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, Lebanon; {pa07,mj54}@aub.edu.lb
[2] École Polytechnique Fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland; firstname.lastname@epfl.ch

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 P.
Architecture composability is a basic and common problem faced by system designers. Consider two architectures A_1, A_2, enforcing respectively properties P_1, P_2 on a set of components $\mathcal{B}$. That is, $A_1(\mathcal{B})$ and $A_2(\mathcal{B})$ satisfy respectively the properties P_1 and P_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 both P_1 and P_2?
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 invariants P_1 and P_2, the architecture $A_1 \oplus A_2$ enforces the state invariant $P_1 \wedge P_2$, that is both invariants are preserved by architecture composition. We also discuss preservation of liveness properties of 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 $\mathcal{B} = \{B_1, \ldots, B_n\}$ builds a composite component $A(\mathcal{B})$ meeting a characteristic property P_A. 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. This demand can be simply formulated in the following manner:

> Consider two architectures A_1 and A_2, enforcing respectively properties P_{A_1} and P_{A_2} on a set of components $\mathcal{B}$. That is, $A_1(\mathcal{B})$ and $A_2(\mathcal{B})$ satisfy respectively the properties P_{A_1} and P_{A_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 $P_{A_1} \wedge P_{A_2}$? 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 [1].

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 [2]. The underlying theory about components and their interaction is inspired from BIP [3]. 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 [4].

We consider that a *component framework* consists of a finite set of *atomic components* $\mathcal{B}$ and a *glue* $\Gamma = \{\gamma_k\}_{k \in K}$, which is a set of 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*, i.e. sets of actions of the composed components. The meaning of γ can be specified by using a set of 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).

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 any (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 their characteristic property. It is a solution to a specific coordination problem specified by P_A by using a particular interaction model specified by γ. 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 formal and general framework for architecture composability based on an architecture partial 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 invariants P_{A_1} and P_{A_2}, the architecture $A_1 \oplus A_2$ enforces the state invariant $P_{A_1} \wedge P_{A_2}$, that is both invariants 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 Formal model

2.1 Behaviours and Architectures

Definition 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. 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. *We also denote Q_B, P_B and $\rightarrow_B$ the constituents of a behaviour B.*

Definition 2 (Interaction model). *Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be a finite set of behaviours with $B_i = (Q_i, q_i^0, P_i, \rightarrow)$,[3] 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:

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

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

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.

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

Definition 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 $P_a \subseteq P \stackrel{\Delta}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. The* application of an architecture *A to the behaviours $\mathcal{B}$ is the behaviour*

$$A(\mathcal{B}) \stackrel{\Delta}{=} \left(\gamma \parallel 2^{P \setminus P_a}\right)(\mathcal{C} \cup \mathcal{B}), \tag{3}$$

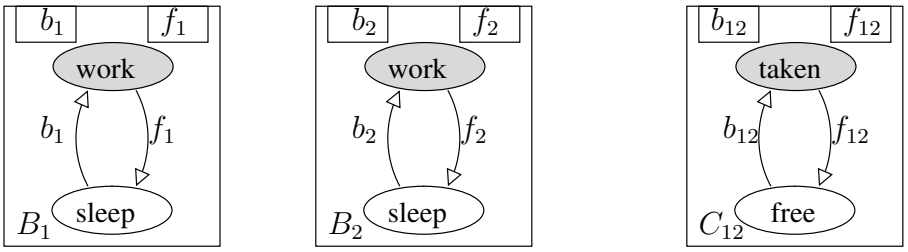
where, for interaction models γ' and γ'' over disjoint domains P' and P'' respectively, $\gamma' \parallel \gamma'' \stackrel{\Delta}{=} \{a' \cup a'' \mid a' \in \gamma', a'' \in \gamma''\}$ is an interaction model over $P' \cup P''$.

Example 1 (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 $\gamma_{12} = \{\emptyset, b_1 b_{12}, b_2 b_{12}, f_1 f_{12}, f_2 f_{12}\}$. We include the empty interaction in γ_{12} in order to allow components to skip a move (cf. (2))—this is essential for modelling asynchronous systems.

Assuming that the initial states of B_1 and B_2 are `sleep`, whereas that of C_{12} is `free`, one can observe that, neither of the two states (`free`, `work`, `work`) and (`taken`, `work`, `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 $\{b_3, f_3\}$. 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.

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



(a) (b)

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

2.2 Architecture composition

As will be further illustrated in Sect. 3, architectures can be intuitively understood as imposing constraints on the global state space of the system [3, 5]. More precisely, component coordination is realised by limiting the allowed synchronisation possibilities, thus imposing 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 formalised by the two definitions below.

Definition 5 (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*

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

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

Definition 6 (Architecture composition). *Let $A_j = (\mathcal{B}_j, P_j, \gamma_j)$, for $j = 1, 2$ be two architectures. The* composition *of A_1 and A_2 is an architecture $A_1 \oplus A_2 = (\mathcal{B}_1 \cup \mathcal{B}_2, P_1 \cup P_2, \gamma_\varphi)$, 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 imposed by both composed architectures.

Lemma 1. *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 $i = 1, 2$.*

Proposition 1. *Architecture composition '$\oplus$' is commutative, associative and idempotent.*

Example 2 (Mutual exclusion (contd.)). Building upon Ex. 1, 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 $\gamma_{i3} = \{\emptyset, b_i b_{i3}, b_3 b_{i3}, f_i f_{i3}, f_3 f_{i3}\}$. The composition of these three architectures, $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$, ensures mutual exclusion among the `work` states of all three behaviours.

Notice that

$$\varphi_{\gamma_{12}} \equiv (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \wedge \\ (b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})\,.$$

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

$$b_1 \Rightarrow b_{12} \wedge b_{13}\,, \quad f_1 \Rightarrow f_{12} \wedge f_{13}\,, \quad b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2\,, \quad f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2\,, \quad b_{12} \Rightarrow \overline{f_{12}}\,,$$
$$b_2 \Rightarrow b_{12} \wedge b_{23}\,, \quad f_2 \Rightarrow f_{12} \wedge f_{23}\,, \quad b_{13} \Rightarrow b_1 \,\texttt{XOR}\, b_3\,, \quad f_{13} \Rightarrow f_1 \,\texttt{XOR}\, f_3\,, \quad b_{13} \Rightarrow \overline{f_{13}}\,,$$
$$b_3 \Rightarrow b_{13} \wedge b_{23}\,, \quad f_3 \Rightarrow f_{13} \wedge f_{23}\,, \quad b_{23} \Rightarrow b_2 \,\texttt{XOR}\, b_3\,, \quad f_{23} \Rightarrow f_2 \,\texttt{XOR}\, f_3\,, \quad b_{23} \Rightarrow \overline{f_{23}}\,.$$

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

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

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 $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$.

2.3 Hierarchical composition of architectures

Proposition 2. *Let $\mathcal{B}$ be a set of behaviours and let $A_1 = (\mathcal{C}_1, P'_a, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P''_a, \gamma_2)$ be two architectures, such that $P'_a \subseteq P' \stackrel{\Delta}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1} P_B$ and $P''_a \subseteq P'' \stackrel{\Delta}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$.*

Proposition 3. *Let $\mathcal{B}_1, \mathcal{B}_2$ be two sets of behaviours, such that $P_1 \cap P_2 = \emptyset$, with $P_i = \bigcup_{B \in \mathcal{B}_i} P_B$, for $i = 1, 2$. Let $A_1 = (\mathcal{C}_1, P'_a, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P''_a, \gamma_2)$ be two architectures, such that $P'_a \cap P_2 = \emptyset$. Then $A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$.*

We now generalise Def. 4 to the case where $P_a \not\subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. This means that the architecture imposes constraints on some ports which are not present in any of the control or base behaviours. In other words, the system obtained by applying the architecture to a given set of behaviours is not *complete*. The result can then itself be considered as an architecture that can be further applied to additional behaviours in order to complete the system.

Definition 7. *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 $A[\mathcal{B}] \stackrel{\Delta}{=} (B', P \cup P_a, \gamma \parallel 2^{P \setminus P_a})$, where $B' \stackrel{\Delta}{=} (\gamma_P \parallel 2^{P \setminus P_a})(\mathcal{C} \cup \mathcal{B})$ with $\gamma_P = \{a \cap P \mid a \in \gamma\}$ and the operator '$\parallel$' as in Def. 4.*

In the above definition, it is important to notice that the interaction model in $A[\mathcal{B}]$ is not the same as in the definition of B'. On the other hand, if $P_a \subseteq P$ (as in Def. 4), we have $\gamma_P = \gamma$ and $A[\mathcal{B}] = (A(\mathcal{B}), P, \gamma \parallel 2^{P \setminus P_a})$.

Proposition 4. *Let $\mathcal{B}$ be set of behaviours and $A = (\mathcal{C}, P_a, \gamma)$ be an architecture, such that $P_a \subseteq P = \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. For any partition $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$ (with $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$), we have $A(\mathcal{B}) = A[\mathcal{B}_1](\mathcal{B}_2)$.*

Example 3 (Mutual exclusion (contd.)). It is clear that Ex. 2 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.

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. Assume furthermore, that an architecture A enforces mutual exclusion between C_{12} and C_{34}. It is clear that the system $A(A_{12}(B_1, B_2), A_{34}(B_3, B_4))$ has mutual exclusion between all four behaviours $(B_i)_{i=1}^4$. Furthermore, by the above propositions, we have

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

Ex. 3 and Def. 7 above 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 8 (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 9 (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 $A(\mathcal{B})$.*

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

Definition 10 (Imposing 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 $(2^P)(\mathcal{B})$, where $P = \bigcup_{B\in\mathcal{B}} P_B$. We say that A* imposes *Φ on $\mathcal{B}$ iff the projection in $\overline{A(\mathcal{B})}$ of any state, reachable in $A(\mathcal{B})$, belongs to Φ.*

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

Theorem 2 (Combining imposed properties). *Let $\mathcal{B}$ be a set of behaviours; let $A_i = (\mathcal{C}_i, P_i^a, \gamma_i)$, for $i = 1, 2$, be two architectures imposing on $\mathcal{B}$ the properties Φ_1 and Φ_2 respectively. The composition $A_1 \oplus A_2$ imposes on $\mathcal{B}$ the property $\Phi_1 \cap \Phi_2$.*

3.2 Liveness properties

We add a *liveness requirement* $\mathcal{L}$ to Def. 3. A liveness requirement is a set of *liveness conditions*, each of which is a (finite) set of ports.

Definition 11 (Architecture with Liveness Conditions). *An* architecture *with liveness conditions is a tuple $A = (\mathcal{C}, P, \gamma, \mathcal{L})$, where $\mathcal{C}$ is the set of* coordinating behaviours*, P is a set of ports, such that $\bigcup_{C\in\mathcal{C}} P_C \subseteq P$, $\gamma \subseteq 2^P$ is an interaction model and $\mathcal{L}$ is a set $\{L_1, \ldots, L_m\}$ of* liveness conditions*, with $L_i \subseteq \bigcup_{C\in\mathcal{C}} P_C$, for $i \in [1, m]$.*

Definition 12 (Live Path). *Let $A = (\mathcal{C}, P_a, \gamma, \mathcal{L})$ 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 $L \in \mathcal{L}$, α contains infinitely many occurrences of interactions containing some port from L. That is, if $\alpha \stackrel{\Delta}{=} \tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} \tilde{q}_i q_i \cdots$ then, for every $L \in \mathcal{L}$, for infinitely many i, $a_i \cap L \neq \emptyset$. Formally: $\forall L \in \mathcal{L}, \forall j \geq 0, \exists i > j : a_i \cap L \neq \emptyset$.*

An architecture is progressively built up from simpler architectures by composition, both "flat" (i.e. $(\oplus)$) and hierarchical. Hence there must be some "initial" architectures that are produced by other means (e.g. manual design, or synthesis from a temporal logic specification). When such initial architectures are composed, we take the union of all of their liveness conditions, so that all the coordinators of each initial architecture are still executed infinitely often in the composed system. This must also hold when we take two or more architectures that are not initial, i.e. are themselves the result of composition, and further compose them to obtain the step in the progressive construction. For example, consider a mutex system constructed hierarchically, where .....tree....

This leads to the following definition for composition.

Definition 13 (Live Architecture composition). *Let $A_j = (\mathcal{C}_j, P_j, \gamma_j, \mathcal{L}_j)$, for $j = 1, 2$ be two live architectures. The* composition *A_1 and A_2 is a live architecture $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi, \mathcal{L}_1 \cup \mathcal{L}_2)$, where $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.*

We also need the notion of feasibility, or machine closure [6]: every finite path can be extended to a live one.

Definition 14 (Feasible Live Architecture). *Let A be a live architecture and $\mathcal{B}$ be a set of behaviours. A is* feasible *w.r.t. $\mathcal{B}$ iff every finite path in $A(\mathcal{B})$ can be extended to a live path.*

Even if $A_1(\mathcal{B})$ and $A_2(\mathcal{B})$ are feasible, it is still possible for $(A_1 \oplus A_2)(\mathcal{B})$ to be infeasible, due to "interference" between the coordinators of A_1 and A_2.

A global state q enables a liveness condition L iff $q \xrightarrow{a}$ for some a such that $a \cap L \neq \emptyset$. An infinite path fragment $\tilde{q}_0 q_0 \xrightarrow{a_1} \tilde{q}_1 q_1 \cdots$ enables L continuously iff, for all $i \geq 0$, $\tilde{q}_i q_i$ enables L. An infinite path α enables L from some point onwards iff some infinite suffix of α enables L continuously. A transition $q \xrightarrow{a} q'$ executes a liveness condition L iff $a \cap L \neq \emptyset$. An infinite path α executes L infinitely often iff α contains an infinite number of transitions that execute L.

Definition 15 (Noninterfering Live Architectures). *Let $A_i = (\mathcal{C}_i, P_i, \gamma_i, \mathcal{L}_i)$, for $i = 1, 2$, be feasible w.r.t. a set of behaviours $\mathcal{B}$. Then, A_1, A_2 are* noninterfering *with respect to $\mathcal{B}$ iff, for every infinite path α in $(A_1 \oplus A_2)(\mathcal{B})$ and every $L \in \mathcal{L}_1 \cup \mathcal{L}_2$, α executes L infinitely often or α enables L from some point onwards.*

Theorem 3 (Noninterfering Live Architectures). *Let $A_i = (\mathcal{C}_i, P_i, \gamma_i, \mathcal{L}_i)$, for $i = 1, 2$ be feasible w.r.t. a set of behaviours $\mathcal{B}$. Assume further that A_1 and A_2 are noninterfering with respect to $\mathcal{B}$ and that $(A_1 \oplus A_2)(\mathcal{B})$ is free of global deadlock. Then $(A_1 \oplus A_2)$ is feasible w.r.t. $(\mathcal{B})$.*

Example 4 (Noninteference in Mutual exclusion). Consider a system $(A_{12} \oplus A_{23})(B_1, B_2, B_3)$, where, as in Ex. 2, A_{12} enforces mutual exclusion between B_1 and B_2 with a coordination behaviour C_{12}, and A_{23} enforces mutual exclusion between B_2 and B_3 with a coordination behaviour C_{23}, that is Ex. 2 with A_{13} removed. Let A_{12} have liveness condition $L_{12} = \{b_{12}, f_{12}\}$ and A_{23} have liveness condition $L_{23} = \{b_{23}, f_{23}\}$. We verify that A_{12} and A_{23} are noninterfering. Consider an infinite path α. If C_{23} is executed infinitely often along α then both b_{23} and f_{23} are executed infinitely often, and we are done. Otherwise, C_{23} is never executed along some infinite suffix α' of α, and so both C_{23} and B_3 do not change state in α'. If C_{23} is in state `free`, then B_3 must be in state `sleep`, and so interaction $b_{23}b_3$ is enabled continuously along α'. If C_{23} is in state `taken`, then B_3 must be in state `work`, and so interaction $f_{23}f_3$ is enabled continuously along α'. In either case, L_{23} is enabled from some point onwards in α. By a symmetric argument, we show that along every infinite path, L_{12} is infinitely often executed or enabled from some point onwards.

Now consider $(A_{12} \oplus A_{23} \oplus A_{13})(B_1, B_2, B_3)$, precisely as in Ex. 2. Here we have interference: there exists an infinite path $\alpha = (b_1 b_{12} b_{13}; f_1 f_{12} f_{13})^\omega$, along which C_{23} is never executed and no interaction of C_{23} is continuously enabled. Hence L_{23} is not continuously enabled. Note that the cyclic arrangement of components is not an issue here: the same phenomenon occurs if we have

four components B_1, B_2, B_3, B_4 with mutual exclusion between B_i and B_{i+1} for $i \in [1,3]$, and with the corresponding coordinators C_{12}, C_{23}, C_{34}. In this case, there exists an infinite path α such that C_{23} is never executed, and is enabled infinitely often but not continuously from some point onwards along α.

This example can be fixed by using an empty interaction. The liveness condition $\{b_{23}, f_{23}\}$ means that C_{23} cycles infinitely often through its `free` and `taken` states. What we actually want is that, when C_{23} is in `taken`, it eventually returns to `free`, i.e. the lock on the critical section is returned. We add a self-loop to the `free` state, labeled with the empty interaction $\emptyset$, and modify the liveness condition to $\{\emptyset, b_{23}, f_{23}\}$, noting that we also use $\emptyset$ as an "empty port".

Pairwise Noninterference To deal with the application of several architectures, we can iterate the above using commutativity and associativity of $\oplus$, i.e. use $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B}) = ((A_1 \oplus A_2) \oplus A_3)(\mathcal{B})$. However, the proof obligation of Th. 3 now must be discharged for the pair of architectures $(A_1 \oplus A_2)$ and A_3. As we successively compose more architectures, this proof obligation becomes more complex, since it involves more and more architectures. It is, in effect, a "global" rather than a local obligation. It would be better to have a proof obligation only over pairs of architectures, e.g. if each pair is non-interfering, then the composition of all the architectures is feasible.

Definition 16 (Subsystem). *Let $\bigoplus_{i=1}^{m} A_i$ be a composition of architectures, as in Def. 6, where each $A_i = (\mathcal{C}_i, P_a^i, \gamma_i)$ is an architecture. Let $\mathcal{B}$ be a set of behaviours. A* subsystem *of $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$ is $(\bigoplus_{i\in I} A_i)(\mathcal{B})$, where $I \subseteq [1,m]$.*

Definition 17 (Local deadlock). *A system $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$ is* free of local deadlock *iff, in every reachable state q of $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$, every subsystem $(\bigoplus_{i\in I} A_i)(\mathcal{B})$ considered in isolation has some enabled interaction, i.e., for some interaction a, we have $\overline{q} \xrightarrow{a}$ in $(\bigoplus_{i\in I} A_i)(\mathcal{B})$, where $\overline{q}$ is the projection of q onto the state space of $(\bigoplus_{i\in I} A_i)(\mathcal{B})$.*

Consider an architecture A with the set of coordinating behaviours $\mathcal{C}$ and a state $q \in \prod_{C\in\mathcal{C}} Q_C$. For a port $p \in \bigcup_{C\in\mathcal{C}} P_C$ we say that A locally enables p in state q, if $q \xrightarrow{a}$ and $p \in a$.

Definition 18 (Strongly Noninterfering Live Architectures). *Let live architectures $A_i = (\mathcal{C}_i, P_i, \gamma_i, \mathcal{L}_i)$, for $i = 1, 2$ be feasible w.r.t. a set of behaviours $\mathcal{B}$. Then A_1 and A_2 are* strongly non-interfering *with respect to $\mathcal{B}$ iff, for every infinite path fragment α in $(A_1 \oplus A_2)(\mathcal{B})$, the following holds:*

- *Assume that A_1 never executes along α, and let P_1^α be the set of ports that A_1 locally enables in every state of α (these do not change along α since A_1 never executes). Then, for every $p \in P_1^\alpha$ and every interaction a such that $p \in a$, a is enabled from some point onwards in α.*
- *Vice-versa for A_2.*

Theorem 4 (Pairwise Strongly Noninterfering Live Architectures). *Let live architectures $A_i = (\mathcal{C}_i, P_i, \gamma_i, \mathcal{L}_i)$, for $i \in [1,m]$ be feasible w.r.t. a set of behaviours $\mathcal{B}$. Assume that (a) for all $j \neq k \in [1,m]$, A_j and A_k are strongly non-interfering with respect to $\mathcal{B}$, (b) $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$ is free of local deadlock, and (c) for each $i \in [1,m]$, $\mathcal{L}_i = \{P_i\}$. Then $(\bigoplus_{i=1}^{m} A_i)$ is feasible w.r.t $\mathcal{B}$.*

A consequence of Th. 4 is that weak interaction fairness (an interaction that is enabled from some point onwards is eventually executed) is sufficient for enforcing all the liveness properties of all the architectures.

Example 5 (Strong Noninterference in Mutual exclusion). Consider mutual exclusion such that coordinators also have request actions. A component can always send a request to all of its coordinators. The coordinators are fair and eventually enable continuously the critical section entry for every component that requests entry. The coordinators in this case are pairwise strongly non-interfering.

Checking Interference and Strong Noninterference Noninterference for finite-state systems, can be checked using fair CTL model checking algorithms of Emerson and Lei [7, Theorem 4.6], as they can handle any boolean combination of "infinitely often" and "continuously from some point onwards" operators.

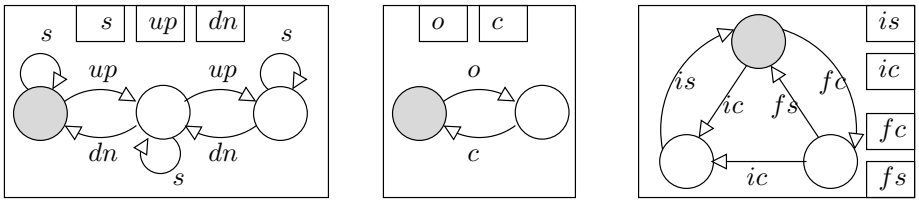
Let $A_j = (\mathcal{B}_j, P_j, \gamma_j)$, for $j = 1,2$ be two architectures. Fig. 2 presents an algorithm that checks, for finite-state systems, whether A_1 and A_2 are strongly non-interfering w.r.t. a set of behaviours $\mathcal{B} = \{B_1, \ldots, B_n\}$. We generate the state transition diagram M of $(A_1 \oplus A_2)(\mathcal{B})$. The idea is that an infinite path has an infinite suffix that lies inside a strongly connected component of M.

```
checkStrongNonIntrf(A1, A2, B1, ..., Bn)
//check that A2 does not interfere with A1
1.  let (A1 ⊕ A2)(B1, ..., Bn) = (Q, q^0, P, →) according to Def. 2, Def. 4, and Def. 6
2.  let →' := → − {(q, a→, q') | (∃p : p ∈ P1 ∩ a)}//remove all transitions involving A1
3.  let Φ be the set of all maximal strongly connected components of →'
4.  forall maximal strongly connected components φ in Φ
5.      let q be an arbitrarily chosen state of φ, and q1 = q ↑ A1
6.      let →A1 be the transition relation of A1 in isolation
7.      Pφ := {p | q1 p→A1}          //Pφ is set of ports that A1 locally enables along φ
        //check all interactions containing p are e.a. enabled in (A1 ⊕ A2)(B)
8.      let →'φ be the transitions in φ
9.      forall p ∈ Pφ
10.         forall interactions a such that p ∈ a
11.             if ¬(∀s ∈ φ : s a→)              //check a enabled in every state of φ
12.                 return(ff)//return false if some a containing p is not e.a. enabled
13.         endfor;
14.     endfor;
15. endfor;
16. return(tt)                                  //return tt if A2 does not interfere with A1
```

Fig. 2: Pseudo-code for checking strong non-interference

Proposition 5. checkStrongNonIntrf$(A_1, A_2, B_1, \ldots, B_n)$ *and* checkStrongNonIntrf$(A_2, A_1, B_1, \ldots, B_n)$ *both return true iff* A_1 *and* A_2 *are strongly non-interfering w.r.t.* $B_1, \ldots, B_n$.

4 Case study: Control of an elevator cabin

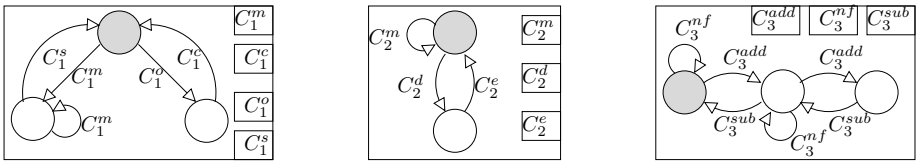


(a) Elevator Engine E (b) Elevator Door D (c) Caller System CS

Fig. 3: Elevator behaviours.

We illustrate our approach with the Elevator case study adapted from the literature [8, 9]. Control of the elevator cabin is modelled as a set of coordinated atomic components shown in Fig. 3. 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 $B = \{E, D, CS_0, CS_1, CS_2\}$ the set of base behaviours.



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

Fig. 4: Coordinating behaviours for the elevator example.

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 B the architecture $A_1 = (\{C_1\}, P_1, \gamma_1)$. C_1 is shown in Fig. 4(a), P_1 contains all ports of C_1,and all ports of base behaviours. γ_1 consists of 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 $A_1(B)$ 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. 4(b), $P_2 = \{C_2^e, C_2^d, C_2^m, C_1^c, C_1^s, C_1^m\}$, and $\gamma_2 = \{C_1^s\, C_2^d, C_1^c\, C_2^e, C_1^m\, C_2^m\}$, which grants priority to the door controller after a C_1^s action. The $A_2(A_1(\mathcal{S}))$ provides the same basic functionality, as A_1 does, but enforce the additional constraint. By Prop. 2, we have $A_2(A_1(\mathcal{S})) = (A_2 \oplus A_1)(\mathcal{S})$.

The feature *"if the elevator is full, it must stop only at floors selected from the cabin and ignore outside calls"* [8, 9], is provided by applying the architecture $A_3 = (\{C_3\}, P_3, \gamma_3)$ with C_3 shown in Fig. 4(c), $P_3 = \{C_3^{add}, C_3^{sub}, C_3^{nf}\ s, fs_i \mid i \in [0, 2]\}$ and $\gamma_3 = \{C_3^{add}, C_3^{sub}\} \cup \{s\, C_3^{nf}\, fs_i \mid i \in [0, 2]\}$.

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 $(A_1 \oplus A_2 \oplus A_3)(\mathcal{S})$, 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. We also verify that A_1, A_2, A_3 are nonintefering w.r.t. $\{E, D, CS\}$, Def. 15, so the elevator is live.

Finally, we consider another feature: *"requests from the second floor have priority over all other requests"* [8, 9]. We compose $A_1 \oplus A_2$ with the architecture $A_4 = (\{C_4\}, P_4, \gamma_4)$ with C_4 shown in Fig. 5; P_4 consisting of ports of C_4, CS_2 and o, s and dn; $\gamma_4 = \{fc_2\, C_4^{req}, ic_2\, C_4^{req}, o\, C_4^{free}, dn\, C_4^{free}, fs_2\, s\, C_4^{finish},\ is_2\, s\, C_4^{finish}\}$.

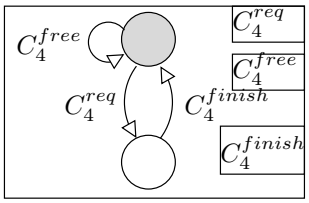


Fig. 5: Executive floor coordinator.

A system $(A_1 \oplus A_2 \oplus A_3 \oplus A_4)(\mathcal{S})$, with contradictory constraints enforced by A_3 and A_4, contains a reachable 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 deadlock state. However, this can be detected using deadlock prevention approach presented in [10].

5 Related Work

There exists an abundant literature on architectures. Most papers propose and study Architecture Description Languages (ADLs) [11]. They focus on technological and syntactic aspects, and mostly disregard semantics and foundational aspects. Nonetheless, 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 [12–14]. 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 [15–23]. 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. [23]. 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 [1]. 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 [24] 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 an invariant and of a liveness property. It considers invariant preservation as an essential condition for architecture composability. Invariants characterize the functionality delivered by the system as safety property. Liveness is a generic property. Its preservation seem 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. Calder, M., Kolberg, M., Magill, E.H., Reiff-Marganiec, S.: Feature interaction: a critical review and considered forecast. Computer Networks **41**(1) (January 2003) 115–141
2. Sifakis, J.: Rigorous System Design. Foundations and Trends in Electronic Design Automation **6**(4) (2012) 293–362
3. Bliudze, S., Sifakis, J.: Synthesizing glue operators from glue constraints for the construction of component-based systems. In Apel, S., Jackson, E., eds.: Software Composition. Volume 6708 of LNCS., Springer Berlin / Heidelberg (2011) 51–67
4. Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. Software, IEEE **28**(3) (2011) 41–48
5. Wegner, P.: Coordination as constrained interaction (extended abstract). In: Coordination Languages and Models. Volume 1061 of LNCS. Springer (1996) 28–33
6. Abadi, M., Lamport, L.: Composing specifications. ACM Trans. Program. Lang. Syst. **15**(1) (January 1993) 73–132
7. Emerson, E.A., Lei, C.L.: Modalities for model checking: Branching time logic strikes back. Sci. Comput. Program. **8**(3) (June 1987) 275–306
8. D'Souza, D., Gopinathan, M.: Conflict-tolerant features. In: CAV. Volume 5123 of LNCS., Springer (2008) 227–239
9. Plath, M., Ryan, M.: Feature integration using a feature construct. Science of Computer Programming **41**(1) (2001) 53–84
10. Attie, P.C., Bensalem, S., Bozga, M., Jaber, M., Sifakis, J., Zaraket, F.A.: An abstract framework for deadlock prevention in BIP. In: FMOODS/FORTE. (2013) 161–177
11. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering **26**(1) (2000) 70–93
12. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A.: Metropolis: an integrated electronic system design environment. IEEE Computer **36**(4) (2003) 45–52
13. Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., Neema, S.: Developing applications using model-driven design environments. IEEE Computer **39**(2) (2006) 33–40

14. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity: The Ptolemy approach. Proceedings of the IEEE **91**(1) (2003) 127–144
15. Arbab, F.: Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science **14**(3) (2004) 329–366
16. Fiadeiro, J.L.: Categories for Software Engineering. Springer-Verlag (April 2004)
17. Ray, A., Cleaveland, R.: Architectural interaction diagrams: AIDs for system modeling. In: ICSE'03: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2003) 396–406
18. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE, IEEE Computer Society (2003) 374–384
19. Bernardo, M., Ciancarini, P., Donatiello, L.: On the formalization of architectural types with process algebras. In: SIGSOFT FSE. (2000) 140–148
20. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theoretical Computer Science **366**(1) (2006) 98–120
21. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall (April 1985)
22. Milner, R.: Communication and Concurrency. Prentice Hall International Series in Computer Science. Prentice Hall (1989)
23. Liu, I., Reineke, J., Lee, E.A.: 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. (2010) 2111–2115
24. Hay, J.D., Atlee, J.M.: Composing features and resolving interactions. SIGSOFT Softw. Eng. Notes **25**(6) (November 2000) 110–119

Appendix: Proofs

6.1 Proofs of results from Sect. 2

Proof (Lem. 1). 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. *Consider a set of behaviours $\mathcal{B}$ and two architectures $A_i = (\mathcal{C}_i, P_i^a, \gamma_i)$, 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 $i = 1, 2$, $\tilde{q}_i q \xrightarrow{a \cap (P_i^a \cup P)} \tilde{q}_i' q'$ 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. $\mathcal{C}_i = \{C_i^a\}$, for $i = 1, 2$.

By Def. 6, $a \cap (P_1^a \cup P_2^a) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lem. 1, $a \cap P_1^a \in \gamma_1$. Hence,

$$\tilde{a} \triangleq a \cap (P_1^a \cup P) = \left(a \cap P_1^a\right) \cup \left(a \cap (P \setminus P_1^a)\right) \in \left(\gamma_1 \parallel 2^{P \setminus P_1^a}\right).$$

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

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

given Since $P_{C_1^a} \subseteq P_1^a$, we have $\tilde{a} \cap P_{C_1^a} = a \cap P_{C_1^a}$. 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. □

Proof (Prop. 1). Follows from the corresponding properties of set union and boolean conjunction. □

Proof (Prop. 2). 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 $\mathcal{C}_1 = \{C_1\}$ and $\mathcal{C}_2 = \{C_2\}$.

By Def. 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 $A_2(A_1(\mathcal{B}))$ iff

1. $q_{C_2} \xrightarrow{a \cap P_{C_2}} q_{C_2}'$ is possible in C_2,
2. $q_{C_1} q_1 \dots q_n \xrightarrow{a \cap P'} q_{C_1}' q_1' \dots q_n'$ is possible in $A_1(\mathcal{B})$ and
3. $a \in \gamma_2 || 2^{P'' \setminus P_a''}$.

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 $i \in [1,n]$, $q_i \xrightarrow{a \cap P_i} q'_i$ is possible in B_i and
6. $a \cap P' \in \gamma_1 || 2^{P' \setminus P'_a}$.

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 $i \in [1,n]$, $q_i \xrightarrow{a \cap P_i} q'_i$ is possible in B_i and
3. $a \in \gamma_{A_1 \oplus A_2} \parallel 2^{P'' \setminus (P'_a \cup P''_a)}$.

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

For $a \subseteq P''$, we have $a \in \gamma_{A_1 \oplus A_2} \parallel 2^{P'' \setminus (P'_a \cup P''_a)}$ iff $a \cap (P'_a \cup P''_a) \in \gamma_{A_1 \oplus A_2}$, i.e. $a \cap (P'_a \cup P''_a) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lem. 1, this is equivalent to $a \cap (P'_a \cup P''_a) \cap P'_a = a \cap P'_a \in \gamma_1$ and $a \cap (P'_a \cup P''_a) \cap P''_a = a \cap P''_a \in \gamma_2$. Since $a \subseteq P''$, we have $a \cap P''_a \in \gamma_2$ iff $a \in \gamma_2 \parallel 2^{P'' \setminus P''_a}$. Finally, since $P'_a \subseteq P'$, we have $a \cap P'_a \in \gamma_1$ iff $a \cap P' \in \gamma_1 \parallel 2^{P' \setminus P'_a}$. □

Proof (Prop. 3). Similarly, as in Prop. 2 the set of states is equal in both composed behaviours, thus we only have to prove the equivalence of transitions set. Without loss of generality, we assume, for $i = 1,2$, $\mathcal{B}_i = \{B_i\}$ and $\mathcal{C}_i = \{C_i\}$.

Let $P = P_{C_1} \cup P_{C_2} \cup P_1 \cup P_2$ be a union of ports from all behaviours including ones from the architectures and let $P' = P_{C_1} \cup P_1 \cup P_2$.

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 $a' \in 2^{P' \setminus P'_a}$. As $P'_a \cap P_2 = \emptyset$, 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.

□

Proof (Prop. 4). Clearly, the state spaces, initial states and interfaces of both composed behaviours coincide. Thus, we only have to show that so do the transition relations. Without loss of generality, we assume $\mathcal{C} = \{B_0\}$. Furthermore, let $P' = \bigcup_{B \in \mathcal{B}1} P_B$ and $P'' = \bigcup_{B \in \mathcal{B}_2} P_B$. (Clearly $P' \cap P'' = \emptyset$.)

By Def. 4, a transition $q_0 q_1 \ldots q_n \xrightarrow{a} q'_0 q'_1 \ldots q'_n$ is possible in $A(\mathcal{B})$ iff

1. for all $i \in [0,n]$, either $a \cap P_i \neq \emptyset$ and $q_i \xrightarrow{a \cap P_i} q'_i$, or $a \cap P_i = \emptyset$ and $q_i = q'_i$;
2. $a = a_\gamma \cup a' \cup a''$, where $a_\gamma \in \gamma$, $a' \subseteq P' \setminus P_a$ and $a'' \subseteq P'' \setminus P_a$.

Similarly, this transition is possible in $A[\mathcal{B}_1](\mathcal{B}_2)$ iff holds the condition 1 above and $a = b \cup b''$, where $b \in \gamma \parallel 2^{P' \setminus P_a}$ (cf. Def. 7) and $b'' \subseteq P'' \setminus P_a$.

It is now sufficient to observe that the two conditions are equivalent, by taking $b = a_\gamma \cup a'$ and $b'' = a''$, to conclude that the transition relations of the two behaviours coincide. □

6.2 Proofs of results from Sect. 3

Proof (Th. 1). Without loss of generality, we can assume that each of the two architectures has only one coordinating behaviour, i.e. $\mathcal{B}_i = \{B_i^a\}$, for $i = 1, 2$.

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. 9, there exist some states $\tilde{q}_1, \tilde{q}_1' \in Q_{B_1^a}$ and $\tilde{q}_2, \tilde{q}_2' \in Q_{B_2^a}$, 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, we have $\tilde{q}_1 q \xrightarrow{\tilde{a}} \tilde{q}_1' q'$ in $A_1(\mathcal{B})$ and, consequently, by Def. 9, $q \xrightarrow{\tilde{a}} q'$ 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. □

Proof (Th. 2). Again, without loss of generality, we can assume that each of the two architectures has only one coordinating behaviour, i.e. $\mathcal{B}_i = \{B_i^a\}$, for $i = 1, 2$.

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 $q^0, \ldots, q^k \in \prod_{i=1}^n Q_i$ and $\tilde{q}_i^0, \ldots, \tilde{q}_i^k \in Q_{B_i^a}$, for $i = 1, 2$.

By Lem. 2, $\tilde{q}_1^0 q^0 \xrightarrow{a_1 \cap (P_1^a \cup P)} \tilde{q}_1^1 q^1 \xrightarrow{a_2 \cap (P_1^a \cup P)} \cdots \xrightarrow{a_k \cap (P_1^a \cup P)} \tilde{q}_1^k q^k$ 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. □

Proof (Th. 3). Let ρ be an arbitrary finite path in $(A_1 \oplus A_2)(\mathcal{B})$. We first establish the following:

There exists a finite extension π of ρ such that:

$$\pi \text{ contains an action from every } L \in \mathcal{L}_1 \cup \mathcal{L}_2\,. \quad (4)$$

We iterate through all the $L \in \mathcal{L}_1 \cup \mathcal{L}_2$ in some arbitrary order. Let α be an infinite extension of ρ. If L is eventually executed along α, then let α^L be the prefix of α up to and including the execution of L. If L is never executed along α, then by Def. 15, L is eventually enabled along α, and so we can again find an extension α^L of α where L is executed.

Repeat this construction for all $L \in \mathcal{L}_1 \cup \mathcal{L}_2$ and let π be the resulting extension of ρ. Hence (4) follows.

We now construct an infinite path α as the sequence $\alpha_0\alpha_1\alpha_2\ldots$ where α_i is an extension of α_{i-1} according to (4), for all $i \geq 1$, and $\alpha_0 = \rho$. Each segment α_i of α contains at least one action from each $L \in \mathcal{L}_1 \cup \mathcal{L}_2$. Hence α contains an infinite number of (occurrences of) actions from each $L \in \mathcal{L}_1 \cup \mathcal{L}_2$. Hence α is a live path. □

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

Definition 19 (Path projection). *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* $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$*, where* $q^0, \ldots, q^k, \ldots \in \prod_{i=1}^{n} Q_i$ *and* $\tilde{q}^0, \ldots, \tilde{q}^k, \ldots \in \prod_{i=1}^{m} \prod_{C \in \mathcal{C}_i} Q_C$*, and* $P_I = (\bigcup_{i=1}^{n} P_i) \cup (\bigcup_{i \in I} \bigcup_{C \in \mathcal{C}_i} P_C)$*. The projection of* α *onto the subsystem* $(\bigoplus_{i \in I} A_i)(\mathcal{B})$*, denoted* $\alpha \restriction I$*, is the path* $\tilde{r}^0 q^0 \xrightarrow{a_1 \cap P} \tilde{r}^1 q^1 \xrightarrow{a_2 \cap P} \cdots \xrightarrow{a_k \cap P} \tilde{r}^k q^k \cdots$*, where* $\tilde{r}^k = \tilde{q}^k \restriction I$ *for all* $k \geq 0$*.*

Proof (Th. 4). Let ρ be an arbitrary finite path in $(\bigoplus_{i \in I} A_i)(\mathcal{B})$. We show that there exists an infinite extension of ρ which is live. Let ρ' be any infinite extension of ρ that satisfies weak interaction fairness (an interaction that is enabled from some point onwards is eventually executed).

Say that an architecture is executed if an interaction in which some coordinator of the architecture participates is executed. Let $\mathcal{A}_{fin}$ be the set of architectures that are executed only finitely often along ρ', and let $\mathcal{A}_{inf}$ be the set of architectures that are executed infinitely often along ρ. Observe that necessarily $\mathcal{A}_{inf} \neq \emptyset$ and suppose that $\mathcal{A}_{fin} \neq \emptyset$ as well.

Let α be an infinite suffix of ρ' such that no architecture in $\mathcal{A}_{fin}$ is executed in α. Hence the local state of every coordinator in $\mathcal{C}_k$, for every architecture $A_k \in \mathcal{A}_{fin}$, does not change along α.

Let P_k^{α} denote the set of ports that are locally enabled by the coordinators in $\mathcal{C}_k$, in every state along α.

Consider arbitrary $A_k \in \mathcal{A}_{fin}$, $A_\ell \in \mathcal{A}_{inf}$ and the projection $\alpha_{k\ell} = \alpha \restriction \{k, \ell\}$. The path $\alpha_{k\ell}$ is infinite, since $A_\ell \in \mathcal{A}_{inf}$. Since A_k, A_ℓ are strongly noninterfering, $(A_k \oplus A_\ell)(\mathcal{B})$ satisfies:

$$\begin{aligned} &\textit{for every } p \in P_k^{\alpha}\textit{, and every interaction } a_{k\ell}^p \textit{ of } (A_k \oplus A_\ell)(\mathcal{B}) \textit{ such that } p \in a_{k\ell}^p\textit{:} \\ &\qquad a_{k\ell}^p \textit{ is enabled by } (A_k \oplus A_\ell)(\mathcal{B}) \textit{ from some point onwards in } \alpha_{k\ell}\,. \end{aligned}$$

Now let $\pi_{k\ell}$ be the infinite suffix of $\alpha_{k\ell}$ in which $a_{k\ell}^p$ is continuously enabled. So we have

$$\begin{aligned} &\textit{for every } p \in P_k^{\alpha}\textit{, and every interaction } a_{k\ell}^p \textit{ of } (A_k \oplus A_\ell)(\mathcal{B}) \textit{ such that } p \in a_{k\ell}^p\textit{:} \\ &\qquad a_{k\ell}^p \textit{ is continuously enabled by } (A_k \oplus A_\ell)(\mathcal{B}) \textit{ in } \pi_{k\ell}\,. \quad (5) \end{aligned}$$

Recall that A_k, A_ℓ were arbitrarily chosen, and so (5) holds for all $A_k \in \mathcal{A}_{fin}$ and all $A_\ell \in \mathcal{A}_{inf}$.

Now consider the subsystem $(\bigoplus \mathcal{A}_{fin})(\mathcal{B})$ consisting of the composition of the architectures in $\mathcal{A}_{fin}$ applied to $\mathcal{B}$. Choose q to be a state on α such that, for all k, ℓ, the projection $q_{k\ell}$ of q onto $(A_k \oplus A_\ell)(\mathcal{B})$ lies on $\pi_{k\ell}$. Let q_{fin} be the projection of q onto $(\bigoplus \mathcal{A}_{fin})(\mathcal{B})$. By the assumption (b) of the lemma, this subsystem, considered in isolation, has some interaction a_{fin} enabled in state q_{fin}. Let p be an arbitrary port in the intersection of a_{fin} and the ports of $\mathcal{A}_{fin}$, i.e. $p \in a_{fin} \cap \bigcup_{k \in fin} P_k$.

By Def. 4 and Def. 6, there exists some interaction a of the overall system $(\bigoplus_{i=1}^{m} A_i)(\mathcal{B})$, such that $a_{fin} \subseteq a$. Moreover, a is the result of the "fusion" of a_{fin} and a set $\{a_{k\ell} \mid (k, \ell) \in R\}$ of interactions such that each $a_{k\ell}$ contains a port

from A_k and a port from A_ℓ, and possibly ports from some of the behaviours in $\mathcal{B}$. Also, $R \subseteq \mathit{fin} \times \mathit{inf}$, i.e. R is a subset of the pairs that can be formed by taking one index from *fin* and one index from *inf*. By Def. 4 and Def. 6, it follows that $\{a_{k\ell} \mid (k,\ell) \in R\}$ can be chosen so that $a_{k\ell}$ is an interaction in $(A_k \oplus A_\ell)(B_1, \ldots, B_n)$.

From (5), we have that each $a_{k\ell}$ is continuously enabled by $(A_k \oplus A_\ell)(\mathcal{B})$ in $\pi_{k\ell}$. Hence each $a_{k\ell}$ is enabled by $(A_k \oplus A_\ell)(\mathcal{B})$ in $q_{k\ell}$. By construction, a_{fin} is enabled by $(\mathcal{A}_{\mathit{fin}})(\mathcal{B})$ in q_{fin}. Since a is the result of fusion of a_{fin} and the $a^p_{k\ell}$'s, and these "component" interactions are all enabled by their respective subsystems in the projections of state q onto these subsystems, we conclude by Def. 2 that a is enabled by the overall system $(\bigoplus_{i=1}^m A_i)(\mathcal{B})$ in state q. Furthermore, by (5), the $a^p_{k\ell}$ are continuously enabled from $q_{k\ell}$ onwards. This means that no transition by any B_i or any $A_\ell \in \mathcal{A}_{inf}$ after q can disable a. The $A_k \in \mathcal{A}_{\mathit{fin}}$ do not execute after state q, by construction. Hence, a is enabled from state q onwards. By weak interaction fairness, a is executed along α, and so some $A_k \in A_{\mathit{fin}}$ is executed along α. But this contradicts the construction of A_{fin}. We conclude that A_{fin} must be empty, and so every architechture is executed infinitely often along α. Hence α is a live extension of ρ. □

Proof (Prop. 5). Proof by double implication.

Left to right: Suppose $\mathsf{checkStrongNonIntrf}(A_1, A_2, B_1, \ldots, B_n)$ returns true. Then, in every strongly connected component of M', every port that A_1 locally enables also has an enabled interaction. Let α be an infinite path fragment along which A_1 does not execute. So, α, with the exception of a finite prefix, must lie entirely inside a single strongly conected component CC of M'. Hence, every port that A_1 locally enables along α also has an enabled interaction along an infinte suffix of α. Hence A_2 does not interfere with A_1. Likewise, if $\mathsf{checkStrongNonIntrf}(A_2, A_1, B_1, \ldots, B_n)$ returns true, then A_1 does not interfere with A_2.

Right to left: Suppose A_2 does not interfere with A_1. Let CC be an arbitrary strongly connected component of M', and let p be a port that A_1 locally enables in every state of CC. There exists an infinite path fragment α that contains an infinite numebr of occurrences of each state of CC. Hence there eixtss an interaction a with $p \in a$ that is enabled in every state of some infinite suffix of α. Hence a is enabled in every state of CC. Hence $\mathsf{checkStrongNonIntrf}(A_1, A_2, B_1, \ldots, B_n)$ returns true. Likewise, if A_1 does not interfere with A_2 then $\mathsf{checkStrongNonIntrf}(A_2, A_1, B_1, \ldots, B_n)$ returns true. □