional constraint. By Prop. 2.13, we have $A_2(A_1(\mathcal{B})) = (A_2 \oplus A_1)(\mathcal{B})$.

The feature *"if the elevator is full, it must stop only at floors selected from the cabin and ignore outside calls"* [11, 20], is provided by applying the architecture $A_3 = (\{C_3\}, P_3, \gamma_3)$ with C_3 shown in Fig. 6(c), $P_3 = \{C_3^{add}, C_3^{sub}, C_3^{nf}, s, fs_i \,|\, i \in [0,2]\}$ and $\gamma_3 = \{\emptyset, C_3^{add}, C_3^{sub}\} \cup \{s\, C_3^{nf}\, fs_i \,|\, i \in [0,2]\}$.

In a composition of A_3 and $A_1 \oplus A_2$, $A_1 \oplus A_2$ does not enforce any constraints on ports $C_3^{add}, C_3^{sub} \notin P_1 \cup P_2$, thus there are singleton interactions C_3^{add}, C_3^{sub} in the composed architecture. Similarly A_3 does not affect interactions $o\, C_1^o$, $c\, C_1^c\, C_2^e$ etc. On the other hand, the ports s and fs_i are forced to synchronise with C_1^s by $A_1 \oplus A_2$ and with C_3^{nf} by A_3. In the combined subsystem $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$, these two interactions get "fused" into $C_1^s\, C_2^d\, C_3^{nf}\, s\, fs_i$, which forces the elevator to ignore the calls from floors when it is full.

To specify liveness properties, we set the idle states of C_1 to the empty set, since we expect the elevator to move infinitely often, under the assumption that users keep arriving. We set the idle states of C_2 to the initial state, since the door must close, once opened. We set the idle states of C_3 to be all of its states, since C_3 enforces a pure safety property. We have implemented our algorithm for checking non-interference, and have used the implementation to prove that each of C_1, C_2, C_3 is non-interfering w.r.t. the other, and the behaviors $\{E, D, CS\}$. Hence by Th. 3.13, $(A_1 \oplus A_2 \oplus A_3)$ is live w.r.t. (E, D, CS).

Finally, we consider another feature: *"requests from the second floor have priority over all other requests"* [11, 20]. We compose $A_1 \oplus A_2$ with the architecture $A_4 = (\{C_4\}, P_4, \gamma_4)$ with C_4 shown in Fig. 7; P_4 consisting of ports of C_4, CS_2 and o, s and dn;

$$\gamma_4 = \{\emptyset, fc_2\, C_4^{req}, ic_2\, C_4^{req}, o\, C_4^{free}, dn\, C_4^{free}, fs_2\, C_4^{finish},\ is_2\, C_4^{finish}\}\,.$$

A system $(A_1 \oplus A_2 \oplus A_3 \oplus A_4)(\mathcal{B})$, with contradictory constraints enforced by A_3 and A_4, contains a reachable local deadlock state. In a situation, when the full elevator is called from the second floor, A_4 enforces a constraint of not going down and A_3 forbids to stop on the floor, not requested from the inside. In this case, when the elevator reaches the second floor, the system is in a local deadlock state involving the elevator engine. This was detected using our implementation of the deadlock prevention approach presented in [3].

5 Related Work

There exists an abundant literature on architectures. Most papers propose and study Architecture Description Languages (ADLs) [18]. They focus on technological and syntactic aspects, and mostly disregard semantics and foundational aspects. Furthermore, they do not deal with correctness by construction which is the main reason for using architectures. Our concept of architecture is

rooted in clean semantics, is equipped with a mathematically elegant definition and an intuitively appealing notion of composition that preserves state invariants.

A number of paradigms for unifying component composition have been studied in [4, 5, 12]. These achieve unification by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behavior. This is also true for the numerous compositional and algebraic frameworks [2, 14, 21, 23, 7, 9, 16, 19, 17]. Most of these frameworks are based on a single operator. This entails poor expressiveness which results in utterly complex architectural designs. Our component framework is inspired from BIP which allows expression of general multiparty interaction and strictly respects separation of concerns. Glue can be studied as a separate entity that admits a simple Boolean characterization that is instrumental for expressing composability.

Existing research on architecture composability deals mainly with resource composability for particular types of architectures, e.g. [17]. 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 modeled as architectural constraints. A survey on feature interaction research is provided in [10]. Existing results focus mainly on modeling 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 [15] 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.

6 Conclusion

The presented work is a first step toward the study of a rigorous concept of architecture and its effective use for achieving correctness by construction in a system design flow. A key idea is that an architecture solves a coordination problem by enforcing a characteristic property which is the conjunction of a safety and a liveness property. It considers preservation of safety properties as an essential condition for architecture composability. Liveness is a generic property. Its preservation seems to be much harder to be guaranteed by simple constructive criteria.

Our work pursues similar objectives as the research on feature interaction, insofar as features can be modeled 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. 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

[1] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, January 1993.

[2] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[3] Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. An abstract framework for deadlock prevention in BIP. In *FMOODS/FORTE*, pages 161–177, 2013.

[4] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[5] K. Balasubramanian, A.S. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema. Developing applications using model-driven design environments. *IEEE Computer*, 39(2):33–40, 2006.

[6] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. *Software, IEEE*, 28(3):41–48, 2011.

[7] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. On the formalization of architectural types with process algebras. In *SIGSOFT FSE*, pages 140–148, 2000.

[8] Simon Bliudze and Joseph Sifakis. Synthesizing glue operators from glue constraints for the construction of component-based systems. In Sven Apel and Ethan Jackson, editors, *Software Composition*, volume 6708 of *LNCS*, pages 51–67. Springer Berlin / Heidelberg, 2011.

[9] Roberto Bruni, Ivan Lanese, and Ugo Montanari. A basic algebra of stateless connectors. *Theoretical Computer Science*, 366(1):98–120, 2006.

[10] Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, January 2003.

[11] Deepak D'Souza and Madhu Gopinathan. Conflict-tolerant features. In *CAV*, volume 5123 of *LNCS*, pages 227–239. Springer, 2008.

[12] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[13] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, June 1987.

[14] José Luis Fiadeiro. *Categories for Software Engineering*. Springer-Verlag, April 2004.

[15] Jonathan D. Hay and Joanne M. Atlee. Composing features and resolving interactions. *SIGSOFT Softw. Eng. Notes*, 25(6):110–119, November 2000.

[16] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall, April 1985.

[17] Isaac Liu, Jan Reineke, and Edward A. Lee. 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 on*, pages 2111–2115, 2010.

[18] Neno Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[19] Robin Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.

[20] Malte Plath and Mark Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 41(1):53–84, 2001.

[21] Arnab Ray and Rance Cleaveland. Architectural interaction diagrams: AIDs for system modeling. In *ICSE'03: Proceedings of the 25th International Conference on Software Engineering*, pages 396–406, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Joseph Sifakis. Rigorous System Design. *Foundations and Trends in Electronic Design Automation*, 6(4):293–362, 2012.

[23] Bridget Spitznagel and David Garlan. A compositional formalization of connector wrappers. In *ICSE*, pages 374–384. IEEE Computer Society, 2003.

[24] Peter Wegner. Coordination as constrained interaction (extended abstract). In *Coordination Languages and Models*, volume 1061 of *LNCS*, pages 28–33. Springer, 1996.