A Semantic Framework for Architecture Modelling

THÈSE N^{O} 7325 (2017)

PRÉSENTÉE LE 20 JANVIER 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE POUR LA CONCEPTION RIGOUREUSE DES SYSTÈMES
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Eduard BARANOV

acceptée sur proposition du jury:

Prof. J.-Y. Le Boudec, président du jury
Prof. J. Sifakis, Dr S. Bliudze, directeurs de thèse
Prof. R. De Nicola, rapporteur
Prof. F. Arbab, rapporteur
Prof. R. Guerraoui, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

Acknowledgements

This thesis is the result of an interesting and challenging part of my life. It would not be possible without help of many people. I would like to use this opportunity to express my gratitude to them.
I would first like to thank my advisor, Prof. Joseph Sifakis, for giving me an opportunity to pursue a PhD in his lab and to explore interesting problems in system design. He provided his guidance and expertise throughout my work. During the PhD I was always inspired by his vision and motivation. I am very grateful to my co-advisor, Dr. Simon Bliudze, for his constant support. He always had time for answering questions and for exchanging ideas. His help with paper writing was invaluable. I'm also grateful for his support in my search of future position. I was extremely lucky to have such mentors.
I would like to thank Prof. Farhad Arbab, Prof. Rocco De Nicola and Prof. Rachid Guerraoui for agreeing to be the jury members of my PhD defence. They generously provided their time to read this work and gave invaluable feedback. I would also like to thank Prof. Jean-Yves Le Boudec who served as a jury president.
I would like to thank Anastasia Mavridou for our collaboration on several papers resulted in a very nice theory that served as a basis for the last part of the thesis. I would like to thank other members of RISD lab Alina Zolotukhina, Stefanos Skalistis, Wang Qiang, Wajeb Saab and Alena Simalatsar for their help during these years, especially for their feedback on presentation dry-runs. They made my time in the lab a great experience. I would also like to thank Ariane Staudenmann for her organisational support. Additional thanks for the office with the view on the Geneva lake and Alps.
Last, but not least, I'm very grateful to my parents for their unconditional love, belief and support throughout my entire life. Thank you very much!

Abstract

Architectures are common means for organising coordination between components in order to build complex systems and to make them manageable. They allow thinking on a higher plane and avoiding low-level mistakes. Architectures provide means for ensuring correctness-by-construction by enforcing global properties characterising the coordination between components. In this work, we consider the following questions of architecture modelling: 1) how to model architectures; 2) how to compose them if several properties enforced by different architectures are required; 3) how to specify architectures styles that generalise the notion of architectures and represent families of architectures satisfying the same property. An architecture can be considered as an operator that, applied to a set of components, builds a composite component meeting a characteristic property. The underlying concepts of components and their interaction originate from the BIP framework.

This thesis is structured in two parts. In the first part, we study the expressiveness of glue operators in the BIP framework. We provide results for classical BIP glue and for several modifications obtained by relaxing the constraints imposed on priority models. We also study an alternative semantics of BIP glue based on the offer predicate. It meets fundamental properties required from component-based frameworks, namely *compositionality*, *incrementality*, *flattening* and *modularity*. We provide the comparison with the classical BIP semantics and the algorithm for the synthesis of connectors from the interaction logic used to describe coordination constraints.

In the second part, we define architectures and propose an architecture composition operator. We study their properties and prove that the composition operator preserves safety properties of its operands. The alternative glue semantics presented in the first part of the thesis allows to extend architectures with priorities. For the specification of architecture styles, we propose configuration logics. We provide a sound and complete axiomatisation of the propositional configuration logic as well as decision procedures for checking that an architecture satisfies a given logical specification. To allow genericity of specifications, we study higher-order extensions of the propositional configuration logic. We illustrate with examples the specification of various architecture styles. We provide an experimental evaluation using the Maude rewriting system to implement the decision procedure for configuration logics. Additionally, we study the relation between the architecture composition operator and the composition of configuration logic formulas.

Keywords: component-based design, correctness-by-construction, architecture, BIP, composability, architecture style.

Résumé

Les architectures sont des moyens communs pour l'organisation de la coordination entre composants en vue de construire des systèmes complexes et de les rendre gérables. Elles permettent de raisonner à un niveau d'abstraction supérieur et d'éviter les erreurs de bas niveau. Les architectures fournissent un moyen pour assurer que les systèmes développés sont corrects par construction en imposant des propriétés globales caractérisant la coordination entre les composants. Dans ce travail, nous considérons les questions suivantes, liées à la modélisation d'architectures : 1) comment modéliser les architectures ; 2) comment les composer si plusieurs propriétés imposées par des architectures différentes sont nécessaires ; 3) comment spécifier des styles architecturaux qui généralisent la notion d'architectures et représentent des familles d'architectures satisfaisant la même propriété. Une architecture peut être considérée comme un opérateur qui, appliqué à un ensemble de composants, construit un composant composite satisfaisant une propriété caractéristique. Les concepts sous-jacents de composants et de leur interaction sont à l'origine du framework BIP.

Cette thèse est structurée en deux parties. Dans la première partie, nous étudions l'expressivité des opérateurs colle dans le framework BIP. Nous fournissons des résultats pour la version classique de colle BIP et pour plusieurs modifications obtenues par l'assouplissement des contraintes imposées sur les modèles de priorité. Nous étudions aussi une sémantique alternative de colle BIP sur la base du prédicat de l'offre. Elle satisfait des propriétés fondamentales requises par les frameworks à base de composants, à savoir *compositionnalité*, *incrémentalité*, *aplatissement* et *modularité*. Nous fournissons une comparaison avec la sémantique BIP classique et l'algorithme de synthèse des connecteurs de la logique d'interaction utilisé pour décrire les contraintes de coordination.

Dans la deuxième partie, nous définissons des architectures et proposons un opérateur de composition d'architectures. Nous étudions ses propriétés et prouvons que l'opérateur de composition conserve les propriétés de sécurité de ses opérandes. La sémantique alternative de colle BIP présentée dans la première partie de la thèse permet d'étendre les architectures aux priorités. Pour la spécification de styles architecturaux, nous proposons des logiques de configuration. Nous fournissons une axiomatisation solide et complète de la logique de configuration propositionnelle ainsi que des procédures de décision pour vérifier qu'une architecture satisfait une spécification logique donnée. Pour permettre la généricité des spécifications, nous étudions des extensions de la logique de configuration propositionnelle à des ordres supérieurs. Nous illustrons la spécification de différents styles architecturaux avec des exemples. Nous fournissons une évaluation expérimentale en utilisant le système de réécriture Maude pour mettre en œuvre la procédure de décision pour les logiques de configuration.

De plus, nous étudions la relation entre l'opérateur de composition d'architectures et la composition des formules logiques de configuration.

Mots-clés : conception à base de composants, développement correct par construction, architecture, BIP, composabilité, style architectural.

Preface

The main results of this thesis were originally presented in the following papers:

- Eduard Baranov, Simon Bliudze. A note on the expressiveness of BIP. In Proc. of Combined 23rd International Workshop on Expressiveness in Concurrency and 13th Workshop on Structural Operational Semantics (EXPRESS/SOS 2016), EPTCS 222, pp. 1-14, 2016.
- Anastasia Mavridou, Eduard Baranov, Simon Bliudze and Joseph Sifakis. Configuration logics: Modelling architecture styles. In Journal of Logical and Algebraic Methods in Programming, 2016.
- Anastasia Mavridou, Eduard Baranov, Simon Bliudze and Joseph Sifakis. Configuration logics: Modelling architecture styles. In Proc. of Formal Aspects of Component Software (FACS 2015), LNCS 9539, pp. 256–274, 2015.
- Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber and Joseph Sifakis. A General Framework for Architecture Composability. In Formal Aspects of Computing, vol. 28, issue 2, pp. 1–25, Springer, 2015.
- Eduard Baranov, Simon Bliudze. Offer semantics: Achieving compositionality, flattening and full expressiveness for the glue operators in BIP. In Science of Computer Programming, vol. 109, pp. 2–35, 2015.
- Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber and Joseph Sifakis. A General Framework for Architecture Composability. In Proc. of the 12th International Conference on Software Engineering and Formal Methods (SEFM 2014), LNCS 8702, pp. 128–143, 2014.
- Eduard Baranov, Simon Bliudze. Extended Connectors: Structuring Glue Operators in BIP. In Proc. of the 6th Interaction and Concurrency Experience (ICE 2013), EPTCS 131, pp. 20–35, 2013.

I was also involved in the work on architecture diagrams presented in

- Anastasia Mavridou, Eduard Baranov, Simon Bliudze and Joseph Sifakis. Architecture diagrams: A graphical language for architecture style specification. In Proc. of the 9th Interaction and Concurrency Experience (ICE 2016), EPTCS 223, pp. 83–97, 2016.

Contents

List of Figures

List of Tables

1 Introduction

Modern software systems are inherently concurrent. They consist of components running simultaneously and communicating with each other through message passing or shared objects. Concurrency is the main cause of the immense complexity of the systems. Analysis of such systems requires considering all possible interleavings of the operations executed by components. Thus, the complexity is exponential in the number of components making the verification infeasible.

An alternative approach is based on building systems that are correct by construction. Rigorous system design [94] achieves correctness by applying a sequence of semantic preserving transformations to obtain an implementation from high-level model preserving the desired properties along the way. Rigorous system design relies on the existence of a unifying *component-based framework* with formal semantics. A component-based framework can be considered as an algebraic structure generated by *components* and a set of coordination mechanisms which we call *glue operators*. Components are abstract building blocks providing basic functionality that can interact with each other. They are characterised by their behaviour and their interface defining interaction points with environment. The notion "component" can cover a very large spectrum, ranging from programs and labelled transition systems, through OSGi bundles and browser plug-ins, to systems of differential equations, etc.

Glue operators specify coordination between components, i.e. how components are allowed to interact. They are usually expressed as a set of connections with or without some coordination components. Connections can represent simple interaction mechanisms like function calls or broadcast as well as more complex ones including protocols and schedulers. They specify two aspects of an interaction: control flow defining synchronisation constraints and data-flow defining how the data is transferred during the interaction. *Glue* is the set of all glue operators in the component-based framework. It is desirable that glue has the following properties:

- *Incrementality.* It allows viewing sub-systems in separation. Given a glue operator for the whole system, incrementality allows extracting operators that coordinate smaller sub-systems.

- *Flattening.* It requires the set of glue operators to be closed under composition and it is complementary to incrementality. Flattening allows replacing a hierarchy of glue operators for different sub-systems with a single operator.
- *Compositionality.* It requires glue operators to preserve component equivalence and allows replacing a component with an equivalent one.

Glue is characterised by its expressive power showing what coordination mechanisms (and possible composed systems) are expressible within the framework.

We distinguish two approaches to system design [24]. In an *architecture-based* approach coordination constraints are described in a purely declarative manner and they can be defined to a large extent independently from components that build the system. This approach requires expressive coordination mechanisms that are capable to describe complex communication protocols. It provides higher abstraction level and consequently stronger correctness-by-construction. In an *architecture-agnostic* approach components are not distinguished from coordination mechanisms. Systems consist of components, some of them provide basic functionality while others ensure coordination. Component interactions are described within its behaviour via function calls, imports etc.. This approach is more error prone, but allows performance optimisations. The distinction between two approaches can be illustrated with Calculus of Communicating Systems (CCS) [81] and Communicating Sequential Processes (CSP) [65]. CCS has a single parallel composition operator, while CSP has a parametrised composition operator defining actions that must synchronise. CCS-like process algebras and most of the programming languages use the architecture-agnostic approach. The architecture-based approach is adopted by various architecture description languages. In [3, 5], authors provide a classification of coordination mechanisms as exogenous or endogenous which is similar to the distinction between architecture-based and architecture-agnostic approaches.

One of the important advantages of the architecture-based approach is the easiness of extraction of reusable solutions. In modern system design approaches, computer systems are never built from scratch. Engineers intensively reuse building blocks and solutions, e.g. libraries, design patterns and communication protocols. Components can specify reusable blocks while *architectures* represent reusable solutions or coordination mechanisms. In rigorous system design an architecture can be considered as an operator that applied to a set of components builds a composite one meeting a characteristic property.

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

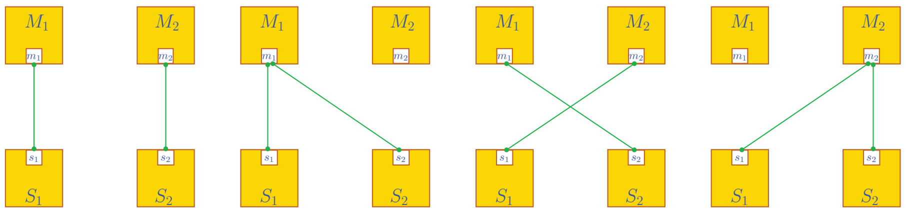


Figure 1.1 – Master/Slave architectures.

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.

Many languages have been proposed for architecture description such as architecture description languages, e.g. [68, 71, 58, 99], coordination languages, e.g. [36, 37, 7], and configuration languages [74, 97, 60]. All these works rely on the distinction between behaviour of individual components and their coordination in the overall system organisation. Informally architectures are characterised by the structure of the interactions between a set of typed components. The structure is usually specified as a relation, e.g. connectors between component ports. Component-based frameworks can be easily adapted to specify architectures.

Nonetheless, the field of software architecture remains relatively immature [57]. A lot of foundational issues remain open. We still lack theory and methods for combining architectures in principled and disciplined fully correct-by-construction design flows.

A theory of architectures must address fundamental questions among which the following are of particular importance:

- How to model architectures? The framework for architecture modelling has to have formal semantics and should be usable with a variety of components.

- How to combine architectures? 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. The composition of solutions has to preserve all characteristic properties of composed architectures. 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 [35].

- How to specify families of architectures? For a single property, there might exist several architectures satisfying it. Consider a simple example: an architecture should be applied to 4 components, two "masters" and two "slaves", and each slave has to interact with one master. Figure 1.1 shows 4 possible architectures: two slaves can interact with the same master or they can interact with two different masters.

A plethora of approaches exists for specification of architectures. For instance, patterns [56, 66] are commonly used for specifying architectures in practical applications. Specifications of architectures are easy to produce, however they lack formal semantics and their meaning may not be clear. The development of a formal framework for architecture modelling requires the rigorous definition of the notion of architectures as well as the rigorous underlying component-based design framework.

1.1 BIP Component Framework

We choose BIP [19] as the underlying theory of components and their interaction. BIP is a component framework rooted in well-defined operational semantics that proposes an expressive and elegant notion of interaction models for component composition. Interaction models can be studied as sets of Boolean constraints expressing interactions between components. BIP has been fully implemented in a language and a supporting toolset, including compilers and code generators [14]. BIP relies on the separation of concerns: components cannot define communication constraints while glue is stateless and cannot perform computations.

Components in BIP are modelled as Labelled Transition Systems (LTS). Transitions are labelled by ports, which are used for synchronisation with other components. *An interface* of the component is the set of its ports. Composition operators defining communication between components are obtained by combining interaction and priority models. Operational semantics of the BIP glue operators is defined by Structural Operational Semantics (SOS) [90] rules. An SOS rule has a form

$$\frac{premises}{conclusion},$$

where "premises" is a set of state predicates on the components composing the system, while "conclusion" is a state predicate on its global state. A rule is interpreted as follows: if all premises are satisfied, then the conclusion is satisfied. The various SOS formats differ in the type of predicates that can be used for either the premises or the conclusion. For the BIP glue, we use a format that is a restriction of GSOS [25], to which we refer as BIP-like SOS.

An *interaction model* γ is a set of *interactions*. Each interaction is a subset of ports of the composed components. A composite component can execute a transition labelled by $a \in \gamma$ iff all components involved in a can execute the corresponding transitions labelled by the actions composing a, whereas other components do not move. Several interaction can be enabled in the same state introducing non-determinism that can be reduced by a *priority model*. A priority model π is a strict partial order on the interaction model. A priority rule $(a, b) \in \pi$ (we write $a \prec b$) forbids the composite component to execute transitions labelled by a from all states where a transition labelled by b is available.

Since interaction models are sets of sets of ports, they can be characterised by a *propositional* *interaction logic (PIL)*—a Boolean algebra on the set of ports of the composed components. The semantics of the interaction logic is defined via satisfaction relation $\overset{i}{\models}$ between interactions and formulas. Each PIL formula represents exactly the set of interactions corresponding to Boolean valuations of ports satisfying the formula. In [22], the authors showed the relation between interaction model and interaction logic and provided a procedure for interaction synthesis from Boolean constraints. Thus, BIP interaction model can be specified in two ways: imperatively with interactions and declaratively with PIL. The former simplifies reasoning about actions the system can perform, while the latter allows reasoning about properties imposed by the glue.

However, PIL does not allow one to encode the priority model. In order to address this issue, a modification of the semantics of BIP glue operators was proposed in [23]. Components are extended with an *offer predicate* defining, for each state, the set of offered ports, i.e. all ports that are used in labels of the outgoing transitions from the state. A priority between interactions $a \prec b$ is defined in this modification as follows: a can be enabled only when at least one port of b is not offered. In order to accommodate the modification of the priority semantics, interactions are extended with two additional parts: an *activation support* and a *negation support* that can reduce the set of states the interaction can be enabled. They define ports that have to be offered or not offered, respectively, in order to enable the interaction. Extended interactions allow to include priority constraints in interaction model through requiring ports of the higher priority interaction to be not offered. Glue operators in this modification consist of a single interaction model. PIL can also be extended allowing to synthesise extended interactions and to define the whole glue including priorities.

1.2 Theory of architectures

We use BIP concepts of components and their interactions for architecture modelling. An architecture A is a solution to a specific coordination problem characterised by a property Φ_A. The application of the architecture $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a set of coordinating components and γ is a configuration—a set of BIP interactions—over a set of ports P_A, to a set of components $\mathcal{B}$ with the corresponding interfaces is the composite component $A(\mathcal{B}) = \gamma(\mathcal{C}, \mathcal{B})$ that satisfies the characteristic property Φ_A. The characteristic property Φ_A assigns the meaning to the architecture that can be informally understood without the need for explicit formalisation (e.g. mutual exclusion, scheduling policy, clock synchronisation).

Communication between components may vary for different properties. 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). Properties could also require state to keep track of previous events. Thus, the restriction of a configuration to be stateless has to be compensated by using the additional set of components $\mathcal{C}$ for coordination.

The PIL representation of configurations allows a simple definition of the composition operator $\oplus$: the composition of two architectures has coordinating components and ports of both operands and its configuration is defined by the conjunction of operands PIL formulas. The composition operator is associative and commutative. Furthermore, it is idempotent on the deterministic components. The composition operator preserves safety properties: for two architectures A_1 and A_2 enforcing respectively safety properties Φ_1 and Φ_2, the architecture $A_1 \oplus A_2$ enforces $\Phi_1 \wedge \Phi_2$.

An architecture implicitly defines the number of components it is applied to. For example, architectures in Figure 1.1 are defined specifically for four components. One way to generalise architectures is to define a *generic architecture* that can be applied to any number of components. This can be done by using *component types* and *higher-order interaction logics* that involve quantification over component variables.

Nevertheless, a property can be enforced by different architectures that are applicable to the same number of components. Figure 1.1 shows four different Master/Slave architectures for two Masters and two Slaves. It is impossible to specify all of them with a single architecture or a single generic architecture.

An *architecture style* is a family of architectures sharing common characteristics such as the type of the involved components and the topology induced by their coordination structure. The relation between architectures and architecture styles is similar to the relation between programs and their specifications. Thus, architectures are specified with configurations while architecture styles with configuration sets. The hierarchy of domains is summarised as follows. Given a set of ports, we consider three different lattices:

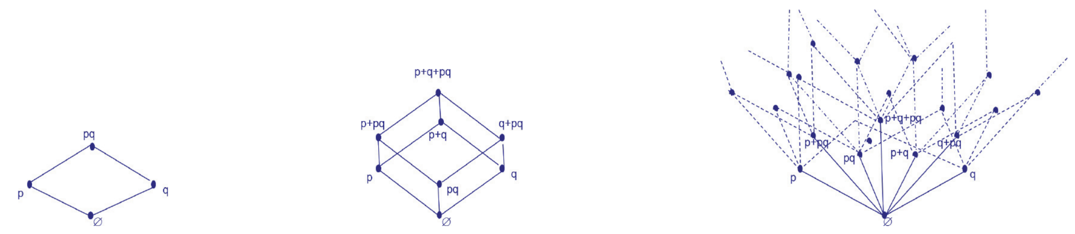
The lattice of interactions. An interaction is a subset of ports of the integrated components. Its execution implies that all components involved in the interaction have to synchronously execute actions associated with the ports of the interaction.

The lattice of configurations. Configurations are sets of interactions defining interaction models of BIP and characterising architectures.

The lattice of configuration sets. Sets of configurations characterise architecture styles.

Figure 1.2 shows the three lattices for $P = \{p, q\}$. For the lattice of configuration sets, we show only how it is generated.

We specify architecture styles with *configuration logic*. Propositional configuration logic (PCL) is a powerset extension of PIL. Its formulas are generated from the formulas of PIL by using the operators union $\sqcup$, intersection $\sqcap$, complementation $\neg$ and *coalescing* $+$. To avoid ambiguity, we refer to the formulas of the configuration logic that syntactically are also formulas of the interaction logics as *interaction formulas*. The semantics of PCL is defined via a satisfaction relation $\models$ between configurations and formulas. An interaction formula f represents any configuration consisting of interactions satisfying it; that is $\gamma \models f$



(a) $I(P) = 2^P$ (b) $C(P) = 2^{I(P)}$ (c) $CS(P) = 2^{C(P)}$

Figure 1.2 – Lattices of interactions (a), configurations (b) and configuration sets (c) for $P = \{p, q\}$.

if, for all $a \in \gamma$, $a \overset{i}{\models} f$. For set-theoretic operators we take the standard meaning. The meaning of formulas of the form $f_1 + f_2$ is all configurations γ that can be decomposed into γ_1 and γ_2 ($\gamma = \gamma_1 \cup \gamma_2$) satisfying, respectively, f_1 and f_2. The formula $f_1 + f_2$ represents configurations obtained as the union of configurations of f_1 with configurations of f_2.

The following simple example illustrates the difference between PIL and PCL. For $P = \{p, q, r, s\}$, the monomial $p \wedge q \wedge \overline{r}$ specifies in interaction logic the interactions $\{p, q\}$ and $\{p, q, s\}$. In configuration logic, it specifies all configurations built from these interactions, i.e. $\{\{p, q\}\}$, $\{\{p, q, s\}\}$ and $\{\{p, q\}, \{p, q, s\}\}$.

Similarly to architectures, PCL formulas are defined for a specific number of components. *Higher-order configuration logics* allow to specify architecture styles for any number of components. First-order logic formulas involve quantification over component variables. We also consider a monadic second-order logic involving quantification over variables for sets of components that allow to express some interesting topological properties, e.g. the existence of cycles of interactions.

1.3 Contributions

This thesis summarises the following contributions:

- In Chapter 3, we study the expressiveness of the classical and the offer semantics of BIP. We show that the BIP glue does not have full expressiveness w.r.t. BIP-like SOS, i.e. there exist glue operators expressible in BIP-like SOS, but inexpressible in the classical BIP. We characterise a subclass of BIP-like SOS glue operators that can be expressed by hierarchical composition of the BIP glue operators. We discuss possible modifications of the priority semantics allowing to recover full expressiveness. We show that, in general, the classical semantics and the offer semantics of BIP are incomparable, i.e. there exist glue operators that are expressible in one semantics and inexpressible in another semantics. Nevertheless, we provide a hierarchy of constraints on components that allows to establish the correspondence between the classical and the offer semantics. In particular, if transition labels of all components consist of a single port (a constraint taken in the BIP implementation [17]), then the offer semantics is strictly more expressive than the classical one. For the offer semantics, in order to support the theory of architectures, we extend representations of the BIP interaction model allowing to define extended interactions with activation and negative support. We extend the synthesis procedure of extended interactions from Boolean constraints.

- In Chapter 4, we define architectures as operators that applied to a set of components build composite ones. An architecture consists of a configuration with a set of *coordinating* components. We define a composition operator and study its properties. We show that hierarchical application of architectures coincide with their composition. In addition, we study partial application of architectures. We show that safety properties defined as a state predicate are preserved by the composition operator. We also show that the offer semantics allows to include priorities in architectures.

- In Chapter 5, we define configuration logics for the specification of architecture styles. We provide a full axiomatisation of the propositional configuration logic and a normal form similar to the disjunctive normal form in Boolean algebras. The existence of such normal form implies the decidability of formula equality and the satisfaction of a formula by an architecture model. To allow genericity of specifications, we study first-order and monadic second-order extensions of the propositional configuration logic. First-order configuration logic formulas involve quantification over component variables. Monadic second-order logic formulas involve additionally quantification over sets of components. It is needed to express some interesting topological properties, e.g. the existence of cycles of interactions. We also study the extension of the first-order logic with ordered components, where we assume that components in models are ordered and use simple constraints based on this order. We illustrate with examples that some interesting styles can be specified in this extension without using second-order logic. The decision procedure for satisfaction of a formula by an architecture model has been implemented in Maude tool for the propositional logic. In addition, we study the relation between the architecture composition operator and conjunction of configuration logic formulas.

2 Background and Related Work

2.1 Properties of Glue

Fundamentally, each component-based design framework can be viewed as a triple $(\mathcal{A}, \sigma, \simeq)$. Here, $\mathcal{A}$ is an algebraic structure generated by a *component type* $\mathcal{B}$ [18] and a set $\mathcal{G} = \bigcup_{n=0}^{\infty}(\mathcal{A}^n \to \mathcal{A})$ of *glue operators* for all arities n:

$$\mathcal{A} ::= B \mid f(C_1, \dots, C_n)\,, \qquad \text{with } B \in \mathcal{B},\ C_1, \dots, C_n \in \mathcal{A} \text{ and } f \in \mathcal{G}. \tag{2.1}$$

We call the elements of $\mathcal{A}$ *systems* and the elements of $\mathcal{B}$ *components*. The structure $\mathcal{A}$ represents the set of all systems constructible within the framework. Component type $\mathcal{B}$ defines the *nature* of the components manipulated by the framework.

The second element of the triple defining a component-based framework is the *semantic mapping* $\sigma : \mathcal{A} \to \mathcal{B}$, which assigns to each system its meaning in terms of the component type $\mathcal{B}$. The semantic mapping must be consistent in the following sense:

$$\textit{for all } B \in \mathcal{B}\,, \qquad \sigma(B) = B\,. \tag{2.2}$$

A trivial consequence of (2.2) is that the application of σ is idempotent, i.e. $\sigma\big(\sigma(C)\big) = \sigma(C)$, for all $C \in \mathcal{A}$. The semantic mapping is called *structural*, if it is defined by associating to each n-ary glue operator $f : \mathcal{A}^n \to \mathcal{A}$ a corresponding operator $\tilde{f} : \mathcal{B}^n \to \mathcal{B}$ and putting

$$\textit{for all } C_1, \dots, C_n \in \mathcal{A} \textit{ and } f \in \mathcal{G}, \quad \sigma\big(f(C_1, \dots, C_n)\big) \overset{def}{=} \tilde{f}\big(\sigma(C_1), \dots, \sigma(C_n)\big)\,. \tag{2.3}$$

Finally, $\simeq\, \subseteq \mathcal{A} \times \mathcal{A}$ is an *equivalence relation*, which allows comparing systems in terms, for example, of their functionality, observable behaviour or capability of interaction with the environment. The equivalence relation must respect the semantics:

$$\textit{for all } C_1, C_2 \in \mathcal{A}\,, \qquad \sigma(C_1) = \sigma(C_2) \implies C_1 \simeq C_2\,. \tag{2.4}$$

Again, a trivial consequence of (2.2) and (2.4) is that a system is always equivalent to its semantics: $C \simeq \sigma(C)$, for all $C \in \mathcal{A}$. In the remainder of the thesis, we assume that (2.2) and (2.4) do hold.

Glue operators used to compose systems in a component-based design framework must possess the following properties [93].

Incrementality
This property represents a generalised form of associativity. It requires that it be possible to view the sub-systems of a system in separation:

$$\begin{aligned}&\textit{for all } i \in [1,n],\ C_1, C_2, \dots, C_n \in \mathcal{A} \textit{ and } f \in \mathcal{G}, \textit{ there exist } g, h \in \mathcal{G} \textit{ such that}\\ &\qquad f(C_1, C_2, \dots, C_n) \simeq g\big(C_i, h(C_1, \dots, C_{i-1}, C_{i+1}, \dots, C_n)\big)\,. \qquad (2.5)\end{aligned}$$

Flattening
This property is complementary to incrementality. It requires that, for any system obtained by hierarchically applying two glue operators to a finite set of sub-systems, there must exist an equivalent system obtained by applying a single glue operator to the *same* sub-systems:

$$\begin{aligned}&\textit{for all } i, j \in [1,n]\ (i \leq j),\ C_1, C_2, \dots, C_n \in \mathcal{A} \textit{ and } f, g \in \mathcal{G}, \textit{ there exists } h \in \mathcal{G}\\ &\qquad \text{such that } f\big(C_1, \dots, C_{i-1}, g(C_i, \dots, C_j), C_{j+1}, \dots, C_n)\big) \simeq h(C_1, \dots, C_n)\,. \qquad (2.6)\end{aligned}$$

In other words, $\mathcal{G}$ must be closed under composition. Flattening enables model transformations, e.g. for optimising code generation or component placement on multicore platforms [26, 30].

Compositionality
This property requires that glue operators preserve the equivalence of their operands:

$$\begin{aligned}&\textit{for all } i \in [1,n],\ C_1, \dots, C_n, C_i' \in \mathcal{A} \textit{ and } f \in \mathcal{G},\\ &\qquad C_i \simeq C_i' \implies f(C_1, \dots, C_i, \dots C_n) \simeq f(C_1, \dots, C_i', \dots C_n)\,. \qquad (2.7)\end{aligned}$$

Another version of this property, which we will call *relaxed compositionality*, only requires that individual glue operators respect component equivalence:

$$\begin{aligned}&\textit{for all } i \in [1,n],\ B_1, \dots, B_n, B_i' \in \mathcal{B} \textit{ and } f \in \mathcal{G},\\ &\qquad B_i \simeq B_i' \implies f(B_1, \dots, B_i, \dots B_n) \simeq f(B_1, \dots, B_i', \dots B_n)\,. \qquad (2.8)\end{aligned}$$

Notice that, combined with flattening, this relaxed notion of compositionality is already quite strong: essentially, compositionality allows replacing sub-systems, whereas relaxed compositionality with flattening allow replacing components in $\mathcal{B}$.

Modularity

By combining the requirement that the equivalence relation $\simeq$ must respect the semantics of the framework (2.4) with compositionality (2.7), we obtain a special case that is important enough to be considered a separate property:

$$\textit{for all } i \in [1,n],\ C_1, \ldots, C_n \in \mathcal{A} \textit{ and } f \in \mathcal{G}, \qquad f(C_1, \ldots, C_i, \ldots, C_n) \simeq f(C_1, \ldots, \sigma(C_i), \ldots, C_n)\,. \quad (2.9)$$

Compositionality and modularity are related to the concepts of encapsulation and information hiding from object-oriented programming. Component-based frameworks provide a disciplined mechanism for restricting access to component's data, exposing only those elements that are explicitly used for communication, e.g. shared memory and buffers used for receiving messages from the environment. However, in order to provide full modularity, designers must have the possibility to bundle several components together with the connecting glue operators into a new component in order to *hide from the user the details of the component implementation.* This achieves two main goals: 1) the use of the component cannot rely on the specifics of its implementation, allowing the component to be replaced with an alternative solution; 2) components can be delivered to the user without disclosing the details of complex solutions constituting intellectual property of the designer.

One can make the following observations about the relations between compositionality and modularity:

1. Compositionality implies modularity and relaxed compositionality.
2. Modularity and relaxed compositionality together imply compositionality:

   Proof. Without loss of generality, let $i = 1$; for all $C_1, \ldots, C_n, C_1' \in \mathcal{A}$ and $f \in \mathcal{G}$, we have $\sigma(C_1) \simeq C_1 \simeq C_1' \simeq \sigma(C_1')$ and, consequently,

   $$f(C_1, \ldots, C_n) \simeq f\big(\sigma(C_1), \ldots, \sigma(C_n)\big) \simeq f\big(\sigma(C_1'), \ldots, \sigma(C_n)\big) \simeq f(C_1', \ldots, C_n)\,.$$

   □

3. If the semantic mapping is structural and its defining operators $\tilde{f}$, for all $f \in \mathcal{G}$, have relaxed compositionality, then the framework has compositionality:

   Proof. Without loss of generality, let $i = 1$; for all $C_1, \ldots, C_n, C_1' \in \mathcal{A}$ and $f \in \mathcal{G}$, we have

$$\begin{aligned} f(C_1,\dots,C_n) &\simeq \sigma(f(C_1,\dots,C_n)) = \tilde{f}(\sigma(C_1),\dots,\sigma(C_n)) \\ &\simeq \tilde{f}(\sigma(C_1'),\dots,\sigma(C_n)) = \sigma(f(C_1',\dots,C_n)) \simeq f(C_1',\dots,C_n)\,. \end{aligned}$$

□

Expressiveness

In [21], a notion of glue expressiveness was proposed. To determine whether one set of glue operators is more expressive than another, authors compared their respective sets of systems composable from the same components. We consider three flavours of expressiveness. Given a set of operators $\mathcal{O} \subseteq \bigcup_{n=0}^{\infty}(\mathcal{B}^n \to \mathcal{B})$, we denote $\mathcal{O}^n = \mathcal{O} \cap (\mathcal{B}^n \to \mathcal{B})$.

Definition 2.1.1. The component-based framework $(\mathcal{A}, \sigma, \simeq)$ has *strong expressiveness*[1] *w.r.t.* $\mathcal{O}$ iff,

for all $B_1,\dots,B_n \in \mathcal{B}$, *for all* $o \in \mathcal{O}^n$, *there exists* $\tilde{o} \in \mathcal{G}$, *such that*

$$\sigma(\tilde{o}(B_1,\dots,B_n)) = o(B_1,\dots,B_n)\,.$$

Definition 2.1.2. The component-based framework $(\mathcal{A}, \sigma, \simeq)$ has *strong full expressiveness w.r.t.* $\mathcal{O}$ iff,

for all $o \in \mathcal{O}^n$, *there exists* $\tilde{o} \in \mathcal{G}$, *such that, for all* $B_1,\dots,B_n \in \mathcal{B}$,

$$\sigma(\tilde{o}(B_1,\dots,B_n)) = o(B_1,\dots,B_n)\,.$$

Definition 2.1.3. The component-based framework $(\mathcal{A}, \sigma, \simeq)$ has *(weak) full expressiveness w.r.t.* $\mathcal{O}$ iff,

for all $o \in \mathcal{O}^n$, *there exists* $\tilde{o} \in \mathcal{G}[Z_1,\dots,Z_n]$, *such that, for all* $B_1,\dots,B_n \in \mathcal{B}$,

$$\sigma(\tilde{o}[B_1/Z_1,\dots,B_n/Z_n]) = o(B_1,\dots,B_n)\,,$$

where $\mathcal{G}[Z_1,\dots,Z_n]$ is the set of expressions on variables $Z_1,\dots,Z_n$ obtained by hierarchically applying the glue operators from $\mathcal{G}$; whereas $\tilde{o}[B_1/Z_1,\dots,B_n/Z_n] \in \mathcal{A}$ is the component obtained by substituting in $\tilde{o}$ the variables Z_i by components B_i, for all $i \in [1,n]$.

One can make the following observations about the relations between expressiveness flavours:

- Strong full expressiveness is the strongest property: it implies both strong expressiveness and weak full expressiveness.
- Weak full expressiveness and flattening together imply strong full expressiveness.

[1] It was called strong expressiveness preorder in [21].

Proof. From weak full expressiveness we have

$$\forall o \in \mathcal{O}^n.\ \exists \tilde{o} \in \mathcal{G}[Z_1,\dots,Z_n].\ \forall B_1,\dots,B_n \in \mathcal{B}.$$
$$\sigma(\tilde{o}[B_1/Z_1,\dots,B_n/Z_n]) = o(B_1,\dots,B_n)\,.$$

Flattening allows to find a glue operator $\tilde{o}_1 \in \mathcal{G}$, such that

$$\tilde{o}_1(B_1,\dots,B_n) \simeq \tilde{o}[B_1/Z_1,\dots,B_n/Z_n].$$

Thus,

$$\forall o \in \mathcal{O}^n.\ \exists \tilde{o}_1 \in \mathcal{G}[Z_1,\dots,Z_n].\ \forall B_1,\dots,B_n \in \mathcal{B}.$$
$$\sigma(\tilde{o}_1(B_1,\dots,B_n)) = \sigma(\tilde{o}[B_1/Z_1,\dots,B_n/Z_n]) = o(B_1,\dots,B_n)\,.$$

□

2.2 Classical BIP

BIP is a component framework for constructing concurrent systems by superposing three layers of modelling: Behaviour, Interaction and Priority. BIP is based on the separation of concerns between coordination and computation, which is essential for component-based design of concurrent systems. This separation allows systems to be built from units processing sequential code insulated from concurrent execution issues. The isolation of coordination mechanisms allows global treatment and analysis. The BIP component framework has been implemented in a language and a tool-set [14].

2.2.1 Classical semantics

In the classical BIP semantics [62, 19], components are modelled by Labelled Transition Systems.

Definition 2.2.1. A component is a *labelled transition system* (LTS) $B = (Q, P, \to)$, where Q is a set of *states*, P is a set of *ports* and $\to\, \subseteq Q \times 2^P \times Q$ is a set of *transitions* labelled by sets of ports, such that only self-loops can be labelled by the empty set of ports, i.e. $(q, \emptyset, q') \in\, \to$ implies $q = q'$. We call P the *interface* of B.

For $q, q' \in Q$ and $a \in 2^P$, we write $q \xrightarrow{a} q'$ iff $(q, a, q') \in\, \to$. A label $a \in 2^P$ is *active* in a state $q \in Q$ (denoted $q \xrightarrow{a}$), iff there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. We abbreviate $q \not\xrightarrow{a} \overset{def}{=} \neg(q \xrightarrow{a})$.

Intuitively, transitions labelled by $\emptyset$ represent idling: a component that remains idle should not change state, hence the restriction to self-loops. Notice that we distinguish idling from unobservable internal transitions, which we do not model explicitly. To model unobservable transitions, one can use a reserved label, e.g. τ or ε, and restrict the ways it can be synchronised with other transitions. This is the approach traditionally taken in the literature [82, 65].

Component equivalence is defined through a bisimulation relation.

Definition 2.2.2. Let $B_1 = (Q_1, P_1, \rightarrow_1)$ and $B_2 = (Q_2, P_2, \rightarrow_2)$ be two components, and let $R \subseteq Q_1 \times Q_2$ be a binary relation.

- R is a *simulation* iff, for all $q_1 R q_2$, $q_1 \xrightarrow{a}_1 q_1'$ implies $q_2 \xrightarrow{a}_2 q_2'$ for some $q_2' \in Q_2$ such that $q_1' R q_2'$.
- R is a *bisimulation* iff both R and R^{-1} are simulations.

Definition 2.2.3. Two components $B_i = (Q_i, P_i, \rightarrow_i)$, for $i = 1, 2$, are *equivalent* if $P_1 = P_2$, and B_1 and B_2 are bisimilar, i.e. there exists a bisimulation relation $R \subseteq Q_1 \times Q_2$ total on both Q_1 and Q_2.

Glue operators are defined using interaction and priority models. Their semantics are given in terms of Structural Operational Semantics (SOS) rules [90] following a certain restricted sub-format of GSOS [25]. The semantics of interaction models is given by rules involving only positive premises, whereas that of priorities introduces additional negative premises. The intuition behind is clear: an enabled interaction can be fired only if all higher-priority interactions are disabled.

Note 2.2.4. In the rest of the thesis, whenever we speak of a set of components $B_i = (Q_i, P_i, \rightarrow_i)$, for $i \in [1, n]$, we assume that all P_i and Q_i are pairwise disjoint, i.e. $i \neq j$ implies $P_i \cap P_j = Q_i \cap Q_j = \emptyset$. We denote $P \overset{def}{=} \bigcup_{i=1}^n P_i$. We will drop the indices on transition relations and denote them by $\rightarrow$, whenever the indices are clear from the context.

Definition 2.2.5. An *interaction* is a set of ports $a \subseteq P$.

Definition 2.2.6. An *configuration* is a set of interactions $\gamma \subseteq 2^P$.

Definition 2.2.7. An *interaction model* is a configuration $\gamma \subseteq 2^P$. The component $\gamma(B_1, \dots, B_n)$ is defined by the LTS $(Q, P, \rightarrow_\gamma)$, with $Q = \prod_{i=1}^n Q_i$ and the transition relation $\rightarrow_\gamma$ inductively defined by the rule

$$\frac{a \in \gamma \quad \left\{q_i \xrightarrow{a \cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\}}{q_1 \dots q_n \xrightarrow{a}_\gamma q_1' \dots q_n'}, \tag{2.10}$$

where $I = \{i \in [1, n] \mid a \cap P_i \neq \emptyset\}$.

Intuitively, this means that an interaction a allowed by the interaction model γ can be fired when all components involved in a are ready to fire the corresponding transitions. All the components that are not involved in a remain in their current state. Notice that, when the interaction model allows idling, i.e. $\emptyset \in \gamma$, the composed component has a self-loop labelled by $\emptyset$ in every state. The fact that components can have idling self-loops does not introduce any ambiguity in the interpretation of (2.10), since, by Definition 2.2.1, $q \xrightarrow{\emptyset} q'$ implies $q = q'$.

Definition 2.2.8. For a component $B = (Q, P, \to)$, a *priority model* is a strict partial order (transitive and irreflexive relation) $\pi \subseteq 2^P \times (2^P \setminus \{\emptyset\})$ (we write $a \prec b$ as a shorthand for $(a, b) \in \pi$). We put $\pi(B) \overset{def}{=} (Q, P, \to_\pi)$, with the transition relation $\to_\pi$ inductively defined by the rule

$$\frac{q \xrightarrow{a} q' \quad \left\{q \not\xrightarrow{b} \;\middle|\; a \prec b\right\}}{q \xrightarrow{a}_\pi q'}\,. \tag{2.11}$$

Intuitively, this means that an interaction can be fired only if no higher-priority interaction is enabled. Notice that we exclude the priority $a \prec \emptyset$. Indeed, if idling is allowed by the interaction model, it will always be possible, effectively suppressing interaction a in all states. If this is the desired outcome, then a should rather be removed from the interaction model. Furthermore, such a priority could induce a kind of “disguised deadlock”, when an interaction is suppressed in favour of doing nothing (cf. also Lemma 2.2.13).

Note 2.2.9. The rules (2.10) and (2.11) defining the semantics of BIP operators require that a partition $\bigcup_{i=1}^n P_i = P$ be defined, but not on the specific components $B_1, \dots, B_n$.

We are now in position to introduce the BIP glue operators.

Definition 2.2.10. An n-ary *BIP glue operator* is a triple $\big((P_i)_{i=1}^n, \gamma, \pi\big)$, where $(P_i)_{i=1}^n$ are disjoint sets of ports and, denoting $P \overset{def}{=} \bigcup_{i=1}^n P_i$, the remaining two elements $\gamma \subseteq 2^P$ and $\pi \subseteq \gamma \times \gamma$ are, respectively, interaction and priority models on P. (In the remainder of the thesis, we omit the sets of ports $(P_i)_{i=1}^n$ when they are clear from the context.)

To simplify the notation, we denote the component obtained by applying the glue operator $\big((P_i)_{i=1}^n, \gamma, \pi\big)$ to sub-components $B_1, \dots, B_n$, by $\pi\gamma(B_1, \dots, B_n)$. Furthermore, when $\pi = \emptyset$, we write directly $\gamma(B_1, \dots, B_n)$, omitting π.

Notice that both interaction and priority models can be neutral. Indeed, a neutral interaction model over the set of ports P is the set 2^P of all possible interactions. A neutral priority model is empty with none of the interactions having higher priority than any other. Thus, both interaction and priority models are also considered as BIP glue operators on their own.

Example 2.2.11. Consider the two components B_1 and B_2 shown in Figure 2.1 (left) with $P_1 = \{p, q\}$ and $P_2 = \{r\}$, and put $\gamma = \{p, q, r, qr\}$ and $\pi = \{q \prec r\}$.[2] The glue operator

[2]To simplify the notation, we use the juxtaposition $\gamma = \{p, q, r, qr\}$ instead of the set notation $\gamma = \big\{\{p\}, \{q\}, \{r\}, \{q, r\}\big\}$ for interactions. Similarly, we directly write $\pi = \{q \prec r\}$ instead of $\pi = \{(q, r)\}$.

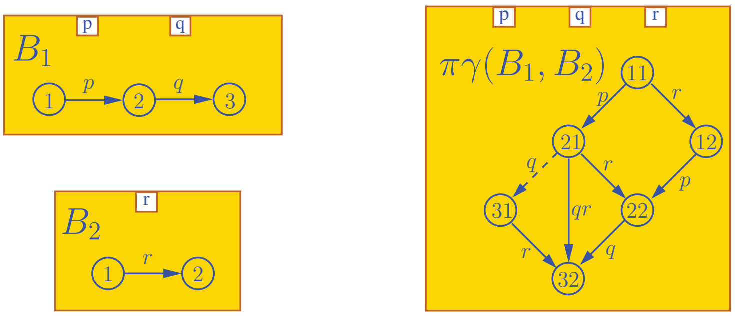


Figure 2.1 – Components for Example 2.2.11.

defined by the combination of the interaction model γ and the priority model π is given by the following four rules:

$$\frac{q_1 \xrightarrow{p} q_1'}{q_1 q_2 \xrightarrow{p} q_1' q_2}, \quad \frac{q_2 \xrightarrow{r} q_2'}{q_1 q_2 \xrightarrow{r} q_1 q_2'}, \quad \frac{q_1 \xrightarrow{q} q_1' \quad q_2 \xrightarrow{r} q_2'}{q_1 q_2 \xrightarrow{qr} q_1' q_2'}, \quad \frac{q_1 \xrightarrow{q} q_1' \quad q_2 \not\xrightarrow{r}}{q_1 q_2 \xrightarrow{q} q_1' q_2}. \tag{2.12}$$

The composed component $\pi\gamma(B_1, B_2)$ is shown in Figure 2.1 (right). The dashed arrow $21 \xrightarrow{q} 31$ shows the transition present only in $\gamma(B_1, B_2)$, but not in $\pi\gamma(B_1, B_2)$. Solid arrows show the transitions of $\pi\gamma(B_1, B_2)$.

Among the transitions labeled by q, only the transition $22 \xrightarrow{q} 32$ is enabled and not $21 \xrightarrow{q} 31$. Indeed, the negative premise in the fourth rule of (2.12), generated by the priority $q \prec r$, suppresses the interaction q when a transition labeled r is possible in the second component. □

It is important to observe that the rules in (2.12) are obtained by composing rules of forms (2.10) and (2.11). In particular, the fourth rule is obtained by the following derivation:

$$\frac{\dfrac{q \in \gamma \quad q_1 \xrightarrow{q} q_1' \quad q_2 = q_2'}{q_1 q_2 \xrightarrow{q}_{\gamma} q_1' q_2'} \qquad \dfrac{r \notin \gamma \quad \vee \quad q_2 \not\xrightarrow{r}}{q_1 q_2 \not\xrightarrow{r}_{\gamma}}\,(*)}{q_1 q_2 \xrightarrow{q}_{\pi} q_1' q_2'}. \tag{2.13}$$

The sub-derivation (*) in (2.13) is obtained by negating the premises of the instance of (2.10) with $a = r$. This is possible because the transition relation in $\gamma(B_1, B_2)$ is defined by (2.10) inductively, i.e. it is the minimal transition relation satisfying (2.10).

In (2.12), we have simplified (2.13) by removing premises, whereof satisfaction does not depend on the state of the operand components: $q \in \gamma$ (satisfied in all states) and $r \notin \gamma$ (dissatisfied in all states), and by replacing q_2' with q_2. Notice that the priority $q \prec r$ affects the behaviour of the composed system only because $r \in \gamma$. Indeed, if r did not belong to γ, the premise $r \notin \gamma$ would always be satisfied independently of the state of the system.

Notice that, after the simplification by removing the constant premises all rules used to define the semantics of BIP glue operators follow the format (a restriction of GSOS [25]):

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_j \overset{b_j^k}{\not\longrightarrow} \,\middle|\, j \in J, k \in K_j\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}, \tag{2.14}$$

where $I = \{i \in [1,n] \,|\, a \cap P_i \neq \emptyset\}$, $J, K_j \subseteq [1,n]$ and, for each $j \in J$ and $k \in K_j$, $b_j^k \in 2^P_j$. We refer to this format (2.14) as *BIP-like SOS*.

Let us now recall an important property of the BIP glue operators with the classical semantics, which was originally shown in [62]: the application of a priority model does not introduce deadlocks.

Definition 2.2.12. For a component $B = (Q, P, \to)$, a state $q \in Q$ is a *deadlock* iff $\forall a \subseteq P$, $q \overset{a}{\not\to}$.

Lemma 2.2.13 ([62]). *Let $B_i = (Q_i, P_i, \to)$, for $i \in [1,n]$, be a set of components, γ and π be respectively interaction and priority models on $P = \bigcup_{i=1}^n P_i$. A state $q \in \prod_{i=1}^n Q_i$ is a deadlock in $\pi\gamma(B_1, \dots, B_n)$ if and only if it is a deadlock in $\gamma(B_1, \dots, B_n)$.*

Proof. The "if" implication is trivial. To prove the "only if" implication, assume that, for some $a \in \gamma$, we have $q \xrightarrow{a}_\gamma$. Let $b \subseteq P$ be an interaction, maximal w.r.t. π, such that $b \in \gamma$, $a \prec b$ and $q \xrightarrow{b}_\gamma$. If such b exists, holds $q \xrightarrow{b}_\pi$. Otherwise holds $q \xrightarrow{a}_\pi$. In both cases, q is not a deadlock in $\pi\gamma(B_1, \dots, B_n)$. □

Notice that this proof does not rely on π being a strict partial order. The lemma can be generalised to any *acyclic* relation $\pi \subseteq \gamma \times \gamma$.

2.2.2 Representations of interaction model

In this subsection, we briefly recall the syntax and the semantics of different representations of the BIP interaction model. All representations are parametrised by a set of ports P. Below, we assume that a set of ports P is given, such that $0, 1 \notin P$.

Algebra of Interactions

The Algebra of Interactions is an auxiliary representation which is used to define the interaction semantics of other algebras. The elements of this algebra can be bijectively mapped to interaction models, i.e. subsets of 2^P.

Syntax. The syntax of the *Algebra of Interactions*, $\mathcal{AI}(P)$, is defined by the following grammar:

$$x \;::=\; 0 \mid 1 \mid p \in P \mid x \cdot x \mid x + x\,, \tag{2.15}$$

where '+' and '·' are binary operators, respectively called *union* and *synchronisation*. Synchronisation binds stronger than union.

As follows from the interaction semantics given below, the additive identity element 0 represents blocking, since it does not authorise any interaction. The multiplicative identity element 1 corresponds to the empty interaction, which represents idling (see the discussion after Definition 2.2.1).

Semantics. The semantics of $\mathcal{AI}(P)$ is given by the function $\|\cdot\| : \mathcal{AI}(P) \to 2^{2^P}$:

$$\begin{aligned}
\|0\| \;=\; \emptyset, \quad \|1\| \;=\; \{\emptyset\}, \quad \|p\| \;&=\; \Big\{\{p\}\Big\}, \\
\|x_1 + x_2\| \;&=\; \|x_1\| \cup \|x_2\|, \\
\|x_1 \cdot x_2\| \;&=\; \Big\{a_1 \cup a_2 \,\Big|\, a_1 \in \|x_1\|, a_2 \in \|x_2\|\Big\},
\end{aligned} \tag{2.16}$$

for $p \in P$, $x_1, x_2 \in \mathcal{AI}(P)$. Terms of $\mathcal{AI}(P)$ represent sets of interactions between the ports P.

The corresponding equivalence relation on $\mathcal{AI}(P)$ is defined as follows: two terms $x, y \in \mathcal{AI}(P)$ are *equivalent* $x \simeq y$ iff $\|x\| = \|y\|$. Sound and complete axiomatisation of $\mathcal{AI}(P)$ with respect to the semantic equivalence is provided in [19]. In a nutshell, $(\mathcal{AI}(P), +, \cdot, 0, 1)$ is a commutative semi-ring idempotent in both $+$ and $\cdot$.

The semantics of the following representations are defined through the Algebra of Interactions: for a representation $\mathcal{A}(P)$, its semantics is defined by the function $|\cdot| : \mathcal{A}(P) \to \mathcal{AI}(P)$. The function $\|\cdot\| : \mathcal{A}(P) \to 2^{2^P}$ is obtained by the composition of $|\cdot| : \mathcal{A}(P) \to \mathcal{AI}(P)$ and $\|\cdot\| : \mathcal{AI}(P) \to 2^{2^P}$. For the following representations, equivalence relations induced by $\|\cdot\|$ and $|\cdot|$ coincide, since the axiomatisation of $\mathcal{AI}(P)$ is sound and complete with respect to $\simeq$.

Algebra of Connectors

The Algebra of Connectors provides an algebraic formalisation for structuring the interaction models. It underlies the graphical notation (e.g. Figure 2.2) and the syntax of the Algebra of Connectors is used in the BIP language.

Syntax. The syntax of the *Algebra of Connectors*, $\mathcal{AC}(P)$, is defined by the following grammar:

$$\begin{aligned}
s \;&::=\; [0] \mid [1] \mid [p] \mid [x] \qquad (\textit{synchrons}) \\
t \;&::=\; [0]' \mid [1]' \mid [p]' \mid [x]' \qquad (\textit{triggers}) \\
x \;&::=\; s \mid t \mid x \cdot x \mid x + x\,,
\end{aligned} \tag{2.17}$$

for $p \in P$, and where '+' is a binary operator called *union*, '·' is a binary operator called *fusion*, and brackets '$[\cdot]$' and '$[\cdot]'$' are unary *typing* operators. Fusion binds stronger than union.

Union has the same meaning as union in $\mathcal{AI}(P)$. Fusion is a generalisation of the synchronisation in $\mathcal{AI}(P)$. Typing is used to form typed connectors: '$[\cdot]$' defines *synchrons* that require synchronisation with other ports in order to interact and '$[\cdot]'$' defines *triggers* that can initiate an interaction.

In order to simplify the notation, we will omit brackets on 0, 1, and ports $p \in P$, as well as '$\cdot$' for the fusion operation.

Definition 2.2.14. In a system with a set of ports P, *connectors* are elements of $\mathcal{AC}(P)$.

The operations of the Algebra of Connectors satisfy the following axioms.

- Union '$+$' is associative, commutative, idempotent and has the identity element 0.
- Fusion '$\cdot$' is associative, commutative and has the identity element 1. It is idempotent on monomial connectors, i.e., for any $x \in \mathcal{AC}(P)$, not involving the union operation, we have $x \cdot x = x$.
- Typing '$[\cdot]^*$' satisfies the following axioms, for $x, y, z \in \mathcal{AC}(P)$ and $[\cdot]^\alpha, [\cdot]^\beta \in \{[\cdot]', [\cdot]\}$ arbitrary typings (trigger or synchron):
  1. $[0] = [0]'$,
  2. $[[x]^\alpha]^\beta = [x]^\beta$,
  3. $[x+y]^\alpha = [x]^\alpha + [y]^\alpha$,
  4. $[x]'[y]' = [x]'[y] + [x][y]'$.

Complete axiomatisation of $\mathcal{AC}(P)$ with respect to the semantic equivalence is provided in [20].

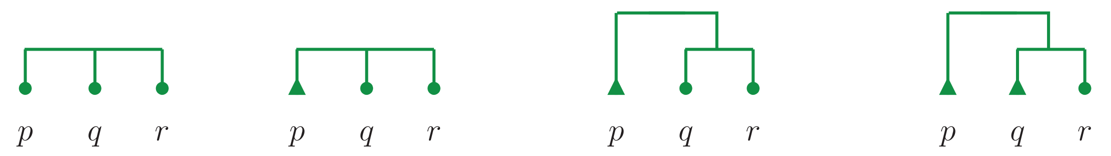
Semantics. The semantics of $\mathcal{AC}(P)$ is given by the function $|\cdot| : \mathcal{AC}(P) \to \mathcal{AI}(P)$ (we use the $\sum$ and $\prod$ notation for the union and fusion of multiple terms of $\mathcal{AC}(P)$):

$$|[p]| = p\,, \qquad |x_1 + x_2| = |x_1| + |x_2|\,, \qquad \left|\prod_{i=1}^{n} [x_i]\right| = \prod_{i=1}^{n} |x_k|\,, \tag{2.18}$$

$$\left|\prod_{i=1}^{n} [x_i]' \prod_{j=1}^{m} [y_j]\right| = \sum_{i=1}^{n} |x_i| \left(\prod_{k \neq i} \Big(1 + |x_k|\Big) \prod_{j=1}^{m} \Big(1 + |y_j|\Big)\right) \tag{2.19}$$

for $n > 0$, $m \geq 0$, $x_1, \dots, x_n, y_1, \dots, y_m \in \mathcal{AC}(P)$ and $p \in P \cup \{0, 1\}$.

Example 2.2.15. Consider a system consisting of three components: a sender with port p and two receivers with ports q and r, respectively. We illustrate with the following examples the specification of various interaction patterns.



(a) Rendezvous pqr (b) Broadcast $p'qr$ (c) Atomic broadcast $p'[qr]$ (d) Causal chain $p'[q'r]$

Figure 2.2 – Basic connector examples.

- *Rendezvous* between the components defines strong synchronisation and it is specified by a single interaction pqr involving all components.
- *Broadcast* defines weak synchronisation among the sender and any number of the receivers: $\{p, pq, pr, pqr\}$.
- *Atomic broadcast* ensures that either all or none of the receivers are involved in the interaction: $\{p, pqr\}$.
- *Causal chain* ensures that second receiver can participate in the interaction only if the first one participates: $\{p, pq, pqr\}$.

Figure 2.2 shows four graphical representations of connectors for the interaction patterns above. Triggers are denoted by triangles, whereas synchrons are denoted by bullets. The syntax of the four connectors is given in the sub-figure captions.

Algebra of Causal Interaction Trees

The Algebra of Causal Interaction Trees serves as pivot for transformations between all other algebraic representations. It makes explicit the causal dependencies between ports contributing to the interactions. In particular, this algebra allows efficient computation of the Boolean representation for connectors and, conversely, the synthesis of connectors from Boolean formulas.

Syntax. The syntax of the *Algebra of Causal Interaction Trees*, $\mathcal{T}(P)$, is given by the following grammar:

$$t ::= a \,|\, a \rightarrow t \,|\, t \oplus t\,, \tag{2.20}$$

where $a \in \mathcal{AI}(P)$ is an interaction, i.e. 0, 1 or a synchronisation of ports (without the use of the union operator), and '$\rightarrow$' and '$\oplus$' are respectively the *causality* and the *parallel composition* operators. Causality binds stronger than parallel composition. Notice that a causal interaction tree can have several roots.

"Atomic" strongly synchronised interactions in the nodes of a causal interaction tree are the building blocks for the interactions provided by the connector. The causality operator defines a dependency between two interactions: $a \rightarrow b$ means that for b to participate in the overall interaction, a must also participate. The parallel composition allows to combine interactions without introducing dependencies: any combination of a and b can participate in $a \oplus b$.

The causality operator is right- (but not left-) associative, for interactions $a_1, \dots, a_n$, we have $a_1 \rightarrow (a_2 \rightarrow (\cdots \rightarrow a_n) \dots)) = a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n$. We call this construction a *causal chain*.

Semantics. The semantics of $\mathcal{T}(P)$ is given by the function $|\cdot| : \mathcal{T}(P) \rightarrow \mathcal{AI}(P)$:

$$|a| = a\,, \qquad |a \rightarrow t| = a\Big(1 + |t|\Big)\,, \qquad |t_1 \oplus t_2| = |t_1| + |t_2| + |t_1|\,|t_2|\,, \tag{2.21}$$

where $a \in 2^P \cup \{0, 1\}$ is an interaction and $t, t_1, t_2 \in \mathcal{T}(P)$.

A sound axiomatisation of $\mathcal{T}(P)$ is provided in [22].

Propositional Interaction Logic

The Propositional Interaction Logic (PIL), studied in [20, 22], is a Boolean algebra used to characterise interactions between components.

Syntax. The propositional interaction logic is defined by the grammar:

$$\phi ::= \mathit{true} \mid p \mid \overline{\phi} \mid \phi \vee \phi\,, \qquad \text{with any } p \in P.$$

Conjunction and implication are defined as usual:

$$\begin{aligned} \phi_1 \wedge \phi_2 &\overset{def}{=} \overline{(\overline{\phi_1} \vee \overline{\phi_2}\,)}\,; \\ \phi_1 \Rightarrow \phi_2 &\overset{def}{=} \overline{\phi_1} \vee \phi_2\,. \end{aligned}$$

To simplify the notation, we omit conjunction in monomials, e.g. writing pqr instead of $p \wedge q \wedge r$.

Semantics. The meaning of a PIL formula ϕ is defined by the following satisfaction relation. For an interaction $a \subseteq P$ we define $a \overset{i}{\models} \phi$ iff ϕ evaluates to *true* for the valuation $p = \mathit{true}$, for all $p \in a$, and $p = \mathit{false}$, for all $p \notin a$. $|\cdot| : \mathit{PIL}(P) \rightarrow \mathcal{AI}(P)$, where $\mathit{PIL}(P)$ is the Boolean algebra over the set of port variables P, is defined by $|\phi| \overset{def}{=} \sum_{a \overset{i}{\models} \phi} a$ the union (in terms of the Algebra of Interactions) of the interactions satisfying ϕ.

The operators meet the usual Boolean axioms. The equivalence $\simeq$ induced by $|\cdot|$ coincide with the Boolean equivalence.

An interaction a can be associated to a *characteristic monomial* $m_a = \bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p}$ such that $a' \overset{i}{\models} m_a$ iff $a' = a$.

Example 2.2.16. The interaction patterns from Example 2.2.15 can be expressed in PIL as follows:

- *Strong synchronisation* is represented by the monomial pqr.
- *Broadcast* is represented by the formula p, which can be expanded to $p\overline{q}\overline{r} \vee pq\overline{r} \vee p\overline{q}r \vee pqr$.
- *Atomic broadcast* can be characterised by the formula $p\overline{q}\,\overline{r} \vee pqr$.
- *Causal chain* can be characterised by the formula $(r \Rightarrow q) \wedge (q \Rightarrow p)$, which can be expanded to $p\overline{q}\,\overline{r} \vee pq\overline{r} \vee pqr$.

Definition 2.2.17. For an interaction model $\gamma \subseteq 2^P$ over a set of ports P, its *characteristic predicate* is a disjunction of characteristic monomials for all interactions in γ:

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

A predicate ϕ uniquely defines an interaction model γ_ϕ such that $\|\phi\| = \gamma_\phi$.

Systems of Causal Rules

Systems of Causal Rules represent an intermediate structure between causal interaction trees and PIL formulas. They directly encode the causality information explicit in the causal interaction trees, by transforming causality relations into dual Horn clauses. Apart from supporting connector synthesis, they provide a convenient way for expressing properties to be enforced by the glue operators. Causal rules have also served as basis for the macro-notation used to specify the glue in Dy-BIP—a dynamic flavour of BIP [29].

Definition 2.2.18. A *causal rule* is a PIL formula of the form $E \Rightarrow C$. The *effect* E is either a constant *true* or a port variable $p \in P$. The *cause* C is either a constant, *true* or *false*, or a disjunction of interactions, i.e. $\bigvee_{i=1}^{n} a_i$ where, for all $i \in [1, n]$, a_i are conjunctions of positive port variables.

Note 2.2.19. Notice that $a_1 \vee a_1\, a_2 = a_1$, and therefore causal rules can be simplified by replacing $p \Rightarrow a_1 \vee a_1\, a_2$ with $p \Rightarrow a_1$. We assume that all causal rules be simplified by this absorption rule.

Definition 2.2.20. A *system of causal rules* is a set $R = \{p \Rightarrow x_p\}_{p \in P \cup \{true\}}$. The meaning of the system of causal rules R coincide with the meaning of the PIL formula obtained by conjunction of causal rules in R, i.e. $|\cdot| : \mathcal{CR}(P) \to \mathcal{AI}(P)$, where $\mathcal{CR}(P)$ is the set of all systems of causal rules over the set of port variables P, is defined by: $|R| = |\phi_R| = \sum_{a \overset{i}{\models} \phi_R} a$, where $\phi_R = \bigwedge_{p \in P \cup \{true\}} (p \Rightarrow x_p)$.

Transformations between representations of interaction models

Transformations $\mathcal{AC}(P) \overset{\tau}{\underset{\sigma}{\rightleftarrows}} \mathcal{T}(P)$, $\mathcal{T}(P) \overset{R}{\rightleftarrows} \mathcal{CR}(P)$ and $\mathcal{CR}(P) \rightleftarrows \mathit{PIL}(P)$ were defined in [22] and have been shown to respect $\simeq$. Below, we briefly recall them.

$\tau : \mathcal{AC}(P) \to \mathcal{T}(P)$ is defined recursively by putting

$$\tau(p) = p\,, \quad \tau\left([x]' \prod_{i=1}^{n} [y_i]\right) = \tau(x) \to \bigoplus_{i=1}^{n} \tau(y_i)\,, \quad \tau\Big([x_1]'[x_2]'\Big) = \tau(x_1) \oplus \tau(x_2)\,,$$
$$\tau\Big([y_1][y_2]\Big) = \bigoplus_{i=1}^{m_1}\bigoplus_{j=1}^{m_2} a_i^1 a_j^2 \to \Big(t_i^1 \oplus t_j^2\Big)\,, \text{ where } \tau(y_k) = \bigoplus_{i=1}^{m_k} a_i^k \to t_i^k, \text{ for } k = 1, 2\,.$$

$\sigma : \mathcal{T}(P) \to \mathcal{AC}(P)$ is defined recursively by putting

$$\sigma(a) = [a]\,, \qquad \sigma(a \to t) = [a]'\,[\sigma(t)]\,, \qquad \sigma(t_1 \oplus t_2) = [\sigma(t_1)]'\,[\sigma(t_2)]'\,. \tag{2.22}$$

$R : \mathcal{T}(P) \to \mathcal{CR}(P)$ is defined by putting

$$R(t) \;=\; \{p \Rightarrow c_p(t)\}_{p \in P \cup \{\mathit{true}\}}\,, \tag{2.23}$$

where the function $c_p : \mathcal{T}(P) \to \mathbb{B}[P]$ is defined recursively as follows. For $a \in 2^P$ (with $p \notin a$) and $t, t_1, t_2 \in \mathcal{T}(P)$, we put

$$\begin{aligned} c_p(0) &= \mathit{false}\,, & c_{\mathit{true}}(0) &= \mathit{false}\,,\\ c_p(p \to t) &= \mathit{true}\,, & c_{\mathit{true}}(1 \to t) &= \mathit{true}\,,\\ c_p(pa \to t) &= a\,, & c_{\mathit{true}}(a \to t) &= a\,,\\ c_p(a \to t) &= a \wedge c_p(t)\,, & &\\ c_p(t_1 \oplus t_2) &= c_p(t_1) \vee c_p(t_2)\,, & c_{\mathit{true}}(t_1 \oplus t_2) &= c_{\mathit{true}}(t_1) \vee c_{\mathit{true}}(t_2)\,. \end{aligned}$$

Observe that this transformation associates to each port $p \in P$ a causal rule $p \Rightarrow C$, where C is the disjunction of all prefixes leading from roots of t to some node containing p, including the ports of this node other than p.

The transformation $\mathcal{CR}(P) \to \mathcal{T}(P)$ consists of two steps. First, we saturate the system of causal rules. Definition 2.2.21, below, defines the notion of a saturated system for causal rules formally. Intuitively, saturation consists in making all causal rules self-contained in terms of information about the constraints imposed by the system.

Definition 2.2.21. A system of causal rules $\{p_i \Rightarrow x_i\}_{i=1}^n$ is *saturated* iff, for all $i \in [1, n]$, $x_i = x_i[x_j/p_j]$, where $x_i[x_j/p_j]$ is obtained by substituting x_j for p_j in x_i, for all $j \neq i$.

For a given system of causal rules $R = \{p_i \Rightarrow x_i\}_{i=1}^n$, we denote by R^S the corresponding saturated system.

In [22] it was shown that R and the corresponding R^S are equivalent.

Given a saturated system of causal rules $R^S = \{p \Rightarrow x_p\}_{p \in P \cup \{true\}}$ with $x_p = \bigvee_{i=1}^{m_p} a_i^p$, we build an auxiliary set

$$Y = \{pa_i^p \,|\, p \in P, i \in [1, m_p]\} \cup \{a_i^{true} \,|\, i \in [1, m_{true}]\} \tag{2.24}$$

by taking all monomials from the causes of the rules conjuncted with the corresponding effects. As shown in [22], this set comprises all "atomic" interactions allowed by the system of causal rules, sets of ports that can only appear together within a valid interaction. An inclusion tree, built from the elements of the set Y, is a corresponding causal tree for the system of causal rules.

The transformation $\mathcal{CR}(P) \to \mathit{PIL}(P)$ is straightforward: for a given system of causal rules $R = \{p_i \Rightarrow x_i\}_{i=1}^n$, the corresponding PIL formula is $\bigwedge_{i=1}^n (p_i \Rightarrow x_i)$.

The transformation $\mathit{PIL}(P) \to \mathcal{CR}(P)$ consist in boolean transformations of the PIL formula to the format similar to causal rules. In order to compute the causal rules for a given PIL formula φ, we take its conjunctive normal form (CNF) $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ with, for $k \in [1, n]$, $C_k = \bigvee_{i \in I_k} p_i \vee \bigvee_{j \in J_k} \overline{p_j}$, where $I_k \cap J_k = \emptyset$, and $p_i, p_j \in P$ for all $i \in I_k$ and $j \in J_k$. Then, we rewrite every clause C_k, with $J_k \neq \emptyset$, as a disjunction of dual Horn clauses $C_k = \bigvee_{j \in J_k} \left(\overline{p_j} \vee \bigvee_{i \in I_k} p_i\right)$. By distributivity, we obtain a representation of φ as a disjunction of dual Horn formulas and, after combining the clauses with the same negative variable, $\varphi = R_1 \vee R_2 \vee \cdots \vee R_m$ with, for $k \in [1, m]$,

$$R_k = \bigwedge_{i \in \widetilde{I}_k} \left(\overline{p_i} \vee \bigvee_{j \in \widetilde{J}_{k,i}} a_j \right) = \bigwedge_{i \in \widetilde{I}_k} \left(p_i \Rightarrow \bigvee_{j \in \widetilde{J}_{k,i}} a_j \right),$$

where, for all $i \in \widetilde{I}_k$, $p_i \in P^t$ and, for all $j \in \widetilde{J}_{k,i}$, a_j is $false$, $true$, or a conjunction of positive variables. For a positive clause $C_k = \bigvee_{i \in I_k} p_i$ ($J_k = \emptyset$), we have $C_k = (true \Rightarrow C_k)$ and $\overline{p} = (p \Rightarrow false)$. Thus, each R_k is a system of causal rules. Notice that this transformation returns a set of systems of causal rules from a single PIL formula.

2.3 Offer Semantics for BIP

In [23], authors proposed a modification of the BIP glue that keeps the information about the active ports of atomic (see Definition 2.3.1 below) components throughout the composition process. In order to have a structural semantics for this modification, the notion of component was enriched.

Definition 2.3.1. An *extended component* is a quadruple $B = (Q, P, \to, \uparrow)$, where $(Q, P, \to)$ is an LTS and $\uparrow$ is an *offer* predicate on $Q \times P$, such that $q{\uparrow}p$ holds (a port $p \in P$ is *offered* in a state $q \in Q$) whenever there is a transition from q containing p, that is $(\exists a \in 2^P : p \in$

$a \wedge q \xrightarrow{a}) \Rightarrow q{\uparrow}p$. If the converse implication also holds, i.e. $(\exists a \in 2^P : p \in a \wedge q \xrightarrow{a}) \iff q{\uparrow}p$, we call the extended component *atomic*.

The offer predicate extends to sets of ports: for $a \in 2^P$, $q \uparrow a \overset{def}{=} \bigwedge_{p \in a} q \uparrow p$. Notice that $q{\uparrow}\emptyset \equiv true$. We denote $q \not\uparrow a \overset{def}{=} \neg(q \uparrow a) = \bigvee_{p \in a} q \not\uparrow p$.

Notice that, for any component, an offer predicate can be defined that makes it atomic [23]. Thus, our notion of atomicity is weaker than the intuitive one. For instance, if a composed component is obtained by putting in parallel two atomic components without any coordination constraints, we consider it as one atomic component. In other words, we use the offer predicate to make explicit part of the information about the transitions of the atomic components that is lost when these are composed by a restrictive operator.

Definition 2.3.2. Two extended components $B_i = (Q_i, P_i, \rightarrow_i, \uparrow_i)$, with $i = 1, 2$, are *equivalent* if $P_1 = P_2$ and there exists a bisimulation relation $R \subseteq Q_1 \times Q_2$, total on both Q_1 and Q_2, such that the offer predicates coincide on bisimilar states, i.e. for all $(q_1, q_2) \in R$ and $p \in P_1$, holds $q_1 \uparrow_1 p \Leftrightarrow q_2 \uparrow_2 p$.

In [23], a more general set of composition operators have been considered. They are defined by the rules in the following format:

$$\frac{\left\{q_i \xrightarrow{a \cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_k \not\uparrow b_k^l \,\middle|\, k \in K, l \in L_k\right\} \quad \left\{q_j \uparrow c_j \,\middle|\, j \in J\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}, \tag{2.25}$$

where $I = \{i \in [1,n] \,|\, a \cap P_i \neq \emptyset\}$, $J, K, L_k \subseteq [1,n]$ and $c_j \in 2_j^P$, $b_k^l \in 2_k^P$, for all $j \in J$, $k \in K$ and $l \in L_k$. In (2.25), there are three types of premises respectively called *firing*, *negative* and *activation* premises. Firing and activation premises are collectively called *positive*. Notice that $q \uparrow c_1 \wedge q \uparrow c_2 = q \uparrow c_1 c_2$. Hence one activation premise per component is sufficient to define any inference rule.

The above set of composition operators can be translated into BIP terms by generalising interaction models. For a set of ports P, we denote $\dot{P} \overset{def}{=} \{\dot{p} \,|\, p \in P\}$ and $\overline{P} \overset{def}{=} \{\overline{p} \,|\, p \in P\}$. We call the elements of P, $\dot{P}$ and $\overline{P}$ respectively *activation*, *firing* and *negative port typings*.

Definition 2.3.3. An *extended interaction* is a subset $a \subseteq P \cup \dot{P} \cup \overline{P}$. An *extended interaction model* is a set of extended interactions $\gamma \subseteq 2^{P \cup \dot{P} \cup \overline{P}}$.

For a given extended interaction a, we define the following sets of ports:

- $\mathbf{act}(a) \overset{def}{=} a \cap P$, the *activation support* of a,
- $\mathbf{fire}(a) \overset{def}{=} \{p \in P \,|\, \dot{p} \in a\}$, the *firing support* of a,

- $\mathbf{neg}(a) \overset{def}{=} \{p \in P \,|\, \overline{p} \in a\}$, the *negative support* of a.

Extended interactions allow us to incorporate priorities into interaction models and, therefore, also extend the theory of algebraic representations of interaction models to encompass priorities.

Definition 2.3.4. Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$[3], with $i \in [1, n]$ and $P = \bigcup_{i=1}^n P_i$, be a set of components. Let $\gamma \subseteq 2^{P \cup \dot{P} \cup \overline{P}}$ be an *extended interaction model.* The composition of $\{B_i\}_{i=1}^n$ with γ is a component $\gamma(B_1, \dots, B_n) \overset{def}{=} (Q, P, \rightarrow_\gamma, \uparrow_\gamma)$ with $Q = \prod_{i=1}^n Q_i$, the offer predicate $\uparrow_\gamma$ defined, for all $p \in P$, by putting $q_1 \dots q_n \uparrow_\gamma p \overset{def}{\Longleftrightarrow} \exists i \in [1, n] : q_i \uparrow p$ and the transition relation $\rightarrow_\gamma$ inductively defined by the rule

$$\frac{\begin{array}{c} a \in \gamma \quad \left\{q_i \xrightarrow{\mathbf{fire}(a) \cap P_i} q_i'\right\}_{i \in I} \quad \left\{q_i = q_i'\right\}_{i \notin I} \\ \left\{q_i \uparrow (\mathbf{act}(a) \cap P_i)\right\}_{i=1}^n \quad \left\{q_i \not\uparrow p \,\middle|\, p \in \mathbf{neg}(a) \cap P_i\right\}_{i=1}^n \end{array}}{q_1 \dots q_n \xrightarrow{\mathbf{fire}(a)}_\gamma q_1' \dots q_n'}, \tag{2.26}$$

where $I = \{i \in [1, n] \,|\, \mathbf{fire}(a) \cap P_i \neq \emptyset\}$.

It is clear, by comparing (2.25) and (2.26), that any BIP glue operator with the offer semantics can be represented by an interaction model with extended interactions.

It is important to observe that, as stated by Lemma 2.3.5 below, sets of interactions can have redundancies.

Lemma 2.3.5. *Let $\gamma_1 \subseteq 2^{P \cup \dot{P} \cup \overline{P}}$ be a set of extended interactions, $\gamma_2 = \gamma_1 \cup \{a\}$, with $a \subseteq P \cup \dot{P} \cup \overline{P}$ such that there exists an interaction $b \in \gamma_1$, $b \subseteq a$ and* $\mathbf{fire}(b) = \mathbf{fire}(a)$. *Then $\gamma_1(B_1, \dots, B_n) = \gamma_2(B_1, \dots, B_n)$.*

Proof. According to rule (2.26) any transition generated by the extended interaction a can also be generated by the extended interaction b. Thus, a does not impact the behaviour of the composed system, and $\gamma_1(B_1, \dots, B_n) = \gamma_2(B_1, \dots, B_n)$. □

Intuitively, this lemma states that any extended interaction allowing the same transition in the composed component as another extended interaction, but under more restrictive conditions, cannot impact the composed system and, therefore, can be removed from the extended interaction model.

An algebra of Boolean formulas $\mathbb{B}[P, \dot{P}]$ over *activation* variables P and the *firing* variables $\dot{P}$ was proposed as a representation of the extended interaction model in [23]. The algebra has the additional axiom:

$$\dot{p} \Rightarrow p, \text{ for all } p \in P. \tag{2.27}$$

[3] As in Remark 2.2.4, we omit the indices on $\uparrow$, whenever they are clear from the context.

Note 2.3.6. A valuation of an activation variable $p \in P$ indicates whether the port p is active, i.e. the corresponding component has an enabled transition containing p in its label, whereas a valuation of a firing variable $\dot{p} \in \dot{P}$ indicates whether the corresponding port p will participate in the next interaction. A formula in $\mathbb{B}[P, \dot{P}]$ defines the constraints on the firing of ports, based on their activation: in a given global state of the system, the valuations of the activation variables are determined by the enabled transitions of the components; a valuation of the firing variables that complements the valuation of the activation ones in such a manner, that the overall valuation satisfies the formula, defines an admissible interaction (for formal presentation, see [23]). Obviously, a port cannot participate in an interaction if it is not active, justifying axiom (2.27).

For an interaction $a = \{\dot{p}_i\}_{i\in I} \cup \{p_j\}_{j\in J} \cup \{\overline{p_k}\}_{k\in K}$, the *c*haracteristic monomial is associated

$$\varphi_a \overset{def}{=} \bigwedge_{i\in I} \dot{p}_i \ \wedge \bigwedge_{i\in P\setminus I} \overline{\dot{p}_i} \ \wedge \bigwedge_{j\in J} p_j \ \wedge \bigwedge_{k\in K} \overline{p_k}\,. \tag{2.28}$$

A formula associated to a glue operator is then the disjunction of formulas associated to interactions defining the extended interaction model.

Notice that the formulas that we obtain in this manner are in *firing-full* Disjunctive Normal Form (DNF), i.e. each firing variable appears in a positive or negative form in each monomial. The firing variables that appear in the negative form are precisely those, for which the respective ports do not appear in the firing premises of the corresponding rule.

In the opposite direction, given a formula $\varphi \in \mathbb{B}[P, \dot{P}]$, we consider its firing-full DNF where each monomial represents an extended interaction.

Definition 2.3.7. Let $\gamma \subseteq 2^{P\cup\dot{P}\cup\overline{P}}$ be an extended interaction model. Its *characteristic predicate* $\varphi_\gamma : \mathbb{B}[P, \dot{P}] \to \mathbb{B}$ is defined by

$$\varphi_\gamma \overset{def}{=} \bigvee_{a\in\gamma} \Big(\bigwedge_{p\in\mathbf{fire}(a)} \dot{p} \ \wedge \bigwedge_{p\notin\mathbf{fire}(a)} \overline{\dot{p}} \ \wedge \bigwedge_{p\in\mathbf{act}(a)} p \ \wedge \bigwedge_{p\in\mathbf{neg}(a)} \overline{p}\Big)$$

2.4 Related Work

2.4.1 Expressiveness of glue

A number of paradigms for unifying component composition have been studied in [11, 12, 52]. These achieve unification by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behaviour. This is also true for the numerous compositional and algebraic frameworks [55, 92, 96, 16, 65, 82, 79]. A comparative survey of various coordination languages has been carried out in [87]. Most of these frameworks are based on a single operator for concurrent composition. This entails poor expressiveness, which results in overly complex architectural designs. In contrast, BIP allows expression of general multiparty interaction and strictly respects separation of concerns. Coordination can be studied as a separate entity that admits a simple Boolean characterisation that is instrumental for expressing composability.

A first framework formally capturing meanings of expressiveness for sequential programming languages and taking into account not only the semantics but also the primitives of languages was provided in [53]. It allows formal reasoning about and distinguishing between *core elements* of a language and *syntactic sugar*. Although a number of studies have taken a similar approach in the context of concurrency, we will only point to [61] and the references therein. The key difference of our approach lies in the strong separation between the computation and coordination aspects of the behaviour of concurrent systems. Indeed, we consider that all sequential computation resides within the components of the system that are not subject to any kind of modification. Thus, we focus on the following question: *what system behaviour can be obtained by coordination of a given set of concurrent components?* In particular, this precludes the expression of parallel composition by choice operators, as in the expansion law [82]. Such notion of expressiveness was proposed in [21].

An extensive overview of SOS formats is provided in [85], including some results comparing their expressiveness. More results comparing different formats of SOS can be found in [84]. The expressiveness property is closely related to the translation between languages. One of the definitions of encoding compared with other approaches can be found in [98]. It should be noted, however, that the above mentioned separation of concerns principle also leads to a very simple rule format. Indeed, the format that we consider is a small subset of GSOS. Our focus is more on the expressiveness of coordination mechanism provided by BIP, than on that of the various SOS rule features.

Though it is not the focus of this thesis, it is also important to mention works comparing BIP with various connector frameworks. A comparative study of three connector frameworks—tile model [32], wire calculus [95] and BIP [15]—was presented in [34]. Recent work [49] relates BIP and Reo. Due to the fact that BIP glue is stateless, it was extended with coordinating components in order to encode Reo connectors, thus relating Reo connectors with BIP architectures. From the operational semantics perspective, all these comparisons only take in account operators with positive premises. In particular, priority in BIP is not considered. It would be interesting to see whether using "local" offer predicate instead of "global" priorities of the classical BIP could help generalising this work.

The approach used in [22] for the Boolean encoding of connectors is close to that used for computing flows in Reo [4] connectors in [43], where it is further extended to data flow. In [70], the authors discuss the extension of the coloring semantics of Reo [41] from the 2-colouring to the 3-colouring model. This extension is necessary to account for context-dependencies. For example, as suggested by its name, a `LossySync` channel can loose data provided on its source end. However, the semantics of Reo channels requires that this happens only if there is no take requests on the sink end of the channel. Context-dependency is encoded by

Table 2.1 – Correspondence between valuations of port variables in BIP and colours in the 3-colouring model of Reo.

p	$\dot{p}$	BIP	Reo
true	*true*	active and firing	data flows
true	*false*	active, but not firing	no flow due to absence of take requests
false	*false*	not active	no flow due to absence of write requests

duplicating all connector nodes: to each *base node* they associate a dual *context node* with complementary flow constraints. This is very similar to our use of firing and negative port typings to encode priority. Furthermore, in [70], one reads: "Whereas 2-coloring models can express synchronisation, they cannot express context-dependency: to model context-sensitive connectors, three colors seem necessary." This observation reflects very closely our use of the additional axiom $\dot{p} \implies p$ in $\mathbb{B}[P, \dot{P}]$. Indeed, this axiom excludes the valuation $\dot{p} = \mathit{true}, p = \mathit{false}$, leaving only three possible valuations of the two variables. It seems that the correspondence between BIP notions of activation and firing and colours in the Reo 3-colouring model can be established as summarised in Table 2.1. This suggests that our notion of activation port typings could also be used in Reo to define nodes that allow the flow of data only if data is available (but will not be consumed) on an additional "control" channel. The authors of [27] model context-dependency with a subclass of guarded automata. Transition labels of these automata consist of two parts: a guard that requires ports to be offered or not offered and a set of firing ports. Two properties required from labels, namely reactivity and uniformity, correspond, respectively, to the axiom $\dot{p} \implies p$ and Lemma 2.3.5.

Connector synthesis for the offer semantics, presented in Section 3.4, is an extension of the procedure in [22]. Other methodologies for synthesis of component coordination have been proposed in the literature, e.g. connector synthesis in [6, 8, 67]. Both approaches are very different from ours. In [6], Reo circuits are generated from constraint automata [10]. This approach is limited, in the first place, by the complexity of building the automaton specification of interactions. An attempt to overcome this limitation is made in [8] by generating constraint automata from UML sequence diagrams. In [67], connectors are synthesised in order to ensure deadlock freedom of systems that follow a very specific architectural style imposing both the interconnection topology and communication primitives (notification and request messages). Our approach, focuses on the properties (expressed as glue constraints) that do not bear computation, which allows us to reduce a very hard and, in general, undecidable problem of synthesising controllers [91] to a tractable one.

Finally, we should mention that the offer predicate used in our formalism has, indeed, some similarity with the concept of barbs [83]. Although, in [83], the barbs do not appear to be used in the premises of the SOS rules defining the semantics of the processes, it would be interesting to further explore this relation.

2.4.2 Architecture modelling

Various architecture description languages have been proposed for architecture modelling, for example [71, 31, 2, 86, 99, 1]. However, some of them do not even have formal semantics and none of them consider a question of architecture composability. Existing research on architecture composability deals mainly with resource composability for particular types of architectures, e.g. [79]. The feature interaction problem is how to rapidly develop and deploy new features without disrupting the functionality of existing features. It can be considered as an architecture composability problem to the extent that features can be modelled as architectural constraints. A survey on feature interaction research is provided in [35]. Existing results focus mainly on modelling aspects and checking feature interaction by using algorithmic verification techniques with well-known complexity limitations. Our work takes a constructive approach. It has some similarities to [63] which presents a formal framework for detecting and avoiding feature interactions by using priorities. Nonetheless, these results do not deal with property preservation through composition. Similarly, existing work on service interaction mainly focuses on modelling and verification aspects, e.g. [47, 77]. In [48], it is shown that under mild assumptions composition of Reo connectors coincides with architecture composition, thus allowing one to extend the results of property preservation to the Reo context.

Among the formal approaches for representing and analysing architecture descriptions, we distinguish two main categories:

- *Extensional approaches*, where one explicitly defines every object that is needed for the specification, i.e. the connections inducing interactions among the components. All connections, other than the ones specified, are excluded. Most architecture description languages, for instance SOFA [71], Wright [2], XCD [86], adopt this approach.

- *Intentional approaches*, where one does not explicitly specify all the connections among the components, but these are derived from a set of logical constraints, formulating the intentions of the designer. Specifications are defined as conjunctions of logical formulas. This approach is taken in [39] and it is often used for characterising architecture styles, for example [58].

The modelling of architectures presented in Chapter 4 allows using both extensional and intentional approaches and configuration logics described in Chapter 5 can even combine them in a single specification of an architecture style.

An architecture style typically specifies a design vocabulary, constraints on how that vocabulary is used and semantic assumptions about that vocabulary [57]. Constraints may be about the allowed interactions between components, e.g. strong synchronisation between components. Semantic assumptions concern the behaviour of the involved components, e.g. loss-less channel, server etc.

A plethora of approaches exist for characterising architecture styles. For instance, patterns are very commonly used for this purpose. Patterns in [46, 66] incorporate explicit constructs for architecture modelling. Nonetheless, they lack formal semantics and they are not amenable to analysis.

Configuration logics presented in Chapter 5 has similarities, but also significant differences, with the use of Alloy [69] and OCL [100] for intentional specification of architecture styles, respectively, in ACME and Darwin [58, 60] and in UML [28]. Our approach achieves a strong semantic integration between architectures and architecture styles. Moreover, configuration logic allows a fine characterisation of the coordination structure by using n-ary connectivity predicates. On the contrary, the connectivity primitives in [58, 60, 100] are binary predicates and cannot tightly characterise coordination structures involving multiparty interaction. To specify an n-ary interaction, these approaches require an additional entity connected by n binary links with the interacting ports. Since the behaviour of such entities is not part of the architecture style, it is impossible to distinguish, e.g., between an n-ary synchronisation and a sequence of n binary ones.

Both Alloy and OCL rely on first-order logics extended with some form of the Kleene closure operator that allows to iterate over a transitive relationship. In particular, this operator allows defining reachability among components. It is known that the addition of the Kleene closure increases the expressive power w.r.t. a first-order logic [72]. To the best of our knowledge, the expressiveness relation between a first-order logic extended with Kleene closure and a corresponding monadic second-order logic remains to be established.

A large body of literature studies transformations or reconfigurations of architectures. Although this work focuses mainly on dynamic reconfiguration of architectures, they can be used to extensionally define architecture styles: a style admits all the configurations that can be obtained by reconfigurations. In [40, 42, 38], the authors propose reconfiguration logic for Reo connectors. Authors of [33, 73, 75] use graph grammars, originating in [64, 76], to define reconfigurations of architectures. The main limitations, outlined already in [76], are the following: 1) the difficulty of understanding the architecture style defined by a grammar; 2) the impossibility of combining several styles in a homogeneous manner; 3) graph grammars are restricted to be context-free, making impossible the specification of certain styles (e.g. trees with unbounded number of components or interactions, square grids). To some extent, the latter two are addressed, respectively, by considering architecture views [88] and synchronised hyperedge replacement [54] or context-sensitive grammars [51, 102].

3 Expressiveness of BIP Glue

BIP glue in the classical semantics does not possess all desired properties defined in Section 2.1. It has incrementality: a composed system $\pi\gamma(B_1, B_2, \dots B_n)$ can be represented as $\pi\gamma(B_1, \pi'\gamma'(B_2, \dots B_n))$, where $\gamma' = \{a \setminus P_1 \mid a \in \gamma\}$, P_1 is the interface of B_1 and π' is empty. The classical BIP semantics defined by (2.10) and (2.11) is structural. Furthermore, since both rule schemata follow the GSOS format, they preserve bisimilarity [25], i.e. the classical BIP semantics has relaxed compositionality and, consequently, compositionality.

Example 3.1.1, below, shows that BIP glue operators in the classical semantics do not possess flattening: in general, when combined hierarchically BIP glue operators in the classical semantics cannot be flattened w.r.t. any bisimilarity-compatible equivalence.

The same Example 3.1.1 shows that BIP glue in the classical semantics does not have strong full expressiveness w.r.t. BIP-like SOS (2.14) either. The hierarchy of BIP glue operators in Example 3.1.1 can be defined by a set of rules in the format (2.14)

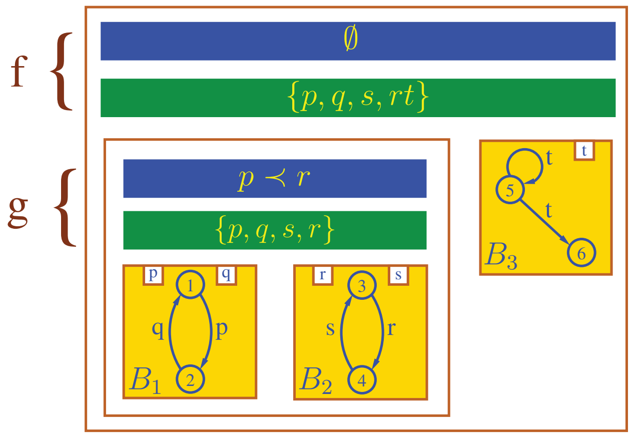
$$\frac{\Big\{q_i \xrightarrow{a\cap P_i} q_i' \,\Big|\, i \in I\Big\} \quad \Big\{q_i = q_i' \,\Big|\, i \notin I\Big\} \quad \Big\{q_j \not\xrightarrow{b_j^k} \,\Big|\, j \in J, k \in K_j\Big\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}\,,$$

but cannot be expressed as a combination of an interaction and a priority models. In Section 3.1, we show that BIP glue in the classical semantics does not possess even weak full expressiveness w.r.t. BIP-like SOS, explain the source of the expressiveness limitation and show relaxations allowing to obtain weak and strong full expressiveness.

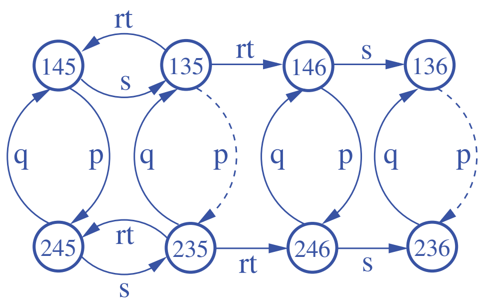
Contrary to the classical semantics, BIP glue in the offer semantics has flattening and strong full expressiveness w.r.t. SOS rules in the format (2.25):

$$\frac{\Big\{q_i \xrightarrow{a\cap P_i} q_i' \,\Big|\, i \in I\Big\} \quad \Big\{q_i = q_i' \,\Big|\, i \notin I\Big\} \quad \Big\{q_k \not\uparrow b_k^l \,\Big|\, k \in K, l \in L_k\Big\} \quad \Big\{q_j \uparrow c_j \,\Big|\, j \in J\Big\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}\,.$$

BIP glue in the offer semantics possess all properties identified in Section 2.1. Notice that



(a) Composed system.



(b) Composed LTS (in dashed, the transitions suppressed by the priority model).

Figure 3.1 – BIP system that cannot be flattened.

the format (2.25) differs from the BIP-like SOS (2.14). Premises of the former require ports to be offered or not offered, while all premises of the latter are defined with interactions. In Section 3.2, we show that BIP glue in the classical semantics and BIP glue in the offer semantics are, in general, incomparable and study the constraints on components allowing the offer semantics to obtain strong expressiveness and strong full expressiveness w.r.t. the classical semantics.

In Section 3.3 and in Section 3.4, we extend interaction model representations and connector synthesis procedure for extended interaction models. This allows to synthesise connectors with priorities from the Boolean constraints.

3.1 Expressiveness of BIP Glue in Classical Semantics

In this section, we consider full expressiveness of the classical semantics of BIP w.r.t. the set $\mathcal{O}$ of operators defined as pairs $\big((P_i)_{i=1}^n, \mathcal{R}\big)$, where n is the arity of the operator, $(P_i)_{i=1}^n$ are pair-wise disjoint sets of ports and $\mathcal{R}$ is a set of BIP-like SOS rules in the format (2.14).

Example 3.1.1. Consider the composed system $f(g(B_1, B_2), B_3)$ (Figure 3.1(a)), with the glue operator g defined by the interaction model $\gamma_1 = \{p, q, r, s\}$ and priority model $\pi_1 = \{p \prec r\}$, and the glue operator f defined by the interaction model $\gamma_2 = \{p, q, s, rt\}$ and the empty priority model. The behaviour of the composite component is shown in Figure 3.1(b) with the transitions, suppressed as the result of applying priority in g, shown as dashed arrows. Composing the rules corresponding to these operators as shown in (2.13), we obtain the four rules

$$\frac{p \in \gamma_1 \cap \gamma_2 \quad q_1 \xrightarrow{p} q_1' \quad (q_2 \not\xrightarrow{r} \vee r \notin \gamma_1)}{q_1q_2q_3 \xrightarrow{p} q_1'q_2q_3}, \qquad \frac{q \in \gamma_1 \cap \gamma_2 \quad q_1 \xrightarrow{q} q_1'}{q_1q_2q_3 \xrightarrow{q} q_1'q_2q_3},$$

$$\frac{s \in \gamma_1 \cap \gamma_2 \quad q_2 \xrightarrow{s} q_2'}{q_1q_2q_3 \xrightarrow{s} q_1q_2'q_3}, \qquad \frac{r \in \gamma_1 \quad rt \in \gamma_2 \quad q_2 \xrightarrow{r} q_2' \quad q_3 \xrightarrow{t} q_3'}{q_1q_2q_3 \xrightarrow{rt} q_1q_2'q_3'}. \tag{3.1}$$

Assume that an interaction model γ and a priority model π are such that $\pi\gamma(B_1, B_2, B_3)$ is equivalent to $f(g(B_1, B_2), B_3)$. By the first rule in (3.1), the transition $14x \xrightarrow{p} 24x$ is possible in $(f \circ g)(B_1, B_2, B_3)$, for any $x \in \{5, 6\}$. Hence, $p \in \gamma$. Clearly, 136 is a deadlock state in $(f \circ g)(B_1, B_2, B_3)$. Hence, 136 must be a deadlock state in $\pi\gamma(B_1, B_2, B_3)$ and, by Lemma 2.2.13, also in $\gamma(B_1, B_2, B_3)$, which is not possible, since all the premises of the rule

$$\frac{p \in \gamma \quad q_1 \xrightarrow{p} q_1'}{q_1q_2q_3 \xrightarrow{p} q_1'q_2q_3},$$

corresponding to p in the semantics (2.10) of γ, are satisfied for $q_1 = 1$ and $q_1' = 2$. Thus, we conclude that, with the classical BIP semantics, there is no glue operator h, such that $f(g(B_1, B_2, B_3)) \simeq h(B_1, B_2, B_3)$, i.e. BIP with the classical semantics does not have the flattening property.

Flattening is not possible due to the fact that the information used by the priority model refers only to interactions authorised by the underlying interaction model. All information about transitions enabled in sub-components is lost (cf. $r \notin \gamma_1$ in the last premise of the first rule in (3.1)).

Simplifying (3.1) by removing the constant premises, we obtain a set of rules in the format (2.14)

$$\frac{q_1 \xrightarrow{p} q_1' \quad q_2 \not\xrightarrow{r}}{q_1q_2q_3 \xrightarrow{p} q_1'q_2q_3}, \quad \frac{q_1 \xrightarrow{q} q_1'}{q_1q_2q_3 \xrightarrow{q} q_1'q_2q_3}, \quad \frac{q_2 \xrightarrow{s} q_2'}{q_1q_2q_3 \xrightarrow{s} q_1q_2'q_3}, \quad \frac{q_2 \xrightarrow{r} q_2' \quad q_3 \xrightarrow{t} q_3'}{q_1q_2q_3 \xrightarrow{rt} q_1q_2'q_3'}, \tag{3.2}$$

defining an operator in $\mathcal{O}$ that cannot be expressed as a BIP glue operator in the classical semantics, which shows that this semantics does not have strong full expressiveness.

Furthermore, the example below shows that the classical semantics of BIP does not have even weak full expressiveness.

Example 3.1.2. Consider a composition operator defined by the following two rules:

$$\frac{q_1 \xrightarrow{p} q_1' \quad q_1 \not\xrightarrow{r}}{q_1 \xrightarrow{p} q_1'}, \qquad \frac{q_1 \xrightarrow{r} q_1' \quad q_1 \not\xrightarrow{p}}{q_1 \xrightarrow{r} q_1'}, \tag{3.3}$$

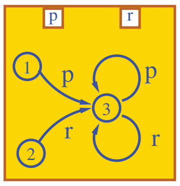


Figure 3.2 – Component for Example 3.1.2.

applied to the component in Figure 3.2. Assume that there exists a hierarchy of BIP glue operators, such that their application to the component in Figure 3.2 results in an equivalent composite component. States 1 and 2 of the composite component have outgoing transitions p and r, respectively, thus all interaction models in the glues have to contain both interactions p and q. State 3 of the composite component is a deadlock. Interaction models do not forbid any transition from this state and priority models cannot introduce deadlock by Lemma 2.2.13. This contradicts the assumption and, consequently, the set of rules (3.3) is not expressible in BIP.

The two fundamental reasons for the lack of expressiveness are related to the definition of the priority model:

- the information used by the priority model refers only to interactions authorised by the underlying interaction model—all information about transitions enabled in sub-components is lost;
- the priority model π must be a strict partial order.

As we explain below, among these two reasons, the first one is easily addressed to achieve weak, rather than strong, full expressiveness, whereas the second one presents the main difficulty.

Characterisation of operators expressible with a hierarchy of BIP glues
Consider an n-ary operator $o : LTS^n \rightarrow LTS$ defined by $(P_i)_{i=1}^n$ and the set of rules

$$\frac{\left\{q_i \xrightarrow{a^l \cap P_i} q_i' \,\middle|\, i \in I^l\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I^l\right\} \quad \left\{q_j \overset{b_{j,k}^l}{\not\longrightarrow} \,\middle|\, j \in J^l, k \in K_j^l\right\}}{q_1 \dots q_n \xrightarrow{a^l} q_1' \dots q_n'}, \qquad \text{for } l \in [1,m], \tag{3.4}$$

where, as above, $I^l = \{i \in [1,n] \,|\, a^l \cap P_i \neq \emptyset\}$. For an interaction $a \in \{a^l \,|\, l \in [1,m]\}$, denote $R_a \overset{def}{=} \{l \in [1,m] \,|\, a = a^l\}$ the set of rules with the conclusion labelled by a. Clearly, for the interaction a to be inhibited by the negative premises, one such premise must be involved for each rule in R_a. We denote by $j : R_a \rightsquigarrow J$ the *choice mappings* $j : R_a \rightarrow \bigcup_{l=1}^m J^l$, such

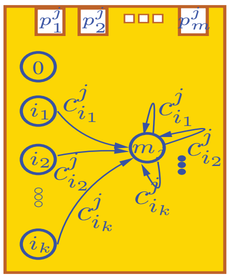


Figure 3.3 – Component for Proposition 3.1.3.

that $j(l) \in J^l$, for all $l \in R_a$.[1]

We define the *inhibiting relation* $\pi \subseteq 2^P \times 2^P$ (where $P = \bigcup_{i=1}^n P_i$) by putting

$$\pi \;=\; \bigcup_{l=1}^{m} \{(a^l, b) \,|\, b = \bigcup_{s \in R_{a^l}} b^s_{j(s),k(s)}, \text{ for some } j : R_{a^l} \rightsquigarrow J,\; k(s) \in K^s_{j(s)}\} \,. \tag{3.5}$$

The following two propositions show that the expressibility of the operator o in BIP depends on the existence of cycles in its inhibiting relation.

Proposition 3.1.3. *If π has cycles, then the operator o cannot be realised by any hierarchical composition of BIP glue operators.*

Proof. Consider a cycle in the inhibiting relation π : $a_1 \prec a_2 \prec \cdots \prec a_l \prec a_1$.

Let $P = \bigcup_{j=1}^n P_j$, where $P_j = \{p_1^j, \dots, p_m^j\}$. Let $c_i^j = a_i \cap P_j$ for $i \in [1,l], j \in [1,n]$ and $C_j = \{c_i^j \,|\, c_i^j \neq \emptyset\}$. For each j consider a component as shown in Figure 3.3. There are no transitions from state 0; from each state i, such that $c_i^j \neq \emptyset$, there is a single transition to state m_j with labels $c_i^j \in C_j$, respectively, and loop transitions in state m_j with labels $c_i^j \in C_j$.

The composition of such components with the operator o allows a single transition a_i from the state $q_1 \dots q_n$, where $q_j = i$ if $c_i^j \neq \emptyset$ or $q_j = 0$ otherwise. In order to allow these transitions, an interaction model of a BIP glue must contain all a_i. In the state $q_1 \dots q_n$, with $q_j = m_j$, all interactions $a_1, \dots, a_l$ are available. The operator o forbids all of them from this state. Interaction models of BIP glues allow all these interactions and priority models cannot introduce deadlock in this state Lemma 2.2.13. Thus, this system is not expressible in BIP. $\square$

Proposition 3.1.4. *If π is acyclic, then the operator o can be realised by a hierarchical composition of BIP glue operators.*

[1]The notion of choice mappings could also be defined as a co-product of mappings $\{l\} \to J^l$ from singleton subsets $\{l\} \subseteq R_a$.

Proof. Since π is acyclic, we can associate a depth $d(a)$ to each interaction a involved in π as the length of the longest path leading to a in the directed acyclic graph defined by π. Denote $d \stackrel{def}{=} \max_a d(a)$. Furthermore, for $i \in [1, d]$, denote $\pi_i \stackrel{def}{=} \{(a, b) \in \pi \,|\, d(a) = i - 1\}$.

Clearly all π_i are strict partial orders. Furthermore, $\pi_i \subseteq \pi \subseteq \gamma_1 \times \gamma_1$, for all $i \in [1, d]$, and

$$\begin{aligned}\gamma_1 &= \gamma_2 \cup \bigcup_{l=1}^{m} \Big\{ \bigcup_{s \in R_{a^l}} b^s_{j(s),k(s)} \,\Big|\, j : R_{a^l} \rightsquigarrow J,\ k(s) \in K^s_{j(s)} \Big\} ,\\ \gamma_2 &= \{a^l \,|\, l \in [1, m]\} .\end{aligned}$$

Hence, for all $i \in [1, d]$, (γ_1, π_i) is a BIP glue operator.

The operator o is equivalent to the composition $(\gamma_2, \emptyset) \circ (\gamma_1, \pi_d) \circ \cdots \circ (\gamma_1, \pi_1)$. We show that for any set of components $B_i = (Q_i, P_i, \to)$, with $i \in [1, n]$, holds

$$\sigma\big(\gamma_2\big(\pi_d\gamma_1(\ldots \pi_1\gamma_1(B_1, \ldots, B_n)) \ldots\big)\big) = o(B_1, \ldots, B_n).$$

We denote

$$B_o = o(B_1, \ldots, B_n) , \qquad B_{\pi\gamma} = \sigma\big(\gamma_2\big(\pi_d\gamma_1(\ldots \pi_1\gamma_1(B_1, \ldots, B_n)) \ldots\big)\big) .$$

The sets of states and ports of these components are the same, thus we only need to check that their transitions coincide.

Let $q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'$ in B_o. This means that, among the rules defining o, i.e. for some $l \in [1, m]$, there is a rule

$$\frac{\Big\{q_i \xrightarrow{a \cap P_i} q_i' \,\Big|\, i \in I^l\Big\} \quad \Big\{q_i = q_i' \,\Big|\, i \not\in I^l\Big\} \quad \Big\{q_j \not\xrightarrow{b^l_{j,k}} \,\Big|\, j \in J^l, k \in K^l_j\Big\}}{q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'} , \tag{3.6}$$

such that $q_i \xrightarrow{a \cap P_i}$, for all $i \in I$, and $q_j \not\xrightarrow{b^l_{j,k}}$ for all $j \in J^l, k \in K^l_j$. By construction both γ_1 and γ_2 contain a. Hence, a is enabled in the state $q_1 \ldots q_n$ of $\gamma_1(B_1, \ldots, B_n)$ and in the same state of $B_{\pi\gamma}$, provided that it is not disabled by any of priorities $\pi_1, \ldots, \pi_d$. Thus, we have to show that no interaction available from this state has higher priority. By construction, priority rules that contain a in the left-hand side can appear only in $\pi_{d(a)-1}$, thus other priority models cannot block a. Priority rules of the form $a \prec b$ have $b = \bigcup_{s \in R_a} b^s_{j(s),k(s)}$, for some $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Since all the premises of (3.6) are satisfied in $q_1 \ldots q_n$, interaction $b^l_{j(l),k(l)}$ is disabled. Hence, b is also disabled. Thus $q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'$ in $B_{\pi\gamma}$.

Let $q_1 \ldots q_n \xrightarrow{a} q_1' \ldots q_n'$ in $B_{\pi\gamma}$. This means that both γ_1 and γ_2 contain the interaction a.

Therefore, by the construction of γ_2, there is at least one rule

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_j \overset{b_{j,k}}{\not\longrightarrow} \,\middle|\, j \in J, k \in K_j\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}, \tag{3.7}$$

among the rules defining o. Furthermore, the priority model $\pi_{d(a)-1}$ contains priorities of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for all $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Notice that a priority rule $b \prec c$, such that $a \prec b$, cannot appear in priorities $\pi_1, \dots, \pi_{d(a)-1}$ since $d(b) \geq d(a) + 1$. Assume that none of the rules defining o, with the conclusion labelled by a, applies in $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$. This necessarily means that each of these rules has a negative premise that is not satisfied. Let $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$ with $b^s_{j(s),k(s)}$, for all $s \in R_a$, being the labels of dissatisfied premises. Then b is an enabled interaction in $\gamma_1(B_1, \dots, B_n)$, such that $a \prec b$ and b cannot be blocked by priorities $\pi_1, \dots, \pi_{d(a)-1}$. Consequently, b is enabled in $\pi_{d(a)-1}\gamma_1\big(\dots\pi_1\gamma_1(B_1, \dots, B_n)\dots\big)$ and blocks a, which contradicts the assumption $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in $B_{\pi\gamma}$. Hence, there is at least one rule of the form (3.7) in the definition of o with all premises satisfied in $q_1 \dots q_n$ and, therefore, $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in B_o. $\square$

Thus, we conclude that BIP has weak full expressiveness w.r.t. the class of BIP-like SOS operators with acyclic inhibiting relations.

Expressiveness of BIP with relaxed priority model

In [13], the following notion of relaxed priority model have been proposed.

Definition 3.1.5. Let P be a set of ports. A *relaxed priority model* on P is a relation $\pi \subseteq 2^P \times (2^P \setminus \{\emptyset\})$. A *relaxed BIP operator* is a triple $\big((P_i)_{i=1}^n, \gamma, \pi\big)$, with $P = \bigcup_{i=1}^n P_i$, such that $\gamma \subseteq 2^P \setminus \{\emptyset\}$ is an interaction model and $\pi \subseteq \gamma \times \gamma$ is a relaxed priority model.

The semantics of relaxed priority models is defined exactly as that of classical priority models, by (2.11). Notice that we do not require the relation π to be acyclic. If all interactions involved in a cyclic dependency in π are enabled simultaneously, they block each other, potentially introducing a deadlock.

Given a BIP-like SOS operator o, we consider its inhibiting relation π (see (3.5)) and the interaction models γ_1, γ_2 as in the proof of Proposition 3.1.4. Since $\pi \subseteq \gamma_1 \times \gamma_1$, the operator (γ_1, π) is a relaxed BIP operator. The operator o is then equivalent to the composition $(\gamma_2, \emptyset) \circ (\gamma_1, \pi)$, where π is considered as a relaxed priority model.

Proposition 3.1.6. *For any set of components $B_i = (Q_i, P_i, \to)$, with $i \in [1, n]$, holds*

$$\sigma\big(\gamma_2(\pi\gamma_1(B_1, \dots, B_n))\big) = o(B_1, \dots, B_n)\,.$$

Proof. For a set of components $B_i = (Q_i, P_i, \to)$, with $i \in [1,n]$, denote

$$B_o = o(B_1, \dots, B_n)\,, \qquad\qquad B_{\pi\gamma} = \sigma\big(\gamma_2\big(\pi\gamma_1(B_1, \dots, B_n)\big)\big)\,.$$

The sets of states and ports of these components are the same, thus we only need to check that their transitions coincide.

Let $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in B_o. This means that, among the rules defining o, i.e. for some $l \in [1,m]$, there is a rule

$$\frac{\Big\{q_i \xrightarrow{a\cap P_i} q_i' \,\Big|\, i \in I^l\Big\} \quad \Big\{q_i = q_i' \,\Big|\, i \notin I^l\Big\} \quad \Big\{q_j \not\xrightarrow{b^l_{j,k}} \,\Big|\, j \in J^l, k \in K^l_j\Big\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}\,, \tag{3.8}$$

such that $q_i \xrightarrow{a\cap P_i}$, for all $i \in I$, and $q_j \not\xrightarrow{b^l_{j,k}}$ for all $j \in J^l, k \in K^l_j$. By construction both γ_1 and γ_2 contain a. Hence, a is enabled in the state $q_1 \dots q_n$ of $\gamma_1(B_1, \dots, B_n)$ and in the same state of $\gamma_2\big(\pi\gamma_1(B_1, \dots, B_n)\big)$, provided that it is not disabled by the priority π. Thus, we have to show that no interaction available from this state has higher priority. Priority rules in π that contain a are of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for some $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Since all the premises of (3.8) are satisfied in $q_1 \dots q_n$, interaction $b^l_{j(l),k(l)}$ is disabled. Hence, b is also disabled. Thus $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in $B_{\pi\gamma}$.

Let $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in $B_{\pi\gamma}$. This means that both γ_1 and γ_2 contain the interaction a. Therefore, by the construction of γ_2, there is at least one rule

$$\frac{\Big\{q_i \xrightarrow{a\cap P_i} q_i' \,\Big|\, i \in I\Big\} \quad \Big\{q_i = q_i' \,\Big|\, i \notin I\Big\} \quad \Big\{q_j \not\xrightarrow{b_{j,k}} \,\Big|\, j \in J, k \in K_j\Big\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}\,, \tag{3.9}$$

among the rules defining o. Furthermore, the priority model π has to contain priorities of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for all $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Assuming now that none of rules defining o, with the conclusion labelled by a, applies in $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$. Since $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in $B_{\pi\gamma}$, this necessarily means that each of these rules has a negative premise that is not satisfied. Let $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$ with $b^s_{j(s),k(s)}$, for all $s \in R_a$, being the labels of dissatisfied premises. Then b is an enabled interaction, such that $a \prec b$, which contradicts the assumption $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in $B_{\pi\gamma}$. Hence, there is at least one rule of the form (3.9) in the definition of o with all premises satisfied in $q_1 \dots q_n$ and, therefore, $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ in B_o. □

Thus, we conclude that BIP with relaxed priority models has weak full expressiveness w.r.t. the set of all BIP-like SOS operators.

Notice that the relaxed priority model does not allow recovering strong full expressiveness.

For instance, consider the operator defined by the single rule

$$\frac{q_1 \xrightarrow{p} q_1' \quad q_1 \not\xrightarrow{r}}{q_1 \xrightarrow{p} q_1'}\,, \tag{3.10}$$

applied to the component in Figure 3.2. The composite component has a single transition $1 \xrightarrow{p} 3$. The interaction model of BIP cannot contain r, as it is not possible to exclude transition $2 \xrightarrow{r} 3$ with a priority model. The transition $3 \xrightarrow{p} 3$ has to be excluded by the priority model, however it cannot use r in the priority relation.

Further relaxation of the definition of the BIP operator by removing the restriction $\pi \subseteq \gamma \times \gamma$ requires a slight modification of the semantics. Clearly, the component $\gamma(B_1, \ldots, B_n)$ does not have transitions that are not in γ and priority rules that can be applied to this component are in $\gamma \times \gamma$. Thus, we need to apply interaction and priority models simultaneously. The semantics of the simultaneous application of an interaction model γ and a priority model π is defined by putting $\sigma(\pi\gamma(B_1, \ldots, B_n)) \overset{def}{=} (Q, P, \rightarrow_{\pi\gamma})$, with $Q = \prod_{i=1}^n Q_i$ and the minimal transition relation $\rightarrow_{\pi\gamma}$ inductively defined by the set of rules

$$\left\{ \frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_j \not\xrightarrow{b\cap P_j} \,\middle|\, b \in K_a\right\}}{q_1 \ldots q_n \xrightarrow{a}_{\pi\gamma} q_1' \ldots q_n'} \,\middle|\, a \in \gamma, j : K_a \rightsquigarrow [1,n] \right\}, \tag{3.11}$$

where $I = \{i \in [1,n] \,|\, a \cap P_i \neq \emptyset\}$, $K_a = \{b | a \prec b\}$ and $j : K_a \rightsquigarrow [1,n]$ is a choice mapping $j : K_a \rightarrow [1,n]$, such that, for all $b \in K_a$, holds $b \cap P_{j(b)} \neq \emptyset$.

With this relaxation we obtain strong full expressiveness, since the operator o is then clearly equivalent to (γ_2, π).

Proposition 3.1.7. *For any set of components $B_i = (Q_i, P_i, \rightarrow)$, with $i \in [1,n]$, holds*

$$\sigma(\pi\gamma_2(B_1, \ldots, B_n)) = o(B_1, \ldots, B_n)\,.$$

Notice that the relaxation of the definition of BIP glue operators, by removing the restriction $\pi \subseteq \gamma \times \gamma$ but requiring π to be a strict partial order (the application is defined by a set of rules (3.11)), does not recover even weak full expressiveness w.r.t. BIP-like SOS operators. Indeed, Example 3.1.2 is still inexpressible.

3.2 Transformation of systems in classical semantics into offer semantics

In [23], it was shown that the expressiveness of BIP glue in the classical and in the offer semantics are incomparable. The system in Example 3.1.2 can be expressed in offer semantics with an extended interaction model $\gamma = \{\dot{p}\overline{r},\, \dot{r}\overline{b}\}$. Example 3.2.1 shows a system in classical

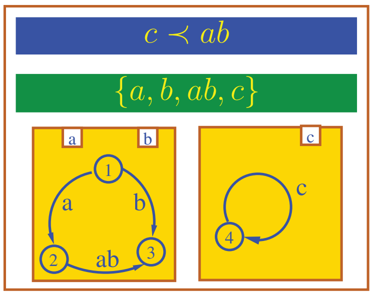


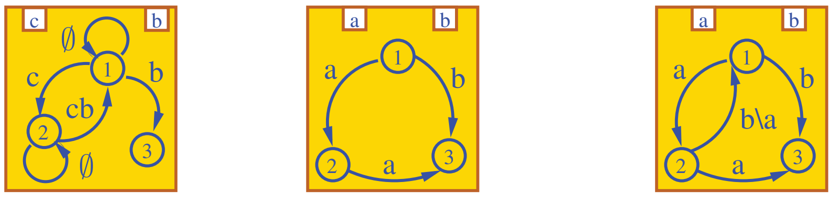
Figure 3.4 – System inexpressible in offer semantics.

semantics that is inexpressible in offer semantics. For the sake of simplicity and in order to better distinguish the BIP glues considered in the classical and in the offer semantics, we will refer to the former through pairs consisting of an interaction model $\gamma \subseteq 2^P$ and a priority model $\pi \subseteq 2^P \times (2^P \setminus \{\emptyset\})$; the latter will be given by extended interaction models $\gamma \subseteq 2^{P \cup \dot{P} \cup \overline{P}}$. Recall that in the classical semantics interactions that do not appear in the interaction model have no effect, when used in the priority model. Therefore, in this section, we will assume that all interactions appearing in a priority model also belong to the corresponding interaction model.

Example 3.2.1. Consider a system built from two components, in the classical semantics, shown in Figure 3.4. The interaction model is $\{a, b, ab, c\}$ and priority model is $\{c \prec ab\}$. Since, classical priority semantics refers to the activation of an interaction, in the composite system the interaction c is available at the state 14, and not available at the state 24. In the offer semantics, all three ports are offered in both states 14 and 24 of this system. Therefore, these states are indistinguishable and c is available or inhibited in both states simultaneously.

Note 3.2.2. In the remainder of this chapter, we will compare composed systems in the classical and the offer semantics obtained by applying glue operators to the same set of components. To simplify the presentation, we will assume that the predicate $\uparrow \subseteq Q \times P$ is also defined, in the classical semantics, on atomic components as in Definition 2.3.1 and, for composited systems as in Definition 2.3.4. Notice that this unambiguously defines the offer predicate in both cases. Hence, we will not explicitly provide it in the examples of this section. Furthermore, sets of states and ports of composite systems, as well as the corresponding offer predicates do not depend on the glue operator used to obtain them. Therefore, to prove that two composite components coincide, we will only have to check that their respective transition relations are equal. (Indeed, in this context, bisimilarity and equality coincide.) The following lemma shows that it is not necessary to consider target states of transitions and it is sufficient to compare labels of outgoing transitions for each state of composed systems.

Lemma 3.2.3. *Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components and let $P = \bigcup_{i=1}^n P_i$. Let $\pi\gamma$ and γ' be glue operators on P in the classical and the offer semantics, respectively. Then, for composed systems $(Q, P, \rightarrow_c, \uparrow) = \pi\gamma(B_1, \ldots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \ldots, B_n)$, the following holds: for any state q and for any transition label a, if $q \xrightarrow{a}_c \Leftrightarrow q \xrightarrow{a}_o$ then $\{q' \mid (q, a, q') \in \rightarrow_c\} = \{q' \mid (q, a, q') \in \rightarrow_o\}$.*



(a) $\emptyset \prec b$ (b) $a \prec b$, $a \neq \emptyset$ and $b \subseteq a$ (c) $a \prec b$, $a \neq \emptyset$ and $b \not\subseteq a$

Figure 3.5 – Components for Theorem 3.2.4.

Proof. If $q \not\xrightarrow{a}_c$ then $q \not\xrightarrow{a}_o$ and both sets are empty.

If $q \xrightarrow{a}_c$ then there is an interaction $a \in \gamma$ and no priority rule forbids the transition a from the state q. By (2.10) $(q, a, q') \in \rightarrow_c$ iff for all $i \in [1, n]$, $q_i \xrightarrow{a \cap P_i} q_i'$, if $a \cap P_i \neq \emptyset$, and $q_i = q_i'$ otherwise. At the same time $q \xrightarrow{a}_o$ and there is an interaction $a' \in \gamma'$, such that $\mathbf{fire}(a') = a$. By (2.26), $(q, a, q') \in \rightarrow_o$ iff for all $i \in [1, n]$, $q_i \xrightarrow{\mathbf{fire}(a') \cap P_i} q_i'$, if $\mathbf{fire}(a') \cap P_i \neq \emptyset$, and $q_i = q_i'$ otherwise. Since $a = \mathbf{fire}(a')$, $\{q' \,|\, (q, a, q') \in\rightarrow_c\} = \{q' \,|\, (q, a, q') \in\rightarrow_o\}$. □

If a priority model of a glue operator is empty, then such glue operator can be easily transformed into an operator in the offer semantics. However, for any non-empty priority model, there exists a set of components, such that the transformation of this glue operator into the offer semantics is not possible.

Theorem 3.2.4. *Let $\pi\gamma$ be a glue operator on a set of ports P in the classical semantics, such that γ contains at least two non-empty interactions and π has at least one priority $a \prec b$ with $a \neq b$. There exists a set of atomic components $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, where $\bigcup_{i=1}^{n} P_i = P$, such that, for any extended interaction model γ' in the offer semantics, the composed systems would not be equivalent, i.e. for $(Q, P, \rightarrow_c) = \pi\gamma(B_1, \ldots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \ldots, B_n)$ holds $\rightarrow_c \; \neq \rightarrow_o$.*

Proof. Let $a \prec b$, with $a \neq b$, be a priority in π. There are three cases. If $a = \emptyset$, let B_1 be the component in Figure 3.5(a) with $c \in \gamma$, $c \neq b$ and $c \neq \emptyset$ (such c exists by the assumption of the theorem). Recall that $a \prec \emptyset$ is not a valid priority. Therefore, we only have to consider two other cases, where neither a nor b are empty interactions: if $b \subseteq a$, let B_1 be the component in Figure 3.5(b); otherwise let B_1 be the component in Figure 3.5(c). The proof below applies identically to all three cases.

States 1 and 2 offer the same sets of ports. There is a transition a from both states. However, transition b is available only in the state 1. Let P_1 be a set of ports in B_1. Consider a second atomic component $B_2 = (\{*\}, P \setminus P_1, \{* \xrightarrow{p} * \,|\, p \in P \setminus P_1\}, \uparrow)$, such that the union of sets of ports of B_1 and B_2 is equal to P. Both $\pi\gamma$ and γ' can be applied to the pair of components (B_1, B_2). In the composed component, transition a is available in the state $2*$, but not available in the state $1*$. Since these states offer the same ports, for any glue operator in the offer semantics transition a is either available in both states or in none of them. □

We are now in position to define three classes of components, for which it is possible to generate an equivalent system in the offer semantics. The first class of components is characterised by Property 3.2.5. For any glue operator in the classical semantics there exists a glue operator in the offer semantics, such that their applications to any set of components satisfying Property 3.2.5 result in equivalent composite systems. Thus, when applied to the class of components satisfying Property 3.2.5, BIP glue in the offer semantics has strong full expressiveness with respect to glue in the classical semantics (Definition 2.1.2). For the two remaining classes of components, the transformation for any glue operator exists, but depends on the set of components. Components from the first of these two classes characterised by Property 3.2.11 allow a transformation without activation port typings. Finally, components characterised by Property 3.2.15 allow a transformation using activation port typings. If allowed components are from any of these two classes, BIP glue in offer semantics has strong expressiveness with respect to the glue in classical semantics (Definition 2.1.1).

Below, we use the following notations: for $a \in 2^P$, we denote $\dot{a} \overset{def}{=} \{\dot{p} \,|\, p \in a\}$ and $\overline{a} \overset{def}{=} \{\overline{p} \,|\, p \in a\}$.

3.2.1 Transformation not depending on component set

Consider a class of components satisfying the following property:

Property 3.2.5. *Let L be a set of transition labels of a component B. For any state q of the component B and any $a \in L \setminus \{\emptyset\}$, holds $q{\uparrow}a \Rightarrow q \xrightarrow{a}$.*

For any system, such that the component of all its sub-systems belong to this class, any glue operator in the classical semantics can be transformed into a glue operator in the offer semantics. Applying the initial and the generated glue operators to any set of components from this class would result in two equal composite systems.

Given an interaction model and a priority model, the algorithm in Figure 3.6 computes an extended interaction model, corresponding to *the same interaction and priority models considered in the offer semantics.* In particular, all interactions, generated starting from an interaction a, have the firing support $\mathbf{fire}(a') = a$, since no firing ports are added after initial generation of the set I.

Lemma 3.2.6. *Let $B_i = (Q_i, P_i, \to, \uparrow)$ for $i \in [1, n]$ be a set of components satisfying Property 3.2.5. Let $(Q, P, \to, \uparrow) = \pi\gamma(B_1, \dots, B_n)$ be a composite system, with γ an interaction model and π a priority model. Then, in any state $q \in Q$, any offered interaction a in γ is active in $(Q, P, \to, \uparrow)$ if and only if it is not inhibited by a priority:*

$$(\nexists b \in \gamma : a \prec b \wedge q \xrightarrow{b}_{\gamma}) \Leftrightarrow q \xrightarrow{a}_{\pi} \; . \tag{3.12}$$

Proof. $\Leftarrow$ is straightforward by (2.11).

Input:	A glue operator in the classical semantics: an interaction model γ and a priority model π.
Output:	A glue operator in the offer semantics: an extended interaction model γ'.

```
1. $I := \{\dot{a} \,|\, a \in \gamma\}$;
2. for each $(a \prec b) \in \pi$
3.     $m := b \setminus a$;
4.     for each $c \in I$, such that $\mathbf{fire}(c) = a$
5.         $C := \{c\,\overline{p} \,|\, p \in m\}$;
6.         $I := (I \setminus \{c\}) \cup C$;
7. $\gamma' := I$;
```

Figure 3.6 – Algorithm transforming a glue operator in the classical semantics into a glue operator in the offer semantics.

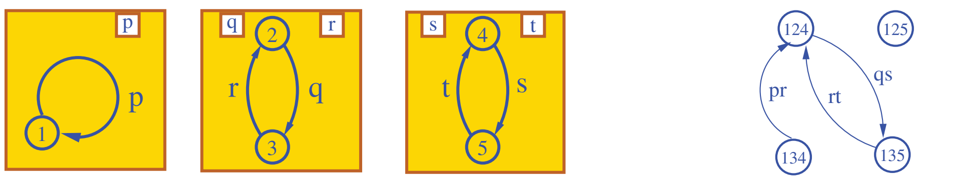
$\underline{\Rightarrow}$: Since, for all $i \in [1, n]$, components B_i satisfy Property 3.2.5, $q_i \uparrow (a \cap P_i) \Rightarrow q_i \xrightarrow{a \cap P_i}$. Hence, by (2.10), for any $a \in \gamma$, $q \uparrow a \Rightarrow q \xrightarrow{a}_\gamma$. By (2.11), $\nexists b \in \gamma : a \prec b \wedge q \xrightarrow{b}_\gamma$ implies $q \xrightarrow{a}_\pi$. $\square$

Theorem 3.2.7. *Let $\pi\gamma$ be a glue operator on a set of ports P in the classical semantics and let γ' be a glue operator obtained by applying the algorithm in Figure 3.6. Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components satisfying Property 3.2.5, and $\bigcup_{i=1}^n P_i = P$. Then for $(Q, P, \rightarrow_c, \uparrow) = \pi\gamma(B_1, \ldots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \ldots, B_n)$ holds $\rightarrow_c \; = \; \rightarrow_o$.*

Proof. 1) $\underline{q \xrightarrow{a}_o \implies q \xrightarrow{a}_c}$: Since $q \xrightarrow{a}_o$, there is an interaction $a' \in \gamma'$, having $\mathbf{fire}(a') = a$. By construction, the generation of a' started from the interaction $a \in \gamma$. Since $q \xrightarrow{a}_o$, $q \uparrow \mathbf{fire}(a')$ and $q \not\uparrow \mathbf{neg}(a')$. For any b, such that $a \prec b$, we have $b \cap \mathbf{neg}(a') \neq \emptyset$ and, consequently, $q \not\xrightarrow{b}_c$, since at least one port of b is not available. By Lemma 3.2.6, we have $q \xrightarrow{a}_c$.

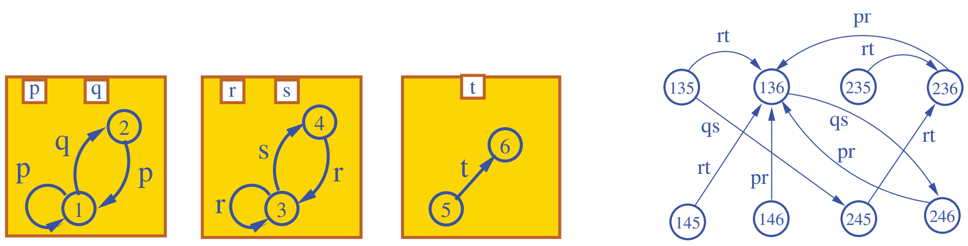
2) $\underline{q \xrightarrow{a}_c \implies q \xrightarrow{a}_o}$: Since $q \xrightarrow{a}_c$, for all $b \in \gamma$, such that $a \prec b$, holds $q \not\xrightarrow{b}_c$. By Lemma 3.2.6, if all ports of b are offered at the state q, then either b is enabled or some $b' : b \prec b'$ is enabled. The priority model is a partial order and, in particular, is transitive. Hence, a has to be suppressed due to availability of b or b'. Thus, at least one port of b is not offered at the state q. Let $p_b \in b$ be a port, such that $q \not\uparrow p_b$. Consider a set of ports c, constructed by choosing one such port p_b for each priority rule $a \prec b$ (the same port can be used for different rules). By construction of γ', we have $a' = \dot{a} \cup \overline{c} \in \gamma'$. Thus, $\mathbf{fire}(a') = a$ and, for all $p \in \mathbf{neg}(a')$, $q \not\uparrow p$. Hence, by Definition 2.3.1, $q \xrightarrow{a}_o$. $\square$

Example 3.2.8. Consider $\gamma = \{pr, qs, rt\}$ and $\pi = \{pr \prec qs,\; pr \prec rt\}$ in the classical semantics. The algorithm in Figure 3.6 generates an equivalent extended interaction model. In the first step, the set $I = \{\dot{p}\dot{r}, \dot{q}\dot{s}, \dot{r}\dot{t}\}$. Considering the first priority rule $pr \prec qs$, we have $m = qs \setminus pr = qs$. For each interaction in I with firing support pr, we generate a set of new interactions, thus from the interaction $\dot{p}\dot{r}$ we obtain a pair of interactions $\dot{p}\dot{r}\overline{q}$ and $\dot{p}\dot{r}\overline{s}$. The new set $I = \{\dot{p}\dot{r}\overline{q}, \dot{p}\dot{r}\overline{s}, \dot{q}\dot{s}, \dot{r}\dot{t}\}$. For the second priority rule $pr \prec rt$, we have $m = rt \setminus pr = t$.



(a) Set of components (b) Composite system

Figure 3.7 – First set of components and composite system for Example 3.2.8.



(a) Set of components (b) Composite system

Figure 3.8 – Second set of components and composite system for Example 3.2.8.

There are two interactions in I with firing support pr: $\dot{p}\dot{r}\overline{q}$ and $\dot{p}\dot{r}\overline{s}$. The algorithm adds $\overline{t}$ to both of them and the final glue in the offer semantics is $\gamma' = \{\dot{p}\dot{r}\overline{q}\,\overline{t}, \dot{p}\dot{r}\overline{s}\,\overline{t}, \dot{q}\dot{s}, \dot{r}\dot{t}\}$.

Consider components in Figure 3.7(a). All of them satisfy Property 3.2.5. Applying glues in the classical and in the offer semantics we obtain equal composite components. Their behaviours are shown in Figure 3.7(b). Transitions qs and rt are available in the states 124 and 135 respectively in both composite components. Ports p and r are offered in the states 134 and 135. The priority rule $pr \prec rt$ forbids the transition pr in the state 135, thus in the system in the classical semantics this transition is available only in the state 134. In the system in the offer semantics in the state 134 ports q and t are not offered, thus the interaction $\dot{p}\dot{r}\overline{q}\,\overline{t}$ allows a transition from this state, whereas in the state 135 t is offered and none of the interactions allows a transition pr from this state.

Consider components in Figure 3.8(a) and the same glue. All of them also satisfy Property 3.2.5. Applying glues in the classical and in the offer semantics we obtain equal composite components. Their behaviours are shown in Figure 3.8(b). Transitions qs and rt are available simultaneously in both composite components, since they depend only on availability of the corresponding ports. There are transitions qs from states 135 and 136, transitions rt from states 135, 145, 235 and 245. Ports p and r are offered in all states. The priority rules forbids transition pr from all states where qs or rt are available, thus there are transitions pr from states 146, 236 and 246. In the system in the offer semantics the interaction $\dot{p}\dot{r}\overline{q}\,\overline{t}$ allows transitions pr from states 236 and 246, the interaction $\dot{p}\dot{r}\overline{s}\,\overline{t}$ allows transitions pr from the state 146.

Theorem 3.2.9. *Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, and $n \geq 2$ be a set of components,* *such that at least one of them violates Property 3.2.5, and let $P = \bigcup_{i=1}^{n} P_i$. There exists $\pi\gamma$, a glue operator on P in the classical semantics, such that for the glue γ' computed by the algorithm in Figure 3.6, and for $(Q, P, \rightarrow_c, \uparrow) = \pi\gamma(B_1, \dots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \dots, B_n)$ holds $\rightarrow_c \neq \rightarrow_o$.*

Proof. Without loss of generality, we assume that B_1 violates Property 3.2.5. Thus there is a state q_1 and a transition a, such that $q_1 \uparrow a$ and $q \not\xrightarrow{a}$. Let b be a transition from a state q_2 in B_2. Let $\gamma = \{a, b\}$ and $\pi = \{b \prec a\}$. The algorithm in Figure 3.6 computes $\gamma' = \{\dot{a}\} \cup \{\dot{b}\,\overline{p} \mid p \in a \setminus b\}$. The transition labelled b is available in the state $q_1 \dots q_n$ of $(Q, P, \rightarrow_c)$, but it is not available in the state $q_1 \dots q_n$ of $(Q, P, \rightarrow_o, \uparrow)$, since all ports of a are offered in this state. ☐

Notice that components in this class allow to represent any BIP-like SOS composition operator (2.14) as a set of SOS rules in the format (2.25) without "positive" premises i.e. in the format:

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_k \not\uparrow b_k^l \,\middle|\, k \in K, l \in L_k\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'} . \tag{3.13}$$

Proposition 3.2.10. *For any BIP-like n-ary SOS composition operator o there exists an operator $\tilde{o}$ represented as a set of SOS rules in the format (3.13), such that for any set of components $B_1, \dots, B_n$ satisfying Property 3.2.5 holds: $\tilde{o}(B_1, \dots, B_n) = o(B_1, \dots, B_n)$.*

Proof. We obtain operator $\tilde{o}$ by replacing all premises $q_j \not\xrightarrow{b_j^k}$ with $q \not\uparrow b_j^k$ in all rules of the operator o. By Property 3.2.5, $q \not\xrightarrow{a} \Rightarrow q \not\uparrow a$. Thus, if in a state a transition is allowed by a rule of the operator o, it is also allowed by a corresponding rule of the operator $\tilde{o}$. By Definition 2.3.1, $q \not\uparrow a \Rightarrow q \not\xrightarrow{a}$. Thus, if in a state a transition is allowed by a rule of the operator $\tilde{o}$, it is also allowed by a corresponding rule of the operator o. ☐

3.2.2 Transformation not using activation port typings

Let us now consider the class of components satisfying the following property:

Property 3.2.11. *Let L be a set of transition labels of B. For any state q, such that $S_q \overset{def}{=} \{a \in L \setminus \{\emptyset\} \mid q \uparrow a \wedge q \not\xrightarrow{a}\} \neq \emptyset$, the following holds: for any state $q' \neq q$, such that $\exists\, a \in S_q : q' \xrightarrow{a}$, there exists a port p, such that $q' \uparrow p$ and $q \not\uparrow p$.*

Intuitively, this property means the following. Assume there is a state that offers an interaction a, which does not correspond to an enabled transition, e.g. a is a proper subset of a label of an enabled transition. Assume, furthermore, that there is also a state that actually has an enabled transition labelled by a. Then Property 3.2.11 requires that these two states be distinguishable by considering whether some other port p is offered or not.

If the Property 3.2.11 holds for a set of components, we can build a composite system in the offer semantics without using activation port typings. Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components, let $P = \bigcup_{i=1}^{n} P_i$ and let $\pi\gamma$ be a glue operator on P in the classical semantics. We start the transformation by applying the algorithm in Figure 3.6. This algorithm generates an extended interaction model γ'' in the offer semantics. However, this property is weaker than Property 3.2.5 and composed components $\pi\gamma(B_1, \ldots, B_n)$ and $\gamma''(B_1, \ldots, B_n)$ can be not equal. A transition relation of the former composite component can contain transitions, which are not present in the latter one. For each such transition a from the state q we add the interaction $\dot{a}\overline{b}$ to the interaction model, where $b = \{p \,|\, q \not\uparrow p\}$. The application of the final extended interaction model γ' results in an equivalent composite system.

Theorem 3.2.12. *Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components, such that all of them satisfy Property 3.2.11 and let $P = \bigcup_{i=1}^{n} P_i$. Let $\pi\gamma$ be a glue operator on P in the classical semantics. Let γ' be an extended interaction model generated in the way it was shown above. Then for $(Q, P, \rightarrow_c, \uparrow) = \pi\gamma(B_1, \ldots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \ldots, B_n)$ holds $\rightarrow_c = \rightarrow_o$.*

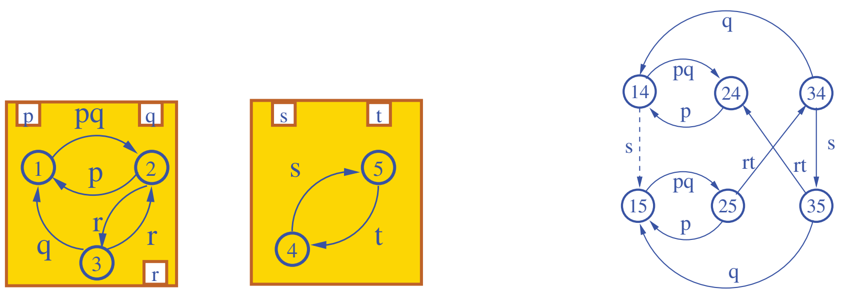
Proof. 1) $\underline{q \xrightarrow{a}_o \implies q \xrightarrow{a}_c}$: By construction, if $q \xrightarrow{a}_o$, then there exists an interaction $a' \in \gamma'$, such that $\mathbf{fire}(a') = a$ and $\mathbf{neg}(a') = \{p \,|\, q \not\uparrow p\}$. There are two possibilities for the moment, when a' was added to γ'. Let γ'' be the extended interaction model computed by the algorithm in Figure 3.6.

If $a' \in \gamma''$, the proof is similar to the one in Theorem 3.2.7. By construction, the generation of a' started from interaction $a \in \gamma$. Since $q \xrightarrow{a}_o$, we have $q \uparrow a$ and $q \not\uparrow \mathbf{neg}(a')$. For any b, such that $a \prec b$, we have $b \cap \mathbf{neg}(a') \neq \emptyset$ and, consequently, $q \overset{b}{\not\rightarrow}_c$, as at least one port of b is not available. Thus, $q \xrightarrow{a}_c$.

If a' was added during the second step of the computation above, a' could have been added to γ' only if there is a state $q' = q'_1 \ldots q'_n$ in the composite system $(Q, P, \rightarrow_c, \uparrow)$, such that $q' \xrightarrow{a}_c$ and $\mathbf{neg}(a') = \{p \,|\, q' \not\uparrow p\}$. Assume that $q = q_1 \ldots q_n \overset{a}{\not\rightarrow}_c$. Since $q \xrightarrow{a}_o$, we have $q \uparrow a$. Thus, a was forbidden by the application of some priority rule $a \prec b$ and $q \xrightarrow{b}_c$. Since $q' \xrightarrow{a}_c$, we also have $q' \overset{b}{\not\rightarrow}_c$. If $q' \not\uparrow b$, then there exists $p \in b$, such that $q' \not\uparrow p$, so $\overline{p} \in a'$ and $q \overset{a}{\not\rightarrow}_o$. If $q' \uparrow b$ then there exists a component B_i, such that $q'_i \uparrow (b \cap P_i)$ and $q'_i \overset{b \cap P_i}{\not\longrightarrow}$. Then $q_i \uparrow (b \cap P_i)$ and, by Property 3.2.11, there exists a port p, such that $q'_i \not\uparrow p$ and $q_i \uparrow p$. Consequently, $q' \not\uparrow p$ and $q \uparrow p$, so $\overline{p} \in a'$ and $q \overset{a}{\not\rightarrow}_o$, which contradicts the assumption $q \overset{a}{\not\rightarrow}_c$.

2) $\underline{q \xrightarrow{a}_c \implies q \xrightarrow{a}_o}$: By construction of γ', either the extended interaction model γ'', obtained after the application of the algorithm in Figure 3.6, contains an interaction generating this transition in the composite system, or an additional interaction is added to γ' that makes this transition present in the composite system. □

Example 3.2.13. Consider components in Figure 3.9(a) and a glue operator in the classical semantics with $\gamma = \{p, pq, q, rt, s\}$ and $\pi = \{s \prec p\}$. Both components satisfy Property 3.2.11.



(a) Set of components (b) Composite system

Figure 3.9 – Components and composite system for Example 3.2.13.

The behaviour of the composite system in the classical semantics is shown in Figure 3.9(b). The algorithm in Figure 3.6 generates the extended interaction model $\gamma'' = \{\dot{p}, \dot{p}\dot{q}, \dot{q}, \dot{r}\dot{t}, \dot{s}\overline{p}\}$. If we apply γ'' to the components in Figure 3.9(a), the composite component would not contain a transition s from the state 14 (dashed in Figure 3.9(b)), as port p is offered at this state. Thus, we need to add an interaction to the extended interaction model, in order to make this transition available, but no other transitions should be added. The interaction $\dot{s}\overline{r}$ adds the transition s from the state 14 in the composite component and it does not add a transition from the state 24, since r is offered at the state 24. Thus, the final extended interaction model is $\gamma' = \{\dot{p}, \dot{p}\dot{q}, \dot{q}, \dot{r}\dot{t}, \dot{s}\overline{p}, \dot{s}\overline{r}\}$.

Theorem 3.2.14. *For any set of components $B_i = (Q_i, P_i, \rightarrow, \uparrow)$ for $i \in [1, n]$ and $n \geq 2$, such that at least one of B_i violates Property 3.2.11, there exists a glue operator $\pi\gamma$ on $P = \bigcup_{i=1}^{n} P_i$ in the classical semantics, such that the composed system $\pi\gamma(B_1, \ldots, B_n)$ cannot be expressed through the offer semantics without using the activation port typings.*

Proof. Without loss of generality, assume that B_1 violates Property 3.2.11. Thus, there exists a state q_1 and a transition a, such that $q_1 \uparrow a$ and $q_1 \not\xrightarrow{a}$ and there exists a state q_1', such that $q_1' \xrightarrow{a}$ and all ports which are not offered in q_1 are also not offered in q_1'.

Let b be some transition label of the component B_2, and let q_2 be a state, such that $q_2 \xrightarrow{b}$. Let $\gamma = \{a, b\}$ and $\pi = \{b \prec a\}$. In a composite system $\pi\gamma(B_1, \ldots, B_n)$, a transition labeled by b is available from the state $q_1 q_2 \ldots q_n$, but not available from the state $q_1' q_2 \ldots q_n$.

Assume that there exists an extended interaction model γ' without activation port typings, such that $\gamma'(B_1, \ldots, B_n) = \pi\gamma(B_1, \ldots, B_n)$. In order to have a transition labelled by b from the state $q_1 \ldots q_n$, γ' has to contain the interaction $\dot{b}\overline{c}$, where $c \subseteq \{p \mid q_1 \not\uparrow p\}$. However, this interaction allows a transition b from the state $q_1' q_2 \ldots q_n$, which contradicts the assumption of the existence of γ'. □

3.2.3 Transformation using activation ports typings

We now consider the class of components characterised by the following property:

Property 3.2.15. *For any two states q_1, q_2 in the component, $\{p \,|\, q_1 {\uparrow} p\} = \{p \,|\, q_2 {\uparrow} p\}$ implies $\{a \neq \emptyset \,|\, q_1 \xrightarrow{a}\} = \{a \neq \emptyset \,|\, q_2 \xrightarrow{a}\}$.*

Proposition 3.2.16. *Property 3.2.11 implies Property 3.2.15.*

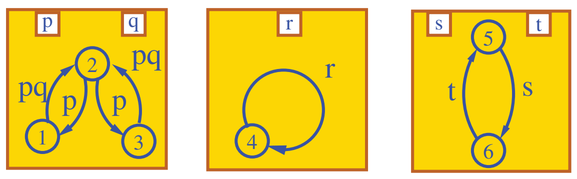
Proof. Consider a component with two states q_1 and q_2 violating Property 3.2.15 and assume that Property 3.2.11 holds for this component. We have $\{p \,|\, q_1 {\uparrow} p\} = \{p \,|\, q_2 {\uparrow} p\}$ and $\{a \neq \emptyset \,|\, q_1 \xrightarrow{a}\} \neq \{a \neq \emptyset \,|\, q_2 \xrightarrow{a}\}$. Without loss of generality, there exists $a \neq \emptyset$, such that $q_1 \xrightarrow{a}$ and $q_2 \not\xrightarrow{a}$. Since $q_1 \xrightarrow{a}$, we also have $q_1 {\uparrow} a$ and, consequently, $q_2 {\uparrow} a$. Therefore, $a \in S_{q_2}$ (see Property 3.2.11). By Property 3.2.11, we then have $\{p \,|\, q_1 {\uparrow} p\} \neq \{p \,|\, q_2 {\uparrow} p\}$, contradicting our assumption. □

Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components satisfying Property 3.2.15 and let $\pi\gamma$ be a glue operator on P in the classical semantics. Consider an initially empty extended interaction model γ'. To each state q of each component, we associate a set $\chi(q) = \{p \,|\, q {\uparrow} p\} \cup \{\overline{p} \,|\, q \not\uparrow p\}$. Notice, that since sets of ports of components are pairwise disjoint, $\chi(q)$ and $\chi(q')$ are disjoint if q and q' are states of different components. Let $(Q, P, \rightarrow, \uparrow) = \pi\gamma(B_1, \dots, B_n)$ be a composite system. To each state $q_1 \dots q_n \in Q$, we associate a set $\chi(q_1 \dots q_n) = \bigcup_{i=1}^n \chi(q_i)$. For each transition $q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'$ of the composite system, we add to γ' the extended interaction $\tilde{a} = \{\dot{p} \,|\, p \in a\} \cup \{p \,|\, p \in \chi(q_1 \dots q_n), p \notin a\}$ (notice that, for all such extended interactions $\tilde{a}$, we have $\mathbf{fire}(\tilde{a}) \cup \mathbf{act}(\tilde{a}) \cup \mathbf{neg}(\tilde{a}) = P$). The theorem below shows that $\gamma'(B_1, \dots, B_n)$ is equivalent to $(Q, P, \rightarrow, \uparrow)$.

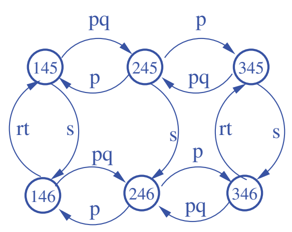
Theorem 3.2.17. *Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$, be a set of components satisfying Property 3.2.15 and let $P = \bigcup_{i=1}^n P_i$. Let $\pi\gamma$ be a glue operator on P in the classical semantics. Let γ' be an extended interaction model obtained as above. Then, for composite components $(Q, P, \rightarrow_c, \uparrow) = \pi\gamma(B_1, \dots, B_n)$ and $(Q, P, \rightarrow_o, \uparrow) = \gamma'(B_1, \dots, B_n)$, holds $\rightarrow_c = \rightarrow_o$.*

Proof. 1) $\underline{q \xrightarrow{a}_o \implies q \xrightarrow{a}_c}$: By (2.26), $q \xrightarrow{a}_o$ implies the existence of an extended interaction $a' \in \gamma'$, such that $\mathbf{fire}(a') = a$. Recall that, for all $\tilde{a} \in \gamma'$, we have $\mathbf{fire}(\tilde{a}) \cup \mathbf{act}(\tilde{a}) \cup \mathbf{neg}(\tilde{a}) = P$. Hence, (2.26) also implies $\mathbf{fire}(a') \cup \mathbf{act}(a') = \{p \,|\, q {\uparrow} p\}$ and $\mathbf{neg}(a') = \{p \,|\, q \not\uparrow p\}$. By construction, each extended interaction $a' \in \gamma'$ corresponds to some transition $q' \xrightarrow{a}_c q''$ (see the algorithm above). Hence, $a \in \gamma$. Furthermore, by construction of γ', we also have $\mathbf{fire}(a') = a$, $\mathbf{fire}(a') \cup \mathbf{act}(a') = \{p \,|\, q' {\uparrow} p\}$ and $\mathbf{neg}(a') = \{p \,|\, q' \not\uparrow p\}$. Therefore, $\{p \,|\, q {\uparrow} p\} = \{p \,|\, q' {\uparrow} p\}$ and, denoting $q = q_1 \dots q_n$ and $q' = q_1' \dots q_n'$, we have $\{p \,|\, q_i {\uparrow} p\} = \{p \,|\, q_i' {\uparrow} p\}$, for all $i \in [1, n]$. By Property 3.2.15, for any $i \in [1, n]$, holds $\{b \neq \emptyset \,|\, q_i \xrightarrow{b}\} = \{b \neq \emptyset \,|\, q_i' \xrightarrow{b}\}$. By (2.10) and (2.11), we then have $\{b \,|\, q \xrightarrow{b}_c\} = \{b \,|\, q' \xrightarrow{b}_c\}$ in $(Q, P, \rightarrow_c, \uparrow)$. Since $q' \xrightarrow{a}_c$, we conclude that $q \xrightarrow{a}_c$.

2) $\underline{q \xrightarrow{a}_c \implies q \xrightarrow{a}_o}$: By construction, $q \xrightarrow{a}_c$ implies $a' = \{\dot{p} \,|\, p \in a\} \cup \{p \,|\, p \in \chi(q), p \notin a\} \in \gamma'$ (with $\mathbf{fire}(a') = a$). By definition of χ, we have $q {\uparrow} p$, for all $p \in \mathbf{act}(a')$, and $q \not\uparrow p$, for all $p \in \mathbf{neg}(a')$, thus $q \xrightarrow{a}_o$. □



(a) Set of components



(b) Composite system

Figure 3.10 – Components and composite system for Example 3.2.18.

Example 3.2.18. Consider components in Figure 3.10(a) and a glue operator in the classical semantics defined by $\gamma = \{p, pq, rt, s\}$ and $\pi = \{rt \prec p\}$. Both components satisfy Property 3.2.15. The behaviour of the composite system in the classical semantics is shown in Figure 3.10(b). This system cannot be expressed in the offer semantics without activation port typings. There should be a transition rt from the state 146, but any interaction allowing it would also allow a transition rt from the state 246, as all ports, which are not offered in the state 146, are also not offered in the state 246.

In order to transform the system into one in the offer semantics we associate a set χ to each state as follows:

$$\begin{aligned} &\chi(145) = \{p, q, r, s, \bar{t}\}, \quad \chi(245) = \{p, \bar{q}, r, s, \bar{t}\}, \quad \chi(345) = \{p, q, r, s, \bar{t}\},\\ &\chi(146) = \{p, q, r, \bar{s}, t\}, \quad \chi(246) = \{p, \bar{q}, r, \bar{s}, t\}, \quad \chi(346) = \{p, q, r, \bar{s}, t\}. \end{aligned}$$

Now, we start generating γ' considering all transition labels in the composed system. From the state 145, there are transitions pq and s, thus we take interactions $\dot{p}\,\dot{q}\,r\,s\,\bar{t}$ and $\dot{s}\,p\,q\,r\,\bar{t}$. From the state 146, there are transitions pq and rt, hence we add interactions $\dot{p}\,\dot{q}\,r\,\bar{s}\,t$ and $\dot{r}\,\dot{t}\,p\,q\,\bar{s}$. Proceeding similarly for the remaining states, we obtain γ':

$$\{\dot{p}\,\dot{q}\,r\,s\,\bar{t},\ \dot{s}\,p\,q\,r\,\bar{t},\ \dot{p}\,\dot{q}\,r\,\bar{s}\,t,\ \dot{r}\,\dot{t}\,p\,q\,\bar{s},\ \dot{p}\,r\,s\,\bar{q}\,\bar{t},\ \dot{p}\,r\,t\,\bar{q}\,\bar{s},\ \dot{s}\,p\,r\,\bar{q}\,\bar{t}\}.$$

Noticing that s and t are mutually exclusive and p, r are offered in all states, this extended interaction model can be simplified to $\gamma'' = \{\dot{p}\,\dot{q}, \dot{p}\,\bar{q}, \dot{s}, \dot{r}\,\dot{t}\,q\}$.

Theorem 3.2.19. *Let $B_i = (Q_i, P_i, \rightarrow, \uparrow)$, for $i \in [1, n]$ and $n \geq 2$, be a set of components, such that at least one of them violates Property 3.2.15, and let $P = \bigcup_{i=1}^{n} P_i$. There exists a glue operator in the classical semantics with an interaction model γ and a priority model π, such that the system $\pi\gamma(B_1, \ldots, B_n)$ cannot be expressed in the offer semantics.*

Proof. Without loss of generality assume that B_1 violates Property 3.2.15. Thus, there is a pair of states q_1 and q_1', such that $\{p | q_1 \uparrow p\} = \{p | q_1' \uparrow p\}$ and $q_1 \xrightarrow{a}$, while $q_1' \not\xrightarrow{a}$. Let b be a transition from a state q_2 in B_2. Let $\gamma = \{a, b\}$ and $\pi = \{b \prec a\}$. A composite system $\pi\gamma(B_1, \ldots, B_n)$ cannot be expressed in the offer semantics.

Consider two states $q_1 q_2 \dots q_n$ and $q_1' q_2 \dots q_n$. In the system $\pi\gamma(B_1, \dots, B_n)$, transition b is forbidden from the first state by the priority rule, but b is allowed from the second state. However, sets of offered ports from both states are equal. Thus, for any interaction b', such that $\mathbf{fire}(b') = b$, either b' can be allowed from both states or it is forbidden from both states. These states cannot be distinguished in the offer semantics, which implies that the system $\pi\gamma(B_1, \dots, B_n)$ cannot be expressed in the offer semantics. □

3.2.4 Hierarchical systems

In hierarchical systems, glue operators can be applied not only to atomic components, but also to composite ones. If we consider components that satisfy Property 3.2.11 or Property 3.2.15, application of any glue operator results in a composite component of the same class.

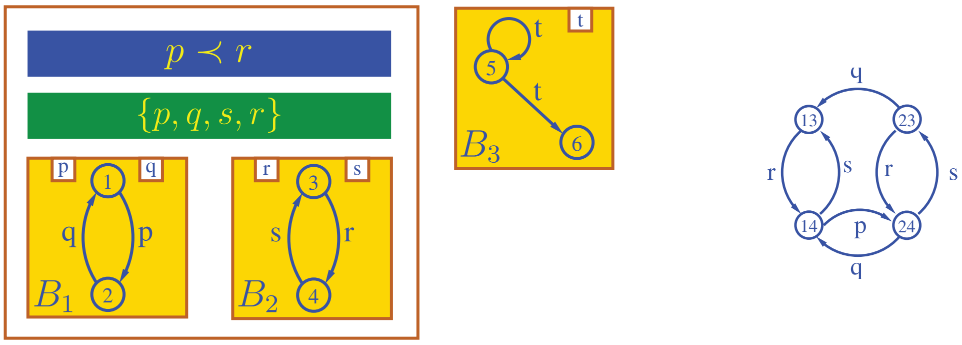
Proposition 3.2.20. *Let $B_i = (Q_i, P_i, \to, \uparrow)$, for $i \in [1, n]$, be a set of components, such that all of them satisfy Property 3.2.11 and let $P = \bigcup_{i=1}^n P_i$. Then, for any interaction model γ on P and for any priority model π on P, the composite component $\pi\gamma(B_1, \dots, B_n)$ satisfies Property 3.2.11.*

Proof. Assume that $\pi\gamma(B_1, \dots, B_n)$ violates this property. There are two states $q = q_1 \dots q_n$, $q' = q_1' \dots q_n'$ and a transition a, such that $q \uparrow a$, $q \not\xrightarrow{a}$, $q' \xrightarrow{a}$ and there exists no port p, such that $q' \uparrow p$ and $q \not\uparrow p$. Since $q \not\xrightarrow{a}$, for some $i \in [1, n]$, $q_i \uparrow a \cap P_i$ and $q_i \not\xrightarrow{a \cap P_i}$. Since B_i satisfies Property 3.2.11 and $q_i' \xrightarrow{a \cap P_i}$, there exists p, such that $q_i \not\uparrow p$ and $q_i' \uparrow p$. By Definition 2.3.1, $q \not\uparrow p$ and $q' \uparrow p$. Thus, states q and q' cannot violate Property 3.2.11. □

Proposition 3.2.21. *Let $B_i = (Q_i, P_i, \to, \uparrow)$, for $i \in [1, n]$, be a set of components, such that all of them satisfy Property 3.2.15, and let $P = \bigcup_{i=1}^n P_i$. Then, for any interaction model γ on P and for any priority model π on P, the composite component $\pi\gamma(B_1, \dots, B_n)$ satisfies Property 3.2.15.*

Proof. Assume that $\pi\gamma(B_1, \dots, B_n)$ violates this property. Thus, there are two states $q = q_1 \dots q_n$ and $q' = q_1' \dots q_n'$, such that $\{p | q \uparrow p\} = \{p | q' \uparrow p\}$ and $\{a | q \xrightarrow{a}\} \neq \{a | q' \xrightarrow{a}\}$. By Definition 2.3.1, we can deduce that, for $i \in [1, n]$, holds $\{p | q_i \uparrow p\} = \{p | q_i' \uparrow p\}$. Since all $B_1, \dots, B_n$ satisfy Property 3.2.15, we have $\{a | q_i \xrightarrow{a}\} = \{a | q_i' \xrightarrow{a}\}$, for $i \in [1, n]$. Consequently, if only the interaction model is applied, $\{a | q \xrightarrow{a}\} = \{a | q' \xrightarrow{a}\}$ by (2.10). The priority model can only remove transitions from these states simultaneously, as the sets of outgoing transitions are equal. Thus, states q and q' cannot violate Property 3.2.15. □

Propositions 3.2.20 and 3.2.21 show that composition of components with glue operators in the classical semantics preserves, respectively, Properties 3.2.11 and 3.2.15. Therefore, for an application of a hierarchical glue operator in the classical semantics to a set of atomic components, all satisfying the same property (recall that Property 3.2.11 implies



(a) Set of components

(b) Composite system

Figure 3.11 – Components and composite system for Example 3.2.22.

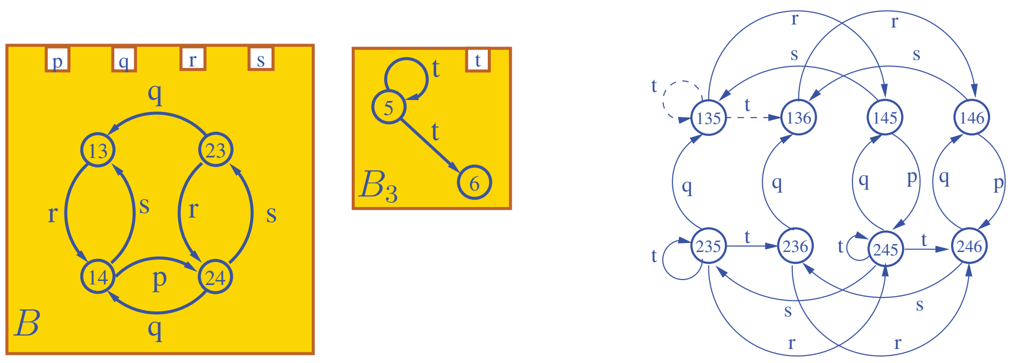
Property 3.2.15), we can iteratively construct the corresponding operator in the offer semantics. We start at the lowest level of operator hierarchy, i.e. from the atomic components. Since atomic components satisfy one of the properties, the corresponding glue operator in the offer semantics can be generated and the composite component also satisfies the property. Repeating this reasoning for the operators on higher levels, we obtain the corresponding hierarchy of glues in the offer semantics.

In general, glue operators do not preserve Property 3.2.5.

Example 3.2.22. All components in Figure 3.11(a) satisfy Property 3.2.5. The application of the glue operator defined, in the classical semantics, by the interaction model $\gamma = \{p, q, r, s\}$ and the priority model $\pi = \{p \prec r\}$ to B_1 and B_2 results in the composite component B. Its behaviour is shown in Figure 3.11(b). In the state 13, p is offered, since it is offered by B_1, however, it is forbidden by the priority model. If we consider a glue operator on the next level of hierarchy with $\gamma = \{p, q, r, s, t\}$ and $\pi = \{t \prec p\}$ that can be applied to B and B_3, the algorithm in Figure 3.6 will generate an incorrect extended interaction model (see the next example).

For any component, Property 3.2.5 implies Property 3.2.11. Thus, hierarchical systems with atomic components that satisfy Property 3.2.5 can always be transformed into offer semantics, but the algorithm in Figure 3.6 can only be applied to the lowest level of hierarchy.

Example 3.2.23. Consider components and glue operators from the previous example. For the first level of hierarchy the algorithm in Figure 3.6 generates the extended interaction model $\gamma_1' = \{\dot{p}\overline{r}, \dot{q}, \dot{r}, \dot{s}\}$. The composite component B (Figure 3.11(b)) does not satisfy Property 3.2.5, but it satisfies Property 3.2.11. Thus, the extended interaction model on the second level of hierarchy can be generated. If we consider components B and B_3 (Figure 3.12(a)) and a glue operator on the next level of hierarchy defined by $\gamma = \{p, q, r, s, t\}$ and $\pi = \{t \prec p\}$, the application of the algorithm in Figure 3.6 will generate an incorrect extended interaction model. The final composite system in the classical semantics is shown in Figure 3.12(b). The algorithm in Figure 3.6 generates the extended interaction model



(a) Set of components (b) Composite system

Figure 3.12 – Components and composite system for Example 3.2.23.

$\gamma_2'' = \{\dot{p}, \dot{q}, \dot{r}, \dot{s}, \dot{t}\,\overline{p}\}$. Two transitions t from the state 135 (dashed in Figure 3.12(b)) are present in the system in the classical semantics, as interaction p is not available in this state. However, p is offered at this state and interaction $\dot{t}\,\overline{p}$ does not allow transitions from this state. The extended interaction model has to be enlarged with the interaction $\dot{t}\,\overline{q}\,\overline{s}$, which allows transitions t from the state 135 and does not add transitions from the state 145, as s is offered at that state. Thus, the hierarchy of extended interaction models $\gamma_1' = \{\dot{p}\,\overline{r}, \dot{q}, \dot{r}, \dot{s}\}$ and $\gamma_2' = \{\dot{p}, \dot{q}, \dot{r}, \dot{s}, \dot{t}\,\overline{p}, \dot{t}\,\overline{q}\,\overline{s}\}$ generates the equivalent composite system.

3.3 Extensions of Interaction Model Representations

Algebras representing BIP interaction model (Section 2.2.2) do not involve priority model and, consequently, cannot synthesise full BIP glue from Boolean constraints. The BIP offer semantics allows to replace a combination of interaction and priority models with an extended interaction model. In this section, we extend the algebras so that they could represent extended interaction model, thus allowing to synthesise connectors with priorities from Boolean constraints.

Recall that, for any algebra $\mathcal{A}(P)$ in Section 2.2.2, we define the equivalence on $\mathcal{A}(P)$ by putting, for $x, y \in \mathcal{A}(P)$, $x \simeq y$ iff $\|x\| = \|y\|$, where $\|\cdot\| : \mathcal{A}(P) \to 2^{2^P}$ is the interaction semantics of the algebra. As a simple corollary of the results in [21], $\|x\| = \|y\|$ is equivalent to $\|x\|(\mathbf{B}) = \|y\|(\mathbf{B})$, for any finite family of components $\mathbf{B}$ (where $\|x\|(\mathbf{B})$ denotes the application of an interaction model $\|x\|$ to the set of components $\mathbf{B}$). However, this is not the case for extended interaction models, where $\|x\| = \|y\|$ implies $\|x\|(\mathbf{B}) = \|y\|(\mathbf{B})$, for any finite family of components $\mathbf{B}$, but the converse implication does not hold (cf. Lemma 2.3.5).

Definition 3.3.1. Let $\mathcal{A}(P)$ be an algebra, $\|\cdot\| : \mathcal{A}(P) \to 2^{2^P}$. Two terms $x, y \in \mathcal{A}(P)$ are *equivalent* $x \sim y$ iff, for any finite family of components $\mathbf{B}$, $\|x\|(\mathbf{B}) = \|y\|(\mathbf{B})$.

The Algebra of Interactions, $\mathcal{AI}(P)$, extends in a straightforward manner. Indeed, it is sufficient to consider $\mathcal{AI}(P \cup \dot{P} \cup \overline{P})$ with the equivalence $\sim$.

$$\begin{tikzcd}
& \dot{p} \arrow[d] & \\
& \overline{q} \arrow[dl] \arrow[dr] & \\
\dot{r} & & \dot{s}
\end{tikzcd}
\qquad
\begin{tikzcd}
& \dot{p} \arrow[dl] \arrow[dr] & \\
\overline{q}\,\dot{r} & & \overline{q}\,\dot{s}
\end{tikzcd}$$

Figure 3.13 – A pair of equivalent causal interaction trees.

We can now similarly extend the other algebras. The interaction semantics of the causal interaction trees $|\cdot| : \mathcal{T}(P) \to \mathcal{AI}(P)$ is transposed without any change to $|\cdot| : \mathcal{T}(P \cup \dot{P} \cup \overline{P}) \to \mathcal{AI}(P \cup \dot{P} \cup \overline{P})$. Similarly, the functions $\tau : \mathcal{AC}(P) \to \mathcal{T}(P)$ and $\sigma : \mathcal{T}(P) \to \mathcal{AC}(P)$ are transposed identically to $\mathcal{AC}(P \cup \dot{P} \cup \overline{P})$ and $\mathcal{T}(P \cup \dot{P} \cup \overline{P})$. The same goes for the mapping $R(t)$ associating to a causal interaction tree $t \in \mathcal{T}(P)$ the corresponding system of causal rules [22]. The only difference is that in $\mathcal{CR}(P \cup \dot{P} \cup \overline{P})$ we introduce the following additional axiom: $\dot{p} \Rightarrow p$, for all $p \in P$ (cf. the discussion leading up to (2.27)).

The first consequence of this extension is that, rather than extending the existing graphical representation of connectors, it can be used as is to express priorities and activation conditions (the use of the offer predicate in the positive premises of the rule (2.26)) by adding a trivalued attribute to ports: *firing*, *activation* and *negative*. It is important to observe the difference between, on one hand, adding an attribute to ports and, on the other hand, modifying the typing operator (*synchron* vs. *trigger* typing), since the latter is applied at each level of the connector hierarchy, whereas the former is applied to ports, that is only at the leaves of the connector.

Example 3.3.2. Let $P = \{p, q, r, s\}$ and consider the (non-extended) interaction model $\gamma = \{p, q, pr, ps, prs\}$ and the priority model $\pi = \{pr \prec q, ps \prec q, prs \prec q\}$. The glue operator $\pi\gamma$ can be equivalently represented in the extended algebras as follows. The corresponding extended interaction model is $\{\dot{p},\, \dot{q},\, \dot{p}\,\overline{q}\,\dot{r}\,,\, \dot{p}\,\overline{q}\,\dot{s}\,,\, \dot{p}\,\overline{q}\,\dot{r}\,\dot{s}\}$, which can be represented by the union of two extended connectors: $\dot{q} + \dot{p}'[\overline{q}'\,\dot{r}\,\dot{s}]$ or, equivalently, $\dot{q} + \dot{p}'[\dot{r}\,\overline{q}][\dot{s}\,\overline{q}]$. The causal interaction trees corresponding to the second summands in these connectors are shown in Figure 3.13.

3.3.1 Refinement of extension

When we use $x, y \in \mathcal{AI}(P \cup \dot{P} \cup \overline{P})$ to compose components in offer semantics (Definition 2.3.4), $\|x\|(\mathbf{B}) = \|y\|(\mathbf{B})$ does not imply $x = y$. $\mathcal{AI}$ axioms are not complete (although still sound) with respect to $\sim$, since this equivalence is weaker than $\simeq$. Consequently, on $\mathcal{T}(P \cup \dot{P} \cup \overline{P})$, $\sim$ is also weaker than $\simeq$.

Example 3.3.3. Let $P = \{p, q, r, s\}$ and consider the $\mathcal{T}(P \cup \dot{P} \cup \overline{P})$ trees shown in Figure 3.13.

The left causal interaction tree in Figure 3.13 defines a redundant interaction. Indeed,

$$\|\dot{p} \to \overline{q} \to (\dot{r} \oplus \dot{s})\| \;=\; \{\dot{p},\, \dot{p}\,\overline{q}\,,\, \dot{p}\,\overline{q}\,\dot{r}\,,\, \dot{p}\,\overline{q}\,\dot{s}\,,\, \dot{p}\,\overline{q}\,\dot{r}\,\dot{s}\,\}\;.$$

Although the interaction $\dot{p}\,\overline{q}$ does contain a firing port $\dot{p}$, it is redundant (Lemma 2.3.5). We conclude, therefore, that the causal interaction trees in Figure 3.13 are equivalent, since

$$\|\dot{p} \to (\overline{q}\,\dot{r} \oplus \overline{q}\,\dot{s})\| \;=\; \{\dot{p},\, \dot{p}\,\overline{q}\,\dot{r}\,,\, \dot{p}\,\overline{q}\,\dot{s}\,,\, \dot{p}\,\overline{q}\,\dot{r}\,\dot{s}\,\}\;.$$

The above example illustrates the idea that the nodes of causal interaction trees, which do not contain firing ports, can be "pushed" down the tree.

Another notable case leading to redundant interactions corresponds to trees containing *contradictory port typings*. For example, either of the two equivalent trees $\overline{p} \to \dot{p}$ and $\overline{p}\,\dot{p}$ authorises the interaction $\overline{p}\,\dot{p}$. However, when considered in the context of the rule (2.26), this interaction generates two conflicting premises $q_i \xrightarrow{p} q_i'$ and $q_i \not\uparrow p$. Thus, this instance of the rule (2.26) does not authorise any transitions and the interaction $\overline{p}\,\dot{p}$ can be safely discarded. This example corresponds to the additional axiom $\dot{p} \Rightarrow p$ imposed in [23] on the Boolean formulas in $\mathbb{B}[P, \dot{P}]$. Similarly, redundant interactions appear when a tree contains other distinct port typings of the same port: p and $\overline{p}$, generating conflicting premises $q_i\!\uparrow\! p$ and $q_i \not\uparrow p$; p and $\dot{p}$, whereof the former generates the premise $q_i\!\uparrow\! p$ redundant alongside the premise $q_i \xrightarrow{p} q_i'$ generated by the latter.

Below, we provide a set of axioms reducing interaction redundancy. We enrich axioms for $\mathcal{T}(P \cup \dot{P} \cup \overline{P})$ from [22] by adding some new ones, specific for the trivalued port attribute.

Axioms.

1. For all $p \in P$ and $a \subseteq P \cup \dot{P} \cup \overline{P}$, such that $a \neq \emptyset$,
   (a) $a \cdot 0 = 0$,
   (b) $a \cdot 1 = a$, for $a \neq 0$,
   (c) $\dot{p} \cdot p = \dot{p}$ (cf. the additional axiom $\dot{p} \Rightarrow p$ in $\mathbb{B}(P \cup \dot{P})$),
   (d) $\dot{p} \cdot \overline{p} = p \cdot \overline{p} = 0$.
2. Parallel composition, '$\oplus$', is associative, commutative, idempotent, and its identity element is 0.
3. $a \to 0 = a$, for all $a \subseteq P \cup \dot{P} \cup \overline{P}$.
4. $0 \to t = 0$, for all $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$.
5. $ap \to b = ap \to bp$ for all $a, b \subseteq P \cup \dot{P} \cup \overline{P}$, $p \in P \cup \dot{P} \cup \overline{P}$.

6. $c \to a \to b \to t = c \to ab \to t$ for all $a, b, c \subseteq P \cup \dot{P} \cup \overline{P}$, such that $\mathbf{fire}(a) = \emptyset$, and $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$.

7. $a \to (t_1 \oplus t_2) = a \to t_1 \ \oplus \ a \to t_2$, for all $a \subseteq P \cup \dot{P} \cup \overline{P}$, $t_1, t_2 \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$.

Axioms 1 equalise redundant interactions due to contradictory port typings, whereas Axiom 5 propagates ports down in order to find contradictory port typings. Axiom 6 eliminates the nodes with empty firing support. Axioms 2, 3, 4 and 7 are the same as in [22]. The two remaining axioms from [22] are replaced by Lemmata 3.3.6 and 3.3.7.

Proposition 3.3.4. *The equivalence relation* $\sim$ *on* $\mathcal{T}(P \cup \dot{P} \cup \overline{P})$ *is a congruence.*

Proof. The proof is the same as for $\mathcal{T}(P)$ [22]. For any two trees $t_1, t_2 \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$ and for any context $C(z) \in \mathcal{T}(P \cup \dot{P} \cup \overline{P} \cup \{z\})$, we have to show that the equivalence $t_1 \sim t_2$ implies $C(t_1/z) \sim C(t_2/z)$, where $C(t_i/z)$ is the tree obtained, by replacing in $C(z)$ all occurrences of z by t_i. Since the semantics $\mathcal{T}$ is compositional, structural induction on the context $C(z)$ proves the proposition. Thus, we need to prove two implications: $t_1 \sim t_2 \Rightarrow (a \to t_1) \sim (a \to t_2)$ and $t_1 \sim t_2 \Rightarrow (t_1 \oplus t) \sim (t_2 \oplus t)$.

- Let $\gamma_i \overset{def}{=} \|a \to t_i\| = \|a(1 + |t_i|)\| = \{a\} \cup \{a \cup a_i \mid a_i \in \|t_i\|\}$, for $i \in \{1, 2\}$. Consider a set of components $B_1, \ldots, B_n$. For $\gamma_1(B_1, \ldots, B_n)$ and $\gamma_2(B_1, \ldots, B_n)$, their sets of states and ports, as well as the corresponding offer predicates, coincide. We have to check that so do their transition relations.

  Let $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(b)} q_1' \ldots q_n'$ in $\gamma_1(B_1, \ldots, B_n)$. Thus, there is an interaction $b \in \gamma_1$, such that $q_j \xrightarrow{\mathbf{fire}(b) \cap P_j} q_j'$ if $\mathbf{fire}(b) \cap P_j \neq \emptyset$ and $q_j = q_j'$ otherwise, $q \uparrow \mathbf{act}(b)$ and $q \not\uparrow p$ for all $p \in \mathbf{neg}(b)$. If $b = a$, then $b \in \gamma_2$, and, since all conditions are satisfied, $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(b)} q_1' \ldots q_n'$ in $\gamma_2(B_1, \ldots, B_n)$. If $b \neq a$ then $b = a \cup a_1$, where $a_1 \in \|t_1\|$. Notice that $q_1 \ldots q_n \uparrow \mathbf{act}(a) \cup \mathbf{act}(a_1)$ and $q_1 \ldots q_n \not\uparrow p$ for all $p \in \mathbf{neg}(a) \cup \mathbf{neg}(a_1)$. However, this does not necessary mean that $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(a_1)}$, since, for some component B_j, it could be $q_j \xrightarrow{\mathbf{fire}(a \cup a_1) \cap P_j}$, but $q_j \not\xrightarrow{\mathbf{fire}(a_1) \cap P_j}$. Consider a set of components $B_1', \ldots, B_n'$ that has the same states, interfaces and offer predicates as $B_1, \ldots, B_n$ but their transition relations have extra transitions $q_j \xrightarrow{\mathbf{fire}(a_1) \cap P_j} q_j'$, for $j \in [1..n]$, if $\mathbf{fire}(a_1) \cap P_j \neq \emptyset$ (if B_j already has this transition, then $B_j = B_j'$). These additional transitions ensure $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(a_1)}$ in $\|t_1\|(B_1', \ldots, B_n')$. Since $t_1 \sim t_2$, $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(a_2)}$ in $\|t_2\|(B_1', \ldots, B_n')$ for some $a_2 \in \|t_2\|$, such that $\mathbf{fire}(a_1) = \mathbf{fire}(a_2)$ and, consequently, $\mathbf{fire}(a \cup a_1) = \mathbf{fire}(a \cup a_2)$. By (2.25), $q_1 \ldots q_n \uparrow \mathbf{act}(a_2)$ and $q_1 \ldots q_n \not\uparrow p$ for all $p \in \mathbf{neg}(a)$. Recalling that $q_1 \ldots q_n \uparrow \mathbf{act}(a)$, $q_1 \ldots q_n \not\uparrow p$ for all $p \in \mathbf{neg}(a_2)$ and $q_j \xrightarrow{\mathbf{fire}(a \cup a_2) \cap P_j} q_j'$ if $\mathbf{fire}(b) \cap P_j \neq \emptyset$, all premises for the interaction $a \cup a_2$ are satisfied and $q_1 \ldots q_n \xrightarrow{\mathbf{fire}(a \cup a_2)} q_1' \ldots q_n'$ in $\gamma_2(B_1, \ldots, B_n)$. The proof that for any transition in $\gamma_2(B_1, \ldots, B_n)$ there is a transition in $\gamma_1(B_1, \ldots, B_n)$ is similar.

- Let $\gamma_i \overset{def}{=} \|t_i \oplus t\| = \|t_i\| \cup \|t\| \cup \{a_i \cup a | a_i \in \|t_i\|, a \in \|t\|\}$. Consider a set of components $B_1, \dots, B_n$. For $\gamma_1(B_1, \dots, B_n)$ and $\gamma_2(B_1, \dots, B_n)$, their sets of states and ports, as well as the corresponding offer predicates, coincide. We have to check that so do their transition relations.
  Let $q_1 \dots q_n \xrightarrow{\mathbf{fire}(b)} q'_1 \dots q'_n$ in $\gamma_1(B_1, \dots, B_n)$. Thus, there is an interaction $b \in \gamma_1$, such that $q_j \xrightarrow{\mathbf{fire}(b) \cap P_j} q'_j$ if $\mathbf{fire}(b) \cap P_j \neq \emptyset$ and $q_j = q'_j$ otherwise, $q \uparrow \mathbf{act}(b)$ and $q \not\uparrow p$ for all $p \in \mathbf{neg}(b)$. If $b \in \|t\|$, then $b \in \gamma_2$ and $q_1 \dots q_n \xrightarrow{\mathbf{fire}(b)} q'_1 \dots q'_n$ in $\gamma_2(B_1, \dots, B_n)$. If $b \in \|t_1\|$, then $q_1 \dots q_n \xrightarrow{\mathbf{fire}(b)}$ in $\|t_1\|(B_1, \dots, B_n)$. Since $t_1 \sim t_2$, $q_1 \dots q_n \xrightarrow{\mathbf{fire}(b)}$ in $\|t_2\|(B_1, \dots, B_n)$ and, consequently, in $\gamma_2(B_1, \dots, B_n)$. If $b = a_1 \cup a$ for some $a_1 \in \|t_1\|$ and $a \in \|t\|$, then the proof is similar to the case $a \to t$. We consider components $B'_1, \dots, B'_n$, such that $q_1 \dots q_n \xrightarrow{\mathbf{fire}(a_1)}$ in $\|t_1\|(B'_1, \dots, B'_n)$. Since $t_1 \sim t_2$, $q_1 \dots q_n \xrightarrow{\mathbf{fire}(a_2)}$ in $\|t_2\|(B'_1, \dots, B'_n)$ for some $a_2 \in \|t_2\|$. All premises for the interaction $a \cup a_2$ are satisfied in the state $q_1 \dots q_n$, therefore $q_1 \dots q_n \xrightarrow{\mathbf{fire}(a_2 \cup a)} q'_1 \dots q'_n$ in $\gamma_2(B_1, \dots, B_n)$. The proof that for any transition in $\gamma_2(B_1, \dots, B_n)$ there is a transition in $\gamma_1(B_1, \dots, B_n)$ is similar.

We have shown that for $t_1 \sim t_2$ holds $a \to t_1 \sim a \to t_2$ and $t_1 \oplus t \sim t_2 \oplus t$. Thus, we can conclude that for any context $C(z)$, $C(t_1/z) \sim C(t_2/z)$. □

Notice that the same example as in [19] illustrates the fact that neither $\sim$ nor $\simeq$ are congruences on $\mathcal{AC}(P \cup \dot{P} \cup \overline{P})$. Indeed, we clearly have $p' \sim p$ and $p'q \not\sim pq$, for any distinct $p, q \in P \cup \dot{P} \cup \overline{P}$.

Proposition 3.3.5. *The above axiomatisation is sound with respect to $\sim$.*

Proof. Since, by Proposition 3.3.4, the equivalence relation $\sim$ is a congruence, it is sufficient to show that all axioms respect $\sim$. This is proved by verifying that the semantics for left and right sides coincide.

Axioms 2, 3, 4 and 7 are the same as in [22]. Hence, their respective left- and right-hand sides are related by $\simeq$, which is stronger than $\sim$. Axiom 1(a) and Axiom 1(b) are trivial. Axiom 1(c) is a consequence of Lemma 2.3.5. In the Axiom 1(d), both pairs p and $\overline{p}$, and $\dot{p}$ and $\overline{p}$ produce conflicting premises in the rule (2.26) and, therefore, do not generate any transitions. For the Axiom 6, we have

$$\|c \to a \to b \to t\| = \{c,\, a\,c,\, a\,b\,c\} \cup \{a\,b\,c\,a_2 \,|\, a_2 \in \|t\|\}\,, \tag{3.14}$$

$$\|c \to ab \to t\| = \{c,\, a\,b\,c\} \cup \{a\,b\,c\,a_2 \,|\, a_2 \in \|t\|\}\,. \tag{3.15}$$

Hence, $\|c \to a \to b \to t\| = \|c \to ab \to t\| \cup \{ac\}$. Since $c \subseteq ac$ and $\mathbf{fire}(a) = \emptyset$, we conclude, by Lemma 2.3.5, that the two causal interaction trees are equivalent: $c \to a \to b \to t \sim c \to ab \to t$.

For the Axiom 5, we have $\|ap \to b\| = \{ap, abp\} = \|ap \to bp\|$. Thus, $ap \to b \simeq ap \to bp$, which implies $ap \to b \sim ap \to bp$. □

Notice that, for the soundness of Axiom 6, it is essential that a is not a root node, since application of Lemma 2.3.5 is made possible by the presence of the interaction $c \in \|c \to a \to b \to t\|$. For a counter-example, consider two interaction trees $\overline{p} \to \dot{q}$ and $\overline{p}\dot{q}$. The former allows self-loops in states of a composed system, where p is not offered, whereas the latter does not.

Lemma 3.3.6. *For all $a, b \subseteq P \cup \dot{P} \cup \overline{P}$, such that* $\mathbf{fire}(b) = \emptyset$*, holds the equality $a \to b = a$.*

Proof. $a \to b = a \to b \to 0 \to 0 = a \to b \cdot 0 \to 0 = a \to 0 \to 0 = a$ (Axioms 3, 6) □

Lemma 3.3.7. *For all $a \subseteq P \cup \dot{P} \cup \overline{P}$ and $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$, holds the equality $a \to 1 \to t = a \to t$.*

Proof. For $t = 0$, the statement of this lemma is a special case of Lemma 3.3.6 with $b = 1$. If $t \neq 0$, it can be represented as a parallel composition of non-zero trees $t = \bigoplus_{i=1}^n r_i \to t_i$, with $r_i \subseteq P \cup \dot{P} \cup \overline{P}$. By Axioms 6 and 7, we have

$$a \to 1 \to t \;=\; \bigoplus_{i=1}^n (a \to 1 \to r_i \to t_i) \;=\; \bigoplus_{i=1}^n (a \to r_i \to t_i) \;=\; a \to \bigoplus_{i=1}^n (r_i \to t_i) \;=\; a \to t\,.$$

□

Lemma 3.3.8. *For all $a, b_i, c \subseteq P \cup \dot{P} \cup \overline{P}$, such that* $\mathbf{fire}(a) = \emptyset$ *and $t_i \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$, holds the equality*

$$c \to a \to \bigoplus_{i=1}^n (b_i \to t_i) = c \to \bigoplus_{i=1}^n (ab_i \to t_i)\,.$$

Proof. As above, applying Axioms 6 and 7, we have

$$c \to a \to \bigoplus_{i=1}^n (b_i \to t_i) \;=\; \bigoplus_{i=1}^n (c \to a \to b_i \to t_i) \;=\; \bigoplus_{i=1}^n (c \to ab_i \to t_i) \;=\; c \to \bigoplus_{i=1}^n (ab_i \to t_i)\,.$$

□

3.3.2 Normalisation of extended algebras

As it was shown in Example 3.3.3, causal interaction trees can contain nodes generating redundant interactions. These nodes can be removed by consecutively applying semantics-preserving transformations based on the axioms of the Algebra of Causal Interaction Trees.

Definition 3.3.9. A causal interaction tree $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$ is in *normal form* if it satisfies the following properties:

1. All nodes of t except roots have non-empty firing support.

2. In any causal chain of t a port p can appear more than once only in the form $ap \to \cdots \to b\dot{p}$, where $a, b \subseteq P \cup \dot{P} \cup \overline{P}$ and $p \in P$.

In the proof of Proposition 3.3.10 below, we provide a constructive procedure for normalising causal interaction trees.

Proposition 3.3.10. *Every causal interaction tree $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$ has a normal form $t = \tilde{t} \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$.*

Proof. Consider $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$. We start by computing $t_1 = t$ with all nodes, except potentially the roots, having non-empty firing support.

Let a be a non-root node of t with $\mathbf{fire}(a) = \emptyset$, such that the tree s rooted in a does not have any nodes with empty firing support. If s is empty, that is a is a leaf, then remove a from the tree (Lemma 3.3.6). Otherwise, let c be the parent of a, which exists since a is not a root and move the parallel composition operator up using Axiom 7:

$$c \to \left((a \to s) \oplus \bigoplus_{i=1}^{n} t_i \right) = (c \to a \to s) \oplus \left(\bigoplus_{i=1}^{n} c \to t_i \right) . \tag{3.16}$$

The sub-tree s can be further decomposed as $s = \bigoplus_{i=1}^{n}(b_i \to s_i)$, so, by Lemma 3.3.8, we have

$$c \to a \to s \;=\; c \to a \to \bigoplus_{i=1}^{n}(b_i \to s_i) \;=\; c \to \bigoplus_{i=1}^{n}(ab_i \to s_i) \,. \tag{3.17}$$

Each of nodes ab_i has non-empty firing support, since $\mathbf{fire}(b_i) \neq \emptyset$ by the choice of a. Substituting (3.17) into (3.16) and applying Axiom 7, we obtain

$$\left(c \to \bigoplus_{i=1}^{n}(ab_i \to s_i) \right) \oplus \left(\bigoplus_{i=1}^{n} c \to t_i \right) = c \to \left(\left(\bigoplus_{i=1}^{n} ab_i \to s_i \right) \oplus \bigoplus_{i=1}^{n} t_i \right) .$$

In the resulting tree, there is one node with empty firing support less than in t. Hence, repeating this procedure as long as there are such nodes, we will compute a tree t_1, where all nodes except roots have non-empty firing support. This computation is confluent, since the order is irrelevant among causally independent nodes, whereas among causally dependent ones it is fixed by the algorithm.

Consider a causal chain $a\tilde{p} \to \cdots \to b\hat{p}$ within t_1, with $\tilde{p}$ and $\hat{p}$ being two typings of the same port. If $\tilde{p} = p$ and $\hat{p} = \dot{p}$, there is nothing to do, since such dependencies are allowed by Definition 3.3.9. Otherwise, we propagate $\tilde{p}$ down by applying Axiom 5:

$$\begin{aligned} a\tilde{p} \rightarrow c_1 \rightarrow \cdots \rightarrow c_k \rightarrow b\hat{p} \;&=\; a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k \rightarrow b\hat{p} = \;\ldots \\ &=\; a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k\tilde{p} \rightarrow b\hat{p}\tilde{p}\,. \end{aligned}$$

Case 1: $\tilde{p} = \hat{p}$ or both $\tilde{p}, \hat{p} \neq \overline{p}$. We apply Axioms 1(c) and 5:

$$\begin{aligned} a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k\tilde{p} \rightarrow b\hat{p}\tilde{p} \;&=\; a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k\tilde{p} \rightarrow b\tilde{p} \\ &= a\tilde{p} \rightarrow c_1 \rightarrow \cdots \rightarrow c_k \rightarrow b\,. \end{aligned}$$

Case 2: $\tilde{p} \neq \hat{p}$ and either $\tilde{p} = \overline{p}$ or $\hat{p} = \overline{p}$. We apply Axioms 1(d), 3 and 5:

$$\begin{aligned} a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k\tilde{p} \rightarrow b\hat{p}\tilde{p} \;&=\; a\tilde{p} \rightarrow c_1\tilde{p} \rightarrow \cdots \rightarrow c_k\tilde{p} \rightarrow 0 \;= \\ =\; a\tilde{p} \rightarrow c_1 \rightarrow \cdots \rightarrow c_k \rightarrow 0 \;&=\; a\tilde{p} \rightarrow c_1 \rightarrow \cdots \rightarrow c_k\,. \end{aligned}$$

To compute $\tilde{t}$, we apply this transformation to all relevant causal chains within t_1. □

When synthesising connectors from causal interaction trees, their complexity can be reduced by tree normalisation. Furthermore, since semantics-preserving transformations can be applied in both directions, a normal form on causal interaction trees induces a normal form on connectors.

Definition 3.3.11. A connector $x \in \mathcal{AC}(P \cup \dot{P} \cup \overline{P})$ is in *normal form* iff $x = \sigma(t)$, where t is a causal interaction tree in normal form and σ is the function defined in (2.22).

The following proposition is a direct consequence of the definitions of the normal form of causal interaction trees and function σ.

Proposition 3.3.12 (Normal form for connectors)**.** *A connector $x \in \mathcal{AC}(P \cup \dot{P} \cup \overline{P})$ in normal form has the following properties:*

1. *Nodes at every hierarchical level of the connector, except the bottom one, have at least one trigger.*
2. *Each node at the bottom hierarchical level is a strong synchronisation of pairwise distinct ports.*
3. *Every node at the bottom hierarchical level without firing ports has only triggers as ancestors.*

Proof. By the definition of normal form there exists a causal interaction tree t in normal form, such that $x = \sigma(t)$.

1. Nodes at every hierarchical level of the connector, except the bottom one, are obtained by one of the two rules $\sigma(a \to t) = [a]' [\sigma(t)]$ or $\sigma(t_1 \oplus t_2) = [\sigma(t_1)]' [\sigma(t_2)]'$. Both of them create at least one trigger.

2. By (2.22), nodes at the bottom hierarchical level are nodes of the causal interaction tree in normal form. Thus, they are strong synchronisations of pairwise distinct ports.

3. A node at the bottom hierarchical level without firing ports can only be obtained from a root of the tree t. A non-trigger ancestor of a can be obtained only from a causality operator $b \to a$, which is not possible for a root.

□

3.3.3 Simplification of systems of causal rules

The port typings in the algebraic representations of extended interaction models increase the complexity of systems of causal rules: without additional simplifications, the number of rules in the system is essentially tripled. The goal of this sub-section is to prove that we can consider only rules with firing port typings or *true* as effects and other rules can be removed as redundant.

The generation of systems of causal rules from Boolean formula starts with $\phi \in \mathbb{B}[P \cup \dot{P}]$ with additional axiom $\dot{p} \Rightarrow p$. This formula is transformed into conjunctive normal form (CNF). At this point we change the domain to consider ϕ as a formula from $\mathbb{B}[P \cup \dot{P} \cup \overline{P}]$ with two additional axioms: $\dot{p} \Rightarrow p$ and $\overline{p}$ `XOR` p. Syntactically, the formula ϕ remains exactly the same, whereas semantically, we consider the negative occurrences of variables from P, i.e. $\overline{p}$ with $p \in P$, as positive occurrences of variables from $\overline{P}$ (recall that $\overline{P} = \{\overline{p} \,|\, p \in P\}$). Thus, seen as a formula from $\mathbb{B}[P \cup \dot{P} \cup \overline{P}]$, ϕ only has firing port variables in negative form. All other variables appear only in positive form.

We then proceed exactly as in [22]: we have $\phi' = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ with, for $k \in [1, n]$, $C_k = \bigvee_{i \in I_k} p_i \vee \bigvee_{j \in J_k} \overline{p_j}$, where $I_k \cap J_k = \emptyset$, $p_i \in P \cup \dot{P} \cup \overline{P}$ and $p_j \in \dot{P}$ for all $i \in I_k$ and $j \in J_k$. We can now rewrite every clause C_k, with $J_k \neq \emptyset$, as a disjunction of dual Horn clauses $C_k = \bigvee_{j \in J_k} \left(\overline{p_j} \vee \bigvee_{i \in I_k} p_i\right)$. By distributivity, we obtain a representation of ϕ' as a disjunction of dual Horn formulas and, after combining the clauses with the same negative variable, we obtain $\phi' = R_1 \vee R_2 \vee \cdots \vee R_m$ with, for $k \in [1, m]$,

$$R_k = \bigwedge_{i \in \widetilde{I}_k} \left(\overline{p_i} \vee \bigvee_{j \in \widetilde{J}_{k,i}} a_j \right) = \bigwedge_{i \in \widetilde{I}_k} \left(p_i \Rightarrow \bigvee_{j \in \widetilde{J}_{k,i}} a_j \right),$$

where, for all $i \in \widetilde{I}_k$, $p_i \in \dot{P} \cup true$ and, for all $j \in \widetilde{J}_{k,i}$, a_j is *false*, *true*, or a conjunction of positive variables. Thus, each R_k is a system of causal rules, with only firing variables in the effects.

The algorithm synthesising causal interaction trees from systems of causal rules (see Section 2.2.2) expects that the input system is complete in the sense that it should have one rule for each port variable. Thus, for each $p \in P \cup \overline{P}$ the rule $p \Rightarrow true$ has to be added to the system. However, the rules with non-firing effects do not impose additional constraints on the system. Theorem 3.3.13 shows that such rules do not affect the causal interaction tree generated from the system of causal rules. Therefore, the synthesis algorithm remains correct, even when simplified by excluding all causal rules with effect $p \in P \cup \overline{P}$.

Theorem 3.3.13. *Let R be a system of causal rules over $P \cup \dot{P} \cup \overline{P}$, where all rules with the effect $p \in P \cup \overline{P}$ have the form $p \Rightarrow true$, and let R' be a set of causal rules containing only rules from R with effects $p \in \dot{P} \cup true$. Then, the causal interaction trees, generated for R and R' with the procedure described in Section 2.2.2, are equivalent with respect to $\sim$.*

Proof. The construction of causal interaction trees consists of two steps: saturation of the system of causal rules and building the tree. Clearly, rules of the form $p \Rightarrow true$ do not affect the saturation of other rules. On the other hand, such rules are saturated to $p \Rightarrow C$, where C is the saturated cause of the rule with the effect *true*.

Let Y and Y' be the auxiliary sets (2.24) containing monomials of the causes composed with the effects of the corresponding rules, for R and R' respectively. Clearly, $Y' \subseteq Y$ and $Y \setminus Y' \subseteq \{pc \mid p \in P \cup \overline{P}, c \in Y' \cup \{\emptyset\}\}$.[2] Hence, in the inclusion tree corresponding to R, elements of $Y \setminus Y'$ can generate additional nodes compared to the inclusion tree corresponding to R'. Every such node necessarily appears in a context of the form $c \to pc \to \bigoplus(pcq_i \to t_i)$ for some port variables q_i and sub-trees t_i. However, by Axioms 1 and 6, $c \to pc \to \bigoplus(pcq_i \to t_i) = c \to \bigoplus(pcq_i \to t_i)$, which is a fragment of the tree corresponding to R'. □

The complexity of causal interaction tree synthesis algorithm [22] greatly depends on the number of rules in the system. Indeed, the saturation phase consists in substituting each port in the cause part of each rule with the cause of the corresponding rule, where this port is the effect. This is repeated until a fixpoint is reached. Theorem 3.3.13 removes two thirds of the rules, thus greatly reducing the synthesis complexity.

We have shown above that, while synthesising causal interaction trees from Boolean formulas, we can discard rules with non-firing effects in the intermediate systems of causal rules. Theorem 3.3.14 below shows that we can also discard rules with non-firing effects, when generating systems of causal rules from causal interaction trees, thus considerably reducing the obtained Boolean formulas.

Theorem 3.3.14. *Consider a causal interaction tree $t \in \mathcal{T}(P \cup \dot{P} \cup \overline{P})$ and a system of causal rules $R(t) = \{p \Rightarrow C_p(t)\}_{p \in P \cup \dot{P} \cup \overline{P} \cup \{true\}}$ obtained by the transformation $R : \mathcal{T}(P \cup \dot{P} \cup \overline{P}) \to \mathcal{CR}(P \cup \dot{P} \cup \overline{P})$ described in Section 2.2.2. Let $\tilde{R}(t) = \{p \Rightarrow C_p(t)\}_{p \in \dot{P} \cup \{true\}}$ be a system of*

[2] It is possible that $c = \emptyset$ if the rule for the effect *true* is $true \Rightarrow true$. Recall that the empty interaction corresponds to 1 in the algebra of Causal Interaction Trees.

causal rules, obtained from $R(t)$ by omitting rules for port variables in $P \cup \overline{P}$. Then holds the equivalence $R(t) \sim \tilde{R}(t)$.

Proof. Recall that the application of the transformation $R : \mathcal{T}(P) \to \mathcal{CR}(P)$ described in Section 2.2.2 to a tree $t \in \mathcal{T}(P)$ gives a system of causal rules of the form $p \Rightarrow C$, where C is a DNF Boolean formula and each monomial is a conjunction of the nodes on the way from a root of t to p (some prefix in t leading to p, excluding p).

$\tilde{R}(t)$ has less constraints than $R(t)$. Hence, it allows more interactions, i.e. $\|R(t)\| \subseteq \|\tilde{R}(t)\|$. Let $a \in \|\tilde{R}(t)\| \setminus \|R(t)\|$, i.e. there exists $p \in P \cup \overline{P}$, such that $p \in a$ and the rule $p \Rightarrow C_1$ is violated by a. First of all, notice that this implies immediately that $a \neq \emptyset$. Furthermore, we have $\emptyset \in \|R(t)\| \Leftrightarrow \emptyset \in \|\tilde{R}(t)\|$. Let $\tilde{a} = a \setminus p$.

Assume that $\tilde{a} \notin \|\tilde{R}(t)\|$, i.e. there exists $\dot{q} \in \dot{P}$ and a rule $(\dot{q} \Rightarrow C_2) \in \tilde{R}(t)$, such that $\dot{q} \in \tilde{a}$ and the rule $\dot{q} \Rightarrow C_2$ is violated by $\tilde{a}$. This rule is not violated by a. Hence, $C_2 = pC_2'$ and, consequently, p lies on all prefixes in t, leading to $\dot{q}$. $a \in \|\tilde{R}(t)\|$ and $\dot{q} \in \tilde{a} \subseteq a$, thus there is at least one prefix in t, leading to $\dot{q}$ and contained in a. As p lies on this prefix, the rule $(p \Rightarrow C_1)$ is satisfied by a, contradicting the conclusion above. Therefore our assumption is wrong and $\tilde{a} \in \|\tilde{R}(t)\|$. Notice that $\tilde{a}$ is a proper subset of a (i.e. $\tilde{a} \subset a$ and $\tilde{a} \neq a$) and $\mathbf{fire}(\tilde{a}) = \mathbf{fire}(a)$.

If $\tilde{a} \notin \|R(t)\|$, we can apply the same reasoning to $\tilde{a}$ to obtain $\tilde{\tilde{a}} \in \|\tilde{R}(t)\|$, a proper subset of $\tilde{a}$ with $\mathbf{fire}(\tilde{\tilde{a}}) = \mathbf{fire}(\tilde{a}) = \mathbf{fire}(a)$ and so on. Thus, we obtain a strictly decreasing (in terms of set inclusion) chain of extended interactions belonging to $\|\tilde{R}(t)\|$, each having the firing support $\mathbf{fire}(a)$. Since $\|\tilde{R}(t)\|$ is finite, this chain must also be finite. Let $a' \in \|\tilde{R}(t)\|$ be the last (smallest) element in the chain. Again, a' is a proper subset of a and $\mathbf{fire}(a') = \mathbf{fire}(a)$. Since a' is the last element of the chain, we deduce that the above reasoning cannot be applied to it, which means that $a' \in \|R(t)\|$.

Thus, for all $a \in \|\tilde{R}(t)\| \setminus \|R(t)\|$, there exists $a' \subseteq a$, such that $\mathbf{fire}(a') = \mathbf{fire}(a)$ and $a' \in \|R(t)\|$. Hence, by Lemma 2.3.5, we have $\|\tilde{R}(t)\|(\mathbf{B}) = \|R(t)\|(\mathbf{B})$, for any finite family of components $\mathbf{B}$, i.e. $R(t) \sim \tilde{R}(t)$. ☐

3.4 Connector Synthesis Example

In order to synthesise $\mathcal{AC}(P \cup \dot{P} \cup \overline{P})$ connectors from $\mathbb{B}[P \cup \dot{P}]$ constraints, one must perform the following steps.

1. Take the conjunction of all the constraints;
2. By adding the axioms $\dot{p} \Rightarrow p$ and p `XOR` $\overline{p}$, transform the obtained formula into a set of systems of causal rules over $P \cup \dot{P} \cup \overline{P}$, as described in the previous section;
3. Saturate the obtained systems of causal rules;

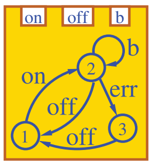


Figure 3.14 – Main module for Example 3.4.1.

4. Convert each saturated system of causal rules into a causal interaction tree;

5. Normalise all trees;

6. Generate corresponding connectors from causal interaction trees.

This procedure is illustrated by the following example.

Example 3.4.1. Consider a system providing some given functionality in two modes: *normal* and *backup*. The system consists of four modules: the Backup module A can only perform one action a; the Main module B (Figure 3.14) can perform an action b corresponding to the normal mode activity, it can also be switched on and off, as well as perform an internal (unobservable) error transition err; the Monitor module M is a black box, which performs some internal logging by observing the two actions a and b through the corresponding ports a_l and b_l; finally, the black box Controller module C takes the decisions to switch on or off the main module through the corresponding ports on_c and off_c, furthermore, it can perform a $test$ to check whether the main module can be restarted.

We want to synthesise connectors ensuring the properties below (encoded by Boolean constraints).

- The main and backup actions must be logged: $\dot{a} \Leftrightarrow \dot{a}_l$ and $\dot{b} \Leftrightarrow \dot{b}_l$;
- Only Controller can turn on the Main module: $\dot{on} \Leftrightarrow \dot{on}_c$;
- When Controller switches off, the Main module must stop operation: $\dot{off}_c \Rightarrow \dot{off}$ and $\dot{b} \Rightarrow \overline{\dot{off}_c}$;
- Controller can only test the execution of Backup: $\dot{test} \Rightarrow \dot{a}$;
- Backup can only execute when Main is not possible: $\dot{a} \Rightarrow \overline{b} \vee \dot{off}$;
- Main can only switch off when ordered to do so or after a failure: $\dot{off} \Rightarrow \overline{b} \vee \dot{off}_c$;

In order to compute the required glue, we take the conjunction of the above constraints together with the *progress* constraint $\dot{a} \vee \dot{b} \vee \dot{on} \vee \dot{off} \vee \dot{test} \vee \dot{a}_l \vee \dot{b}_l \vee \dot{off}_c \vee \dot{on}_c$ stating that at every round some action must be taken. Notice that, combined with the above constraints, the progress constraint can be immediately simplified by absorption to $\dot{a} \vee \dot{b} \vee \dot{on} \vee \dot{off}$. In order to simplify the resulting connectors, we also use part of the information about the behaviour of the Main module, namely the fact that on, on one hand, and b or off, on the other, are mutually exclusive: $on \Rightarrow \overline{b} \wedge \overline{off}$. We obtain the following global constraint:

$$\begin{aligned}&(\dot{a} \Rightarrow \dot{a}_l\,\overline{b} \;\vee\; \dot{a}_l\, o\dot{f}f) \wedge (\dot{a}_l \Rightarrow \dot{a}) \wedge (\dot{b} \Rightarrow \dot{b}_l\, \overline{o\dot{f}f_c}) \wedge (\dot{b}_l \Rightarrow \dot{b})\\ &\quad \wedge (o\dot{f}f \Rightarrow \overline{b} \;\vee\; o\dot{f}f_c) \wedge (o\dot{f}f_c \Rightarrow o\dot{f}f) \wedge (te\dot{s}t \Rightarrow \dot{a}) \wedge (o\dot{n} \Rightarrow o\dot{n}_c)\\ &\qquad\qquad \wedge (o\dot{n}_c \Rightarrow o\dot{n}) \wedge (on \Rightarrow \overline{b}\;\overline{off}) \wedge (\dot{a} \;\vee\; \dot{b} \;\vee\; o\dot{n} \;\vee\; o\dot{f}f)\,.\end{aligned}$$

Recall now that causal rules must have the form $p \Rightarrow C$, where $p \in \dot{P} \cup \{true\}$ and C is a DNF Boolean formula on $P \cup \dot{P} \cup \overline{P}$ without negative firing variables or a logical constant. A system of causal rules is a conjunction of such clauses. Among the constraints above, there are two that do not have this form: $on \Rightarrow \overline{b}\;\overline{off}$ and $\dot{b} \Rightarrow \dot{b}_l\;\overline{o\dot{f}f_c}$. We rewrite them as $\overline{on} \vee \overline{b}\;\overline{off}$ and $\overline{\dot{b}} \vee \dot{b}_l\;\overline{o\dot{f}f_c}$, and distribute over the conjunction of the rest of the constraints. Finally, we implicitly apply the additional axioms $\dot{p} \Rightarrow p$ and p `XOR` $\overline{p}$ and, making some straightforward simplifications, obtain a disjunction of three systems of causal rules. In Table 3.1, these systems are shown in the first column and their corresponding saturated equivalents are shown in the second column.

The corresponding auxiliary sets (2.24) obtained by combining the effects with the causes are then:

$$\begin{aligned}&\{\dot{a}\,\dot{a}_l\,\overline{b}\,\overline{off},\ o\dot{n}\,o\dot{n}_c\,\overline{b}\,\overline{off}\ \dot{a}\,\dot{a}_l\,\overline{b}\,\overline{off}\,te\dot{s}t\}\,, \quad \{\dot{b}\,\dot{b}_l\,\overline{on}\}\,,\\ &\{\dot{a}\dot{a}_l\overline{b}\overline{on},\ \dot{a}\dot{a}_l\overline{b}\overline{on}te\dot{s}t,\ o\dot{f}f\overline{b}\overline{on},\ o\dot{f}f\,o\dot{f}f_c\overline{on},\ \dot{a}\dot{a}_l o\dot{f}f\,o\dot{f}f_c\overline{on},\ \dot{a}\dot{a}_l o\dot{f}f\,o\dot{f}f_c\overline{on}te\dot{s}t\}\,.\end{aligned}$$

$\mathcal{T}(P \cup \dot{P} \cup \overline{P})$ trees, shown on Figure 3.15, are obtained by normalising the inclusion trees corresponding to these sets. Applying (2.22) we obtain $\mathcal{AC}(P \cup \dot{P} \cup \overline{P})$ connectors in Figure 3.16.

In terms of classical BIP, one can, for example, easily distinguish here two priorities: $x\ a\ a_l \prec b\ b_l$ and $x\ off \prec b\ b_l$ for all x not containing $off\ off_c$. In general, priorities are replaced by local inhibitors. In this example, these appear to characterise states of the Main module. For instance, $\dot{a}\ \dot{a}_l\ \overline{b}\ \overline{off}$ defines possible interactions involving $a\ a_l$ when neither b nor off are possible, i.e. in state 1 (see Figure 3.14).

Table 3.1 – Systems of causal rules for Example 3.4.1.

true	$\Rightarrow \dot{a}\,\overline{b}\,\overline{off} \lor \dot{on}\,\overline{b}\,\overline{off}$			true	$\Rightarrow \dot{a}\,\dot{a}_l\,\overline{b}\,\overline{off} \lor \dot{on}\,\dot{on}_c\,\overline{b}\,\overline{off}$		
$\dot{a}$	$\Rightarrow \dot{a}_l\,\overline{b} \lor \dot{a}_l\,\dot{off}$			$\dot{a}$	$\Rightarrow \dot{a}_l\,\overline{b}\,\overline{off}$		
$\dot{a}_l$	$\Rightarrow \dot{a}$	$\dot{b}$	$\Rightarrow \mathit{false}$	$\dot{a}_l$	$\Rightarrow \dot{a}\,\overline{b}\,\overline{off}$	$\dot{b}$	$\Rightarrow \mathit{false}$
$\dot{on}$	$\Rightarrow \dot{on}_c$	$\dot{b}_l$	$\Rightarrow \dot{b}$	$\dot{on}$	$\Rightarrow \dot{on}_c\,\overline{b}\,\overline{off}$	$\dot{b}_l$	$\Rightarrow \mathit{false}$
$\dot{on}_c$	$\Rightarrow \dot{on}$	$\dot{off}$	$\Rightarrow \mathit{false}$	$\dot{on}_c$	$\Rightarrow \dot{on}\,\overline{b}\,\overline{off}$	$\dot{off}$	$\Rightarrow \mathit{false}$
$\dot{test}$	$\Rightarrow \dot{a}$	$\dot{off}_c$	$\Rightarrow \dot{off}$	$\dot{test}$	$\Rightarrow \dot{a}\,\dot{a}_l\,\overline{b}\,\overline{off}$	$\dot{off}_c$	$\Rightarrow \mathit{false}$
true	$\Rightarrow \dot{b}\,\dot{b}_l\,\overline{on}$			true	$\Rightarrow \dot{b}\,\dot{b}_l\,\overline{on}$		
$\dot{a}$	$\Rightarrow \dot{a}_l\,\overline{b} \lor \dot{a}_l\,\dot{off}$			$\dot{a}$	$\Rightarrow \mathit{false}$		
$\dot{a}_l$	$\Rightarrow \dot{a}$	$\dot{b}$	$\Rightarrow \mathit{true}$	$\dot{a}_l$	$\Rightarrow \mathit{false}$	$\dot{b}$	$\Rightarrow \dot{b}_l\,\overline{on}$
$\dot{on}$	$\Rightarrow \mathit{false}$	$\dot{b}_l$	$\Rightarrow \dot{b}$	$\dot{on}$	$\Rightarrow \mathit{false}$	$\dot{b}_l$	$\Rightarrow \dot{b}\,\overline{on}$
$\dot{on}_c$	$\Rightarrow \dot{on}$	$\dot{off}$	$\Rightarrow \overline{b} \lor \dot{off}_c$	$\dot{on}_c$	$\Rightarrow \mathit{false}$	$\dot{off}$	$\Rightarrow \mathit{false}$
$\dot{test}$	$\Rightarrow \dot{a}$	$\dot{off}_c$	$\Rightarrow \mathit{false}$	$\dot{test}$	$\Rightarrow \mathit{false}$	$\dot{off}_c$	$\Rightarrow \mathit{false}$
true	$\Rightarrow \dot{a}\,\overline{on} \lor \dot{off}\,\overline{on}$			true	$\Rightarrow \dot{a}\,\dot{a}_l\,\overline{b}\,\overline{on} \lor \dot{off}\,\dot{off}_c\,\overline{on} \lor \dot{off}\,\overline{b}\,\overline{on}$		
$\dot{a}$	$\Rightarrow \dot{a}_l\,\overline{b} \lor \dot{a}_l\,\dot{off}$			$\dot{a}$	$\Rightarrow \dot{a}_l\,\overline{b}\,\overline{on} \lor \dot{a}_l\,\dot{off}\,\dot{off}_c\,\overline{on}$	$\dot{b}$	$\Rightarrow \mathit{false}$
$\dot{a}_l$	$\Rightarrow \dot{a}$	$\dot{b}$	$\Rightarrow \mathit{false}$	$\dot{a}_l$	$\Rightarrow \dot{a}\,\overline{b}\,\overline{on} \lor \dot{a}\,\dot{off}\,\dot{off}_c\,\overline{on}$	$\dot{b}_l$	$\Rightarrow \mathit{false}$
$\dot{on}$	$\Rightarrow \mathit{false}$	$\dot{b}_l$	$\Rightarrow \dot{b}$	$\dot{off}$	$\Rightarrow \overline{b}\,\overline{on} \lor \dot{off}_c\,\overline{on}$	$\dot{on}$	$\Rightarrow \mathit{false}$
$\dot{on}_c$	$\Rightarrow \dot{on}$	$\dot{off}$	$\Rightarrow \overline{b} \lor \dot{off}_c$	$\dot{off}_c$	$\Rightarrow \dot{off}\,\overline{on}$	$\dot{on}_c$	$\Rightarrow \mathit{false}$
$\dot{test}$	$\Rightarrow \dot{a}$	$\dot{off}_c$	$\Rightarrow \dot{off}$	$\dot{test}$	$\Rightarrow \dot{a}\,\dot{a}_l\,\overline{b}\,\overline{on} \lor \dot{a}\,\dot{a}_l\,\dot{off}\,\dot{off}_c\,\overline{on}$		

$$\begin{tikzcd} \dot{a}\ \dot{a}_l\ \bar{b}\ \overline{off} \arrow[dd] & & \\ & \oplus & \dot{o}n\ \dot{o}n_c\ \bar{b}\ \overline{off} \\ te\dot{s}t & & \end{tikzcd}$$

(a) First tree

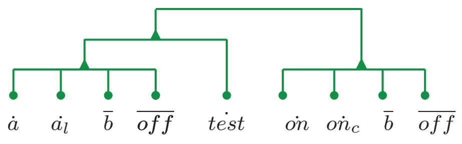
$$\dot{b}\ \dot{b}_l\ \overline{on}$$

(b) Second tree

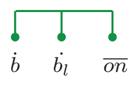
$$\begin{tikzcd} o\dot{f}f\ o\dot{f}f_c\ \overline{on} \arrow[d] & & \dot{a}\ \dot{a}_l\ \bar{b}\ \overline{on} \arrow[dd] & & \\ \dot{a}\ \dot{a}_l \arrow[d] & \oplus & & \oplus & o\dot{f}f\ \bar{b}\ \overline{on} \\ te\dot{s}t & & te\dot{s}t & & \end{tikzcd}$$

(c) Third tree

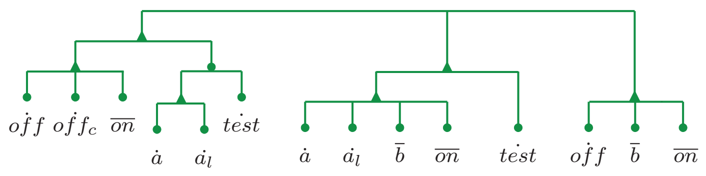
Figure 3.15 – Causal interaction trees for Example 3.4.1.



(a) First connector



(b) Second connector



(c) Third connector

Figure 3.16 – Connectors corresponding to trees from Figure 3.15.

3.5 Discussion

The expressiveness relation between BIP, its modifications and operators defined by SOS rules[3] can be summarised as follows:

- BIP glue in classical semantics (denoted as "BIP" in Figure 3.17 and Figure 3.18) does not have even weak full expressiveness w.r.t. BIP-like SOS (2.14) ("BSOS") (cf. Example 3.1.2).
- BIP glue in classical semantics has weak full expressiveness w.r.t. BIP-like SOS with acyclic inhibiting relations ("$BSOS^a$") (cf. Proposition 3.1.4).
- BIP glue with relaxed priority model ("RBIP") has strong full expressiveness w.r.t. BIP-like SOS (cf. Proposition 3.1.7).
- BIP glue in offer semantics ("OBIP") has strong full expressiveness w.r.t. SOS rules in the format (2.25) ("FNWSOS").
- BIP-like SOS is incomparable with SOS rules in the format (2.25) [23].

[3] In this chapter we have considered the same sets of components (cf. 3.2.2) that can be used to build systems, the same semantics mapping and the same equivalence relation. Thus, instead of saying that component-based framework has some kind of expressiveness w.r.t. the set of operators, we directly say that one set of operators or glue has some kind of expressiveness w.r.t. another set of operators.

$$\begin{tikzcd}
BSOS^a \arrow[r, leftrightarrow, "w"] \arrow[dr] & BIP \arrow[d] \arrow[r, leftrightarrow, dashed] \arrow[drr, leftrightarrow, dashed] & FNSOS \arrow[dl] \arrow[d] & \\
RBIP \arrow[r, leftrightarrow] & BSOS \arrow[r, leftrightarrow, dashed] & FNWSOS \arrow[r, leftrightarrow] & OBIP
\end{tikzcd}$$

Figure 3.17 – Expressiveness relation between composition operators.

$$\begin{tikzcd}
BSOS^a \arrow[r, leftrightarrow, "w"] \arrow[dr] & BIP \arrow[d] \arrow[dr] & & \\
RBIP \arrow[r, leftrightarrow] & BSOS \arrow[r, leftrightarrow] & FNSOS \arrow[d] & \\
& & FNWSOS \arrow[r, leftrightarrow] & OBIP
\end{tikzcd}$$

Figure 3.18 – Expressiveness relation for components satisfying Property 3.2.5.

- SOS rules in the format

$$\frac{\left\{q_i \xrightarrow{a \cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_k \not\rightarrow b_k^l \,\middle|\, k \in K, l \in L_k\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}, \tag{3.18}$$

  defined in [23] is less expressive than BIP-like SOS [23], i.e. it does ot have weak full expressiveness w.r.t. BIP-like SOS. We denote SOS rules in the format (3.18) with "FNSOS".

- BIP glue in classical semantics and BIP glue in offer semantics are incomparable (cf. Examples3.1.2 and 3.2.1). Notice that the same examples can be used to show that BIP glue in classical semantics is incomparable with SOS rules in the format (3.18).

These relations are illustrated in Figure 3.17. Arrows connect less expressive sets of operators with more expressive ones; double-headed arrows connect sets that have strong full expressiveness (or weak full expressiveness if there is a mark "w") w.r.t. to each other; dashed lines connect incomparable sets. The figure shows two extra relations since BIP-like SOS with acyclic inhibiting relation and SOS rules in the format (3.18) are sub-formats of BIP-like SOS and SOS rules in the format (2.25), respectively.

Considering a class of components satisfying Property 3.2.5, we can transform BIP glue in the classical semantics into the offer semantics by the algorithm in Figure 3.6. Within this class, any system in the classical semantics is expressible in the offer semantics. Proposition 3.2.10 shows that any BIP-like SOS operator can be represented as an operator in the format (3.18). Assuming that components satisfy Property 3.2.5 expressiveness relations between composition operators are summarised in Figure 3.18.

4 Architecture Composability

In this chapter, we show how architectures can be modelled and combined. An architecture is defined as an operator that applied to a set of components builds a composite one. It consists of a configuration with a set of *coordinating* components. For the sake of simplicity, we use BIP interaction model (cf. Definition 2.2.7) to specify configurations throughout this chapter. In Section 4.4, we show that the results are also valid for extended interaction model (cf. Definition 2.3.4).

Architectures can be intuitively understood as enforcing constraints on the global state space of the system [23, 101]. From this perspective, architecture composition can be understood as the conjunction of their respective constraints. Propositional Interaction Logic as a representation of BIP interaction model allows to define composition in a simple manner: the composition operator returns an architecture with coordinating components of all operands and a conjunction of PIL formulas representing operands' interaction models. This composition operator is commutative and associative. Furthermore, it is idempotent if all coordinating components are deterministic.

An architecture is a solution to a specific coordination problem characterised by a property. Any property can be decomposed as the conjunction of safety and liveness properties. The composition operator preserves safety properties: for two architectures with characteristic safety properties Φ_1 and Φ_2, their composition has a characteristic property $\Phi_1 \wedge \Phi_2$. Some results on liveness properties preservation are provided in [9], however they are out of the scope of this thesis.

The results are illustrated in the case study in Section 4.5.

4.1 Architecture operator

An architecture can be seen as a composition operator that transforms a set of components into a new composite component. It generalises BIP interaction models, by introducing stateful coordinating components. The interface of an architecture is a set of ports that

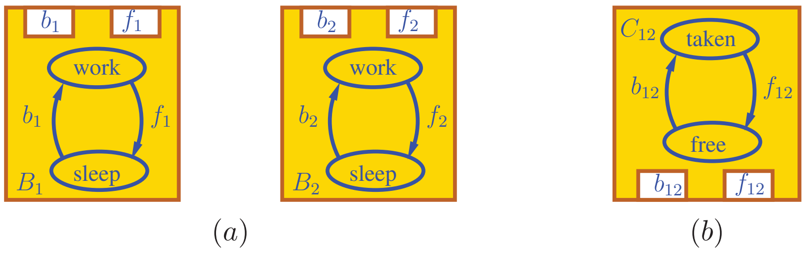


Figure 4.1 – Components (a) and coordinator (b) for Example 4.1.3.

comprises both the ports of the coordinating components and additional *dangling* ports that must belong to operand components, to which the architecture is applied.

Definition 4.1.1. An *architecture* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite set of *coordinating components* with pairwise disjoint sets of ports, P_A is a set of ports, such that $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A}$ is an interaction model over P_A.

An architecture A can be applied to any set of components $\mathcal{B}$ that contains all dangling ports of A. Intuitively, an architecture enforces coordination constraints on the components in $\mathcal{B}$. The interface P_A of an architecture A contains all ports of the coordinating components $\mathcal{C}$ and some additional ports, which must belong to the components in $\mathcal{B}$. In the application $A(\mathcal{B})$, the ports belonging to P_A can only participate in the interactions defined by the interaction model γ of A. Ports that do not belong to P_A are not restricted and can participate in any interaction. In particular, they can join the interactions in γ (see (4.1) below).

Definition 4.1.2. Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and let $\mathcal{B}$ be a set of components, such that $\bigcup_{B \in \mathcal{B}} P_B \cap \bigcup_{C \in \mathcal{C}} P_C = \emptyset$ and $P_A \subseteq P \overset{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. The *application of an architecture* A to the components $\mathcal{B}$ is the component

$$A(\mathcal{B}) \overset{def}{=} \left(\gamma \bowtie 2^{P \setminus P_A}\right)(\mathcal{C} \cup \mathcal{B})\,, \tag{4.1}$$

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

$$\gamma' \bowtie \gamma'' \overset{def}{=} \{a' \cup a'' \,|\, a' \in \gamma', a'' \in \gamma''\}$$

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

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

Example 4.1.3. Consider the components B_1 and B_2 in Figure 4.1(a). In order to ensure mutual exclusion of their `work` states, we apply the architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$, where the component C_{12} is shown in Figure 4.1(b), $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \{\emptyset,\ b_1b_{12}, b_2b_{12},\ f_1f_{12},\ f_2f_{12}\}$.

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, i.e. the mutual exclusion property $(q_1 \neq \texttt{work}) \vee (q_2 \neq \texttt{work})$—where q_1 and q_2 are state variables of B_1 and B_2 respectively—holds in $A_{12}(B_1, B_2)$.

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

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

Lemma 4.1.4. *Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and denote by $\gamma_{\mathcal{C}} \overset{def}{=} \{a \cap P_{\mathcal{C}} \mid a \in \gamma\}$, with $P_{\mathcal{C}} = \bigcup_{C \in \mathcal{C}} P_C$, the projection of γ onto the ports of the coordinating components of A. Consider an architecture $A' = (\{C'\}, P_A, \gamma)$, where $C' = \gamma_{\mathcal{C}}(\mathcal{C})$. For any set of components $\mathcal{B}$, satisfying the conditions of Definition 4.1.2, we have $A(\mathcal{B}) = A'(\mathcal{B})$.*

Proof. First of all, notice that $P_{C'} = P_{\mathcal{C}}$. Hence, the conditions of Definition 4.1.1 are satisfied and A' is indeed an architecture. Furthermore, $\mathcal{B}$ satisfies the conditions of Definition 4.1.2 w.r.t. A'. Hence, the component $A'(\mathcal{B})$ is well defined.

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

By Definition 4.1.2, a transition $\tilde{q}_1 \dots \tilde{q}_m q_1 \dots q_n \xrightarrow{a} \tilde{q}_1' \dots \tilde{q}_m' q_1' \dots q_n'$ is possible in $A(\mathcal{B})$ iff $a \neq \emptyset$ and

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

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

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

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

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

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

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

4.2 Composition of architectures

Architectures can be intuitively understood as enforcing constraints on the global state space of the system [23, 101]. More precisely, component coordination is realised by limiting the allowed interactions, thus enforcing constraints on the transitions components can take. From this perspective, architecture composition can be understood as the conjunction of their respective constraints. This intuitive notion is formalised by the following definition.

Definition 4.2.1. Let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two architectures and let $\varphi_{\gamma_1}, \varphi_{\gamma_2}$ be characteristic predicates (Definition 2.2.17) of γ_1, γ_2, respectively. The *composition* of A_1 and A_2 is an architecture $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_{A_1} \cup P_{A_2}, \gamma_\varphi)$, where $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$ and $\gamma_\varphi = \|\varphi\|$ is an interaction model defined by the predicate φ.

The following lemma states that the interaction model of the composed component consists precisely of the interactions a such that the projections of a onto the interfaces of the composed architectures (A_1 and A_2, resp.) belong to the corresponding interaction models (γ_1 and γ_2, resp.). In other words, these are precisely the interactions that satisfy the coordination constraints enforced by both composed architectures. In particular, as we will show in Theorem 4.3.5, this means that, for two architectures A_1, A_2 and a set of components $\mathcal{B}$, the execution traces allowed by $A_1 \oplus A_2$ on $\mathcal{B}$ are those that are allowed by both A_1 and A_2, which guarantees the preservation of safety properties by the composition of architectures.

Lemma 4.2.2. *Consider two interaction models $\gamma_i \subseteq 2^{P_i}$, for $i = 1, 2$, and let $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. For an interaction $a \subseteq P_1 \cup P_2$, $a \in \gamma_\varphi$ iff $a \cap P_i \in \gamma_i$, for $i = 1, 2$. More specifically, $\gamma_\varphi = \{a_1 \cup a_2 \mid a_1 \in \gamma_1,\, a_2 \in \gamma_2,\, a_1 \cap P_2 = a_2 \cap P_1\}$.*

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

Thus, for $a \in \gamma_\varphi$, $a \cap P_i \in \gamma_i$ $(i \in \{1, 2\})$ and, trivially, $(a \cap P_1) \cap P_2 = (a \cap P_2) \cap P_1$.

For any $a_1 \in \gamma_1$ and $a_2 \in \gamma_2$, such that $a_1 \cap P_2 = a_2 \cap P_1$, holds $(a_1 \cup a_2) \cap P_1 = a_1$ and $(a_1 \cup a_2) \cap P_2 = a_2$. Consequently, $a_1 \cup a_2 \in \gamma_\varphi$. ☐

Remark 4.2.3. Every interaction allowed by $A_1 \oplus A_2$ must comprise both an interaction allowed by A_1 and an interaction allowed by A_2. To allow architecture A_1 to progress independently from A_2, one must have $\emptyset \in \gamma_2$ and vice-versa.

Lemma 4.2.4. *Consider a set of components $\mathcal{B}$ and two architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$. Let $\tilde{q}_1\tilde{q}_2 q \xrightarrow{a} \tilde{q}_1'\tilde{q}_2' q'$ be a transition in $(A_1 \oplus A_2)(\mathcal{B})$, where, for $i = 1, 2$, $\tilde{q}_i, \tilde{q}_i' \in \prod_{C \in \mathcal{C}_i} Q_C$ and $q, q' \in \prod_{B \in \mathcal{B}} Q_B$. Then, for $i = 1, 2$, if $a \cap (P_{A_i} \cup P) \neq \emptyset$, then $\tilde{q}_i q \xrightarrow{a \cap (P_{A_i} \cup P)} \tilde{q}_i' q'$ is a transition in $A_i(\mathcal{B})$, where $P = \bigcup_{B \in \mathcal{B}} P_B$.*

Proof. By Lemma 4.1.4, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$.

By Definition 4.2.1, $a \cap (P_{A_1} \cup P_{A_2}) \overset{i}{\models} \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lemma 4.2.2, $a \cap P_{A_1} \in \gamma_1$. Hence,

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

By the assumption of the lemma, $\tilde{a} \neq \emptyset$. Furthermore, since $\tilde{q}_1\tilde{q}_2 q \xrightarrow{a} \tilde{q}_1'\tilde{q}_2' q'$, we have by (2.10),

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

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 (2.10) 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 4.2.5. *Architecture composition $\oplus$ is commutative and associative; it is idempotent if all coordinating components are deterministic; $A_{id} = (\emptyset, \emptyset, \{\emptyset\})$ is its neutral* *element, i.e. for any architecture A, we have $A \oplus A_{id} = A$. Furthermore, for any component B, we have $A_{id}(B) = B$.*

Proof. Commutativity and associativity follow from the corresponding properties of set union and boolean conjunction. Suppose we have two architectures $A = A'$. As illustrated by Lemma 4.1.4, this does not necessarily mean that their sets of coordinating components coincide. However, if all involved coordinating components are deterministic, then, in any state of $(A \oplus A')(\mathcal{B})$, both architectures will impose the same restrictions, enabling the same interactions between the coordinating and operand components. Hence, we have $(A \oplus A')(\mathcal{B}) = A(\mathcal{B}) = A'(\mathcal{B})$. Since this holds for any set of components $\mathcal{B}$, we conclude that $A \oplus A' = A = A'$. The properties of A_{id} follow immediately from the definitions of architecture application and composition. $\square$

Notice that, by (4.1), for an arbitrary set of components $\mathcal{B}$ with $P = \bigcup_{B \in \mathcal{B}} P_B$, we have $A_{id}(\mathcal{B}) = (2^P)(\mathcal{B})$.

Example 4.2.6. Building upon Example 4.1.3, let B_3 be a third component, similar to B_1 and B_2, with the interface $\{b_3, f_3\}$. We define two additional architectures A_{13} and A_{23} similar to A_{12}: for $i = 1, 2$, $A_{i3} = (\{C_{i3}\}, P_{i3}, \gamma_{i3})$, where, up to the renaming of ports, C_{i3} is the same as C_{12} in Figure 4.1(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 characteristic predicate of γ_{12} is:

$$\begin{aligned}
\varphi_{\gamma_{12}} &= \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \ \vee\ b_1\,\overline{b_2}\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \ \vee\ \overline{b_1}\,b_2\,b_{12}\,\overline{f_1}\,\overline{f_2}\,\overline{f_{12}} \\
&\quad \vee\ \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,f_1\,\overline{f_2}\,f_{12} \ \vee\ \overline{b_1}\,\overline{b_2}\,\overline{b_{12}}\,\overline{f_1}\,f_2\,f_{12} \\
&= (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \\
&\quad \wedge\ (b_{12} \Rightarrow b_1 \,\mathtt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\mathtt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})\,.
\end{aligned} \tag{4.2}$$

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 [22].

By considering, for $\varphi_{\gamma_{13}}$ and $\varphi_{\gamma_{23}}$, expressions similar to (4.2), it is easy to compute $\varphi_{\gamma_{12}} \wedge \varphi_{\gamma_{13}} \wedge \varphi_{\gamma_{23}}$ as the conjunction of the following implications:

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

Finally, the interaction model γ_{123} for $A_{12} \oplus A_{13} \oplus A_{23}$ is:

$$\gamma_{123} \ = \ \{\emptyset,\ b_1 b_{12} b_{13},\ f_1 f_{12} f_{13},\ b_2 b_{12} b_{23},\ f_2 f_{12} f_{23},\ b_3 b_{13} b_{23},\ f_3 f_{13} f_{23}\}\,.$$

Notice that this interaction model is different from the union of the interaction models of the three architectures. For any interaction $a \in \gamma_{123}$, $a \cap P_{ij} \in \gamma_{ij}$, for $ij \in \{12, 13, 23\}$ (cf. Lemma 4.2.2).

Assuming that the initial states of B_1, B_2 and B_3 are `sleep`, whereas those of C_{12}, C_{13} and C_{23} are `free`, one can observe that none of the states $(\cdot, \cdot, \cdot, \texttt{work}, \texttt{work}, \cdot)$, $(\cdot, \cdot, \cdot, \texttt{work}, \cdot, \texttt{work})$ and $(\cdot, \cdot, \cdot, \cdot, \texttt{work}, \texttt{work})$ are reachable in $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$. Thus, we conclude that the composition of the three architectures, $(A_{12} \oplus A_{13} \oplus A_{23})(B_1, B_2, B_3)$, enforces mutual exclusion among the `work` states of all three components. In Section 4.3, we provide a general result stating that architecture composition preserves the enforced state properties.

4.2.1 Hierarchical composition of architectures

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

Proposition 4.2.7. *Let $\mathcal{B}$ be a set of components and let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that 1) $P_{A_1} \subseteq P_1 \overset{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1} P_B$ and 2) $P_{A_2} \subseteq P_2 \overset{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B}))$ is defined and equal to $(A_1 \oplus A_2)(\mathcal{B})$.*

Proof. Clearly, the state spaces and interfaces of both components coincide. Thus we only have to prove that so do the transition relations. By Lemma 4.1.4, we assume $\mathcal{C}_1 = \{C_1\}$, $\mathcal{C}_2 = \{C_2\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$.

By Definition 4.1.2 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 $a \neq \emptyset$ and

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

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

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

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

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

For $a \subseteq P_2$, we have $a \in \gamma_{A_1 \oplus A_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$ iff $a \cap (P_{A_1} \cup P_{A_2}) \in \gamma_{A_1 \oplus A_2}$, i.e. $a \cap (P_{A_1} \cup P_{A_2}) \models \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lemma 4.2.2, this is equivalent to $a \cap (P_{A_1} \cup P_{A_2}) \cap P_{A_1} = a \cap P_{A_1} \in \gamma_1$ and $a \cap (P_{A_1} \cup P_{A_2}) \cap P_{A_2} = a \cap P_{A_2} \in \gamma_2$. Since $a \subseteq P_2$, we have $a \cap P_{A_2} \in \gamma_2$ iff $a \in \gamma_2 \bowtie 2^{P_2 \setminus P_{A_2}}$. Finally, since $P_{A_1} \subseteq P_1$, we have $a \cap P_{A_1} \in \gamma_1$ iff $a \cap P_1 \in \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}$. □

The first condition in Proposition 4.2.7 states that A_1 can be applied to the components in $\mathcal{B}$ (cf. Definition 4.1.2). Similarly, the second condition states that A_2 can be applied to $A_1(\mathcal{B})$. Note that, when $P_{A_i} \subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}_i} P_B$ holds for both $i \in \{1, 2\}$—for $i = 1$, this is the first condition of Proposition 4.2.7—and none of the architectures involves the ports of the other, i.e. $P_{A_i} \cap \bigcup_{C \in \mathcal{C}_j} P_C = \emptyset$, for $i \neq j \in \{1, 2\}$, then the two architectures are independent and their composition is commutative: $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B}) = A_1(A_2(\mathcal{B}))$.

The following proposition shows that the application of an architecture only affects the components that have ports belonging to its interface. Components that do not involve such ports are not affected, even if they interact with the operand components of the architecture. In Proposition 4.2.8, such potential interactions are modelled by applying the architecture A_2, which also provides a context for the comparison of the resulting systems. In the special case, where such independent components do not interact with the architecture operands, one can consider $A_2 = A_{id}$.

Proposition 4.2.8. *Let $\mathcal{B}_1, \mathcal{B}_2$ be two sets of components, such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$. Let $A_1 = (\mathcal{C}_1, P_{A_1}, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_{A_2}, \gamma_2)$ be two architectures, such that $P_{A_1} \subseteq P_1 \overset{def}{=} \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{C}_1} P_B$ and $P_{A_2} \subseteq P_2 \overset{def}{=} \bigcup_{B \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{C}_1 \cup \mathcal{C}_2} P_B$. Then $A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$.*

Proof. As in the proof of Proposition 4.2.7, we notice that the sets of states and interfaces are equal in both composed components, thus we only have to prove the equality of transition relations. By Lemma 4.1.4, we assume $\mathcal{C}_1 = \{C_1\}$ and $\mathcal{C}_2 = \{C_2\}$. Furthermore, let $\mathcal{B}_1 = \{B_1, \dots, B_k\}$ and $\mathcal{B}_2 = \{B_{k+1}, \dots, B_n\}$. We then have $P_1 = P_{C_1} \cup \bigcup_{i=1}^k P_{B_i}$ and $P_2 = P_{C_1} \cup P_{C_2} \cup \bigcup_{i=1}^n P_{B_i}$.

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

$a_{\mathcal{B}_2} = a \cap \bigcup_{i=k+1}^{n} P_{B_i}$. Then either $a_{\gamma_1} \cup a_2 = \emptyset$ or it is enabled in $A_1(\mathcal{B}_1)$. Hence, interaction $a = a_{C_2} \cup a_{\gamma_1} \cup a_2 \cup a_{\mathcal{B}_2}$ is enabled in $A_2\big(A_1(\mathcal{B}_1), \mathcal{B}_2\big)$.

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

Intuitively, Proposition 4.2.8 states that one only has to apply the architecture A_1 to those components that have ports involved in its interface. Notice that, in order to compare the semantics of two sets of components, one has to compose them into compound components, by applying *some* architecture. Hence, the need for A_2 in Proposition 4.2.8. As a special case, one can consider the "most liberal" identity architecture A_{id} (see Proposition 4.2.5). A_{id} does not impose any coordination constraints, allowing all possible interactions between the components it is applied to.

Example 4.2.9. Example 4.2.6 can be generalised to an arbitrary number n of components by repeating the architecture application pairwise. However, this solution requires $n(n-1)/2$ architectures, and so does not scale well. Instead, we apply architectures hierarchically.

Let $n = 4$ and consider two architectures A_{12}, A_{34}, with the respective coordination components C_{12}, C_{34}, that respectively enforce mutual exclusion between B_1, B_2 and B_3, B_4 as in Example 4.2.6. Assume furthermore, that an architecture A enforces mutual exclusion between the `taken` states of C_{12} and C_{34}. It is clear that the system $A\big(A_{12}(B_1, B_2), A_{34}(B_3, B_4)\big)$ ensures mutual exclusion between all four components $(B_i)_{i=1}^4$. Furthermore, by the above propositions,

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

4.2.2 Partial application of architectures

Notice that the main condition, limiting the application of Proposition 4.2.7 and Proposition 4.2.8, is that the architectures must be applicable, i.e. every port of the architecture interface must belong to some component. Below we lift this restriction by introducing the notion of *partial application.* We generalise Definition 4.1.2 for architectures $A = (\mathcal{C}, P_A, \gamma)$ applied to sets of components $\mathcal{B}$, such that $P_A \not\subseteq \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. This means that the architecture enforces constraints on some ports which are not present in any of the coordinating or base components. In other words, the system obtained by applying the architecture to the set of components $\mathcal{B}$ is not *complete.* The result can then itself be considered as an architecture where the coordinating component is the one obtained by applying to $\mathcal{B} \cup \mathcal{C}$ the projection of interactions in γ.

Definition 4.2.10. Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture and $\mathcal{B}$ be a set of components. Let $P = \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. A *partial application* of A to $\mathcal{B}$ is an architecture $A[\mathcal{B}] \overset{def}{=} (\{C'\}, P \cup P_A, \gamma \bowtie$

$2^{P\setminus P_A})$, where $C' \overset{def}{=} (\gamma^P \bowtie 2^{P\setminus P_A})(\mathcal{C} \cup \mathcal{B})$ with $\gamma^P = \{a \cap P \,|\, a \in \gamma\}$ and the operator $\bowtie$ as in Definition 4.1.2.

Notice that an architecture obtained by partial application has precisely one coordinating component C'. It is also important to notice that the interaction model in $A[\mathcal{B}]$ is not the same as in the definition of C'. On the other hand, if $P_A \subseteq P$ (as in Definition 4.1.2), we have $\gamma^P = \gamma$ and $A[\mathcal{B}] = (\{A(\mathcal{B})\}, P, \gamma \bowtie 2^{P\setminus P_A})$.

Lemma 4.2.11. *Let $\mathcal{B}$ be a set of components and $A = (\mathcal{C}, P_A, \gamma)$ be an architecture, such that $P_A \subseteq \bigcup_{B\in\mathcal{B}\cup\mathcal{C}} P_B$. Then $A(\mathcal{B}) = A[\mathcal{B}](\emptyset)$.*

Proof. Follows immediately from Definitions 4.1.2 and 4.2.10. □

Proposition 4.2.12. *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two sets of components, such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$, and let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture. Then $A[\mathcal{B}_1 \cup \mathcal{B}_2] = \big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$.*

Proof. Clearly the interfaces of both architectures coincide. Furthermore, since the two architectures are obtained by partial application, each has only one coordinating component (see Definition 4.2.10). Thus, we have to show that the coordinating components and the interaction models of both architectures coincide.

Let $P_1 = \bigcup_{B\in\mathcal{C}\cup\mathcal{B}_1} P_B$ and $P_2 = \bigcup_{B\in\mathcal{C}\cup\mathcal{B}_2} P_B$. By Definition 4.2.10, the interaction models of $A[\mathcal{B}_1 \cup \mathcal{B}_2]$ and $\big(A[\mathcal{B}_1]\big)[\mathcal{B}_2]$ are, respectively $\gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}$ and $\big(\gamma \bowtie 2^{P_1\setminus P_A}\big) \bowtie 2^{P_2\setminus P_A}$. Since $2^{P_1\setminus P_A} \bowtie 2^{P_2\setminus P_A} = 2^{(P_1\cup P_2)\setminus P_A}$, we conclude that the interaction models coincide.

It is also clear that the state spaces and interfaces of both coordination components coincide. Thus, we only have to show that so do the transition relations. Let us consider the coordinating components of the two architectures. By Definition 4.2.10, we have

$$\begin{aligned} A[\mathcal{B}_1 \cup \mathcal{B}_2] &= \big(\{C_{12}\}, P_A \cup P_1 \cup P_2, \gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)\,,\\ \text{with } C_{12} &= \big(\gamma^{P_1\cup P_2} \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1 \cup \mathcal{B}_2)\,,\\ \text{where } \gamma^{P_1\cup P_2} &= \{a \cap (P_1 \cup P_2) \,|\, a \in \gamma\}\,. \end{aligned} \tag{4.3}$$

Similarly,

$$\begin{aligned} A[\mathcal{B}_1] &= \big(\{C_1\}, P_A \cup P_1, \gamma \bowtie 2^{P_1\setminus P_A}\big)\,,\\ \text{with } C_1 &= \big(\gamma^{P_1} \bowtie 2^{P_1\setminus P_A}\big)(\mathcal{C} \cup \mathcal{B}_1)\,, \text{ where } \gamma^{P_1} = \{a \cap P_1 \,|\, a \in \gamma\} \text{ and} \qquad (4.4)\\ \big(A[\mathcal{B}_1]\big)[\mathcal{B}_2] &= \big(\{C_2\}, P_A \cup P_1 \cup P_2, \gamma \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)\,,\\ \text{with } C_2 &= \big(\gamma^{P_1\cup P_2} \bowtie 2^{(P_1\cup P_2)\setminus P_A}\big)(\{C_1\} \cup \mathcal{B}_2)\,. \qquad (4.5) \end{aligned}$$

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

Consider $a \in \gamma^{P_1 \cup P_2} \bowtie 2^{(P_1 \cup P_2) \setminus P_A}$. By definition of $\bowtie$, $a = a_1 \cup a_2$, with $a_1 \in \gamma^{P_1 \cup P_2}$ and $a_2 \subseteq (P_1 \cup P_2) \setminus P_A$. By (4.3), we have $a_1 = \tilde{a}_1 \cap (P_1 \cup P_2)$ with some $\tilde{a}_1 \in \gamma$. We deduce that $a_1 \cap P_1 = \tilde{a}_1 \cap (P_1 \cup P_2) \cap P_1 = \tilde{a}_1 \cap P_1$ and, therefore $a_1 \cap P_1 \in \gamma^{P_1}$. Since, $a \cap P_1 = (a_1 \cap P_1) \cup (a_2 \cap P_1)$ and $a_2 \cap P_1 \subseteq ((P_1 \cup P_2) \setminus P_A) \cap P_1 = P_1 \setminus P_A$, we have $a \cap P_1 \in \gamma^{P_1} \bowtie 2^{P_1 \setminus P_A}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (4.4). By (2.10), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

Proposition 4.2.12 generalises Proposition 4.2.8. In order to generalise Proposition 4.2.7, we first define the application of one architecture to another, by putting

$$A_1[A_2] \overset{def}{=} (A_1 \oplus A_2)[\emptyset]\,. \tag{4.6}$$

Lemma 4.2.13. *For any set of components $\mathcal{B}$ and any architectures A_1 and A_2, we have $(A_1 \oplus A_2)[\mathcal{B}] = (A_1[\mathcal{B}] \oplus A_2)[\emptyset] = (A_1 \oplus A_2[\mathcal{B}])[\emptyset]$.*

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

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

Clearly the interfaces of both architectures coincide. Furthermore, since the two architectures are obtained by partial application, each has only one coordinating component (see Definition 4.2.10). Thus we have to show that the coordinating components and the interaction models of both architectures coincide.

Let us consider the characteristic predicates of the interaction models. Notice, first, that for any two interaction models $\gamma' \subseteq 2^{P'}$ and $\gamma'' \subseteq 2^{P''}$, over disjoint sets of ports $P' \cap P'' = \emptyset$, one has (cf. Definition 4.1.2)

$$\varphi_{\gamma'} \wedge \varphi_{\gamma''} \equiv \varphi_{\gamma' \bowtie \gamma''}. \tag{4.7}$$

Denote the interaction model of $A_1[\mathcal{B}]$ by $\gamma_1' = \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}$. Clearly, $\varphi_{\left(2^{P_1 \setminus P_{A_1}}\right)} = true$. Hence, by (4.7), we have $\varphi_{\gamma_1'} \equiv \varphi_{\gamma_1} \wedge \varphi_{\left(2^{P_1 \setminus P_{A_1}}\right)} \equiv \varphi_{\gamma_1}$ and, consequently, the characteristic predicate of the interaction model of $(A_1[\mathcal{B}] \oplus A_2)[\emptyset]$ is $\varphi_{\gamma_1'} \wedge \varphi_{\gamma_2} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By a similar argument, we can conclude that the characteristic predicate of the interaction model of $(A_1 \oplus A_2)[\mathcal{B}]$ is also $\varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. Since the interfaces of the two architectures coincide, this implies that so do their interaction models. We denote the interaction model in question by γ_{12}. Recall that $\varphi_{\gamma_{12}} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$.

Let us consider the coordinating components of the two architectures. By Definition 4.2.10, we have

$$\begin{aligned}(A_1 \oplus A_2)[\mathcal{B}] &= \big(\{C_{12}\}, P_2 \cup P_{A_1} \cup P_{A_2}, \gamma_{12} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)\,,\\ \text{with } C_{12} &= \big(\gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)(\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{B})\,, \\ \text{where } \gamma_{12}^{P_2} &= \{a \cap P_2 \,|\, a \in \gamma_{12}\}\,.\end{aligned} \tag{4.8}$$

Similarly,

$$\begin{aligned}A_1[\mathcal{B}] &= \big(\{C_1\}, P_1 \cup P_{A_1}, \gamma_1 \bowtie 2^{P_1 \setminus P_{A_1}}\big)\,,\\ \text{with } C_1 &= \big(\gamma_1^{P_1} \bowtie 2^{P_1 \setminus P_{A_1}}\big)(\mathcal{C}_1 \cup \mathcal{B})\,,\ \text{where } \gamma_1^{P_1} = \{a \cap P_1 \,|\, a \in \gamma_1\} \text{ and} \end{aligned} \tag{4.9}$$

$$\begin{aligned}(A_1[\mathcal{B}] \oplus A_2)[\emptyset] &= \big(\{C_2\}, P_2 \cup P_{A_1} \cup P_{A_2}, \gamma_{12} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)\,,\\ \text{with } C_2 &= \big(\gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}\big)(\{C_1\} \cup \mathcal{C}_2)\,.\end{aligned} \tag{4.10}$$

Notice that the interaction models and the constituent atomic components in (4.8) and (4.10) coincide. Therefore, any transition allowed in C_2 is also allowed in C_{12}. Hence, to prove that $C_{12} = C_2$, we have to show that any interaction allowed in C_{12}, after projection, is allowed in C_1. Notice, further, that the interface of C_1 is P_1, whereas those of C_2 and C_{12} are both P_2.

Consider $a \in \gamma_{12}^{P_2} \bowtie 2^{P_2 \setminus (P_{A_1} \cup P_{A_2})}$. By definition of $\bowtie$, $a = a_1 \cup a_2$ with $a_1 \in \gamma_{12}^{P_2}$ and $a_2 \subseteq P_2 \setminus (P_{A_1} \cup P_{A_2}) \subseteq P_1 \setminus P_{A_1}$. By (4.8), we have $a_1 = \tilde{a}_1 \cap P_2$ with some $\tilde{a}_1 \in \gamma_{12}$. Since $\varphi_{\gamma_{12}} \equiv \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$, by Lemma 4.2.2, we have $\tilde{a}_1 \cap P_{A_1} \in \gamma_1$ and $\tilde{a}_1 \cap P_1 \in \gamma_1^{P_1}$. Notice that $P_1 \subseteq P_2$. Hence $a_1 \cap P_1 = \tilde{a}_1 \cap P_2 \cap P_1 = \tilde{a}_1 \cap P_1 \in \gamma_1^{P_1}$. We conclude that $a \cap P_1 = (a_1 \cap P_1) \cup (a_2 \cap P_1) = (a_1 \cap P_1) \cup a_2 \in \gamma_1^{P_1} \bowtie 2^{P_1 \setminus P_{A_1}}$. Thus, the part of a relevant to the atomic components comprising C_1 belongs to the interaction model in (4.9). By (2.10), we conclude that any transition labelled by a in C_{12} is also a transition of C_2. □

As a consequence of Lemma 4.2.13, we immediately obtain the following generalisation of Proposition 4.2.7.

Proposition 4.2.14. *For any set of components $\mathcal{B}$ and any architectures A_1 and A_2, we have $A_2[A_1[\mathcal{B}]] = A_1[A_2[\mathcal{B}]]$.*

Proof. By (4.6) and Lemma 4.2.13, we have

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

□

Notice, furthermore, that (4.6) generalises Definition 4.2.10. Indeed, to a given set of components $\mathcal{B}$, we can associate the architecture $A_{\mathcal{B}} \overset{def}{=} A_{id}[\mathcal{B}]$ (cf. Proposition 4.2.5). By

(4.6) and Lemma 4.2.13, we obtain, for any architecture A,

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

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

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

4.3 Property preservation

In this section, unless explicitly stated otherwise, we consider a set of architectures $A_1, \dots, A_m$ and a set of operand components $\mathcal{B}$. For each component we define an *initial state* q^0, i.e. a component is defined as a quadruple $B = (Q, q^0, P, \rightarrow\ )$ with $q^0 \in Q$. For a set of indices $I \subseteq [1, m]$, we will denote by $\bigoplus_{i \in I} A_i$ the composition of all architectures with indices in I. This is well-defined, since $\oplus$ is associative and commutative. In particular, $\bigoplus_{i\in[1,m]} A_i \overset{def}{=} A_1 \oplus \ldots \oplus A_m$.

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

Definition 4.3.1. Let $B = (Q, q^0, P, \rightarrow\ )$ be a component. A finite or infinite sequence $q_0 \rightarrow a_1 q_1 \rightarrow a_2 \cdots \rightarrow a_k q_k \cdots$ is a *path fragment* in B. If in addition $q_0 = q^0$, then it is also a *path.* A state $q \in Q$ is *reachable* iff there exists a finite path in B terminating in q. A path fragment is *reachable* iff its first state is reachable.

Definition 4.3.2. Let $B = (Q, q^0, P, \rightarrow\ )$ be a component. A *safety property* (in the rest of this section, simply *property*) of B is a state predicate $\Phi : Q \rightarrow \mathbb{B}$. We write $q \models \Phi$ iff $\Phi(q) = \mathit{true}$. A property Φ is *initial* if $q^0 \models \Phi$; it is *reachable* iff there exists a possibly empty path $q^0 \rightarrow a_1 q^1 \rightarrow a_2 \cdots \rightarrow a_n q^n$, such that $q^n \models \Phi$.

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

Definition 4.3.3 (Enforcing properties)**.** Let $A = (\mathcal{C}, P_A, \gamma)$ be an architecture; let $\mathcal{B}$ be a set of components and Φ be an initial property of their parallel composition $A_{id}(\mathcal{B})$ (see Proposition 4.2.5). We say that A *enforces* Φ *on* $\mathcal{B}$ iff, for every state $q = (q_b, q_c)$ reachable in $A(\mathcal{B})$, with $q_b \in \prod_{B\in\mathcal{B}} Q_B$ and $q_c \in \prod_{C\in\mathcal{C}} Q_C$, we have $q_b \models \Phi$.

According to the above definition, when we say that an architecture enforces some property

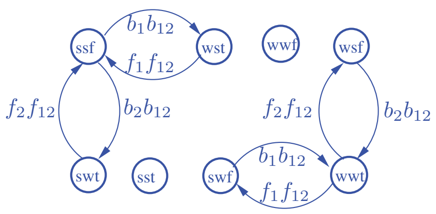


Figure 4.2 – Composite component $A_{12}(B_1, B_2)$ for Example 4.3.4.

Φ, it is implicitly assumed that Φ is initial for the coordinated components. Below, we omit mentioning this explicitly.

Example 4.3.4. Consider again the mutual exclusion in Example 4.1.3. States *sleep* and *free* in components B_1, B_2 and C_{12} are initial. Component $A_{12}(B_1, B_2)$ is shown in Figure 4.2 (we abbreviate `sleep`, `work`, `free` and `taken` to `s`, `w`, `f` and `t` respectively). Clearly A_{12} enforces on $\{B_1, B_2\}$ the mutual exclusion property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$, where q_1 and q_2 are state variables of B_1 and B_2 respectively.

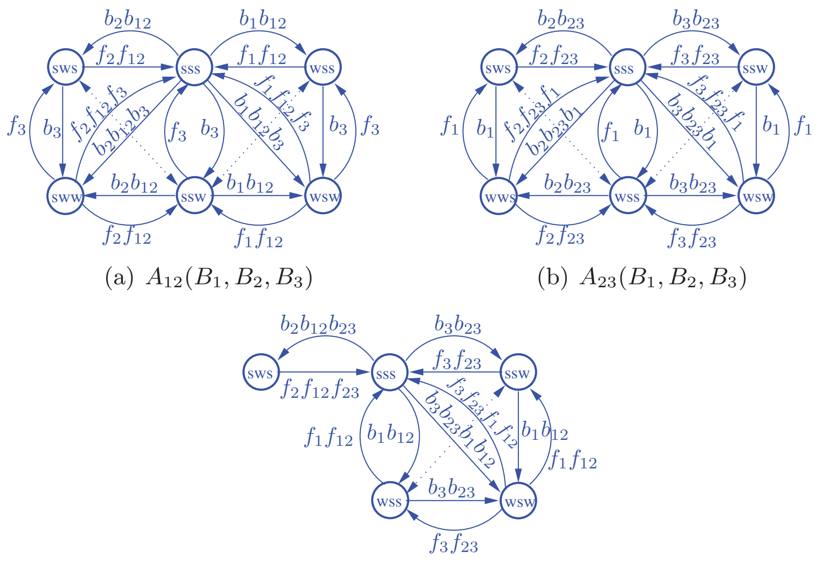
Theorem 4.3.5. *Let $\mathcal{B}$ be a set of components; let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two architectures enforcing on $\mathcal{B}$ the properties Φ_1 and Φ_2, respectively. The composition $A_1 \oplus A_2$ enforces on $\mathcal{B}$ the property $\Phi_1 \wedge \Phi_2$.*

Proof. Again, by Lemma 4.1.4, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$. We also denote, for $i = 1, 2$, $P_i = P_{C_i} \cup \bigcup_{B \in \mathcal{B}} P_B$.

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

Consider a path $\tilde{q}_1^0\tilde{q}_2^0q^0 \xrightarrow{a_1} \tilde{q}_1^1\tilde{q}_2^1q^1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} \tilde{q}_1^k\tilde{q}_2^kq^k$ in $(A_1 \oplus A_2)(\mathcal{B})$, where $q^0, \dots, q^k \in \prod_{B \in \mathcal{B}} Q_B$ and $\tilde{q}_i^0, \dots, \tilde{q}_i^k \in Q_{C_i}$, for $i = 1, 2$. By Lemma 4.2.4, $\tilde{q}_1^0q^0 \xrightarrow{a_1 \cap P_1} \tilde{q}_1^1q^1 \xrightarrow{a_2 \cap P_1} \cdots \xrightarrow{a_k \cap P_1} \tilde{q}_1^kq^k$ is a path in $A_1(\mathcal{B})$. (If, for some $i \in [1, k]$, $a_i \cap P_1 = \emptyset$, the corresponding transition can be omitted from the path.) Thus the state $\tilde{q}_1^kq^k$ is reachable in $A_1(\mathcal{B})$. Since A_1 enforces Φ_1 on $\mathcal{B}$, this implies that $q^k \models \Phi_1$. Symmetrically, $q^k \models \Phi_2$, which concludes the proof. □

Example 4.3.6. In the context of Example 4.2.6, consider the application of architectures A_{12} and A_{23} to the components B_1, B_2 and B_3. The former enforces the property $\Phi_{12} = (q_1 \neq \mathtt{w}) \vee (q_2 \neq \mathtt{w})$ (the projections of reachable states of $A_{12}(B_1, B_2, B_3)$ onto the state-space of the atomic components are shown in Figure 4.3(a)), whereas the latter enforces $\Phi_{23} = (q_2 \neq \mathtt{w}) \vee (q_3 \neq \mathtt{w})$ (the projections of reachable states of $A_{23}(B_1, B_2, B_3)$ onto the state-space of the atomic components are shown in Figure 4.3(b)). By Theorem 4.3.5, the composition $A_{12} \oplus A_{23}$ enforces $\Phi_{12} \wedge \Phi_{23} = (q_2 \neq \mathtt{w}) \vee ((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w}))$, i.e. mutual exclusion between, on one hand, the `work` state of B_2 and, on the other hand, the `work` states of B_1 and B_3 (see Figure 4.3(c)). Mutual exclusion between the `work` states of B_1 and B_3 is not enforced. Furthermore, it is easy to check that $A_{12} \oplus A_{23} \oplus A_{13}$ enforces mutual



(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 4.3 – Projections of reachable states of Example 4.3.6 components onto $A_{id}(B_1, B_2, B_3)$ (for ease of reading, we omit the transitions indicated by dotted blue arrows).

exclusion between the `work` states of B_1, B_2 and B_3 as $\Phi_{12} \wedge \Phi_{13} \wedge \Phi_{23} = \big((q_1 \neq \mathtt{w}) \wedge (q_2 \neq \mathtt{w})\big) \vee \big((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w})\big) \vee \big((q_1 \neq \mathtt{w}) \wedge (q_3 \neq \mathtt{w})\big)$.

4.4 Including priorities in architectures

The definition of architectures does not include BIP priority model. Interaction priorities can only be imposed by coordinating components and only when the involved components are all deterministic. In Chapter 3, we have shown that both interactions and priorities can be modelled with extended interaction model in the BIP offer semantics. The modelling of architectures with extended interaction models allows to include priorities in architectures. In this section, we consider that all components are extended with the offer predicate (cf. Definition 2.3.1), i.e. $B = (Q, q^0, P, \rightarrow, \uparrow)$.

Recall that for a set of ports P we have firing $\dot{P}$, activation P and negative $\overline{P}$ port typings.

Definitions of extended architectures, their application and composition are similar to the non-extended ones.

Definition 4.4.1. An *extended architecture* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite set of *coordinating components* with pairwise disjoint sets of ports, P_A is a set of ports, such that $\bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A \cup \dot{P}_A \cup \overline{P_A}}$ is an extended interaction model.

Definition 4.4.2. Let $A = (\mathcal{C}, P_A, \gamma)$ be an extended architecture and let $\mathcal{B}$ be a set of components, such that $\bigcup_{B \in \mathcal{B}} P_B \cap \bigcup_{C \in \mathcal{C}} P_C = \emptyset$ and $P_A \subseteq P \overset{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B$. The *application* *of an extended architecture* A to the components $\mathcal{B}$ is the component

$$A(\mathcal{B}) \overset{def}{=} \left(\gamma \bowtie 2^{\dot{P} \setminus \dot{P_A}}\right)(\mathcal{C} \cup \mathcal{B})\,, \tag{4.11}$$

where, $\bowtie$ is defined as in Definition 4.1.2.

Ports that do not belong to P_A are not restricted and can be fired alongside the interactions in γ.

Example 4.4.3. Consider again the components B_1 and B_2 in Figure 4.1(a). In order to ensure mutual exclusion of their `work` states, we apply the architecture $A_{12} = (\emptyset, P_{12}, \gamma_{12})$, where $P_{12} = \{b_1, b_2, f_1, f_2\}$ and $\gamma_{12} = \{\emptyset, \dot{b_1}\overline{f_2}, \dot{b_2}\overline{f_1}, \dot{f_1}, \dot{f_2}\}$.

The interface P_{12} of A_{12} covers all ports of B_1 and B_2, hence, the only possible interactions are those explicitly belonging to γ_{12}. None of the components can take a transition b_i if the second component is in the state `work`, since f_j, for $j \neq i$, is offered in this state. Assuming that the initial states of B_1 and B_2 are `sleep`, the state (`work`, `work`) is unreachable. Notice that we do not need a coordinating component contrary to Example 4.1.3.

In order to define the composition operator, we use characteristic predicates of extended interaction models (cf. Definition 2.3.7).

Definition 4.4.4. Let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two extended architectures and let $\varphi_{\gamma_1}, \varphi_{\gamma_2}$ be characteristic predicates (Definition 2.3.7) of γ_1, γ_2, respectively. The *composition* of A_1 and A_2 is an extended architecture $A_1 \oplus A_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, P_{A_1} \cup P_{A_2}, \gamma_\varphi)$, where $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$ and $\gamma_\varphi = \|\varphi\|$ is an extended interaction model defined by the predicate φ.

Lemma 4.4.5 is a generalisation of Lemma 4.2.2. It characterises the extended interaction model of the architecture composition. If none of the composed architectures has negative or activation ports in their interaction models, then Lemma 4.4.5 and Lemma 4.2.2 coincide. In the general case, for extended architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, a projection of an extended interaction a of the interaction model of $A_1 \oplus A_2$ onto P_1 (resp. P_2) might not belong to γ_1 (resp. γ_2). Nevertheless, γ_1(resp. γ_2) has to contain a less restrictive extended interaction b with the same firing support as the projection of a, i.e. $b \subseteq a$ and $\mathbf{fire}(b) = \mathbf{fire}(a) \cap P_1$ (resp. P_2).

Lemma 4.4.5. *Consider two extended interaction models* $\gamma_i \subseteq 2^{P_i}$*, for* $i = 1, 2$*, and let* $\varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$*. For an extended interaction* $a \subseteq P_1 \cup \dot{P_1} \cup \overline{P_1} \cup P_2 \cup \dot{P_2} \cup \overline{P_2}$, $a \in \gamma_\varphi$ *iff for* $i = 1, 2$*, there exists* $a_i \in \gamma_i$ *such that* $a_i \subseteq a$ *and* $\mathbf{fire}(a_i) = \mathbf{fire}(a) \cap P_i$*. Assuming that* γ_1, γ_2 *and* γ_φ *do not contain redundant extended interactions (cf. Lemma 2.3.5),*

$$\gamma_\varphi = \left\{ a_1 \cup a_2 \;\middle|\; \begin{array}{l} a_1 \in \gamma_1,\ a_2 \in \gamma_2, \\ \mathbf{fire}(a_1) \cap P_2 = \mathbf{fire}(a_2) \cap P_1, \\ \mathbf{act}(a_1) \cap \mathbf{neg}(a_2) = \mathbf{neg}(a_1) \cap \mathbf{act}(a_2) = \emptyset \end{array} \right\}.$$

Proof. Let $v(p)$ be any valuation $P_1 \cup P_2 \cup \dot{P}_1 \cup \dot{P}_2 \to \mathbb{B}$ corresponding to a, such that $v(\dot{p}) = (p \in \mathbf{fire}(a))$, $v(p) = \mathit{true}$, for all $p \in \mathbf{fire}(a) \cup \mathbf{act}(a)$ and $v(p) = \mathit{false}$, for all $p \in \mathbf{neg}(a)$. Contrary to the proof of Lemma 4.2.2 there can be several valuations, since $v(p)$ can take any value for $p \notin \mathbf{fire}(a) \cup \mathbf{act}(a) \cup \mathbf{neg}(a)$.

We have $a \overset{i}{\models} \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$ iff $(\varphi_{\gamma_1} \wedge \varphi_{\gamma_2})(v) = \mathit{true}$, which is equivalent to $\varphi_{\gamma_1}(v) = \mathit{true}$ and $\varphi_{\gamma_2}(v) = \mathit{true}$. Consider a restriction $v' : P_1 \cup \dot{P}_1 \to \mathbb{B}$ of v to $P_1 \cup \dot{P}_1$ defined by putting, for $p \in P_1 \cup \dot{P}_1$, $v'(p) = v(p)$. Since the variables $p \in (P_2 \cup \dot{P}_2) \setminus (P_1 \cup \dot{P}_1)$ do not appear in φ_{γ_1}, we have $\varphi_{\gamma_1}(v) = \mathit{true}$ iff $\varphi_{\gamma_1}(v') = \mathit{true}$ for any valuation $v(p)$ corresponding to a. Thus, there exists an extended interaction $a_1 \in \gamma_1$ such that $\mathbf{fire}(a_1) = \mathbf{fire}(a) \cap P_1$, $\mathbf{act}(a_1) \subseteq \mathbf{act}(a) \cap P_1$ and $\mathbf{neg}(a_1) \subseteq \mathbf{neg}(a) \cap P_1$, i.e. $a_1 \subseteq a$. The same holds for γ_2.

Consider two extended interactions $a_1 \in \gamma_1$, $a_2 \in \gamma_2$, such that $\mathbf{fire}(a_1) \cap P_2 = \mathbf{fire}(a_2) \cap P_1$ and $\mathbf{act}(a_1) \cap P_2 \cap \mathbf{neg}(a_2) = \mathbf{neg}(a_1) \cap \mathbf{act}(a_2) = \emptyset$. For an interaction $a_1 \cup a_2$ and for $i \in \{1, 2\}$, holds $a_i \subseteq a$ and $\mathbf{fire}(a_i) = \mathbf{fire}(a) \cap P_i$. Thus, $a_1 \cup a_2 \in \gamma_\varphi$.

For any extended interaction $a \in \gamma_\varphi$, there exist $a_1 \in \gamma_1$ and $a_2 \in \gamma_2$, such that $a_i \subseteq a$ and $\mathbf{fire}(a_i) = \mathbf{fire}(a) \cap P_i$, for $i \in \{1, 2\}$. Trivially, $\mathbf{fire}(a_1) \cap P_2 = \mathbf{fire}(a) \cap P_1 \cap P_2 = \mathbf{fire}(a_2) \cap P_1$. Assume that there exists $p \in \mathbf{act}(a_1) \cap \mathbf{neg}(a_2)$, such that both $p \in a$ and $\overline{p} \in a$. In this case a can never be enabled, so a is redundant and can be removed from γ_φ. Thus $\mathbf{act}(a_1) \cap \mathbf{neg}(a_2) = \emptyset$ and, similarly, $\mathbf{neg}(a_1) \cap \mathbf{act}(a_2) = \emptyset$. If $a \neq a_1 \cup a_2$, then, by the reasoning in the previous paragraph, $a_1 \cup a_2 \in \gamma_\varphi$ and $\mathbf{fire}(a) = \mathbf{fire}(a_1 \cup a_2)$. Since $a_1 \cup a_2 \subset a$, by Lemma 2.3.5, a is redundant and can be removed from γ_φ.

$\square$

Example 4.4.6. Mutual exclusion between two components B_i and B_j can be enforced by the extended architecture $A_{i,j} = (\emptyset, P_{i,j} = \{b_i, f_i, b_j, f_j\}, \gamma_{i,j} = \{\emptyset, \dot{b}_i\overline{f_j}, \dot{b}_j\overline{f_i}, \dot{f}_i, \dot{f}_j\})$ (cf. Example 4.4.3). Mutual exclusion between three components can be enforced by a composition of extended architectures $A_{1,2} \oplus A_{1,3} \oplus A_{2,3}$. Their characteristic predicates, simplified by the axiom $\dot{p} \Rightarrow p$, are

$$\begin{aligned}
\varphi_{\gamma_{12}} =& \overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_1}\,\overline{\dot{f}_2} \;\vee\; \dot{b}_1\overline{\dot{b}_2}\,\overline{\dot{f}_1}\,\overline{f_2} \;\vee\; \dot{b}_2\overline{\dot{b}_1}\,\overline{\dot{f}_2}\,\overline{f_1} \;\vee\; \dot{f}_1\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_2} \;\vee\; \dot{f}_2\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_1}\\
\varphi_{\gamma_{13}} =& \overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}\,\overline{\dot{f}_3} \;\vee\; \dot{b}_1\overline{\dot{b}_3}\,\overline{\dot{f}_1}\,\overline{f_3} \;\vee\; \dot{b}_3\overline{\dot{b}_1}\,\overline{\dot{f}_3}\,\overline{f_1} \;\vee\; \dot{f}_1\overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_3} \;\vee\; \dot{f}_3\overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}\\
\varphi_{\gamma_{23}} =& \overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_2}\,\overline{\dot{f}_3} \;\vee\; \dot{b}_2\overline{\dot{b}_3}\,\overline{\dot{f}_2}\,\overline{f_3} \;\vee\; \dot{b}_3\overline{\dot{b}_2}\,\overline{\dot{f}_3}\,\overline{f_2} \;\vee\; \dot{f}_2\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_3} \;\vee\; \dot{f}_3\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_2}.
\end{aligned}$$

Conjunction of the $\varphi_{\gamma_{12}}$ with $\varphi_{\gamma_{13}}$ is equal to

$$\begin{aligned}
&\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}\,\overline{\dot{f}_2}\,\overline{\dot{f}_3} \vee \dot{b}_3\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_2}\,\overline{f_1} \vee \dot{f}_3\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_1}\,\overline{\dot{f}_2} \vee \dot{b}_2\overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_3}\,\overline{f_1} \vee \dot{f}_2\overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}\,\overline{\dot{f}_3} \vee \dot{b}_1\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}\,\overline{f_2}\,\overline{f_3}\\
&\vee \dot{b}_2\dot{b}_3\overline{\dot{b}_1}\,\overline{\dot{f}_2}\,\overline{\dot{f}_3}\,\overline{f_1} \vee \dot{b}_2\dot{f}_3\overline{\dot{b}_1}\,\overline{\dot{b}_3}\,\overline{\dot{f}_2}\,\overline{f_1} \vee \dot{f}_1\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_2}\,\overline{\dot{f}_3} \vee \dot{b}_3\dot{f}_2\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{f}_3}\,\overline{f_1} \vee \dot{f}_2\dot{f}_3\overline{\dot{b}_1}\,\overline{\dot{b}_2}\,\overline{\dot{b}_3}\,\overline{\dot{f}_1}.
\end{aligned}$$

Notice that the composition of two architectures ensures mutual exclusiveness of components B_1, B_2 and of components B_1, B_3, but allows to fire b_2 and b_3 simultaneously provided that component B_1 is not in the working state. The conjunction of all three characteristic predicates is:

$$\begin{aligned}\overline{\dot b_1}\,\overline{\dot b_2}\,\overline{\dot b_3}\,\overline{\dot f_1}\,\overline{\dot f_2}\,\overline{\dot f_3} \;\vee\; \dot b_1\overline{\dot b_2}\,\overline{\dot b_3}\,\overline{\dot f_1}\,\overline{f_2}\,\overline{f_3} \;\vee\; \dot f_1\overline{\dot b_1}\,\overline{\dot b_2}\,\overline{\dot b_3}\,\overline{\dot f_2}\,\overline{\dot f_3} \;\vee\; \dot b_2\overline{\dot b_1}\,\overline{\dot b_3}\,\overline{\dot f_2}\,\overline{f_1}\,\overline{f_3}&\\ \vee\; \dot b_3\overline{\dot b_1}\,\overline{\dot b_2}\,\overline{\dot f_3}\,\overline{f_1}\,\overline{f_2} \;\vee\; \dot f_2\overline{\dot b_1}\,\overline{\dot b_2}\,\overline{\dot b_3}\,\overline{\dot f_1}\,\overline{\dot f_3} \;\vee\; \dot f_3\overline{\dot b_1}\,\overline{\dot b_2}\,\overline{\dot b_3}\,\overline{\dot f_1}\,\overline{\dot f_2}.&\end{aligned}$$

Finally, it is straightforward to obtain an extended interaction model:

$$\gamma_{1,2,3} = \{\emptyset, \dot b_1\overline{f_2}\,\overline{f_3}, \dot b_2\overline{f_1}\,\overline{f_3}, \dot b_3\overline{f_1}\,\overline{f_2}, \dot f_1, \dot f_2, \dot f_3\}.$$

Notice that a projection of $\dot b_1\overline{f_2}\,\overline{f_3}$ onto $P_{2,3}$ is $\overline{f_2}\,\overline{f_3}$ which is not in the extended interaction model $\gamma_{2,3}$, however, for $\emptyset \in \gamma_{2,3}$, $\emptyset \subset \overline{f_2}\,\overline{f_3}$ and $\mathbf{fire}(\emptyset) = \mathbf{fire}\big(\overline{f_2}\,\overline{f_3}\big)$.

The rest of the results presented in Sections 4.2 and 4.3 can be easily generalised to extended architectures. Rather than reproduce all of them we only show the generalisation of the main result (Theorem 4.3.5).

Lemma 4.4.7. *Consider a set of components $\mathcal{B}$ and two extended architectures $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$. Let $\tilde q_1\tilde q_2 q \xrightarrow{\mathbf{fire}(a)} \tilde q_1'\tilde q_2' q'$ be a transition in $(A_1 \oplus A_2)(\mathcal{B})$, where, for $i = 1, 2$, $\tilde q_i, \tilde q_i' \in \prod_{C\in\mathcal{C}_i} Q_C$ and $q, q' \in \prod_{B\in\mathcal{B}} Q_B$. Then, for $i = 1, 2$, if $\mathbf{fire}(a)\cap(P_{A_i}\cup P) \neq \emptyset$, then $\tilde q_i q \xrightarrow{\mathbf{fire}(a)\cap(P_{A_i}\cup P)} \tilde q_i' q'$ is a transition in $A_i(\mathcal{B})$, where $P = \bigcup_{B\in\mathcal{B}} P_B$.*

Proof. By Lemma 4.1.4, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$.

By Definition 4.4.4, $a \cap (\dot P_{A_1} \cup P_{A_1} \cup \overline{P_{A_1}} \cup \dot P_{A_2} \cup P_{A_2} \cup \overline{P_{A_2}}) \overset{i}{\models} \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$. By Lemma 4.4.5, there exists $a_1 \in \gamma_1$, such that $\mathbf{fire}(a) \cap P_{A_1} = \mathbf{fire}(a_1)$ and $a_1 \subseteq a$. Hence,

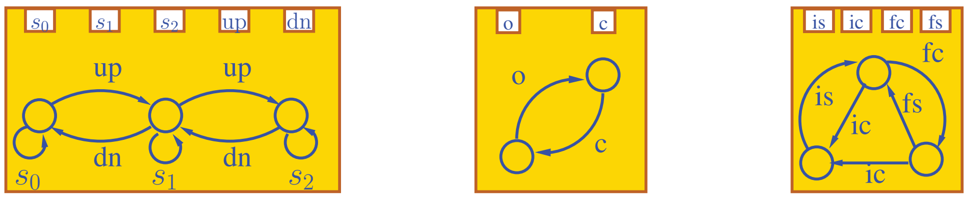
$$\mathbf{fire}(a)\cap(P_{A_1}\cup P) \;=\; (\mathbf{fire}(a)\cap P_{A_1})\cup(\mathbf{fire}(a)\cap(P\setminus P_{A_1})) \;=\; \mathbf{fire}(a_1)\cup(\mathbf{fire}(a)\cap(P\setminus P_{A_1}))$$

and $\tilde a \overset{def}{=} a_1 \cup (\mathbf{fire}(a) \cap (P \setminus P_{A_1})) \in \big(\gamma_1 \bowtie 2^{\dot P\setminus\dot P_{A_1}}\big)$. By the assumption of the lemma, $\mathbf{fire}(\tilde a) \neq \emptyset$. Furthermore, since $\tilde q_1\tilde q_2 q \xrightarrow{\dot a} \tilde q_1'\tilde q_2' q'$, we have by (2.26),

$$\begin{cases}\tilde q_1 \xrightarrow{\mathbf{fire}(a)\cap P_{C_1}} \tilde q_1', & \text{if } \mathbf{fire}(a)\cap P_{C_1} \neq \emptyset,\\ \tilde q_1 = \tilde q_1', & \text{if } \mathbf{fire}(a)\cap P_{C_1} = \emptyset,\end{cases} \quad \text{and } \tilde q_1 {\uparrow} \mathbf{act}(a)\cap P_{C_1}\,,\ \tilde q_1 \not\uparrow \mathbf{neg}(a)\cap P_{C_1}\,;$$

$$\text{for } i\in[1,n], \begin{cases}q_i \xrightarrow{\mathbf{fire}(a)\cap P_i} q_i', & \text{if } \mathbf{fire}(a)\cap P_i \neq \emptyset,\\ q_i = q_i', & \text{if } \mathbf{fire}(a)\cap P_i = \emptyset.\end{cases} \quad \text{and } q_i {\uparrow} \mathbf{act}(a)\cap P_i\,,\ q_i \not\uparrow \mathbf{neg}(a)\cap P_i\,.$$

Since $P_{C_1} \subseteq P_{A_1}$, we have $\tilde a \cap (\dot P_{C_1} \cup P_{C_1} \cup \overline{P_{C_1}}) = a \cap (\dot P_{C_1} \cup P_{C_1} \cup \overline{P_{C_1}})$. Similarly, for any $i \in [1, n]$, $P_i \subseteq P$, hence $\tilde a \cap (\dot P_i \cup P_i \cup \overline{P_i}) = a \cap (\dot P_i \cup P_i \cup \overline{P_i})$. Thus, all premises of the



(a) Elevator Engine E

(b) Elevator Door D

(c) Caller System CS

Figure 4.4 – Atomic components for elevator example.

instance of the rule (2.26) for $\tilde{a}$ in $A_1(\mathcal{B})$ are satisfied and we have $\tilde{q}_1 q \xrightarrow{\mathbf{fire}(\tilde{a})} \tilde{q}_1' q'$ in $A_1(\mathcal{B})$. For $A_2(\mathcal{B})$, the result is obtained by a symmetrical argument. □

This lemma generalises Lemma 4.2.4. Theorem 4.4.8 shows the preservation of safety properties by the composition of extended architectures. Its proof is identical to the one of Theorem 4.3.5 up to the reference to the Lemma 4.4.7 instead of Lemma 4.2.4 and the use of firing support of interactions in the path fragment.

Theorem 4.4.8. *Let $\mathcal{B}$ be a set of components; let $A_i = (\mathcal{C}_i, P_{A_i}, \gamma_i)$, for $i = 1, 2$, be two extended architectures enforcing on $\mathcal{B}$ safety properties Φ_1 and Φ_2, respectively. The composition $A_1 \oplus A_2$ enforces on $\mathcal{B}$ the property $\Phi_1 \wedge \Phi_2$.*

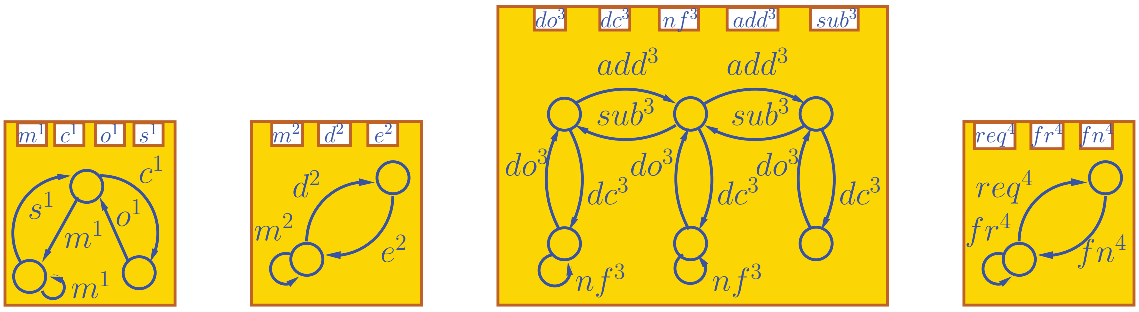
Proof. By Lemma 4.1.4, we can assume that each of the two architectures has only one coordinating component, i.e. $\mathcal{C}_i = \{C_i\}$, for $i = 1, 2$. We also denote, for $i = 1, 2$, $P_i = P_{C_i} \cup \bigcup_{B \in \mathcal{B}} P_B$.

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

Consider a path $\tilde{q}_1^0 \tilde{q}_2^0 q^0 \xrightarrow{\mathbf{fire}(a_1)} \tilde{q}_1^1 \tilde{q}_2^1 q^1 \xrightarrow{\mathbf{fire}(a_2)} \cdots \xrightarrow{\mathbf{fire}(a_k)} \tilde{q}_1^k \tilde{q}_2^k q^k$ in $(A_1 \oplus A_2)(\mathcal{B})$, where $q^0, \dots, q^k \in \prod_{B \in \mathcal{B}} Q_B$ and $\tilde{q}_i^0, \dots, \tilde{q}_i^k \in Q_{C_i}$, for $i = 1, 2$. By Lemma 4.4.7, $\tilde{q}_1^0 q^0 \xrightarrow{\mathbf{fire}(a_1) \cap P_1} \tilde{q}_1^1 q^1 \xrightarrow{\mathbf{fire}(a_2) \cap P_1} \cdots \xrightarrow{\mathbf{fire}(a_k) \cap P_1} \tilde{q}_1^k q^k$ is a path in $A_1(\mathcal{B})$. (If, for some $i \in [1, k]$, $\mathbf{fire}(a_i) \cap P_1 = \emptyset$, the corresponding transition can be omitted from the path.) Thus the state $\tilde{q}_1^k q^k$ is reachable in $A_1(\mathcal{B})$. Since A_1 enforces Φ_1 on $\mathcal{B}$, this implies that $q^k \models \Phi_1$. Symmetrically, $q^k \models \Phi_2$, which concludes the proof. □

4.5 Case study: control of an elevator cabin

We illustrate our results with the case study adapted from the literature [50, 89], which models an elevator in a building with three floors. Control of the elevator cabin is modelled as a set of coordinated atomic components shown in Figure 4.4. 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



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

Figure 4.5 – Coordinating components for the elevator example.

is and *fs* represent cabin stops in response to these calls. Furthermore, port names m, c, o, *do*, *dc*, s, *dn*, *up* and *nf* stand, respectively, for "move", "close", "open", "door open", "door close", "stop", "move down", "move up" and "not full". For the coordinating components of the architectures in the case study, we will use super-indices to show explicitly which port belongs to which coordinating component, as in s^1 for the port "stop" of coordinator C_1 (see Figure 4.5(a)). Caller system components and their ports are indexed by floor numbers. We denote $\mathcal{B} = \{E, D, CS_0, CS_1, CS_2\}$ the set of atomic components. To enforce required properties, we successively apply and compose architectures.

In order to provide the basic functionality of the elevator, we apply to $\mathcal{B}$ the architecture $A_1 = (\{C_1\}, P_1, \gamma_1)$. Component C_1 is shown in Figure 4.5(a). P_1 contains all ports of C_1 and all ports of $\mathcal{B}$. γ_1 comprises the empty interaction $\emptyset$ and the following interactions (for $i \in [0, 2]$):

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

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

Nonetheless, $A_1(\mathcal{B})$ allows the elevator to stop at a floor, and then to leave without having opened the door. The property *"if the elevator stops, it has to open doors before continue moving"* can be enforced by architecture $A_2 = (\{C_2\}, P_2, \gamma_2)$ where C_2 is shown in Figure 4.5(b), $P_2 = \{e^2, d^2, m^2, c^1, m^1, s_0, s_1, s_2\}$, and $\gamma_2 = \{\emptyset,\ c^1e^2,\ m^1m^2\} \cup \{s_i d^2 \mid i \in [0, 2]\}$. This grants priority to the door controller after a "stop" action. By Proposition 4.2.7, $A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$. $(A_2 \oplus A_1)(\mathcal{B})$ provides the same functionality as $A_1(\mathcal{B})$ and also this additional property.

The property *"if the elevator is full, it must stop only at floors selected from the cabin and ignore outside calls"* [50, 89], is enforced by applying architecture $A_3 = (\{C_3\}, P_3, \gamma_3)$ with C_3 shown in Figure 4.5(c), $P_3 = \{add^3, sub^3, nf^3, do^3, dc^3, o, c\} \cup \{s_i, fs_i \,|\, i \in [0,2]\}$ and $\gamma_3 = \{\emptyset,\ add^3,\ sub^3,\ do^3 o,\ dc^3 c\} \cup \{s_i\, fs_i\, nf^3 \,|\, i \in [0,2]\}$. An elevator is full in our example if it has two passengers on board, i.e. C_3 has reached the bottom right-most state (see Figure 4.5(c)) by twice firing the port add^3 without firing the port sub^3 in the meantime. By Proposition 4.2.7, $A_3\big((A_1 \oplus A_2)(\mathcal{B})\big) = (A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$. By Theorem 4.3.5, $(A_1 \oplus A_2 \oplus A_3)(\mathcal{B})$ satisfies all three properties.

Finally, we consider the additional property: *"requests from the second floor have priority over all other requests"* [50, 89]. This is enforced by the architecture $A_4 = (\{C_4\}, P_4, \gamma_4)$ with C_4 shown in Figure 4.5(d). P_4 consists of all ports of C_4 and CS_2, and ports o and dn of E, whereas $\gamma_4 = \{\emptyset,\ fc_2\, req^4,\ ic_2\, req^4,\ o\, fr^4,\ dn\, fr^4,\ fs_2\, fn^4,\ is_2\, fn^4\}$. Notice that the system $(A_1 \oplus A_2 \oplus A_3 \oplus A_4)(\mathcal{B})$ has a local deadlock, i.e. a reachable state such that no transition involving one of the composed components can be taken after reaching this state. This deadlock occurs when a full elevator is called from the second floor. Once the second floor is reached, A_4 enforces the constraint of not going down, while A_3 forbids stopping at this floor. Hence, the system is in a local deadlock state involving the elevator engine.

4.6 Discussion

Reusable solutions can be modelled by architectures. They are the step towards correct-by-construction system design. Architecture operators restrict the behaviour of their arguments enforcing a characteristic property. They can be composed and studied independently. Using BIP to describe architectures allows to keep a separation between computation and coordination. The application of architectures does not require any modification of the atomic components.

Architecture composition allows to build complex architectures from simple ones. It always preserves safety properties. The preservation of liveness properties is studied in [9]. In short, the system is live iff it is free of global deadlock and composed architectures are pairwise non-interfering. Non-interference of two architectures requires that, for each path, coordinating components of both architectures either be invoked infinitely often or be in idle states infinitely often or be enabled continuously from some point onwards, i.e. each state of the path enables every interaction that the coordinator is ready to participate in. Pairwise non-interference can be checked algorithmically.

In this work, architectures define only synchronisation constraints between coordinated components but they do not specify data transfer. The extension of architectures with data is proposed in [48].

However, architectures have an important limitation: an architecture is defined for a specific number of components. Thus, in order to provide generic solution for mutual exclusion, one has to define architectures for two, three, etc. components or to compose architectures for each pair of components during the modelling of the system. An alternative solution is the specification of architecture styles that model generic architectures.

5 Configuration Logics

Architectures implicitly define the number of components they can be applied to. Thus, in order to enforce a property on two components and the same property on three components, one need two different architectures. The fourth component would require another architecture. Moreover, there might exist several architectures enforcing the same property on the same number of components (cf. Figure 1.1). In this chapter, we study modelling of architecture styles that characterise not a single architecture but a family of architectures sharing common characteristics such as the type of the involved components and the topology induced by their coordination structure.

We propose configuration logic, formulas of which specify configuration sets. A configuration on a set of components represents a particular architecture. Configuration logic is a powerset extension of interaction logic. First, we introduce a propositional configuration logic whose formulas represent, for a given set of components, the allowed configuration sets. We provide its sound and complete axiomatisation and a normal form similar to the disjunctive normal form in Boolean algebras. The existence of such normal form implies the decidability of formula equality and the satisfaction of a formula by an architecture model.

To allow genericity of specifications, we study first-order and second-order logics as extensions of the propositional logic. They are defined for types of components and involve quantification over component variables and variables for sets of components in the second-order logic. Second-order logic is needed to express some interesting topological properties, e.g. the existence of cycles of interactions. We also study an alternative extension of the first-order logic with ordered components. In this extension, components in the models are ordered linearly and formulas can have constraints relying on the order. This allows to specify some styles that are inexpressible in the first-order logic. Specifications of various architecture styles are illustrated with examples.

We also study the relation between the composition of architectures and conjunction of configuration logic formulas. For a class of formulas that corresponds to safety properties, we show that, for two formulas and two architectures satisfying them, the composition of architectures satisfies conjunction of formulas.

5.1 Propositional configuration logic

Syntax. The propositional configuration logic (PCL) is an extension of PIL (cf. Section 2.2.2) defined by the grammar:

$$f ::= \mathit{true} \mid \phi \mid \neg\, f \mid f + f \mid f \sqcup f\,, \tag{5.1}$$

where ϕ is a PIL formula; $\neg$, $+$ and $\sqcup$ are, respectively, the *complementation*, *coalescing* and *union* operators.

Additionally, we define the usual notation for intersection and implication:

$$\begin{aligned} f_1 \sqcap f_2 &\overset{def}{=} \neg\,(\neg\, f_1 \sqcup \neg\, f_2)\,, \\ f_1 \Rightarrow f_2 &\overset{def}{=} \neg\, f_1 \sqcup f_2\,. \end{aligned}$$

The language of PCL formulas is generated from PIL formulas by using union, coalescing and complementation operators. The binding strength of the operators is as follows (in the decreasing order): PIL negation, complementation, PIL conjunction, PIL disjunction, coalescing, union.

Henceforth, to avoid confusion, we refer as *interaction formulas* to the subset of PCL formulas that syntactically are also PIL formulas. Furthermore, we will use Latin letters f, g, h for general PCL formulas and Greek letters ϕ, ψ, ξ for interaction formulas. Interaction formulas inherit all axioms of PIL.

Semantics. Let P be a set of ports. The semantic domain of PCL is the lattice of configuration sets $CS(P) = 2^{C(P)\setminus\{\emptyset\}}$ (Figure 1.2(c)). The meaning of a PCL formula f is defined by the following satisfaction relation. Let $\gamma \in C(P)$ be a non-empty configuration. We define:

$\gamma \models \mathit{true}\,,$	always,	(5.2)
$\gamma \models \phi\,,$	if for all $a \in \gamma,\ a \overset{i}{\models} \phi$, where ϕ is an interaction formula and $\overset{i}{\models}$ is the satisfaction relation of PIL,	(5.3)
$\gamma \models f_1 + f_2\,,$	if there exist $\gamma_1, \gamma_2 \in C(P) \setminus \{\emptyset\}$, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models f_1$ and $\gamma_2 \models f_2$,	(5.4)
$\gamma \models f_1 \sqcup f_2\,,$	if $\gamma \models f_1$ or $\gamma \models f_2$,	(5.5)
$\gamma \models \neg\, f\,,$	if $\gamma \not\models f$ (i.e. $\gamma \models f$ does not hold).	(5.6)

In particular, the meaning of an interaction formula ϕ in PCL is the set $2^{I_a} \setminus \{\emptyset\}$, with

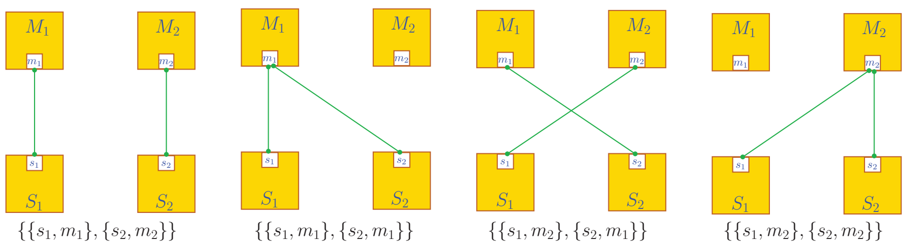

M_1
M_2
m_1
m_2
s_1
s_2
S_1
S_2
$\{\{s_1, m_1\}, \{s_2, m_2\}\}$
$\{\{s_1, m_1\}, \{s_2, m_1\}\}$
$\{\{s_1, m_2\}, \{s_2, m_1\}\}$
$\{\{s_1, m_2\}, \{s_2, m_2\}\}$

Figure 5.1 – Master/Slave architectures.

$I_a = \{a \in I(P) \,|\, a \overset{i}{\models} \phi\}$, of all configurations involving any number of interactions satisfying ϕ in PIL.

The semantics of intersection and implication can also be stated directly as follows:

$$\gamma \models f_1 \sqcap f_2\,, \qquad \text{if } \gamma \models f_1 \text{ and } \gamma \models f_2, \tag{5.7}$$
$$\gamma \models f_1 \Rightarrow f_2\,, \qquad \text{if } \gamma \not\models f_1 \text{ or } \gamma \models f_2. \tag{5.8}$$

We say that two formulas are equivalent $f_1 \equiv f_2$ iff, for all $\gamma \in C(P)$ such that $\gamma \neq \emptyset$, $\gamma \models f_1 \Leftrightarrow \gamma \models f_2$.

We denote by $|f| \overset{def}{=} \{\gamma \in C(P) \setminus \{\emptyset\} \,|\, \gamma \models f\}$ the characteristic configuration set of the formula f. Clearly $f_1 \equiv f_2$ iff $|f_1| = |f_2|$.

Proposition 5.1.1. *Equivalence $\equiv$ is a congruence w.r.t. all PCL operations.*

Proof. In order to prove the proposition, it is sufficient to show that for each binary operator *op* from the PCL grammar (5.1), the characteristic configuration set of the formula $f_1\, op\, f_2$ can be expressed as a function of characteristic configuration sets of f_1 and f_2. In other words, we have to exhibit a binary operator op' on sets, such that $|f_1\, op\, f_2| = op'(|f_1|, |f_2|)$. Similarly, we have to exhibit an unary operator on sets, expressing the characteristic configuration set of the formula $\neg\, f$ in terms of the characteristic configuration set of f.

Clearly, the set operators corresponding to $\neg$ and $\sqcup$ are, respectively, complementation with respect to $C(P) \setminus \{\emptyset\}$ and set union. For the coalescing operator $+$, it is easy to see that, defining

$$op'_+(X, Y) \overset{def}{=} \{\gamma_1 \cup \gamma_2 \,|\, \gamma_1 \in X, \gamma_2 \in Y\}\,,$$

we have $|f_1 + f_2| = op'_+(|f_1|, |f_2|)$. □

Example 5.1.2. The Master/Slave architecture style for two masters M_1, M_2 and two slaves S_1, S_2 with ports $m_1,\ m_2,\ s_1,\ s_2$, respectively, characterises the four configurations of Figure 5.1 as the union:

$$\bigsqcup_{i,j\in\{1,2\}} (\phi_{1,i} + \phi_{2,j}),$$

where $\phi_{i,j} = s_i \wedge m_j \wedge \overline{s_{i'}} \wedge \overline{m_{j'}}$, for $i \neq i', j \neq j'$, is a monomial defining a binary interaction between ports s_i and m_j.

This formula can be alternatively written as a coalescing of interactions for each slave:

$$(\phi_{1,1} \sqcup \phi_{1,2}) + (\phi_{2,1} \sqcup \phi_{2,2}).$$

Any configuration satisfying this formula consists of two parts, which satisfy, respectively, the left and the right terms of the coalescing operator. The left term requires either an interaction $\{s_1, m_1\}$ or an interaction $\{s_1, m_2\}$. Similarly, the right term requires exactly one interaction among $\{s_2, m_1\}$ and $\{s_2, m_2\}$. Therefore, there are four possible pairs of interactions corresponding to the four configurations of Figure 5.1.

From the PCL semantics of interaction formulas (5.3), it follows immediately that PCL is a conservative extension of PIL. Below we extend the PIL conjunction and disjunction operators to PCL.

PCL intersection is a conservative extension of PIL conjunction.

Proposition 5.1.3. *$\phi_1 \wedge \phi_2 \equiv \phi_1 \sqcap \phi_2$, for any interaction formulas ϕ_1, ϕ_2.*

Proof. For any two interaction formulas ϕ_1 and ϕ_2, $\phi_1 \wedge \phi_2$ is also an interaction formula. Hence, by (5.3), $\gamma \models \phi_1 \wedge \phi_2$ iff $\gamma \subseteq \{a \,|\, a \overset{i}{\models} \phi_1 \wedge \phi_2\} = \{a \,|\, a \overset{i}{\models} \phi_1 \wedge a \overset{i}{\models} \phi_2\}$. By (5.7), $\gamma \models \phi_1 \sqcap \phi_2$ iff $\gamma \models \phi_1$ and $\gamma \models \phi_2$, that is $\gamma \subseteq \{a \,|\, a \overset{i}{\models} \phi_1\} \cap \{a \,|\, a \overset{i}{\models} \phi_2\} = \{a \,|\, a \overset{i}{\models} \phi_1 \wedge a \overset{i}{\models} \phi_2\}$. Since characteristic configuration sets of formulas coincide, $\phi_1 \wedge \phi_2 \equiv \phi_1 \sqcap \phi_2$. □

Thus, conjunction and intersection coincide on interaction formulas. In the rest of the chapter, we use the same symbol $\wedge$ to denote both operators.

Disjunction can be conservatively extended to PCL with the following semantics: for any PCL formulas f_1 and f_2,

$$\gamma \models f_1 \vee f_2\,, \qquad \text{if } \gamma \models f_1 \sqcup f_2 \sqcup f_1 + f_2. \tag{5.9}$$

Proposition 5.1.4. *For any interaction formulas ϕ_1 and ϕ_2 and any $\gamma \in C(P) \setminus \emptyset$, we have $\gamma \models \phi_1 \vee \phi_2$ iff $\forall a \in \gamma, a \overset{i}{\models} \phi_1 \vee \phi_2$.*

Proof. The PCL semantics defines $\gamma \models \phi_1 \vee \phi_2$ if $\gamma \models \phi_1$ or $\gamma \models \phi_2$ or there exist γ_1 and γ_2, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models \phi_1$ and $\gamma_2 \models \phi_2$, where $\gamma \models \phi$ if for all $a \in \gamma$, $a \overset{i}{\models} \phi$. Thus, in all three cases all interactions in γ either satisfy ϕ_1 or ϕ_2 and, consequently, for all $a \in \gamma$, $a \overset{i}{\models} \phi_1 \vee \phi_2$.

Conversely, if γ consists of interactions a, such that $a \overset{i}{\models} \phi_1 \vee \phi_2$, these interactions can be split into two possibly empty sets γ_1 and γ_2, such that for all $a \in \gamma_j$, where $j \in [1, 2]$, $a \overset{i}{\models} \phi_j$. If one of these groups is empty then the second one contains all interactions and $\gamma \models \phi_j$. Otherwise, $\gamma_1 \models \phi_1$ and $\gamma_2 \models \phi_2$, where $\gamma_1 \cup \gamma_2 = \gamma$. In all cases $\gamma \models \phi_1 \vee \phi_2$. $\square$

5.1.1 Properties of PCL

In this subsection, we present the key properties of PCL operators, which allow us to define a normal form and a sound and complete axiomatisation of PCL.

Union, complementation and conjunction have the standard set-theoretic meaning.

Proposition 5.1.5. *The operators $\sqcup$, $\neg$, $\wedge$ satisfy the usual axioms of propositional logic.*

Proof. The proof is immediate from the semantics (5.5), (5.6) and (5.7). $\square$

Notice that coalescing $+$ combines configurations, as opposed to union $\sqcup$, which combines configuration sets. Coalescing has the following properties:

Proposition 5.1.6. *$+$ is associative, commutative and has an absorbing element $false \overset{def}{=} \neg true$.*

Proof. The proof is immediate from the semantics (5.4). $\square$

Coalescing distributes over union, as shown in the following proposition.

Proposition 5.1.7. *For any formulas f, f_1, f_2, the following distributivity result holds:*

$$f + (f_1 \sqcup f_2) \equiv f + f_1 \sqcup f + f_2 \,.$$

Proof. If $\gamma \models f + (f_1 \sqcup f_2)$, then there exist γ_1 and γ_2, such that $\gamma_1 \cup \gamma_2 = \gamma$, $\gamma_1 \models f$ and $\gamma_2 \models f_1 \sqcup f_2$. If $\gamma_2 \models f_1$, then $\gamma \models f + f_1$. Otherwise, $\gamma_2 \models f_2$ and $\gamma \models f + f_2$. Combining these two cases we obtain $\gamma \models f + f_1 \sqcup f + f_2$.

If $\gamma \models f + f_1 \sqcup f + f_2$, then either $\gamma \models f + f_1$ or $\gamma \models f + f_2$. In the first case there exist γ_1 and γ_2, such that $\gamma_1 \cup \gamma_2 = \gamma$, $\gamma_1 \models f$ and $\gamma_2 \models f_1$. Since $\gamma_2 \models f_1$ implies $\gamma_2 \models f_1 \sqcup f_2$, $\gamma \models f + (f_1 \sqcup f_2)$. The second case is similar. $\square$

Associativity of coalescing and union, together with the distributivity of coalescing over union, immediately imply the following generalisation of the extended semantics of disjunction (5.9).

Corollary 5.1.8. *For any set of formulas $\{f_i\}_{i\in I}$, we have*

$$\bigvee_{i\in I} f_i \equiv \bigsqcup_{\emptyset\neq J\subseteq I} \sum_{j\in J} f_j\,,$$

where $\sum_{j\in J} f_j$ denotes the coalescing of formulas f_j, for all $j \in J$.

Example 5.1.9. A configuration γ satisfying the formula $f = f_1 \vee f_2 \vee f_3$ can be partitioned into $\gamma = \gamma_1 \cup \gamma_2 \cup \gamma_3$, such that $\gamma_i \models f_i$. However, by the semantics of disjunction, some γ_i can be empty. On the contrary, the semantics of coalescing requires all elements of such partition to be non-empty. Hence, in order to rewrite f without the disjunction operator, we take the union of all possible coalescings of f_1, f_2 and f_3. Thus, we have $f \equiv f_1 \sqcup f_2 \sqcup f_3 \sqcup (f_1 + f_2) \sqcup (f_1 + f_3) \sqcup (f_2 + f_3) \sqcup (f_1 + f_2 + f_3)$.

The following proposition shows distributivity results involving disjunction. In particular, it shows that disjunction distributes over union and coalescing distributes over disjunction.

Proposition 5.1.10. *For any formulas f, f_1, f_2, the following distributivity results hold:*

1. $f \vee (f_1 \sqcup f_2) \equiv (f \vee f_1) \sqcup (f \vee f_2)$,
2. $f + (f_1 \vee f_2) \equiv (f + f_1) \vee (f + f_2)$.

Proof. We have

$$\begin{aligned} f \vee (f_1 \sqcup f_2) &\equiv f \sqcup (f_1 \sqcup f_2) \sqcup f + (f_1 \sqcup f_2) \\ &\equiv f \sqcup f_1 \sqcup f + f_1 \sqcup f \sqcup f_2 \sqcup f + f_2 \equiv (f \vee f_1) \sqcup (f \vee f_2) \end{aligned}$$

and

$$\begin{aligned} f + (f_1 \vee f_2) &\equiv f + (f_1 \sqcup f_2 \sqcup f_1 + f_2) \\ &\equiv f + f_1 \sqcup f + f_2 \sqcup f + f_1 + f_2 \equiv (f + f_1) \vee (f + f_2)\,. \end{aligned}$$

□

The following example shows that coalescing does not distribute over conjunction.

Example 5.1.11. Let $P = \{p, q\}$ and consider $f = p \sqcup q$, $f_1 = p$ and $f_2 = q$. We then have $(f + f_1) \wedge (f + f_2) = ((p \sqcup q) + p) \wedge ((p \sqcup q) + q)$ and $f + (f_1 \wedge f_2) = (p \sqcup q) + (p \wedge q)$. The configuration $\{\{p\}, \{q\}\}$ satisfies the former, but not the latter.

Proposition 5.1.12. *For any formulas f, f_1, f_2, the following implication is true:*

$$f + (f_1 \wedge f_2) \Rightarrow (f + f_1) \wedge (f + f_2)\,.$$

Proof. If $\gamma \models f + (f_1 \wedge f_2)$ then there exist γ_1 and γ_2, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models f$, $\gamma_2 \models f_1$ and $\gamma_2 \models f_2$. Hence, we have both $\gamma \models f + f_1$ and $\gamma \models f + f_2$. $\square$

In general, neither conjunction distributes over coalescing nor coalescing over conjunction. To provide more distributivity results, we introduce the following classes of PCL formulas.

Definition 5.1.13.

- A formula f is *downward-closed* iff $\gamma \models f$ implies for all non-empty $\gamma_1 \subseteq \gamma, \gamma_1 \models f$.
- A formula f is *upward-closed* iff $\gamma \models f$ implies for all $\gamma_1 \supseteq \gamma, \gamma_1 \models f$.
- A formula f is $\cup$-*closed* iff $\gamma_1 \models f$ and $\gamma_2 \models f$ implies $\gamma_1 \cup \gamma_2 \models f$.

Example 5.1.14.

- $p \sqcup q$ is downward-closed,
- $\neg\,(p \sqcup q)$ is upward-closed,
- $p \vee q$ is $\cup$-closed.

The following propositions show properties of these classes and their relations.

Proposition 5.1.15. *If f and g are downward- (resp. upward-) closed, then $f \sqcup g$ and $f \wedge g$ are also downward- (resp. upward-) closed.*

Proof. In the first two parts of the proof formulas f and g are downward-closed, while in the last two parts they are upward-closed.

If $\gamma \models f \sqcup g$, then $\gamma \models f$ or $\gamma \models g$. If $\gamma \models f$, then $\forall \gamma_1 \subseteq \gamma$, $\gamma_1 \models f$. Thus, $\gamma_1 \models f \sqcup g$. The case $\gamma \models g$ is similar.

If $\gamma \models f \wedge g$, then $\gamma \models f$ and $\gamma \models g$. If $\gamma \models f$, then $\forall \gamma_1 \subseteq \gamma$, $\gamma_1 \models f$ and similarly for g. Thus, $\gamma_1 \models f \wedge g$.

If $\gamma \models f \sqcup g$, then $\gamma \models f$ or $\gamma \models g$. If $\gamma \models f$, then $\forall \gamma_1 \supseteq \gamma$, $\gamma_1 \models f$. Thus, $\gamma_1 \models f \sqcup g$. The case $\gamma \models g$ is similar.

If $\gamma \models f \wedge g$, then $\gamma \models f$ and $\gamma \models g$. If $\gamma \models f$, then $\forall \gamma_1 \supseteq \gamma$, $\gamma_1 \models f$ and similarly for g. Thus, $\gamma_1 \models f \wedge g$. $\square$

Proposition 5.1.16. *For any formula f, the formula $f + true$ is upward-closed.*

Proof. Let $\gamma \models f + true$. There exists $\gamma_1 \subseteq \gamma$, such that $\gamma_1 \models f$. For any $\gamma_2 \supseteq \gamma$ holds $\gamma_2 \supseteq \gamma_1$ and $\gamma_2 \models f + true$, since *true* is satisfied by any configuration. $\square$

Proposition 5.1.17. *If f is upward-closed then $f \equiv f + true$.*

Proof. If $\gamma \models f$, then $\gamma \cup \gamma = \gamma \models f + true$.
If $\gamma \models f + true$, then there exists $\gamma_1 \subseteq \gamma$, such that $\gamma_1 \models f$. Since f is upward-closed, for any $\gamma \supseteq \gamma_1$, holds $\gamma \models f$. $\square$

Proposition 5.1.18. *If f and g are $\cup$-closed then $f + g$ is also $\cup$-closed.*

Proof. If $\gamma_1 \models f + g$ and $\gamma_2 \models f + g$, then there exist $\gamma_{1,1}$, $\gamma_{1,2}$, $\gamma_{2,1}$ and $\gamma_{2,2}$, such that $\gamma_i = \gamma_{i,1} \cup \gamma_{i,2}$, $\gamma_{i,1} \models f$ and $\gamma_{i,2} \models g$ for $i \in \{1, 2\}$. Since f and g are $\cup$-closed, $\gamma_{1,1} \cup \gamma_{2,1} \models f$ and $\gamma_{1,2} \cup \gamma_{2,2} \models g$ and, consequently, $\gamma_1 \cup \gamma_2 \models f + g$. $\square$

The following proposition shows that the complement of a downward-closed formula is an upward-closed formula.

Proposition 5.1.19. *A formula f is downward-closed iff the formula $\neg f$ is upward-closed.*

Proof. Assume that f is downward-closed and $\neg f$ is not upward-closed. The latter means that there exist γ_1 and $\gamma_2 \supseteq \gamma_1$, such that $\gamma_1 \models \neg f$ and $\gamma_2 \not\models \neg f$. This is equivalent to $\gamma_1 \not\models f$ and $\gamma_2 \models f$, which contradicts the fact that f is downward-closed.

Conversely, assume that $\neg f$ is upward-closed and f is not downward-closed. The latter means that there exist γ_1 and $\gamma_2 \subseteq \gamma_1$, such that $\gamma_1 \models f$ and $\gamma_2 \not\models f$. This is equivalent to $\gamma_1 \not\models \neg f$ and $\gamma_2 \models \neg f$, which contradicts the fact that $\neg f$ is upward-closed. $\square$

The following proposition characterises downward-closed formulas.

Proposition 5.1.20. *Any downward-closed formula f can be expressed in the form $\bigsqcup_{\phi \in \Phi} \phi$, where Φ is a set of interaction formulas.*

Proof. For a configuration γ, we denote ϕ_γ an interaction formula syntactically equal to the characteristic predicate of γ (Definition 2.2.17). ϕ_γ is satisfied by γ and any sub-configuration of γ.

Let $\Gamma = |f|$ be a characteristic configuration set of f. Consider a formula $f' = \bigsqcup_{\gamma \in \Gamma} \phi_\gamma$. By (5.5) and (5.3), for all $\gamma \in \Gamma$, $\gamma \models f'$. By (5.5), a configuration γ' satisfies f' iff $\gamma' \models \phi_\gamma$ for some $\gamma \in \Gamma$. By (5.3), $\gamma' \subseteq \gamma$. Since f is downward-closed, $\gamma' \in \Gamma$. Thus, $f' \equiv f$. $\square$

Proposition 5.1.21. *A formula is $\cup$-closed and downward-closed iff it is an interaction formula.*

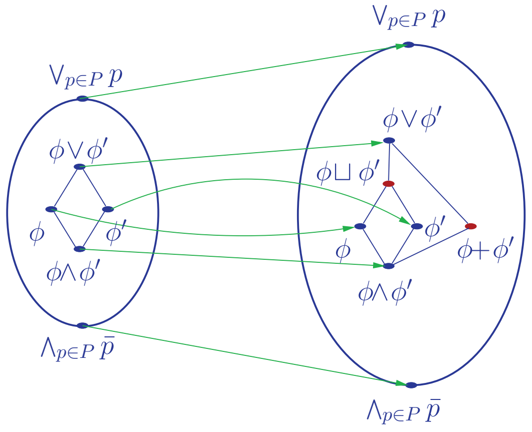


Figure 5.2 – Correspondence between the lattices of PIL and PCL.

Proof. Let ϕ be an interaction formula. Consider two configurations $\gamma_1 \models \phi$ and $\gamma_2 \models \phi$. Any $\gamma' \subseteq \gamma_1$ contains only interactions from γ_1, thus, $\gamma' \models \phi$. For all $a \in \gamma_1 \cup \gamma_2$ holds $a \overset{i}{\models} \phi$, consequently $\gamma_1 \cup \gamma_2 \models \phi$. This shows that ϕ is downward-closed and $\cup$-closed.

Conversely, suppose that f is a $\cup$-closed and downward-closed formula and consider its characteristic configuration set $|f| = \{\gamma \in C(P) \setminus \{\emptyset\} \,|\, \gamma \models f\}$. Let $I = \bigcup_{\gamma \in |f|} \gamma$ be the set of all interactions belonging to configurations satisfying f. Since f is downward-closed, $\{a\} \models f$ for any $a \in I$. By the definition of $\cup$-closed formulas, the union of models is also a model. Thus, $\gamma \models f$, for any $\emptyset \neq \gamma \subseteq I$. Consequently, $|f| = \{\gamma \subseteq I \,|\, \gamma \neq \emptyset\}$ and $f = \bigvee_{a\in I} m_a$, where m_a denotes the characteristic monomial of the interaction a. $\square$

Thus, interaction formulas are represented by formulas that are both downward-closed and $\cup$-closed. Figure 5.2 shows the correspondence between the PIL lattice and the PCL lattice. Notice that, in general, $\phi \sqcup \phi'$ is not $\cup$-closed and $\phi + \phi'$ is not downward-closed. For example, for $P = \{p, q\}$, $f_1 = p\bar{q} \ \sqcup \bar{p}\, q$ is not $\cup$-closed, since $\{\{p\}\}$ and $\{\{q\}\}$ are models of f_1, but $\{\{p\}, \{q\}\}$ is not a model of f_1. Similarly, $f_2 = p\bar{q} \ + \bar{p}\, q$ is not downward-closed, since $\{\{p\}, \{q\}\}$ is a model of f_2, but neither $\{\{p\}\}$ nor $\{\{q\}\}$ is.

As shown before, conjunction does not distribute over coalescing. Nevertheless, it distributes for interaction formulas as shown in the following proposition.

Proposition 5.1.22. *For any formulas f_1, f_2 and interaction formula ϕ, we have:*

$$\phi \wedge (f_1 + f_2) \equiv (\phi \wedge f_1) + (\phi \wedge f_2)\,.$$

Proof. If γ is a configuration satisfying $\phi \wedge (f_1 + f_2)$, then $\gamma \models \phi$ and there exist γ_1, γ_2, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models f_1$ and $\gamma_2 \models f_2$. Since ϕ is an interaction formula, it is also downward-closed (Proposition 5.1.21). Thus, $\gamma \models \phi$ implies $\gamma_1 \models \phi$ and $\gamma_2 \models \phi$. Consequently, $\gamma_1 \models \phi \wedge f_1$ and $\gamma_2 \models \phi \wedge f_2$.

Conversely, if γ is a configuration satisfying $(\phi \wedge f_1) + (\phi \wedge f_2)$ then $\gamma = \gamma_1 \cup \gamma_2$, such that $\gamma_1 \models f_1, \gamma_1 \models \phi, \gamma_2 \models f_2$ and $\gamma_2 \models \phi$. Since ϕ is $\cup$-closed, $\gamma \models \phi$ and, consequently, $\gamma \models \phi \wedge (f_1 + f_2)$. □

Notice that coalescing is not idempotent in general, as it is shown in the following example.

Example 5.1.23. $(p \sqcup q) + (p \sqcup q) \not\equiv p \sqcup q$. The configuration $\{\{p\}, \{q\}\}$ satisfies $(p \sqcup q) + (p \sqcup q)$, but it does not satisfy $p \sqcup q$.

Nevertheless, coalescing is idempotent on $\cup$-closed formulas.

Proposition 5.1.24. *$f + f \equiv f$ for any $\cup$-closed formula f.*

Proof. The implication $\gamma \models f \Rightarrow \gamma \models f + f$ for any γ is trivial.

Conversely, consider a configuration $\gamma \models f + f$. By the semantics of coalescing, there exist γ_1, γ_2, such that $\gamma = \gamma_1 \cup \gamma_2$, $\gamma_1 \models f$ and $\gamma_2 \models f$. Since f is $\cup$-closed, $\gamma_1 \cup \gamma_2 \models f$. Consequently, $\gamma \models f$. □

Coalescing with *true* presents a particular interest for writing specifications, since they allow adding any set of interactions to the configurations satisfying f. Notice that *true* is not a neutral element of coalescing: only the implication $f \Rightarrow f + true$ holds in general.

Definition 5.1.25. For any formula f, the *closure operator* $\sim$ is defined by putting $\sim f \overset{def}{=} f + true$. We give $\sim$ the same binding power as $\neg$.

Although closure is not a primitive operator of PCL, it is easy to see that the semantics of closure can be directly defined by putting $\gamma \models \sim f$ iff exists $\gamma_1 \subseteq \gamma$ such that $\gamma_1 \models f$.

Example 5.1.26. For $P = \{p, q, r\}$ the formula f characterising all configurations such that p must interact with both q and r, is $f = pq + pr + true = \sim(pq + pr)$. Notice that the only constraint imposed by the formula f is that configurations that satisfy it must contain an interaction pqr or both interactions pq and pr. Configurations satisfying f can contain any additional interactions.

Proposition 5.1.27. *$\sim\sim f \equiv \sim f$ for any formula f.*

Proof. $\sim\sim f \equiv \sim f + true \equiv f + true + true \equiv f + true \equiv \sim f$. □

Notice that, as an immediate corollary of Proposition 5.1.17, the closure of any formula is upward-closed. The following proposition shows that $\sim f$ is the smallest upward-closed formula greater than f in the lattice of PCL formulas ordered by implication.

Proposition 5.1.28. *For any formula f, holds $f \Rightarrow \sim f$. Furthermore, for any upward-closed formula f', such that $f \Rightarrow f'$, holds $\sim f \Rightarrow f'$.*

Proof. $f \Rightarrow \sim f$ follows directly from the semantics of the $\sim$ operator. Assume that there exists an upward-closed f', such that $f \Rightarrow f'$, and a configuration γ, such that $\gamma \models \sim f$ and $\gamma \not\models f'$. Since $\gamma \models \sim f$, there exists $\gamma_1 \subseteq \gamma$, such that $\gamma_1 \models f$. Since $f \Rightarrow f'$, we have $\gamma_1 \models f'$. The formula f' is upward-closed, therefore $\gamma_1 \models f'$ implies $\gamma \models f'$, which contradicts our assumption. $\square$

The closure operator can be interpreted as a modal operator with existential quantification. The formula $\sim f$ characterises configurations γ, such that there *exists* a sub-configuration of γ satisfying f. Thus, $\sim f$ means "possible f". Dually $\neg \sim \neg f$ means "always f" in the following sense: if a configuration γ satisfies $\neg \sim \neg f$, *all* sub-configurations of γ satisfy f.

Corollary 5.1.29. *For any formula f, holds $\neg \sim \neg f \Rightarrow f$. Furthermore, for any downward-closed formula f', such that $f' \Rightarrow f$, holds $f' \Rightarrow \neg \sim \neg f$.*

Proof. By 5.1.28, for any formula f, we have $\neg f \Rightarrow \sim \neg f$, which immediately implies $\neg \sim \neg f \Rightarrow f$. For any downward-closed f', such that $f' \Rightarrow f$, we observe that, by Proposition 5.1.19, $\neg f'$ is upward-closed. Hence, by Proposition 5.1.28, $\sim \neg f \Rightarrow \neg f'$ and, consequently, $f' \Rightarrow \neg \sim \neg f$. $\square$

Clearly, if f is downward-closed then $\neg \sim \neg f \equiv f$. However, this is not true in general. Consider $f = m_a + m_b$, where m_a and m_b are characteristic monomials of interactions a and b, respectively. The only configuration satisfying f is $\gamma = \{a, b\}$. In particular, none of the sub-configurations $\{a\}, \{b\} \subset \gamma$ satisfies f. Thus, $\neg \sim \neg (m_a + m_b) \equiv$ *false*.

Proposition 5.1.30. *For any formulas f_1, f_2, the following distributivity results hold:*

1. $\sim(f_1 \sqcup f_2) \equiv \sim f_1 \sqcup \sim f_2 \equiv \sim(f_1 \vee f_2)$,
2. $\sim f_1 + \sim f_2 \equiv \sim(f_1 + f_2) \equiv \sim f_1 \wedge \sim f_2$.

Proof. We have the following equalities:

$$\begin{aligned}
\sim(f_1 \sqcup f_2) &\equiv (f_1 \sqcup f_2) + \mathit{true} \equiv f_1 + \mathit{true} \sqcup f_2 + \mathit{true} \equiv \sim f_1 \sqcup \sim f_2\,, \\
\sim(f_1 \vee f_2) &\equiv f_1 + \mathit{true} \sqcup f_2 + \mathit{true} \sqcup f_1 + f_2 + \mathit{true} \\
&\equiv f_1 + \mathit{true} \sqcup f_2 + \mathit{true} \equiv \sim f_1 \sqcup \sim f_2\,, \\
\sim f_1 + \sim f_2 &\equiv f_1 + \mathit{true} + f_2 + \mathit{true} \equiv f_1 + f_2 + \mathit{true} \equiv \sim(f_1 + f_2)\,, \\
\sim f_1 \wedge \sim f_2 &\equiv (f_1 + \mathit{true}) \wedge (f_2 + \mathit{true}) \equiv f_1 + f_2 + \mathit{true} \equiv \sim(f_1 + f_2)\,.
\end{aligned}$$

$\square$

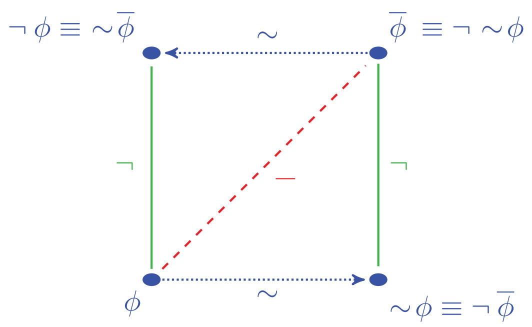

$\neg\,\phi \equiv \sim\overline{\phi}$
$\sim$
$\overline{\phi} \equiv \neg\sim\phi$
$\neg$
$-$
$\neg$
ϕ
$\sim$
$\sim\phi \equiv \neg\,\overline{\phi}$

Figure 5.3 – Correspondence between negation and complementation of interaction formulas.

The following results allow us to address the relation between complementation and negation.

Lemma 5.1.31. *For any interaction formula ϕ, the following holds:*

$$\phi \sqcup \overline{\phi} \sqcup (\overline{\phi} + \phi) \equiv \mathit{true}\,. \tag{5.10}$$

Proof. The proof is immediate from Corollary 5.1.8 and the fact that $\phi \vee \overline{\phi} \equiv \mathit{true}$, for any interaction formula ϕ. □

Notice that the three terms in the left-hand side of (5.10) are mutually disjoint.

Proposition 5.1.32. *For any interaction formula ϕ, holds $\neg\,\phi \equiv \sim\overline{\phi}$.*

Proof. By Lemma 5.1.31, we have $\neg\,\phi \equiv \overline{\phi} \sqcup (\phi + \overline{\phi}) \equiv \overline{\phi} + \mathit{true} \equiv \sim\overline{\phi}$. □

In particular, this means that complementation can also be interpreted as a modality. Proposition 5.1.32 shows that the complementation of ϕ represents all configurations that contain $\overline{\phi}$. Equivalences $\neg\,\overline{\phi} \equiv \sim\phi$, $\neg\sim\phi \equiv \overline{\phi}$, $\neg\sim\overline{\phi} \equiv \phi$ and $\sim\neg\,\phi \equiv \neg\,\phi$, for interaction formulas ϕ, are direct corollaries of Proposition 5.1.32 and, for the latter, Proposition 5.1.27. Figure 5.3 depicts the relations between complementation and negation of the interaction formulas.

The following lemma expresses coalescing through extended disjunction. Coalescing is more restrictive than extended disjunction requiring the existence of sub-configurations that satisfy all operands.

Lemma 5.1.33. *For any formulas f, g, we have:*

$$f + g \equiv \sim f \wedge \sim g \wedge (f \vee g)\,.$$

Proof. By (5.9) and Proposition 5.1.30, we have

$$\sim f \wedge \sim g \wedge (f \vee g) \equiv \sim(f + g) \wedge (f \sqcup g \sqcup f + g)\,.$$

Notice that $\gamma \models \sim(f+g) \wedge f$ iff $\gamma \models f$ and there exists $\gamma_1 \subseteq \gamma$, such that $\gamma_1 \models g$. Thus, $\sim(f+g) \wedge f \equiv f + (f \wedge g)$. By applying a similar transformation to g, we obtain

$$\sim(f+g) \wedge (f \sqcup g \sqcup f+g) \equiv \big(f + (f \wedge g)\big) \sqcup \big(g + (f \wedge g)\big) \sqcup (f+g) \equiv f+g\,,$$

where the last equality is an immediate consequence of the fact that $f \wedge g \Rightarrow f$ and $f \wedge g \Rightarrow g$. □

Proposition 5.1.34. *For any interaction formulas ϕ, ψ, the following two formulas are equivalent:*

$$\neg\,(\phi+\psi) \equiv \overline{\phi} \sqcup \overline{\psi} \sqcup \sim(\overline{\phi} \wedge \overline{\psi})\,.$$

Proof. By Proposition 5.1.33, $\phi+\psi \equiv \sim\phi \wedge \sim\psi \wedge (\phi \vee \psi)$. Thus, $\neg\,(\phi+\psi) \equiv \neg\,(\sim\phi \wedge \sim\psi \wedge (\phi \vee \psi)) \equiv \neg \sim\phi \sqcup \neg \sim\psi \sqcup \neg\,(\phi \vee \psi)$. Since ϕ, ψ and $\phi \vee \psi$ are interaction formulas, the application of Proposition 5.1.32 gives $\neg\,(\phi+\psi) \equiv \overline{\phi} \sqcup \overline{\psi} \sqcup \sim(\overline{\phi} \wedge \overline{\psi})$ □

Proposition 5.1.34 allows the elimination of complementation as shown in the following example.

Example 5.1.35. Consider a formula $f = \neg\,(pq+pr)$ and a configuration $\gamma \models f$. The PCL semantics requires that γ cannot be split into two non-empty parts $\gamma_1 \models pq$ and $\gamma_2 \models pr$. This can happen in two cases: 1) there exists $a \in \gamma$ such that a does not satisfy neither pq nor pr; 2) one of the monomials is not satisfied by any interaction in γ. The former case can be expressed as $\sim(\overline{pq}\ \overline{pr})$ and the latter as $\overline{pq} \sqcup \overline{pr}$. The union of these formulas gives the equivalence $\neg\,(pq+pr) \equiv \overline{pq} \sqcup \overline{pr} \sqcup \sim(\overline{pq}\ \overline{pr})$.

Lemma 5.1.33 and Proposition 5.1.34 can be generalised as follows:

Lemma 5.1.36. *For any set of formulas F, we have:*

$$\sum_{f \in F} f \equiv \bigwedge_{f \in F} \sim f \wedge \bigvee_{f \in F} f\,.$$

Proposition 5.1.37. *For any set of interaction formulas Φ, the following two formulas are equivalent:*

$$\neg \sum_{\phi \in \Phi} \phi \equiv \bigsqcup_{\phi \in \Phi} \overline{\phi} \sqcup \sim \bigwedge_{\phi \in \Phi} \overline{\phi}\,.$$

Proofs of Propositions 5.1.36 and 5.1.37 are similar to the proofs of Propositions 5.1.33 and 5.1.34, respectively.

Conjunction of coalescings of interaction formulas can be eliminated by using the following distributivity result pushing it down within the formula tree.

Proposition 5.1.38. *If Φ and Ψ are sets of interaction formulas, then*

$$\sum_{\phi\in\Phi}\phi \wedge \sum_{\psi\in\Psi}\psi \;\equiv\; \sum_{\xi\in\Phi\cup\Psi}\Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi)\Big).$$

Proof. Notice that

$$\sum_{\phi\in\Phi}\phi \wedge \sum_{\psi\in\Psi}\psi \;\equiv\; \neg\neg\Big(\sum_{\phi\in\Phi}\phi \wedge \sum_{\psi\in\Psi}\psi\Big) \equiv \neg\Big(\neg\sum_{\phi\in\Phi}\phi \sqcup \neg\sum_{\psi\in\Psi}\psi\Big).$$

By Proposition 5.1.37, this can be further transformed into

$$\neg\left(\bigsqcup_{\phi\in\Phi}\overline{\phi} \sqcup \sim\bigwedge_{\phi\in\Phi}\overline{\phi} \;\sqcup\; \bigsqcup_{\psi\in\Psi}\overline{\psi} \sqcup \sim\bigwedge_{\psi\in\Psi}\overline{\psi}\right) \;\equiv\; \neg\left(\bigsqcup_{\xi\in\Phi\cup\Psi}\overline{\xi} \sqcup \sim\bigwedge_{\phi\in\Phi}\overline{\phi} \sqcup \sim\bigwedge_{\psi\in\Psi}\overline{\psi}\right),$$

which we further transform by applying twice the De Morgan's law (once for complementation and union and once for negation and disjunction) and Proposition 5.1.32:

$$\bigwedge_{\xi\in\Phi\cup\Psi}\neg\overline{\xi} \wedge \neg\Big(\sim\bigwedge_{\phi\in\Phi}\overline{\phi}\Big) \wedge \neg\Big(\sim\bigwedge_{\psi\in\Psi}\overline{\psi}\Big) \;\equiv\; \bigwedge_{\xi\in\Phi\cup\Psi}{\sim}\xi \wedge \overline{\bigwedge_{\phi\in\Phi}\overline{\phi}} \wedge \overline{\bigwedge_{\psi\in\Psi}\overline{\psi}}\;.$$

By Proposition 5.1.30 and another application of De Morgan's law, we obtain

$${\sim}\sum_{\xi\in\Phi\cup\Psi}\xi \wedge \bigvee_{\phi\in\Phi}\phi \wedge \bigvee_{\psi\in\Psi}\psi \;\equiv {\sim}\sum_{\xi\in\Phi\cup\Psi}\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi)\,.$$

Let γ be a configuration satisfying the formula in the right-hand side of this equation. By (5.7), any interaction $a \in \gamma$ satisfies the second conjunct in this formula. Hence, there exists a pair $(\phi, \psi) \in \Phi \times \Psi$, such that $a \overset{i}{\models} \phi \wedge \psi$ and, a fortiori, there exists $\xi \in \Phi \cup \Psi$, such that $a \overset{i}{\models} \xi$. Thus, the closure operator in the first conjunct of this formula can be discarded. Finally, by Proposition 5.1.22, we have

$$\Big(\sum_{\xi\in\Phi\cup\Psi}\xi\Big) \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi) \;\equiv\; \sum_{\xi\in\Phi\cup\Psi}\Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi)\Big).$$

$\square$

Example 5.1.39. Consider a formula $f = (\phi_1 + \phi_2) \wedge (\phi_3 + \phi_4)$, where ϕ_1, ϕ_2, ϕ_3 and ϕ_4 are interaction formulas, and a configuration $\gamma \models f$. The semantics requires that there exist two partitions of γ: $\gamma = \gamma_1 \cup \gamma_2$ and $\gamma = \gamma_3 \cup \gamma_4$, such that $\gamma_i \models \phi_i$ for $i \in [1, 4]$. Considering an intersection $\gamma_{i,j} = \gamma_i \cap \gamma_j$, we have $\gamma_{i,j} \models \phi_i \wedge \phi_j$. Thus, $\gamma = \bigcup \gamma_{i,j}$ satisfies $\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4$ even if some $\gamma_{i,j}$ are empty. Nevertheless, disjunction allows configurations such that no interaction satisfy one of the disjunction terms and consequently some ϕ_i. A coalescing of ϕ_i allows only configurations such that each ϕ_i is satisfied by at least

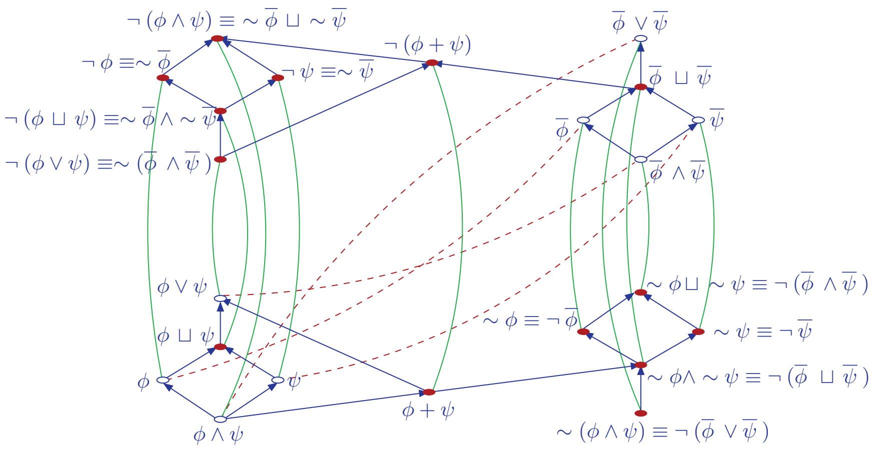

$\neg(\phi \wedge \psi) \equiv \sim\overline{\phi} \sqcup \sim\overline{\psi}$
$\neg\phi \equiv \sim\overline{\phi}$
$\neg\psi \equiv \sim\overline{\psi}$
$\neg(\phi + \psi)$
$\neg(\phi \sqcup \psi) \equiv \sim\overline{\phi} \wedge \sim\overline{\psi}$
$\neg(\phi \vee \psi) \equiv \sim(\overline{\phi} \wedge \overline{\psi})$
$\overline{\phi} \vee \overline{\psi}$
$\overline{\phi} \sqcup \overline{\psi}$
$\overline{\phi}$
$\overline{\psi}$
$\overline{\phi} \wedge \overline{\psi}$
$\phi \vee \psi$
$\phi \sqcup \psi$
ϕ
ψ
$\phi \wedge \psi$
$\phi + \psi$
$\sim\phi \sqcup \sim\psi \equiv \neg(\overline{\phi} \wedge \overline{\psi})$
$\sim\phi \equiv \neg\overline{\phi}$
$\sim\psi \equiv \neg\overline{\psi}$
$\sim\phi \wedge \sim\psi \equiv \neg(\overline{\phi} \sqcup \overline{\psi})$
$\sim(\phi \wedge \psi) \equiv \neg(\overline{\phi} \vee \overline{\psi})$

Figure 5.4 – PCL lattice.

one interaction. Thus, the conjunction of these formulas gives the equivalent representation:

$$
\begin{aligned}
f &\equiv (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) \wedge (\phi_1 + \phi_2 + \phi_3 + \phi_4) \\
&= \phi_1 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) + \phi_2 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) \\
&+ \phi_3 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) + \phi_4 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4)\,.
\end{aligned}
$$

The PCL lattice is illustrated in Figure 5.4. Notice that the PCL lattice has two sub-lattices generated by monomials:

- through disjunction and negation (isomorphic to the PIL lattice);
- through union and complementation (disjunction is not expressible).

Notice that coalescing cannot be expressed in any of these two sub-lattices. Although some formulas involving the closure operator can be expressed in the second sub-lattice, e.g. $\sim\overline{\phi} \equiv \neg\phi$, in general this is not the case, e.g. the formulas $\sim(\overline{\phi} \wedge \overline{\psi})$ and $\sim\phi \sqcup \sim\psi$ are not part of either sub-lattice. However, the closure operator is expressible by taking as generators the interaction formulas:

Proposition 5.1.40. *The lattice generated by interaction formulas through union and complementation is closed under the closure operator* $\sim$.

Proof. We must prove that, for any formula f in this lattice, the formula $\sim f$ is also in the lattice.

Since union and complementation satisfy the usual axioms of propositional logic, f can be represented in the equivalent of the disjunction normal form:

$$f \equiv \bigsqcup_{i \in I} \Big(\bigwedge_{k \in K_i} \phi_k \wedge \bigwedge_{j \in J_i} \neg\phi_j \Big)\,,$$

where all ϕ_j and ϕ_k are interaction formulas. Furthermore, since the conjunction of interaction formulas $\bigwedge_{k\in K_i}\phi_k$ is also an interaction formula, we can assume, without loss of generality, that all K_i are singleton sets and

$$f \equiv \bigsqcup_{i\in I}\Big(\phi_i \wedge \bigwedge_{j\in J_i} \neg\,\phi_j\Big)\,.$$

Applying the closure operator, we then have

$$\begin{aligned}
\sim f &\equiv \sim\bigsqcup_{i\in I}\Big(\phi_i \wedge \bigwedge_{j\in J_i} \neg\,\phi_j\Big) \\
&\equiv \bigsqcup_{i\in I}\sim\Big(\phi_i \wedge \bigwedge_{j\in J_i} \neg\,\phi_j\Big) && \text{// by Proposition 5.1.30} \\
&\equiv \bigsqcup_{i\in I}\sim\Big(\phi_i \wedge \sim\Big(\sum_{j\in J_i} \overline{\phi_j}\Big)\Big) && \text{// by Propositions 5.1.32 and 5.1.30} \\
&\equiv \bigsqcup_{i\in I}\sim\Big(\phi_i + \sum_{j\in J_i} (\phi_i \wedge \overline{\phi_j}\,)\Big) && \text{// by Proposition 5.1.22} \\
&\equiv \bigsqcup_{i\in I}\Big(\sim\phi_i \wedge \bigwedge_{j\in J_i} \sim(\phi_i \wedge \overline{\phi_j}\,)\Big) && \text{// by Proposition 5.1.30} \\
&\equiv \bigsqcup_{i\in I}\Big(\neg\,\overline{\phi_i} \wedge \bigwedge_{j\in J_i} \neg\,\overline{\phi_i \wedge \overline{\phi_j}}\,\Big) && \text{// by Proposition 5.1.32} \\
&\equiv \bigsqcup_{i\in I}\neg\Big(\overline{\phi_i} \sqcup \bigsqcup_{j\in J_i} \overline{\phi_i \wedge \overline{\phi_j}}\,\Big)\,.
\end{aligned}$$

Since, for all i and j, both $\overline{\phi_i}$ and $\overline{\phi_i \wedge \overline{\phi_j}}$ are interaction formulas, we conclude that $\sim f$ belongs to the lattice generated by interaction formulas through union and complementation.

□

5.1.2 Normal form and axiomatisation of PCL formulas

To simplify the presentation, we assume in this subsection that formulas do not contain closure operators, i.e. all $\sim f$ are replaced by $f + true$, and disjunction can appear only within interaction formulas, i.e. we do not consider the extension (5.9) of the disjunction operator to general PCL formulas. We prove that any PCL formula can be expressed in the following normal form: $\bigsqcup_{i\in I}\sum_{j\in J_i}\bigvee_{k\in K_{i,j}} m_{i,j,k}$, where all $m_{i,j,k}$ are monomials. This normal form can be obtained by using the rewriting system given in Figure 5.5 and usual Boolean transformations for interaction formulas. Normal form of a formula is computed by applying the procedure in Figure 5.6 to the root of its Abstract Syntax Tree (AST). Notice that no two rules in Figure 5.5 can be simultaneously applicable to the same node.

An application of a rule to a node of an AST modifies only the subtree rooted at this node. In order to simplify the reasoning, we impose the following additional constraint on the application order of the rules from the rewriting system in Figure 5.5.

$$1.\ \frac{g \wedge \bigsqcup_{i\in I} f_i}{\bigsqcup_{i\in I} g \wedge f_i} \qquad 2.\ \frac{g + \bigsqcup_{i\in I} f_i}{\bigsqcup_{i\in I} f_i + g} \qquad 3.\ \frac{\neg \bigsqcup_{i\in I} f_i}{\bigwedge_{i\in I} \neg f_i}$$

(Proposition 5.1.5) (Proposition 5.1.7) (Proposition 5.1.5)

$$4.\ \frac{\neg \sum_{\phi\in\Phi} \phi\,, \quad \text{all } \phi \text{ are interaction formulas}}{\bigsqcup_{\phi\in\Phi} \overline{\phi} \ \sqcup \Big(\bigwedge_{\phi\in\Phi} \overline{\phi} + \mathit{true}\Big)} \qquad \text{(Proposition 5.1.37)}$$

$$5.\ \frac{\sum_{\phi\in\Phi} \phi \wedge \sum_{\psi\in\Psi} \psi\,, \quad \text{all } \phi\in\Phi \text{ and } \psi\in\Psi \text{ are interaction formulas}}{\sum_{\xi\in\Phi\cup\Psi} \Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi} (\phi\wedge\psi)\Big)} \qquad \text{(Proposition 5.1.38)}$$

Figure 5.5 – Rewriting system for computing the normal form by the procedure in Figure 5.6.

Constraint 5.1.41. We require that any rule be applied to a node n only if no rule can be applied to any other node in the subtree of n.

Remark 5.1.42. We extend Constraint 5.1.41 to include usual Boolean transformations. Hence, at every step of the process, all interaction sub-formulas are maintained in Disjunctive Normal Form.

Example 5.1.43. The following example illustrates the normalisation process:

$$\begin{aligned}
(pq \sqcup r) \wedge (pr + \neg q) &\equiv (pq \sqcup r) \\
&\quad \wedge (pr + (\overline{q} \sqcup \overline{q} + \mathit{true})) && // \text{ rule 4 with } \Phi = \{q\} \\
&\equiv (pq \sqcup r) \wedge (pr + \overline{q} + \mathit{true}) && // \text{ absorption laws}^1 \\
&\equiv (pq \wedge (pr + \overline{q} + \mathit{true})) \\
&\quad \sqcup (r \wedge (pr + \overline{q} + \mathit{true})) && // \text{ rule 1} \\
&\equiv ((pq \wedge pr) + (pq \wedge \overline{q}) + (pq \wedge \mathit{true})) \\
&\quad \sqcup ((r \wedge pr) + (r \wedge \overline{q}) + (r \wedge \mathit{true})) && // \text{ rule 5} \\
&\equiv (pqr + \mathit{false} + pq) \sqcup (pr + r\overline{q} + r) && // \text{ Boolean laws} \\
&\equiv pr + r\overline{q} + r\,. && // \text{ absorption and identity laws}
\end{aligned}$$

The first step removes complementation. Then, the application of distributivity rules pushes conjunction down in the AST of the formula, to the level of monomials. Finally, the formula is simplified by observing that $false$ is the absorbing element of coalescing and the identity of union.

[1]Absorption laws are not essential for the normalisation process. We use them here to simplify the example.

```
procedure normalise (node)
  if (node is an interaction formula)
    transform node into DNF;
    return;
  endif

  foreach child of node do
    normalise(child);
  od

  if (no rule applicable to node)
    return;
  else
    apply rule to node;
    normalise(node);
  endif
end
```

Figure 5.6 – Procedure for computing the normal form using the rewriting system of Figure 5.5.

Theorem 5.1.44. *Under Constraint 5.1.41, the rewriting system in Figure 5.5 has the following properties:*

1. *The rewriting system is terminating and confluent.*
2. *For any formula f' derived from a formula f by the application of rewriting rules, we have $f' \equiv f$.*
3. *Any irreducible formula is in the normal form $\bigsqcup_{i\in I}\sum_{j\in J_i}\bigvee_{k\in K_{i,j}} m_{i,j,k}$.*

Proof.
1. In order to prove that the rewriting system is terminating, we define a ranking function on the AST of a formula, with leaves representing interaction sub-formulas. First, we introduce the following notations:

- Denote $p(n)$ the number of nodes in the subtree with the root n.
- Denote $d(n)$ the depth of the node n in the AST of the formula.
- Let $N = \sum_n p(n)^{p(n)}$, where the sum is taken over all $\neg$-nodes.
- Let $C = \sum_n p(n)^{p(n)}$, where the sum is taken over all $\wedge$-nodes.
- Let $U = \sum_n d(n)$, where the sum is taken over all $\sqcup$-nodes.

- Denote A the number of $+$-nodes in the AST of the formula.

The ranking function associates a tuple to a tree $\langle N, C, U, A\rangle$. We use lexicographical order to compare the values of the function, i.e. $\langle a_1, a_2, a_3, a_4\rangle < \langle b_1, b_2, b_3, b_4\rangle$ iff there exists $i \leq 4$ such that $a_j = b_j$, for all $j < i$, and $a_i < b_i$. We show that the application of each rewriting rule strictly reduces the value of the ranking function.

- Rule 1 does not change N and reduces C. Let n be the $\wedge$-node, to which we apply Rule 1. For each $\wedge$-node n', generated by the application of the rule, we have $p(n') < p(n)$. The number of generated $\wedge$-nodes n' is less than $p(n)$. Hence, $p(n)^{p(n)} > p(n) * p(n')^{p(n')}$, which implies that the value of C decreases after the application of the rule. Although, application of Rule 1 increases the value of U, the ranking function decreases by the definition of the lexicographical order.
- The application of Rule 2 increases A, but decreases U as it transforms a non-empty set of $\sqcup$-nodes into one with smaller depth. This rule does not change the values of N or C.
- The application of Rule 3 decreases N. A $\neg$-node with value $p(n)^{p(n)}$ is transformed into less than $p(n)$ nodes of value less than $p(n')^{p(n')}$ with $p(n') < p(n)$.
- The application of Rule 4 decreases N. It transforms a $\neg$-node into a union of conjunctions and coalescing.
- The application of Rule 5 decreases C and does not change N. It transforms a $\wedge$-node into a coalescing of interaction formulas.
- The application of usual Boolean transformations makes modifications only inside leaves, thus this rule does not affect the function value.

Since all rewriting rules decrease the rank of the tree and each value is a tuple of finite natural numbers, any sequence of applications of rewriting rules is finite.

Notice that applications of rules in different subtrees do not interfere and the order of rule applications between subtrees does not affect the resulting formula. This observation, together with Constraint 5.1.41, guarantees the confluence of the rewriting system.

2. Since all rewriting rules in Figure 5.5 preserve equivalence and $\equiv$ is a congruence (Proposition 5.1.1), the formula obtained by application of these rules is necessarily equivalent to the initial one.

3. Let f be an irreducible formula and let T be an AST of f. Any non-leaf node of T can be represented by the expression $x \to (n_1, \ldots, n_k)$, where $x \in \{\sqcup, +, \neg, \wedge\}$ is an operator and $(n_1, \ldots, n_k)$ is the list of child nodes. We call such a node $x \to (n_1, \ldots, n_k)$ an *x-node*. Notice

that, since all operators of the Configuration Logic are associative, an x-node can always be merged with its immediate child x-nodes: let $n_1 = x \to (m_1, \dots, m_l)$, then $x \to (n_1, \dots, n_k)$ can be substituted by $x \to (m_1, \dots, m_l, n_2, \dots, n_k)$ without changing the meaning of the formula (similarly for all n_i). Henceforth, we assume that no child node of an x-node is an x-node.

Let n be a $\neg$-node in T, such that none of the nodes in the sub-tree rooted in n is a $\neg$-node. Let n' be a child node of n. Since Rules 3 and 4 cannot be applied to n, the node n' is neither a $\sqcup$-node, nor a node representing an interaction formula. Hence, n' corresponds to a conjunction or a coalescing of PCL formulas, among which at least one is not an interaction formula. Notice that in the subtree rooted at n' there are neither $\neg$-nodes by the assumption that n is the deepest one nor $\sqcup$-nodes since Rules 1, 2 and 3 cannot be applied. Let n'' be the deepest $+$-node in the subtree rooted at n'. Children of n'' are interaction formulas as subtrees rooted at n'' cannot contain $\neg$-, $\sqcup$- or $+$-nodes. The parent node of n'' cannot be a negation or a union since they cannot appear in the subtree rooted at n', it is not a coalescing due to the form of AST and it is not a conjunction since Rule 5 is not applicable. This contradicts to the assumption that there are $\neg$-nodes in the AST.

Since none of the Rules 1, 2 and 3 are applicable, a $\sqcup$-node can only be the root of the AST of f. Hence, since Rule 5 is not applicable and there are no $\neg$-nodes in the AST of f, a $+$-node can only be the root or a child of the $\sqcup$-node. Furthermore, for the same reason, the children of a $+$-node can only be interaction formulas.

Since all interaction sub-formulas are in their DNF forms (see Remark 5.1.42), we conclude that f is in normal form. $\square$

A *full monomial* is a monomial, which involves all ports, i.e. $m = \bigwedge_{p \in P_+} p \wedge \bigwedge_{p \in P_-} \overline{p}$ such that $P = P_+ \cup P_-$ and $P_+ \cap P_- = \emptyset$. We define a *full normal form* as $\bigsqcup_{i \in I} \sum_{j \in J} m_{i,j}$, where $m_{i,j}$ are full monomials, $m_{i,j} \not\equiv m_{i,j'}$, for $j \neq j'$, and $\sum_{j \in J_i} m_{i,j} \not\equiv \sum_{j \in J_{i'}} m_{i',j}$, for $i \neq i'$. We show that any formula has an equivalent full normal form.

Lemma 5.1.45.

1. *A formula $f = \sum_{i \in I} m_i$, where m_i are full monomials, is satisfied by exactly one configuration $\gamma = \{a_i\}_{i \in I}$, where a_i is an interaction, such that m_i is its characteristic monomial.*

2. *For a configuration $\gamma = \{a_i\}_{i \in I}$, there exists a unique*[2] *formula $f = \sum_{i \in I} m_i$ such that $\gamma \models f$.*

Proof.

[2] In this lemma and the following theorem, uniqueness is up to the order of ports in monomials and the order of operands in coalescing and union.

1. Since m_i is a full monomial, there exists exactly one valuation of ports such that the monomial evaluates to true, i.e. there exists exactly one interaction a_i, such that $a_i \overset{i}{\models} m_i$.

   $\gamma \models \sum_{i\in I} m_i$ iff there exists $\{\gamma_i\}_{i\in I}$, such that $\gamma = \bigcup_{i\in I} \gamma_i$ and, for all $i \in I$, $\gamma_i \models m_i$. For each m_i, there exists only one interaction and consequently only one configuration $\gamma_i \models m_i$. Thus, there exists exactly one γ, such that $\gamma \models f$.

2. A characteristic monomial m_i of an interaction $a_i \in \gamma$ is a full monomial. Thus, $\gamma = \{a_i\}_{i\in I} \models \sum_{i\in I} m_i$. Let $f' = \sum_{j\in J} m'_j$ be another formula such that $\gamma \models f'$. By the first part of this lemma, f' is satisfied by a single configuration. Consequently, each interaction $a_j \in \gamma$ corresponds to the full monomial m_j. Therefore, f and f' consist of the same full monomials.

□

Theorem 5.1.46. *Any formula f has a unique full normal form.*

Proof. By Theorem 5.1.44 any formula f can be rewritten as a formula $f' \equiv f$ in normal form. In f', any non-full monomial can be transformed into a disjunction of full monomials, which, by Corollary 5.1.8, can be further transformed into a union of coalesced full monomials. The application of Proposition 5.1.7 and absorption of equivalent terms lead to the full normal form. Uniqueness is a corollary of Lemma 5.1.45. □

Example 5.1.47. Let $P = \{p, q, r\}$ and consider the formula in normal form $pr + r\overline{q}$. It can be transformed into full normal form as follows:

$$\begin{aligned} pr + r\overline{q} &\equiv (pqr \sqcup p\overline{q}\,r \sqcup pqr + p\overline{q}\,r) + (p\overline{q}\,r \sqcup \overline{p}\;\overline{q}\,r \sqcup p\overline{q}\,r + \overline{p}\;\overline{q}\,r) \\ &\equiv (pqr + p\overline{q}\,r) \sqcup (pqr + \overline{p}\;\overline{q}\,r) \sqcup (pqr + p\overline{q}\,r + \overline{p}\;\overline{q}\,r) \sqcup p\overline{q}\,r \sqcup (p\overline{q}\,r + \overline{p}\;\overline{q}\,r). \end{aligned}$$

Axioms. PCL operators satisfy the following axioms:

1. The PIL axioms for interaction formulas.
2. The usual axioms of propositional logic for $\sqcup$, $\wedge$, $\neg$.
3. $+$ is associative, commutative and has an absorbing element *false*.
4. For any formulas f, f_1 and f_2, holds $f + (f_1 \sqcup f_2) \equiv f + f_1 \sqcup f + f_2$.
5. For any formulas f_1 and f_2, holds $f_1 \vee f_2 \equiv f_1 \sqcup f_2 \sqcup f_1 + f_2$.
6. For any sets of interaction formulas Φ and Ψ, holds

$$\sum_{\phi\in\Phi} \phi \wedge \sum_{\psi\in\Psi} \psi \equiv \sum_{\xi\in\Phi\cup\Psi} \Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi} (\phi\wedge\psi)\Big).$$

```
Input:   A sub-formula $f = \sum_{j \in J} \bigvee_{k \in K_j} m_{j,k}$, and a configuration $\gamma = \{a_1, \ldots, a_n\}$.
Output:  $true$ if $\gamma \models f$, $false$ otherwise.
1.  $J' := \emptyset$; $l := 1$; $b := true$;
2.  while ($l \leq n$ and $b$) do
3.      $X := \{j \in J \,|\, a_l \overset{i}{\models} \bigvee_{k \in K_j} m_{j,k}\}$;
4.      if ($X \neq \emptyset$)
5.        $J' := J' \cup X$;
6.      else
7.        $b := false$;
8.      endif
9.      $l := l + 1$;
10. od
11. return $J' = J$;
```

Figure 5.7 – Algorithm for checking satisfaction of formulas.

7. For any set of interaction formulas Φ, holds

$$\neg \Big(\sum_{\phi \in \Phi} \phi \Big) \equiv \bigsqcup_{\phi \in \Phi} \overline{\phi} \sqcup \sim \Big(\bigwedge_{\phi \in \Phi} \overline{\phi} \Big) .$$

Theorem 5.1.48. *The above axiomatization is sound and complete for the equivalence in PCL.*

Proof. Soundness of all the above axioms has been proved in Subsection 5.1.1. Completeness is an immediate consequence of the existence of a unique full normal form. □

5.1.3 Checking satisfaction of formulas

We provide a method for checking that a configuration of the form $\gamma = \{a_1, \ldots, a_n\}$ satisfies a formula f. Without loss of generality, we assume that the formula is in normal form $f = \bigsqcup_{i \in I} \sum_{j \in J_i} \bigvee_{k \in K_{i,j}} m_{i,j,k}$. We have to check that γ satisfies at least one of the terms $\sum_{j \in J_i} \bigvee_{k \in K_{i,j}} m_{i,j,k}$, for some $i \in I$. The algorithm in Figure 5.7 performs this verification for one term (index i is omitted).

We have to check the validity of the following two statements: 1) each interaction satisfies at least one of the coalesced interaction formula and 2) each interaction formula is satisfied by at least one interaction. The algorithm iterates through the interactions, checking the first part and memorising the satisfied interaction formulas. After the iteration stops, it checks whether all interaction formulas were satisfied by at least one interaction. Configuration γ satisfies the formula f iff the disjunction of the results of the algorithm in Figure 5.7, for all terms of the union evaluates to true.

An alternative method for checking satisfaction of a formula f by a configuration γ is based on the existence of a normal form and the completeness theorem.

Consider a formula f and a configuration $\gamma = \{a_1, ..., a_n\}$. In order to decide whether $\gamma \models f$, we associate with γ a characteristic formula $\varphi_\gamma = m_1 + \cdots + m_n$, where $m_i = \bigwedge_{p \in a_i} p \wedge \bigwedge_{p \notin a_i} \overline{p}$ are characteristic monomials of the respective interactions a_i. Notice that φ_γ has exactly one model γ (Lemma 5.1.45). If formulas φ_γ and f have a common model then γ is a model of f. Thus, $\gamma \models f$ iff $\varphi_\gamma \wedge f \not\equiv false$. This latter non-equivalence can be decided by verifying whether at least one term of the normal form of $\varphi_\gamma \wedge f$ is not equivalent to $false$. Recall that the terms of a formula in normal form are coalescings of interaction formulas. Therefore, for a term to be not equivalent to $false$, it is necessary that all of its participating interaction formulas be not equivalent to $false$. Finally, as in Boolean logics, a disjunction of monomials is not equivalent to $false$ iff at least one monomial does not contain one of the variables twice in opposite (positive and negative) forms.

Example 5.1.49. Let $P = \{p, q, r\}$ and consider $f = p\,q + r\,\overline{p}$ and $\gamma = \{\{p, q, r\}, \{q, r\}\}$. In order to decide whether $\gamma \models f$, we first apply the algorithm in Figure 5.7. This algorithm iterates through the interactions of γ and monomials of f: $\{p, q, r\}$ satisfies $p\,q$, whereas $\{q, r\}$ satisfies $r\,\overline{p}$. For both interactions the sets of monomials are not empty and all monomials were visited. Hence, $\gamma \models f$.

Alternatively, we consider the characteristic formula $\varphi_\gamma = p\,q\,r + q\,r\,\overline{p}$ and check whether $\varphi_\gamma \wedge f = (p\,q\,r + q\,r\,\overline{p}\,) \wedge (p\,q + r\,\overline{p}\,) \not\equiv false$. We have

$$
\begin{aligned}
&(pqr + qr\overline{p}\,) \wedge (pq + r\overline{p}\,) \\
&\qquad \equiv (pqr \wedge (pqr \vee false \vee false \vee qr\overline{p}\,)) + (qr\overline{p} \wedge (pqr \vee false \vee false \vee qr\overline{p}\,)) \\
&\qquad\quad + (pq \wedge (pqr \vee false \vee false \vee qr\overline{p}\,)) + (r\overline{p} \wedge (pqr \vee false \vee false \vee qr\overline{p}\,)) \\
&\qquad \equiv pqr + qr\overline{p} \not\equiv false\,.
\end{aligned}
$$

5.2 Higher order extensions of PCL

PCL is defined for a given set of components and a given set of ports. On the contrary, architecture styles are defined for arbitrary number of components. In order to specify architecture styles, we introduce types of components and quantification over component variables. We make the following assumptions:

- A finite set of component types $\mathcal{T} = \{T_1, \ldots, T_n\}$ is given. Instances of a component type have the same interface and behaviour. We write $c : T$ to denote a component c of type T.

- The interface of each component type has a distinct set of ports. We write $c.p$ to denote the port p of component c and $c.P$ to denote the set of ports of component c.

5.2.1 First-order configuration logic

Syntax. The language of the formulas of the first-order configuration logic extends the PCL language by allowing an existential quantifier $\exists c{:}T$ and a specific coalescing quantifier $\Sigma c{:}T$ on component variables.

$$F ::= \mathit{true} \mid \phi \mid \exists c{:}T\big(\Phi(c)\big).F \mid \Sigma c{:}T\big(\Phi(c)\big).F \mid F \sqcup F \mid \neg\, F \mid F + F\,,$$

where ϕ is an interaction formula, c is a component variable and $\Phi(c)$ is some set-theoretic predicate on c (omitted when $\Phi = \mathit{true}$).

Additionally, we define the usual notation for universal quantifier:

$$\forall c{:}T\big(\Phi(c)\big).F \overset{def}{=} \neg\, \exists c{:}T\big(\Phi(c)\big).\neg\, F.$$

Semantics. The semantics is defined for closed formulas, where, for each variable in the formula, there is a quantifier over this variable in an upper nesting level. Models are pairs $\langle B, \gamma\rangle$, where B is a set of component instances of types from $\mathcal{T}$ and γ is a configuration on the set of ports P of these components. For quantifier-free closed formulas the semantics is the same as for PCL formulas. For closed formulas with quantifiers the satisfaction relation is defined by the following rules:

$$\langle B, \gamma\rangle \models \exists c{:}T\,\big(\Phi(c)\big)\ .\ F\,, \qquad \text{iff} \quad \gamma \models \bigsqcup_{c'{:}T\in B\,\wedge\,\Phi(c')} F[c'/c]\,,$$

$$\langle B, \gamma\rangle \models \Sigma c{:}T\,\big(\Phi(c)\big)\ .\ F\,,$$
$$\text{iff} \quad \{c' : T \in B \mid \Phi(c')\} \neq \emptyset\ \wedge\ \gamma \models \sum_{c'{:}T\in B\,\wedge\,\Phi(c')} F[c'/c]\,,$$

where $c' : T$ ranges over all component instances of type $T \in \mathcal{T}$ satisfying Φ and $F[c'/c]$ is obtained by replacing all occurrences of c in F by c'.

For a more concise representation of formulas, we introduce the following additional notation:

$$\sharp(c_1.p_1, \ldots, c_n.p_n) \overset{def}{=} \bigwedge_{i=1}^{n} c_i.p_i \ \wedge \bigwedge_{i=1}^{n} \bigwedge_{p\in c_i.P\setminus\{p_i\}} \overline{c_i.p}$$
$$\wedge \bigwedge_{T\in\mathcal{T}} \left(\forall c{:}T(c \notin \{c_1, \ldots, c_n\}).\ \bigwedge_{p\in c.P} \overline{c.p}\right).$$

The $\sharp(c_1.p_1, \ldots, c_n.p_n)$ notation expresses an exact interaction, i.e. all ports in the arguments must participate in the interaction and all other ports of the system cannot participate in the interaction. If $\langle B, \gamma\rangle$ is a model, it can be shown that:

$$\langle B, \gamma\rangle \models \sharp(c_1.p_1, c_2.p_2, \ldots, c_n.p_n) \\ \text{iff } c_1, c_2, \ldots, c_n \in B \text{ and } \gamma = \{\{c_1.p_1,\ c_2.p_2, \ldots, c_n.p_n\}\}\,.$$

The following three examples illustrate the specification of simple interactions.

Example 5.2.1 (Single interaction)**.** Assume that there is only one type of components T with a single port p. We characterise models with a single interaction $\{c_1.p, c_2.p\}$.

The formulas $c_1.p\ c_2.p$ and $\sim(c_1.p\ c_2.p)$ do not ensure the presence of interaction $\{c_1.p, c_2.p\}$, since the model with $\gamma = \{\{c_1.p, c_2.p, c_3.p\}\}$ satisfies these formulas. The correct specification can be expressed by a monomial, which contains all the negated ports that are not included in the interaction:

$$c_1.p \wedge c_2.p \wedge \forall c{:}T(c \notin \{c_1, c_2\}).\ \overline{c.p}\ .$$

This formula is can be equivalently rewritten using the $\sharp$ notation introduced above: $\sharp(c_1.p,\ c_2.p)$.

Example 5.2.2 (Binary interactions)**.** Assume that there is only one type of components T with a single port p. We require that all binary interactions among components be included. The formula $\sharp(c_1.p, c_2.p)$ represents a binary interaction involving ports $c_1.p$ and $c_2.p$. To obtain the required specification, we use a coalescing quantifier for each pair of components as follows:

$$\Sigma c_1 : T.\ \Sigma c_2 : T(c_1 \neq c_2).\ \sharp(c_1.p, c_2.p).$$

Example 5.2.3 (No interaction of arity greater than two)**.** Assume again that all components are of type T with a single port p. We want to express the property that all interactions involve at most two ports.

If we have three components c_1, c_2, c_3 the formula $\overline{c_1.p\ c_2.p\ c_3.p}$ forbids interactions involving all of the components. The desired specification is obtained by requiring that this formula holds for any triple of components:

$$\forall c_1 : T.\ \forall c_2 : T(c_1 \neq c_2).\ \forall c_3{:}T(c_3 \notin \{c_1, c_2\}).(\overline{c_1.p\ c_2.p\ c_3.p}\ ).$$

Alternatively, this property can be reformulated as follows: all interactions are either unary or binary. The formulas $\sharp(c_1.p)$, $\sharp(c_2.p)$, $\sharp(c_1.p,\ c_2.p)$ allow unary or binary interaction between the $c_1.p$ and $c_2.p$ ports. The formula $\overline{c_1.p\ c_2.p}$ forbids interactions involving both components. Disjunction (5.9) of the four aforementioned formulas allows only unary or binary interactions between c_1 and c_2. The desired specification is obtained by requiring

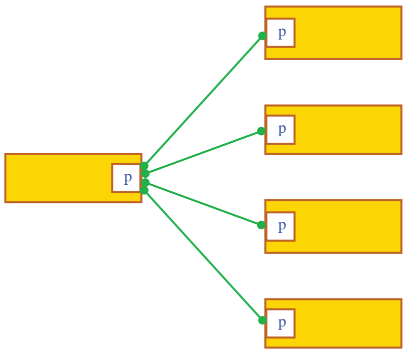


Figure 5.8 – Star architecture.

that disjunction holds for any pair of components:

$$\forall c_1 : T.\ \forall c_2 : T(c_1 \neq c_2).\ (\sharp(c_1.p) \vee \sharp(c_2.p) \vee \sharp(c_1.p,\ c_2.p) \vee \overline{c_1.p\ c_2.p}\,)).$$

The following examples illustrate the specification of architecture styles and patterns.

Example 5.2.4. The Star architecture style, illustrated in Figure 5.8, is defined for a set of components of the same type. One central component s is connected to every other component through a binary interaction and there are no other interactions. It can be specified as follows:

$$\exists s{:}T.\ \forall c{:}T(c \neq s).\ \Big(\sim(c.p\ s.p)\ \wedge\ \forall c'{:}T(c' \notin \{c, s\}).\ (\overline{c'.p\ c.p}\,)\Big) \\ \wedge \neg\big(\exists c : T.\ \sim\sharp(c.p)\big)\,. \tag{5.11}$$

The three conjuncts of this formula express, respectively, the properties: 1) any component is connected to the center; 2) components other than the center are not connected among themselves; and 3) unary interactions are forbidden.

Notice that the semantics of the first conjunct in (5.11), $\forall c : T(c \neq s).\ \sim(c.p\ s.p)$, is a conjunction of closure formulas. In this conjunct, the closure operator also allows interactions in addition to the ones explicitly defined. Therefore, to correctly specify this style, we forbid all other interactions by using the second and third conjuncts of the specification. A simpler alternative specification uses the Σ quantifier:

$$\exists s{:}T.\ \Sigma c{:}T(c \neq s).\ \sharp(c.p,\ s.p)\,. \tag{5.12}$$

The $\sharp$ notation requires interactions to be binary and the Σ quantifier allows configurations that contain only interactions satisfying $\sharp(c.p,\ s.p)$, for some c. Thus, contrary to (5.11), we do not need to explicitly forbid unary interactions and connections between non-center components.

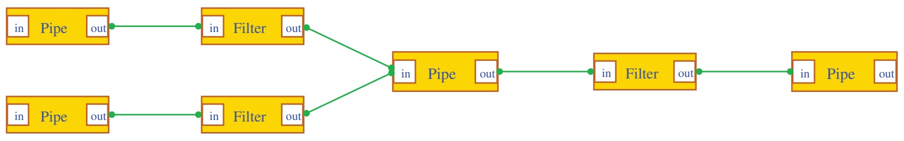


Figure 5.9 – Pipes and Filters architecture.

Example 5.2.5. The Pipes and Filters architecture style [59] involves two types of components, P and F, each having two ports *in* and *out*. Each input (resp. output) of a filter is connected to an output (resp. input) of a single pipe. The output of any pipe can be connected to at most one filter. One possible configuration is shown in Figure 5.9.

This style can be specified as follows:

$$\forall f{:}F.\ \exists p{:}P.\ \sim(f.in\ p.out) \land \forall p'{:}P(p \neq p').\ (\overline{f.in\ p'.out}) \tag{5.13}$$
$$\land\ \forall f{:}F.\ \exists p{:}P.\ \sim(f.out\ p.in) \land \forall p'{:}P(p \neq p').\ (\overline{f.out\ p'.in}) \tag{5.14}$$
$$\land\ \forall p{:}P.\ \exists f{:}F.\ \forall f'{:}F(f \neq f').\ (\overline{p.out\ f'.in}) \tag{5.15}$$
$$\land\ \forall p{:}P.\ \left(\overline{p.in\,p.out} \land \forall p'{:}P(p \neq p').\ (\overline{p.in\,p'.in} \land \overline{p.in\,p'.out})\right) \tag{5.16}$$
$$\land\ \forall f{:}F.\ \left(\overline{f.in\,f.out} \land \forall f'{:}F(f \neq f').\ (\overline{f.in\,f'.in} \land \overline{f.in\,f'.out})\right), \tag{5.17}$$

The first conjunct (5.13) requires that the input of each filter be connected to the output of a single pipe. The second conjunct (5.14) requires that the output of each filter be connected to the input of a single pipe. The third conjunct (5.15) requires that the output of a pipe be connected to at most one filter. Finally, the fourth and fifth conjuncts (5.16) and (5.17) require that pipes only be connected to filters and vice-versa.

Example 5.2.6. In the Blackboard architecture style [45], a blackboard component of type B holds data[3] that may be updated by a group of knowledge sources of type S. A controller of type C enforces mutual exclusion of write access. Figure 5.10 depicts a model with three knowledge sources. We provide specifications of models composed of: 1) a single blackboard b with two ports *sh* (share) and *ctrl* (control); 2) a single controller c with a port *ctrl*; and 3) a set of knowledge sources with a port *acc* (access). No knowledge can be shared without taking control of the blackboard through the *ctrl* port.

The Blackboard architecture style can be specified as follows:

$$b.ctrl \land c.ctrl \land \sim\left(\Sigma s{:}S.\ (s.acc\ b.sh)\right) \land \left(\forall s_1 : S.\ \forall s_2 : S(s_1 \neq s_2).\ (\overline{s_1.acc\ s_2.acc})\right).$$

The first two conjuncts require that the control ports of blackboard and controller components participate in all interactions. The third conjunct requires that all knowledge sources be connected to the blackboard. The last conjunct requires that there be no interactions

[3]We omit the data representation in this example, since only the fact that the data is updated is relevant and not the data itself.

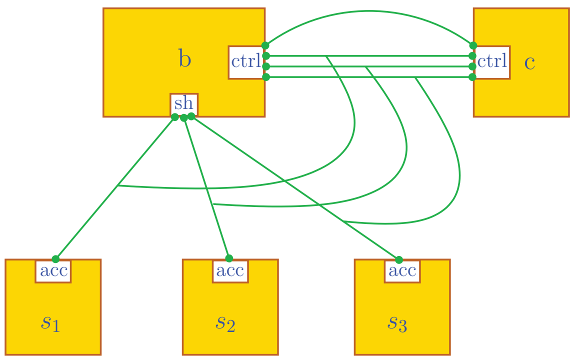


Figure 5.10 – Blackboard architecture.

involving two or more knowledge sources.

Example 5.2.7. The Request/Response pattern involves Clients and Services. It is defined as follows [46]:

“Request/Response begins when the client establishes a connection to the service. Once a connection has been established, the client sends its request and waits for a response. The service processes the request as soon as it is received and returns a response over the same connection. This sequence of client-service activities is considered to be synchronous because the activities occur in a coordinated and strictly ordered sequence. Once the client submits a request, it cannot continue until the service provides a response."

From this informal description we can infer the following. There are two types of components: a client Cl and a service S. Clients have three ports: $Cl.con$, $Cl.req$ and $Cl.rec$ that correspond to the connect, request and receive actions defined in the pattern, respectively. Service components have two ports $S.get$ for receiving a request and $S.send$ for sending a reply to the client that raised a request.

We use a coordinator of type C to enforce the properties: 1) only one client can be connected at a time to a service; and 2) a client has to connect to the service before sending a request. A unique coordinator is needed per service and therefore, the number of coordinators must match the numbers of services. There can be arbitrarily many clients. Each coordinator has three ports con, get and dsc that correspond to connect, get a request and disconnect actions. Notice that the behaviour of a coordinator is cyclic involving the sequence $con \rightarrow get \rightarrow dsc \rightarrow con$. The Request/Reply pattern is illustrated in Figure 5.11.

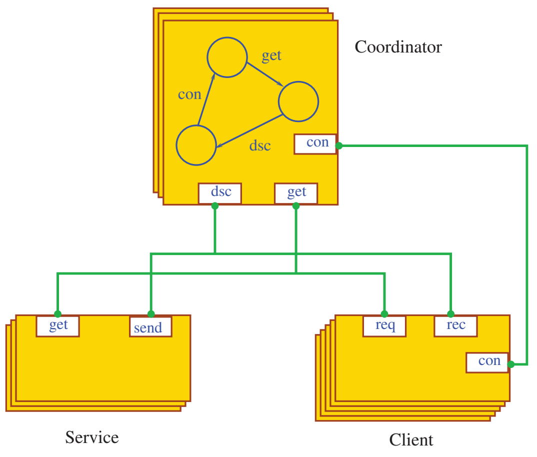


Figure 5.11 – Request/Response architecture.

This pattern can be specified as follows:

$$\begin{aligned}&\Sigma cl\!:\!Cl.\ \Sigma s\!:\!S.\ \exists c\!:\!C.\ \Big(\sharp(cl.con,\ c.con) + \sharp(cl.req,\ s.get,\ c.get)\\&\qquad\qquad\qquad\qquad\qquad + \sharp(cl.rec,\ s.send,\ c.dsc)\Big)\\&\wedge\ \Sigma cl\!:\!Cl.\ \Sigma c\!:\!C.\ \exists s\!:\!S.\ \Big(\sharp(cl.con,\ c.con) + \sharp(cl.req,\ s.get,\ c.get)\\&\qquad\qquad\qquad\qquad\qquad + \sharp(cl.rec,\ s.send,\ c.dsc)\Big)\,.\end{aligned} \tag{5.18}$$

Notice that the $\exists$ quantifier has the semantics of union. Coalescing distributes over union. Therefore, the meaning of the nested existential quantifier in the first conjunct is several configurations, where in each configuration a service is connected to a single coordinator.

The property *"a unique coordinator is needed per service"* is enforced by the formula as follows: 1) the first conjunct requires that each service be connected to a single coordinator; and 2) the second conjunct requires that each coordinator be connected to a single service.

Example 5.2.8. The Repository architecture style [44] consists of a repository component r with a port p and a set of data-accessor components of type A with ports q. The basic property *"there exists a single repository and all interactions involve it"* is specified as follows:

$$SingleRepo \overset{def}{=} \exists r\!:\!R.\ (r.p) \wedge \forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r')\,,$$

where the subterm $\forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r')$ can be expressed in the logic as $\forall r\!:\!R.\ \forall r'\!:\!R(r' \neq r).\ false$. The additional property *"there are some data-accessors and any data-accessor must be connected to the repository"* is enforced by extending the formula as follows:

$$SingleRepo \wedge \exists a\!:\!A.\ true \wedge \forall a\!:\!A.\ \exists r\!:\!R.\ \sim(r.p\ a.q)\,.$$

5.2.2 Monadic second-order configuration logic

Properties stating that two components are connected through a chain of interactions, are essential for architecture style specification. For instance, the property that all components form a single ring and not several disjoint ones can be reformulated as such a property. In [78], it is shown that such reachability properties cannot be expressed in the first-order logic. This motivates the introduction of the monadic second-order configuration logic with quantification over sets of components.

This logic extends the first-order logic with variables ranging over component sets. We write $C{:}T$ to express the fact that all components belonging to C are of type T.

Syntax. The syntax of the monadic second-order configuration logic is defined by:

$$\begin{aligned} S ::= &\ \mathit{true} \mid \phi \mid \exists c{:}T(\Phi(c)).S \mid \Sigma c{:}T(\Phi(c)).S \mid S \sqcup S \mid \neg\, S \mid S + S \\ &\mid \exists C : T(\Psi(C)).S \mid \Sigma C : T(\Psi(C)).S\,, \end{aligned}$$

where ϕ is an interaction formula, c is a component variable, C is a component set variable and $\Phi(c)$, $\Psi(C)$ are some set-theoretic predicates (omitted when *true*). Additionally, we define the usual notation for universal quantifier:

$$\forall C{:}T(\Psi(C)).S \overset{def}{=} \neg\, \exists C{:}T(\Psi(C)).\neg\, S.$$

Semantics. The semantics is defined for closed formulas, where, for each variable in the formula, there is a quantifier over this variable in an upper nesting level. Models are pairs $\langle B, \gamma\rangle$, where B is a set of component instances of types from $\mathcal{T}$ and γ is a configuration on the set of ports P of these components. The meaning of quantifier-free formulas or formulas with quantification only over component variables is as for first-order logic. We define the meaning of quantifiers over component set variables as follows:

$$\langle B, \gamma\rangle \models \exists C{:}T\ (\Psi(C))\ .\ S\,, \qquad \text{iff} \quad \gamma \models \bigsqcup_{C'{:}T\in B\,\wedge\,\Psi(C')} S[C'/C]\,,$$

$$\langle B, \gamma\rangle \models \Sigma C{:}T\ (\Psi(C))\ .\ S\,,$$
$$\text{iff} \quad \{C' : T \in B \mid \Psi(C')\} \neq \emptyset\ \wedge\ \gamma \models \sum_{C'{:}T\in B\,\wedge\,\Psi(C')} S[C'/C]\,,$$

where $C'{:}T$ ranges over all sets of components of type T that satisfy Ψ.

In the following three examples, we consider systems consisting of components of a single type T with two ports *in* and *out*. We assume that every interaction has at least one *in* port and at least one *out* port. Alternatively, this assumption can be enforced by the constraint $\neg\, (\forall c{:}T.\ \overline{c.out}\,) \wedge \neg\, (\forall c{:}T.\ \overline{c.in}\,)$.

Example 5.2.9. The property that the graph, formed by components belonging to a set C

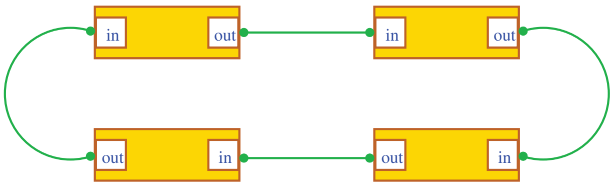


Figure 5.12 – Ring architecture.

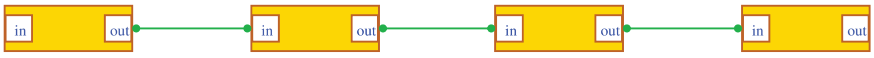


Figure 5.13 – Linear architecture.

and interactions among their ports, is connected can be expressed as follows:

$$\mathit{Connected}(C) \overset{def}{=} \forall C'{:}T(C' \subsetneq C).$$
$$\Big(\exists c'{:}T(c' \in C').\ \exists c{:}T(c \in C \setminus C').\ \sim(c.in\ c'.out) \sqcup \sim(c'.in\ c.out)\Big)\,.$$

In particular, the formula requires that for any subset C' of C there exist an interaction that involves a component that belongs to C' and a component that belongs to $C \setminus C'$.

Example 5.2.10. The component connection graph respects the Ring architecture style (Figure 5.12) if the following predicate is satisfied:

$$\mathit{Connected}(T) \ \wedge\ \Sigma c{:}T.\ \exists c'{:}T(c \neq c').\ \sharp(c.in,\ c'.out)$$
$$\wedge\ \Sigma c{:}T.\ \exists c'{:}T(c \neq c').\ \sharp(c.out,\ c'.in)\,,$$

The constraint $\mathit{Connected}(T)$ is used to ensure that all components form a single ring, rather than several disconnected ones. The second and third conjuncts require that each input port be connected to a unique output port.

Example 5.2.11. The Linear architecture style, illustrated in Figure 5.13, involves serially connected components. It is similar to the Ring architecture style: the difference being that in the Linear architecture style, there are two distinguished components that are the ends of the line such that the input of the first component and the output of the last component are not connected. The following formula requires that the components in the C set form a linear architecture.

$$\mathit{Linear}(C, out, in) \overset{def}{=}$$
$$\mathit{Connected}(C) \ \wedge\ \exists c_1{:}T(c_1 \in C).\ \exists c_2{:}T(c_2 \neq c_1 \wedge c_2 \in C).$$
$$\Big(\Sigma c{:}T(c \neq c_1 \wedge c \in C).\ \exists c'{:}T(c' \notin \{c, c_2\} \wedge c' \in C).\ \sharp(c.in,\ c'.out)$$
$$\wedge\ \Sigma c{:}T(c \neq c_2 \wedge c \in C).\ \exists c'{:}T(c' \notin \{c, c_1\} \wedge c' \in C).\ \sharp(c.out,\ c'.in)\Big)\,.$$

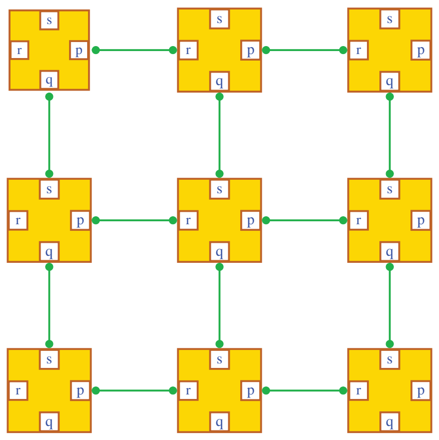


Figure 5.14 – Grid architecture.

Example 5.2.12. The Square Grid architecture style, illustrated in Figure 5.14, involves n^2 components of type T, each with four ports p, q, r and s. Adjacent components are connected through ports p and r in each row of the grid and through ports q and s in each column. It can be expressed as follows:

$$\begin{aligned}
&\Big(\forall c{:}T.\quad \Big(\overline{c.p}\ \sqcup \exists c'{:}T(c\neq c').\,\sharp(c.p,c'.r)+\overline{c.p}\Big)\\
&\qquad\qquad \wedge\Big(\overline{c.q}\ \sqcup \exists c'{:}T(c\neq c').\,\sharp(c.q,c'.s)+\overline{c.q}\Big)\\
&\qquad\qquad \wedge\Big(\overline{c.r}\ \sqcup \exists c'{:}T(c\neq c').\,\sharp(c.r,c'.p)+\overline{c.r}\Big)\\
&\qquad\qquad \wedge\Big(\overline{c.s}\ \sqcup \exists c'{:}T(c\neq c').\,\sharp(c.s,c'.q)+\overline{c.s}\Big)\Big)\\
&\bigwedge\Big(\forall c{:}T.\,\exists C{:}T(c\in C).\,\mathit{MaxLinear}(C,p,r)\\
&\qquad\qquad \wedge\exists C'{:}T(C'\cap C=\{c\}\wedge|C'|=|C|).\,\mathit{MaxLinear}(C',q,s)\Big)\\
&\bigwedge\Big(\forall c_1:T.\forall c_2:T(c_1\neq c_2).\forall c_3{:}T\big(c_3\notin\{c_1,c_2\}\big).\\
&\qquad \sim(c_1.p\,c_2.r+c_1.q\,c_3.s)\Rightarrow\exists c_4{:}T\big(c_4\notin\{c_1,c_2,c_3\}\big).\ \sim(c_2.q\,c_4.s+c_3.p\,c_4.r)\\
&\qquad \wedge\sim(c_1.q\,c_2.s+c_1.r\,c_3.p)\Rightarrow\exists c_4{:}T\big(c_4\notin\{c_1,c_2,c_3\}\big).\ \sim(c_2.r\,c_4.p+c_3.q\,c_4.s)\\
&\qquad \wedge\sim(c_1.r\,c_2.p+c_1.s\,c_3.q)\Rightarrow\exists c_4{:}T\big(c_4\notin\{c_1,c_2,c_3\}\big).\ \sim(c_2.s\,c_4.q+c_3.r\,c_4.p)\\
&\qquad \wedge\sim(c_1.s\,c_2.q+c_1.p\,c_3.r)\Rightarrow\exists c_4{:}T\big(c_4\notin\{c_1,c_2,c_3\}\big).\ \sim(c_2.p\,c_4.r+c_3.s\,c_4.q)\Big)\\
&\bigwedge\mathit{Connected}(T),
\end{aligned}$$

where

$$\mathit{MaxLinear}(C,p_1,p_2)\overset{def}{=}\mathit{Linear}(C,p_1,p_2)\ \wedge\forall C'{:}T(C\subset C').\neg\,\mathit{Linear}(C',p_1,p_2).$$

The four big conjuncts represent, respectively, the following constraints:

1. Each port participates in at most one interaction.
2. Each component belongs in one row and one column of equal sizes. The conjunction with the first constraint ensures that, for any two components, the rows (columns) in which they belong either coincide or do not intersect.
3. If two components are connected to a third one and all three components do not belong in the same row or column then there exists a fourth component that is connected to the first two. The conjunction with the second constraint ensures that given two adjacent components that belong in the same row (column), all other components that belong in the columns (rows) of the first two components are pairwise connected.
4. Components form a single grid instead of several ones. Notice that it is not possible to distinguish a single grid from several small ones in the first-order logic and thus, this architecture style cannot be expressed in first-order logic.

5.2.3 First-order configuration logic with ordered components

In this subsection, we consider an alternative extension of the first-order logic. We assume that components in models are ordered linearly and, thus, formulas can contain constraints based on the order. As a result, several architecture styles, for instance Ring and Linear architecture styles, that were not expressible in the first-order logic, can be expressed in this extension.

Syntax. The syntax of the first-order logic with ordered components is defined by:

$$F ::= \ true \mid \phi \mid \exists c[i]\!:\!T\,(\Phi(i)).F \mid \Sigma c[i]\!:\!T\,(\Phi(i)).F \mid F \sqcup F \mid \neg F \mid F + F\,,$$

where ϕ is an interaction formula; $c[i]$ refers to the i^{th} instance of the component type T; $\Phi(i)$ is a predicate based on arithmetic operations on indices (omitted when $\Phi = true$).

The universal quantifier is introduced as usual:

$$\forall c[i]\!:\!T\,(\Phi(i))\ .\ F \overset{def}{=} \neg\Big(\exists c[i]\!:\!T\,(\Phi(i))\ .\ \neg F\Big)\,.$$

Semantics. The semantics is defined for closed formulas, where, for each variable in the formula, there is a quantifier over this variable in an upper nesting level. Models are pairs $\langle B, \gamma\rangle$, where B is an ordered set of component instances of types from $\mathcal{T}$ and γ is a configuration on the set of ports P of these components.

We denote n_T the number of components of type T. Within each component type, components are ordered linearly, i.e. they can be represented by an array with the index ranging from 1 to n_T. We write c_i to denote the i^{th} component instance of type T.

For quantifier-free closed formulas the semantics is the same as for PCL formulas. For quantifiers, the satisfaction relation is defined as follows:

$$\langle B, \gamma \rangle \models \exists c[i]\!:\!T\,(\Phi(i))\ .\ F\,, \qquad \text{iff} \quad \gamma \models \bigsqcup_{\substack{j\in[1,n_T]\\ \text{s.t. } \Phi(j)}} F[c_j/c[i]]\,,$$

$$\langle B, \gamma \rangle \models \Sigma c[i]\!:\!T\,(\Phi(i))\ .\ F\,,$$
$$\text{iff} \quad \{j \in [1, n_T] \,|\, \Phi(j)\} \neq \emptyset \ \wedge\ \gamma \models \sum_{\substack{j\in[1,n_T]\\ \text{s.t. } \Phi(j)}} F[c_j/c[i]]\,,$$

where j ranges over all indices of component instances of type $T \in \mathcal{T}$, such that j satisfy Φ; and $F[c_j/c[i]]$ is the formula obtained by replacing all occurrences of the variable $c[i]$ in F by the instance c_j.

Notice that models in this logic differ from models in logics presented in the previous subsections: their components are ordered linearly. For models with unordered sets of components, the semantics can be defined by requiring the existence of a component order, with which the formula is satisfied.

In [78], it was shown that this extension of the first-order logic increases its expressive power, but it is still less expressive than the monadic second-order logic. In particular, Example 5.2.14 and Example 5.2.15 show specification of styles that are inexpressible in the first-order configuration logic. Nevertheless, Example 5.2.9 is inexpressible in this extension.

The following examples illustrate the specification of architecture styles with this extension.

Example 5.2.13. Consider the Request/Response pattern described in Example 5.2.7. Despite that it was specified in the first-order logic, the specification of this pattern can be simplified by requiring connections between pairs of services and coordinators with equal indices. This makes the second conjunct of (5.18) unnecessary. The Request/Response pattern can be specified in the first-order configuration logic with ordered components as follows:

$$\Sigma cl[i]\!:\!Cl.\ \Sigma s[j]\!:\!S.\ \exists c[k]\!:\!C(k=j).\ \Big(\sharp(cl[i].con,\ c[k].con)$$
$$+\ \sharp(cl[i].req,\ s[j].get,\ c[k].get) + \sharp(cl[i].rec,\ s[j].send,\ c[k].dsc)\Big)\,.$$

In the following two examples, we consider systems consisting of components of a single type T with two ports *in* and *out*. We assume that every interaction has at least one *in* port and at least one *out* port. Alternatively, this assumption can be enforced by the constraint

$$\neg\ \Big(\forall c[i]\!:\!T\ .\ \overline{c[i].out}\,\Big) \wedge \neg\ \Big(\forall c[i]\!:\!T\ .\ \overline{c[i].in}\,\Big)\,.$$

Example 5.2.14. The Ring architecture style (cf. Example 5.2.10), inexpressible in the first-order logic, can be specified in the first-order configuration logic with ordered components as follows:

$$\Sigma c[i]:T \;.\; \Sigma c[j]:T \; (j = i+1 \bmod n_T) \;.\; \sharp(c[i].out, c[j].in).$$

The constraint allows only interactions between neighbour components.

Example 5.2.15. The Linear architecture style (cf. Example 5.2.11) can be specified as follows:

$$\Sigma c[i]:T \;.\; \Sigma c[j]:T \; (j = i+1) \;.\; \sharp(c[i].out, c[j].in)\,.$$

The formula is similar to the specification of the Ring architecture style. Taking equality instead of the modular one in the constraint forbids the interaction between the first and the last component.

Example 5.2.16. The Grid architecture style (cf. Example 5.2.12) can be specified as follows:

$$\Sigma c[i]:T \;.\; \Sigma c[j]:T \; (j = i+1 \wedge i \neq 0 \bmod n) \;.\; \sharp(c[i].p, c[j].r) \\ + \Sigma c[i]:T \;.\; \Sigma c[j]:T \; (j = i+n) \;.\; \sharp(c[i].q, c[j].s)\,,$$

where $n = \sqrt{n_T}$. The formula is based on the specification of the Linear architecture style. It requires components be arranged in n horizontal and n vertical lines of length n.

5.3 Implementation of decision procedure

The decision procedure is based on the computation of the normal form followed by a decision whether a model satisfies at least one union term of the normal form or not. We implemented the decision procedure for PCL using Maude 2.0. Maude is a language and an efficient rewriting system supporting both equational and rewriting logic specification and programming for a wide range of applications. The set of rewriting rules in Figure 5.5 were encoded in Maude. For example, Rule 2 (distributing coalescing over union) is encoded as follows:

$$\begin{aligned}
&op\ \$Rule2\ :\ Expr\ Set\{Expr\}\ Set\{Expr\}\ \rightarrow\ Set\{Expr\}.\\
&eq\ \$Rule2(A,\ empty,\ SC)\ =\ \sqcup(SC).\\
&eq\ \$Rule2(A,\ (B,\ SB),\ SC)\ =\ \$Rule2(A,\ SB,\ (+((A,\ B)),\ SC)).\\
&eq\ A\ +\ \sqcup(SB) = \$Rule2(A,\ SB,\ empty).
\end{aligned}$$

The first line defines an additional operator for the Rule 2 that takes three arguments: a formula that is coalesced, a set of formulas that are united and an additional set of formulas

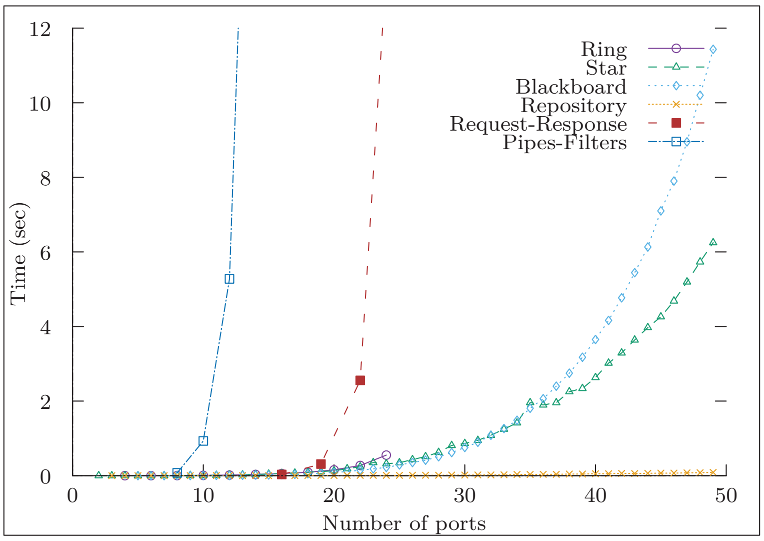


Figure 5.15 – Performance of the decision procedure for architecture styles.

that is used for the accumulation. The two following lines define the behaviour of this operator recursively. A and B are variables for single formulas, while SB and SC are variables for sets of formulas. Given a set of united formulas SB we take one of them, coalesce it with A, store the result in the accumulator SC and recursively repeat until there are no formulas left. The result returned is the union of formulas in the accumulator.

The rest of the rules in Figure 5.5 are defined in a similar manner. The Maude system can apply all rules to a given formula and return the formula in normal form. If we have a configuration logic formula in normal form and an encoded model, we can apply the implementation of the procedure in Figure 5.7 and decide for satisfaction.

In the experimental evaluation we used a set of architecture styles including Star, Ring, Request-Response pattern, Pipes-Filters, Repository and Blackboard. We used configuration logic formulas (cf. Section 5.2) and models of different sizes, including both correct and incorrect models. Quantifiers were eliminated externally and the decision procedure was applied to quantifier-free formulas. All experiments have been performed on a 64-bit Linux machine with a 2.8 Ghz Intel i7-2640M CPU with a memory limit of 1Gb and time limit of 600 seconds.

Figure 5.15 shows the average duration of the decision procedure for the six examples, as a function of the total number of ports involved in the formula. Simple architecture styles like star are decidable within seconds even for 50 ports. For architecture styles requiring more complex specifications, the number of ports that can be managed in 600 seconds is smaller. For the ring architecture the memory limit is attained for the model with 24 ports. This result shows high correlation between the maximal nesting level of quantifiers and the decision time. Pipes and Filters and Request-Response architecture styles have three nested quantifiers, while Star and Blackboard only two. Parsing the formula with eliminated quantifiers (thousands of lines) is a computationally expensive operation and is the reason of the memory limit attainment for the ring architecture. Internal quantifier elimination should eliminate the parsing overhead. Another possible direction for future optimisation is to research the possibility to delay the quantifier elimination.

5.4 Composition of architecture styles

Configuration logics can be used to specify families of architectures enforcing the same characteristic property. For a configuration logic formula f, a model is a pair $\langle \mathcal{B}, \gamma \rangle$. Notice that we neither use components behaviour nor apply glue (configuration) to them. These components are necessary to define a set of ports for the configuration. Thus, for a model $\langle \mathcal{B}, \gamma \rangle$, we can define an architecture $A = (\emptyset, \bigcup_{B \in \mathcal{B}} P_B, \gamma)$, where P_B is the interface of B, such that the application of A to $\mathcal{B}$ gives a composite component satisfying the desired property. We say that A satisfies f.

In general, it is not possible to distinguish coordinating components in a model of a configuration logic formula, thus architectures in this section do not have them. Nevertheless, architecture styles can specify components of some type as coordinating (see Example 5.2.7) making possible to take instances of these types as coordinating components.

In order to build a system that satisfy several properties, on the level of architectures we can compose architectures characterising desired properties with the operator $\oplus$. On the level of architecture styles we can compose the corresponding formulas with the operator $\wedge$. In this section, we study the relation between composed architectures and composed configuration logic formulas.

In Section 5.2, we have assumed that for a configuration logic formula a set of component types is predefined. However, sets of component types might not be equal for different formulas. We define the set of component types for the composition of formulas $f_1 \wedge f_2$ as union of sets for f_1 and f_2.

In Theorem 4.3.5, we have shown that the composition of architectures preserves safety properties. One can make an observation that for an architecture $A = (\mathcal{C}, P_A, \gamma)$ enforcing a safety property Φ, an architecture $A' = (\mathcal{C}, P_A, \gamma')$, with $\gamma' \subset \gamma$ allowing less interactions, also enforces the property Φ. Indeed, any state reachable in the system obtained by the application of A' is also reachable in the system obtained by the application of A. Thus, a family of architectures enforcing a safety property should be specified by a downward-closed formula. Here we recall the definition of downward-closed formulas.

Definition 5.4.1. A formula f is *downward-closed* iff $\langle \mathcal{B}, \gamma \rangle \models f$ implies for all non-empty configuration $\gamma_1 \subseteq \gamma, \langle \mathcal{B}, \gamma_1 \rangle \models f$.

In this section, we use the following notation: given a formula f defined over a set of types $\mathcal{T}_1$, we denote f^e the formula that coincides with f syntactically, but is defined over a set of types $\mathcal{T}_1 \cup \mathcal{T}_2$ for some $\mathcal{T}_2$. We also denote $\langle \mathcal{B}^e, \gamma^e \rangle$ models of f^e.

Proposition 5.4.2, below, shows the relation between the composition of downward-closed formulas and the composition of architectures satisfying them.

Proposition 5.4.2. *Let f_1, f_2 be two downward-closed configuration logic formulas defined over sets of component types $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Let $A_1 = (\emptyset, P_1, \gamma_1)$ and $A_2 = (\emptyset, P_2, \gamma_2)$ be two architectures satisfying f_1 and f_2, respectively. If the interaction model of the architecture $A_1 \oplus A_2$ is not empty, then $A_1 \oplus A_2$ satisfies $f_1^e \wedge f_2^e$, where f_1^e and f_2^e are the formulas defined over a set of component types $\mathcal{T}_1 \cup \mathcal{T}_2$.*

In order to prove this proposition, we use several lemmas. The first lemma shows the relation between models of equal formulas defined over different component type sets. The rest of the lemmas explore properties of downward-closed equal formulas defined over different component type sets.

Definition 5.4.3. For a model $\langle \mathcal{B}^e, \gamma^e \rangle$ with components of types in $\mathcal{T}_1 \cup \mathcal{T}_2$, its *projection* on $\mathcal{T}_1$ is a model $\langle \mathcal{B}, \gamma \rangle$, where $\mathcal{B} = \{B{:}T \mid B \in \mathcal{B}^e, T \in \mathcal{T}_1\}$, $P = \bigcup_{B \in \mathcal{B}} P_B$ is a set of ports in $\mathcal{B}$ and $\gamma = \{a^e \cap P \mid a^e \in \gamma^e\}$. We also say that γ is a projection of γ^e and $a^e \cap P$ is a projection of a^e.

Lemma 5.4.4. *Let f be a configuration formula defined over a set of component types $\mathcal{T}_1$. Then for any set of component types $\mathcal{T}_2$, f^e defined over $\mathcal{T}_1 \cup \mathcal{T}_2$ and for any models $\langle \mathcal{B}, \gamma \rangle$ and $\langle \mathcal{B}^e, \gamma^e \rangle$, where $\mathcal{B}$ has components of types in $\mathcal{T}_1$, $\mathcal{B}^e$ has components of types in $\mathcal{T}_1 \cup \mathcal{T}_2$ and $\langle \mathcal{B}, \gamma \rangle$ is a projection of $\langle \mathcal{B}^e, \gamma^e \rangle$ on $\mathcal{T}_1$, holds:*

$$\langle \mathcal{B}, \gamma \rangle \models f \Leftrightarrow \langle \mathcal{B}^e, \gamma^e \rangle \models f^e.$$

Proof. Recall that the semantics of quantifiers is defined by elimination. Notice that f is initially defined over $\mathcal{T}_1$ and it has only quantifiers over component variables of types in $\mathcal{T}_1$. Thus, quantifier elimination from f and f^e over $\mathcal{B}$ and $\mathcal{B}^e$, respectively, results in equal formulas. Hence, it is sufficient to prove the lemma for quantifier-free formulas. It can be done by structural induction.

- $f = \mathit{true}$. Both $\langle \mathcal{B}, \gamma \rangle \models f$ and $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e$.
- f is an interaction formula. If $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e$, then for any $a^e \in \gamma^e$, $a^e \overset{i}{\models} f^e$. The projection $a \in \gamma$ of a^e induces the same valuation of ports of $\mathcal{B}$, thus $a \overset{i}{\models} f$ and, consequently, $\langle \mathcal{B}, \gamma \rangle \models f$. If $\langle \mathcal{B}, \gamma \rangle \models f$, then for any $a \in \gamma$, $a \overset{i}{\models} f$. Since f and f^e do not constrain ports of components $\mathcal{B}^e \setminus \mathcal{B}$, for any $a^e \in \gamma^e$, such that a is a projection of a^e, holds $a^e \overset{i}{\models} f^e$. Therefore, $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e$.
- $f = \neg f_1$. Since, by induction hypothesis $\langle \mathcal{B}^e, \gamma^e \rangle \not\models f_1^e \Leftrightarrow \langle \mathcal{B}, \gamma \rangle \not\models f_1$, we trivially have $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e \Leftrightarrow \langle \mathcal{B}, \gamma \rangle \models f$.

- $f = f_1 + f_2$. If $\langle \mathcal{B}^e, \gamma^e \rangle \models f_1^e + f_2^e$, then there exist γ_1^e and γ_2^e, such that $\gamma^e = \gamma_1^e \cup \gamma_2^e$, $\langle \mathcal{B}^e, \gamma_1^e \rangle \models f_1^e$ and $\langle \mathcal{B}^e, \gamma_2^e \rangle \models f_2^e$. Let γ_1 and γ_2 be projections of γ_1^e and γ_2^e, respectively. Notice that $\gamma = \gamma_1 \cup \gamma_2$. By induction hypothesis, $\langle \mathcal{B}, \gamma_1 \rangle \models f_1$ and $\langle \mathcal{B}, \gamma_2 \rangle \models f_2$, therefore $\langle \mathcal{B}, \gamma \rangle \models f_1 + f_2$. Similarly, if $\langle \mathcal{B}, \gamma \rangle \models f_1 + f_2$, then there exist γ_1 and γ_2 such that $\gamma = \gamma_1 \cup \gamma_2$, $\langle \mathcal{B}, \gamma_1 \rangle \models f_1$ and $\langle \mathcal{B}, \gamma_2 \rangle \models f_2$. Consider the partition of γ^e into γ_1^e and γ_2^e, such that γ_1 and γ_2 are projections of γ_1^e and γ_2^e, respectively. By induction hypothesis, $\langle \mathcal{B}^e, \gamma_1^e \rangle \models f_1^e$ and $\langle \mathcal{B}^e, \gamma_2^e \rangle \models f_2^e$, therefore $\langle \mathcal{B}^e, \gamma^e \rangle \models f_1^e + f_2^e$.
- $f = f_1 \sqcup f_2$. If both $\langle \mathcal{B}, \gamma \rangle \models f_i$ and $\langle \mathcal{B}^e, \gamma^e \rangle \models f_i^e$, for $i \in \{1, 2\}$, then both $\langle \mathcal{B}, \gamma \rangle \models f_1 \sqcup f_2$ and $\langle \mathcal{B}^e, \gamma^e \rangle \models f_1^e \sqcup f_2^e$. If both do not satisfy neither f_1, f_1^e nor f_2, f_2^e, then $\langle \mathcal{B}, \gamma \rangle \not\models f_1 \sqcup f_2$ and $\langle \mathcal{B}^e, \gamma^e \rangle \not\models f_1^e \sqcup f_2^e$.

Thus, by structural induction, $\langle \mathcal{B}, \gamma \rangle \models f \Leftrightarrow \langle \mathcal{B}^e, \gamma^e \rangle \models f^e$. □

A particular case of Lemma 5.4.4 is obtained by choosing $\langle \mathcal{B}^e, \gamma^e \rangle = \langle \mathcal{B}, \gamma \rangle$: for any model holds $\langle \mathcal{B}, \gamma \rangle \models f \Rightarrow \langle \mathcal{B}, \gamma \rangle \models f^e$.

Notice that, by the definition of notation, $\sharp(c_1.p_1, \ldots, c_n.p_n)$ does not define an exact interaction for extended set of component types: for a formula defined over a set of component types $\mathcal{T}$, the notation allows any port to participate in interactions provided the port belong to a component of type $T \notin \mathcal{T}$. Thus, the $\sharp$ notation implicitly depends on a set of component types. Below, in order to make this dependency explicit, we will denote $\sharp^{\mathcal{T}}(c_1.p_1, \ldots, c_n.p_n)$, showing that it is defined over a set of component types $\mathcal{T}$. For example, consider a component type T_1 with a single port p and a formula $f = \exists c : T_1 \,.\, \sharp^{\{T_1\}}(c.p) \equiv \exists c : T_1 \,.\, c.p \ \wedge (\forall c_1 \!:\! T_1 (c_1 \neq c).\ \overline{c_1.p}\ )$ defined over a set of component types $\{T_1\}$. This formula allows only unary interactions. For the formula f^e defined over a set of component types $\{T_1, T_2\}$ for some component type T_2 with a port q, interactions of the form $c.p \cup \{c_i.q\}_{i=1}^n$, where $c \!:\! T_1$ and $c_i \!:\! T_2$ for $i \in \{1, n\}$, are allowed.

Lemma 5.4.5. *Let f be a downward-closed formula defined over a set of component types $\mathcal{T}_1$. Then for any set of component types $\mathcal{T}_2$, a formula f^e defined over $\mathcal{T}_1 \cup \mathcal{T}_2$ is downward-closed.*

Proof. Consider two models $\langle \mathcal{B}^e, \gamma_1^e \rangle$ and $\langle \mathcal{B}^e, \gamma_2^e \rangle$, such that $\gamma_1^e \subset \gamma_2^e$ and $\langle \mathcal{B}^e, \gamma_2^e \rangle \models f^e$. By Lemma 5.4.4, $\langle \mathcal{B}, \gamma_2 \rangle \models f$, where $\langle \mathcal{B}, \gamma_2 \rangle$ is a projection of $\langle \mathcal{B}^e, \gamma_2^e \rangle$ on $\mathcal{T}_1$. The formula f is downward-closed, therefore $\langle \mathcal{B}, \gamma_1 \rangle \models f$, where $\langle \mathcal{B}, \gamma_1 \rangle$ is a projection of $\langle \mathcal{B}^e, \gamma_1^e \rangle$. By Lemma 5.4.4, $\langle \mathcal{B}^e, \gamma_1^e \rangle \models f^e$. □

Lemma 5.4.6. *Let f be a quantifier-free downward-closed formula defined over a set of component types $\mathcal{T}_1$. Let f^e be a formula defined over $\mathcal{T}_1 \cup \mathcal{T}_2$ for some $\mathcal{T}_2$. Then f^e can be expressed in the form $\bigsqcup_{\phi \in \Phi} \phi$, where Φ is a set of interaction formulas containing only ports of components of types in $\mathcal{T}_1$*

Proof. By Proposition 5.1.20, f can be expressed in the form $g = \bigsqcup_{\phi \in \Phi} \phi$, i.e. $f \equiv g$. Let $g^e = \bigsqcup_{\phi \in \Phi} \phi$ be a formula defined over $\mathcal{T}_1 \cup \mathcal{T}_2$. We will prove that $f^e \equiv g^e$.

By Lemma 5.4.4, if $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e$, then its projection $\langle \mathcal{B}, \gamma \rangle \models f$ and, consequently, $\langle \mathcal{B}, \gamma \rangle \models g$. Let $P = \bigcup_{B \in \mathcal{B}} P_B$ be a set of ports of components in $\mathcal{B}$. By (5.5), there exists $\phi \in \Phi$ such that $\langle \mathcal{B}, \gamma \rangle \models \phi$. Any $a^e \in \gamma^e$ has a projection $a \in \gamma$. Since ϕ constrains only ports in P, for any $a^e \in \gamma^e$, $a^e \overset{i}{\models} \phi$. Therefore, $\langle \mathcal{B}^e, \gamma^e \rangle \models g^e$.

If $\langle \mathcal{B}^e, \gamma^e \rangle \models g^e$, then there exists a term $\phi \in \Phi$ in g^e. such that for all $a^e \in \gamma^e$, $a^e \overset{i}{\models} \phi$. Since ϕ constrains only ports in P, for any $a^e \in \gamma^e$, $a \overset{i}{\models} \phi$, where $a = a^e \cap P$ is a projection of a^e. Thus, a projection of $\langle \mathcal{B}^e, \gamma^e \rangle$ satisfies g. By Lemma 5.4.4, $\langle \mathcal{B}^e, \gamma^e \rangle \models f^e$. □

Proof of Proposition 5.4.2. By the semantics (5.7), $A_1 \oplus A_2$ satisfies $f_1^e \wedge f_2^e$ iff $A_1 \oplus A_2$ satisfies both f_1^e and f_2^e.

Let $\gamma_{1,2}$ be the interaction model of $A_1 \oplus A_2$ and let, for $i \in \{1, 2\}$, $\mathcal{B}_i$ be a set of components, such that $P_i = \bigcup_{B \in \mathcal{B}_i} P_B$ and $\langle \mathcal{B}_i, \gamma_i \rangle \models f_i$. Let g_1^e be a formula obtained from f_1^e by quantifier elimination over $\mathcal{B}_1 \cup \mathcal{B}_2$. By Lemma 5.4.5, f_1^e and g_1^e are downward-closed. By Lemma 5.4.6, g_1^e can be expressed as a union of interaction formulas, thus $g_1^e \equiv \bigsqcup_{\phi \in \Phi} \phi$, where each ϕ only has ports in P_1. Since $\langle \mathcal{B}_1, \gamma_1 \rangle \models f_1$, by Lemma 5.4.4, $\langle \mathcal{B}_1, \gamma_1 \rangle \models f_1^e$ and, consequently, $\langle \mathcal{B}_1, \gamma_1 \rangle \models g_1^e$. By (5.5), there exists $\phi \in \Phi$, such that $\gamma_1 \models \phi$. By Lemma 4.2.2, $a \in \gamma_{1,2}$ if $a \cap P_1 \in \gamma_1$. Since $a \cap P_1 \overset{i}{\models} \phi$ and ϕ has only ports in P_1, $a \overset{i}{\models} \phi$. Thus, $\langle \mathcal{B}_1 \cup \mathcal{B}_2, \gamma_{1,2} \rangle \models \phi$ and, consequently, $A_1 \oplus A_2$ satisfies f_1^e. The proof that $A_1 \oplus A_2$ satisfies f_2^e is symmetrical. □

For the formulas that are not downward-closed, Proposition 5.4.2 does not hold in general. It is possible to find a second formula and a pair of architectures satisfying them, such that their composition does not satisfy the composition of formulas.

Proposition 5.4.7. *Let f_1 be a formula defined over a set of component types $\mathcal{T}_1$ that is not downward-closed. Then there exist a set of component types $\mathcal{T}_2$, a formula f_2 defined over a set of component types $\mathcal{T}_2$ and two architectures $A_i = (\emptyset, P_i, \gamma_i)$ satisfying f_i, for $i \in \{1, 2\}$, such that $A_1 \oplus A_2$ does not satisfy $f_1^e \wedge f_2^e$, where, for $i \in \{1, 2\}$, f_i^e is a formula defined over a set of component types $\mathcal{T}_1 \cup \mathcal{T}_2$.*

Proof. Since f_1 is not downward-closed, there exist two models $\langle \mathcal{B}, \gamma \rangle$ satisfying f_1 and $\langle \mathcal{B}, \gamma' \rangle$ not satisfying f_1, such that $\gamma' \subset \gamma$. Consider $f_2 = \mathit{true}$ for a set of component types $\mathcal{T}_1$ and consider two architectures $A_1 = (\emptyset, \bigcup_{B \in \mathcal{B}} P_B, \gamma)$ and $A_2 = (\emptyset, \bigcup_{B \in \mathcal{B}} P_B, \gamma')$. By Lemma 4.2.2, interaction model of $A_1 \oplus A_2$ is $\gamma_{1,2} = \gamma'$. Since $\langle \mathcal{B}, \gamma' \rangle \not\models f_1$, $A_1 \oplus A_2$ does not satisfy $f_1^e \wedge f_2^e$. □

Architectures satisfying the composition of two formulas cannot always be decomposed into architectures satisfying each of the composed formulas even for downward-closed formulas. The following example illustrates this fact.

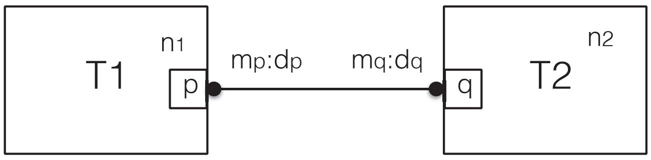


Figure 5.16 – Architecture diagram.

Example 5.4.8. Consider three component types: T_1 with interface $\{p\}$, T_2 with interface $\{q\}$ and T_3 with interface $\{r\}$. Consider a formula $f_1 = \exists c : T_1.\ c.p$ defined over $\{T_1, T_2\}$ and a formula $f_2 = \exists c : T_1.\ c.p$ defined over $\{T_1, T_3\}$. The architecture $A = (\emptyset, \{p_1, q_1, q_2, r_1, r_2\}, \{pq_1r_1, pq_2r_2\})$ satisfies $f_1 \wedge f_2$. However, it cannot be represented as $A_1 \oplus A_2$, where A_i satisfies f_i for $i \in \{1, 2\}$.

5.5 Discussion

Among the formal approaches for representing and analysing architecture descriptions, we distinguish two main categories: extensional and intentional. In extensional approaches one explicitly defines every object that is needed for the specification, i.e. the connections inducing interactions among the components. In intentional approaches all connections among the components are not specified explicitly, but they are derived from a set of logical constraints, formulating the intentions of the designer. In intentional approaches specifications are often represented as conjunctions of logical formulas. Configuration logics encompass both approaches. They allow the description of individual interactions as well as the specification of configuration sets. Example 5.2.4 illustrates the use of both approaches.

We have shown that configuration logics are a powerful tool for architecture style specification. Nevertheless, their use may be challenging for engineers. In [80], we have studied an alternative avenue for architecture style specification based on architecture diagrams. An architecture diagram consists of a set of component types equipped with a parameter defining the number of component instances. Connector motifs connect ports of component types and define configurations. A connector motif specify the number of ports participating in each interaction with a multiplicity parameter and the number of interactions involving each port with a degree parameter. A simple architecture diagram is shown in Figure 5.16. This diagram specify two component types T_1 and T_2 that have n_1 and n_2 instances, respectively. A single connector motif involve ports p and q. m_1 and m_2 define multiplicities, i.e. each connector have m_1 instances of the port p and m_2 instances of the port q. d_1 and d_2 define degrees, i.e. each instance of the port p is involved in d_1 interactions and each instance of the port q is involved in d_2 interactions. There exist several extensions of architecture diagrams allowing to use intervals in parameters or constraints based on the order of components. Architecture diagrams provide a way to graphically specify architecture styles, allow to check efficiently the satisfaction of models, but, to the best of our knowledge, are less expressive than configuration logics.

6 Conclusion and Future Work

The presented work is a contribution to a long-term research program that have been pursued for more than 15 years. The program aims at developing the BIP component framework for rigorous systems design [94]. BIP is a language and a set of supporting tools including code generators, verification and simulation tools. The theoretical work has focused on the study of expressive composition frameworks and their algebraic and logical formalisation.

We provided a study of the expressiveness of the BIP glue in two semantics. We provided theoretical results and examples showing that the classical BIP semantics, where glue consists of an interaction and a priority models, has compositionality, but does not have neither flattening, nor full expressiveness w.r.t. BIP-like SOS. We show that the source of expressiveness limitation is the definition of priority model. The offer semantics, where glue is represented with an extended interaction model, allows to define any glue expressible as a set of SOS rules in the format (2.25). In general, the classical and offer semantics are not comparable. We have presented constraints on components, under which offer semantics becomes more expressive than the classical one. We have also provided a synthesis procedure for the extended interaction model from Boolean constraints.

Our work on specification of architectures and architecture styles is a part of a broader research program investigating correct-by-construction approaches. 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.

An architecture A is a solution to a specific coordination problem, characterised by a property. We specify architectures as triples: a set of coordinating components, a set of ports and a configuration on these ports. An application of an architecture builds a composite component that satisfies the characteristic property of the architecture. Architectures can be easily composed and the composition preserves safety properties. However, an architecture is applicable to a specific number of components. In order to specify families of architectures enforcing the same property, we use configuration logics. We study their properties and present a sound and complete axiomatisation for the propositional flavour of logic. Logics are equipped with a decision procedure for checking that a given architecture model meets given style requirements.

There exist several directions for future work. Currently, the offer semantics of BIP is a purely theoretical work. We would like to implement a BIP engine for the offer semantics and related tools in order to include it in the BIP framework. For the specification of architectures and architecture styles, we are planning to incorporate connectors as hierarchically structured interactions and data transfer among the participating ports. We would also like to include priorities in configuration logics. From the analysis perspective, we will study efficient techniques for deciding satisfiability of higher-order configuration logics. We would also like to research methods for proving correctness of architectures. As a result of this, we would be able to create libraries of architectures and architecture styles for various domains. Another interesting direction of future work is to provide means for dynamic reconfigurations of architectures within an architecture style.

Bibliography

[1] Jonathan Aldrich, Craig Chambers, and David Notkin. Archjava: connecting software architecture to implementation. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 187–197. IEEE, 2002.

[2] Robert Allen and David Garlan. Formalizing architectural connection. In *Proceedings of the 16th international conference on Software engineering*, pages 71–80. IEEE Computer Society Press, 1994.

[3] Farhad Arbab. What do you mean, coordination. *Bulletin of the Dutch Association for Theoretical Computer Science, NVTI*, 1122:1–18, 1998.

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

[5] Farhad Arbab. Composition of interacting computations. In *Interactive computation*, pages 277–321. Springer, 2006.

[6] Farhad Arbab, Christel Baier, Frank de Boer, Jan Rutten, and Marjan Sirjani. Synthesis of Reo circuits for implementation of component-connector automata specifications. In *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 236–251. Springer, 2005.

[7] Farhad Arbab, Ivan Herman, and Pål Spilling. An overview of manifold and its implementation. *Concurrency: practice and experience*, 5(1):23–70, 1993.

[8] Farhad Arbab and Sun Meng. Synthesis of connectors from scenario-based interaction specifications. In *CBSE'08*, volume 5282 of *LNCS*, pages 114–129. Springer, 2008.

[9] Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber, and Joseph Sifakis. A general framework for architecture composability. In D. Giannakopoulou and G. Salaün, editors, *12th International Conference on Software Engineering and Formal Methods (SEFM 2014)*, number 8702 in LNCS, pages 128–143. Springer, 2014.

[10] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan J. M. M. Rutten. Modeling component connectors in Reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.

[11] 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.

[12] 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.

[13] Eduard Baranov and Simon Bliudze. Offer semantics: Achieving compositionality, flattening and full expressiveness for the glue operators in BIP. *Science of Computer Programming*, 109(0):2–35, 2015.

[14] 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.

[15] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In 4^{th} *IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM06)*, pages 3–12, 2006.

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

[17] BIP. `http://www-verimag.imag.fr/~async/BIP/bip.html`

[18] Simon Bliudze. Towards a theory of glue. In *ICE 2012: Distributed coordination, execution models, and resilient interaction*, volume 104 of *EPTCS*, pages 48–66, 2012.

[19] Simon Bliudze and Joseph Sifakis. The algebra of connectors — Structuring interaction in BIP. In *Proc. of the EMSOFT'07*, pages 11–20. ACM SigBED, 2007.

[20] Simon Bliudze and Joseph Sifakis. The algebra of connectors—structuring interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.

[21] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008*, volume 5201 of *LNCS*, pages 508–522. Springer, 2008.

[22] Simon Bliudze and Joseph Sifakis. Causal semantics for the algebra of connectors. *Formal Methods in System Design*, 36(2):167–194, 2010.

[23] 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, 10^{th} *International Conference on Software Composition*, volume 6708 of *LNCS*, pages 51–67. Springer, 2011.

[24] Simon Bliudze, Joseph Sifakis, Marius Dorel Bozga, and Mohamad Jaber. Architecture internalisation in BIP. In *Proceedings of the 17th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE'14, pages 169–178. ACM, 2014.

[25] Bard Bloom. *Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages*. PhD thesis, Massachusetts Institute of Technology, 1989.

[26] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. From high-level component-based models to distributed implementations. In *Proceedings of the* 10th *ACM international conference on Embedded software*, EMSOFT'10, pages 209–218. ACM, 2010.

[27] Marcello Bonsangue, Dave Clarke, and Alexandra Silva. A model of context-dependent component connectors. *Science of Computer Programming*, 77(6):685–706, 2012.

[28] Grady Booch, James Rumbaugh, and Ivar Jacobson. The unified modeling language user guide. *Addison-Welsley Longman Inc*, 1999.

[29] Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. Modeling dynamic architectures using Dy-BIP. In Thomas Gschwind, Flavio Paoli, Volker Gruhn, and Matthias Book, editors, *Software Composition*, volume 7306 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[30] Marius Bozga, Mohamad Jaber, and Joseph Sifakis. Source-to-source architecture transformation for performance optimization in BIP. In *Industrial Embedded Systems, 2009. SIES'09. IEEE International Symposium on*, pages 152–160, 2009.

[31] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The Fractal component model and its support in Java. *Software: Practice and Experience*, 36(11-12):1257–1284, 2006.

[32] Roberto Bruni, Ivan Lanese, and Ugo Montanari. A basic algebra of stateless connectors. *Theor. Comput. Sci.*, 366(1):98–120, 2006.

[33] Roberto Bruni, Alberto Lluch-Lafuente, Ugo Montanari, and Emilio Tuosto. Style-based architectural reconfigurations. *Bulletin of the EATCS*, 94:161–180, 2008.

[34] Roberto Bruni, Hernán Melgratti, and Ugo Montanari. Connector algebras, petri nets, and bip. In Edmund Clarke, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics*, volume 7162 of *Lecture Notes in Computer Science*, pages 19–38. Springer, 2012.

[35] 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, 2003.

[36] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[37] Barbara Chapman, Matthew Haines, Piyush Mehrotra, Hans Zima, and John Van Rosendale. Opus: A coordination language for multidisciplinary applications. *Scientific Programming*, 6(4):345–362, 1997.

[38] Dave Clarke. Reasoning about connector reconfiguration ii: Basic reconfiguration logic. In *Electronic Notes in Theoretical Computer Science*, volume 159, pages 61–77. Elsevier, 2006.

[39] Dave Clarke. Coordination: Reo, nets, and logic. In *International Symposium on Formal Methods for Components and Objects*, pages 226–256. Springer, 2007.

[40] Dave Clarke. A basic logic for reasoning about connector reconfiguration. *Fundamenta Informaticae*, 82(4):361–390, 2008.

[41] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency. *Electr. Notes Theor. Comput. Sci.*, 154(1):101–119, 2006.

[42] Dave Clarke and Jose Proenca. Coordination via interaction constraints i: Local logic. In *Electronic Proceedings in Theoretical Computer Science*, volume 12, pages 17–39, 2009.

[43] Dave Clarke, José Proença, Alexander Lazovik, and Farhad Arbab. Deconstructing Reo. *ENTCS*, 229(2):43–58, 2009.

[44] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting software architectures: views and beyond.* Pearson Education, 2002.

[45] Daniel D. Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[46] Robert Daigneau. *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services.* Addison-Wesley, 2011.

[47] Gero Decker, Frank Puhlmann, and Mathias Weske. Formalizing service interactions. In *Business Process Management*, pages 414–419, 2006.

[48] Kasper Dokter, Sung-Shik Jongmans, Farhad Arbab, and Simon Bliudze. Combine and conquer: Relating BIP and Reo. *Journal of Logical and Algebraic Methods in Programming*, 86(1):134–156, 2017.

[49] Kasper Dokter, Sung-Shik T. Q. Jongmans, Farhad Arbab, and Simon Bliudze. Relating BIP and reo. In Sophia Knight, Ivan Lanese, Alberto Lluch-Lafuente, and Hugo Torres Vieira, editors, *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015.*, volume 189 of *EPTCS*, pages 3–20, 2015.

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

[51] Hartmut Ehrig and Barbara Konig. Deriving bisimulation congruences in the DPO approach to graph rewriting. In *FoSSaCS*, volume 2987 of *LNCS*, pages 151–166. Springer, 2004.

[52] 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.

[53] Matthias Felleisen. On the expressive power of programming languages. In 3^{rd} *European Symposium on Programming (ESOP'90)*, volume 432 of *LNCS*, pages 134–151. Springer, 1990.

[54] Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *Formal Methods for Components and Objects*, pages 22–43. Springer, 2006.

[55] José Luis Fiadeiro. *Categories for Software Engineering*. Springer, 2004.

[56] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.

[57] David Garlan. Software architecture: a travelogue. In *Proceedings of the on Future of Software Engineering*, pages 29–39. ACM, 2014.

[58] David Garlan, Robert Monroe, and David Wile. Acme: An architecture description interchange language. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, pages 159–173. IBM Press, 1997.

[59] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1–39. World Scientific Publishing Company, 1993.

[60] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM, 2002.

[61] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

[62] Gregor Gößler and Joseph Sifakis. Priority systems. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4-7, 2003, Revised Lectures*, volume 3188 of *Lecture Notes in Computer Science*, pages 314–329. Springer, 2003.

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

[64] Dan Hirsch, Paola Inverardi, and Ugo Montanari. Modeling software architectures and styles with graph grammars and constraint solving. In Patrick Donohoe, editor, *Software Architecture*, volume 12 of *IFIP*, pages 127–143. Springer, 1999.

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

[66] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2003.

[67] Paola Inverardi and Simone Scriboni. Connectors synthesis for deadlock-free component-based architectures. In *ASE '01*, pages 174–181. IEEE Computer Society, 2001.

[68] ISO/IEC/IEEE 42010. *Systems and software engineering — Architecture description*, 2011.

[69] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[70] Sung-Shik T.Q. Jongmans, Christian Krause, and Farhad Arbab. Encoding context-sensitivity in reo into non-context-sensitive semantic models. In Wolfgang De Meuter and Gruia-Catalin Roman, editors, *Coordination Models and Languages*, volume 6721 of *LNCS*, pages 31–48. Springer, 2011.

[71] Tomas Kalibera and Petr Tuma. Distributed component system based on architecture description: The Sofa experience. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 981–994. Springer, 2002.

[72] Uwe Keller. Some remarks on the definability of transitive closure in first-order logic and Datalog. Internal report, Digital Enterprise Research Institute (DERI), University of Innsbruck, 2004.

[73] Christian Koehler, Alexander Lazovik, and Farhad Arbab. Connector rewriting with high-level replacement systems. *Electronic Notes in Theoretical Computer Science*, 194(4):77–92, 2008.

[74] Jeff Kramer. Configuration programming — A framework for the development of distributable systems. In *CompEuro'90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 374–384. IEEE, 1990.

[75] Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Modeling dynamic reconfigurations in Reo using high-level replacement systems. *Sci. of Comp. Prog.*, 76(1):23–36, 2011.

[76] Daniel Le Métayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998.

[77] Zheng Li, Yan Jin, and Jun Han. A runtime monitoring and validation framework for web service interactions. In *ASWEC*, pages 70–79, 2006.

[78] Leonid Libkin. *Elements of finite model theory.* Springer, 2013.

[79] Isaac Liu, Jan Reineke, and Edward A. Lee. A PRET architecture supporting concurrent programs with composable timing properties. In *Conference Record of the* 44th *Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 2111–2115, 2010.

[80] Anastasia Mavridou, Eduard Baranov, Simon Bliudze, and Joseph Sifakis. Architecture diagrams: A graphical language for architecture style specification. In Massimo Bartoletti, Ludovic Henrio, Sophia Knight, and Hugo Torres Vieira, editors, Proceedings 9th *Interaction and Concurrency Experience,* Heraklion, Greece, 8-9 June 2016, volume 223 of *Electronic Proceedings in Theoretical Computer Science*, pages 83–97. Open Publishing Association, 2016.

[81] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980.

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

[83] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[84] MohammadReza Mousavi, Iain Phillips, Michel A. Reniers, and Irek Ulidowski. Semantics and expressiveness of ordered SOS. *Information and Computation*, 207:85–119, 2009.

[85] MohammadReza Mousavi, Michel A. Reniers, and Jan Friso Groote. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3):238–272, 2007.

[86] Mert Ozkaya and Christos Kloukinas. Design-by-contract for reusable components and realizable architectures. In *Proceedings of the 17th international ACM Sigsoft symposium on Component-based software engineering*, pages 129–138. ACM, 2014.

[87] George A Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in computers*, 46:329–400, 1998.

[88] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

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

[90] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[91] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. *Annual IEEE Symposium on Foundations of Computer Science*, 2:746–757, 1990.

[92] 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. IEEE Computer Society, 2003.

[93] Joseph Sifakis. A framework for component-based construction. In 3^{rd} *IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, 2005.

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

[95] Pawel Sobocinski. A non-interleaving process calculus for multi-party synchronisation. In *ICE*, volume 12 of *EPTCS*, pages 87–98, 2009.

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

[97] Eirik Tryggeseth, Bjørn Gulla, and Reidar Conradi. Modelling systems with variability using the proteus configuration language. In *Software Configuration Management*, pages 216–240. Springer, 1995.

[98] Rob J. van Glabbeek. Musings on encodings and expressiveness. 89:81–98, 2012.

[99] Rob Van Ommering, Frank Van Der Linden, Jeff Kramer, and Jeff Magee. The Koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[100] Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley, 1998.

[101] Peter Wegner. Coordination as constrained interaction (extended abstract). In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models*, volume 1061 of *Lecture Notes in Computer Science*, pages 28–33. Springer, 1996.

[102] Da-Qian Zhang, Kang Zhang, and Jiannong Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, 44(3):186–200, 2001.

Eduard Baranov

Personal Data

Address:	EPFL IC IINFCOM RISD INJ 337 (Batiment INJ) Station 14 CH-1015 Lausanne
Phone:	+41 21 69 32580
email:	eduard.baranov@epfl.ch

Education

2012-present *École polytechnique fédérale de Lausanne*
PhD student in Computer Science
Rigorous System Design laboratory under supervision of Prof. Joseph Sifakis

2007-2012 *Saint Petersburg State University*
Mathematics & Mechanics Faculty
Specialist degree in Software Engineering: graduation with honours

Professional Experience

2007-2012 Lanit-Tercom Software developer (part-time 50%)
Participated in different projects including

- development of a database migration tool
- software assessment of a huge banking system
- creation of a specific Silverlight control

Used technologies: C#, F#, MSSQL Server, Oracle, C++, Silverlight

Projects

- PhD thesis: *A semantic framework for architecture modelling.*
  My work is focused on the architecture modelling. Architectures organise coordination between components in order to build complex systems and to make them manageable. An architecture is considered as an operator that, applied to a set of components, builds a composite component meeting a characteristic property. The underlying concepts of components and their interactions originate from the BIP Framework. I studied preservation of properties by composition of architectures, and configuration logics that allow to specify architecture styles representing families of architectures satisfying the same property.
- Specialist diploma: *Automation of QA in the project of DB migration from SQL Server into Oracle.*
  The goal was to test automatically that migrated database contains the same data as initial one and has the same functionality. The data comparison was made by checking hashes of each table, while the functionality was tested by simultaneous execution of pre-generated traces on both databases comparing the results after each trace step.

Main Publications

- Anastasia Mavridou, Eduard Baranov, Simon Bliudze and Joseph Sifakis. Configuration Logics: Modelling Architecture Styles. In Journal of Logical and Algebraic Methods in Programming, 2016.
- Paul Attie, Eduard Baranov, Simon Bliudze, Mohamad Jaber, Joseph Sifakis. A General Framework for Architecture Composability. In Formal Aspects of Computing, pp. 125, 2015
- Eduard Baranov, Simon Bliudze. Offer semantics: Achieving compositionality, flattening and full expressiveness for the glue operators in BIP. In Science of Computer Programming, vol. 109, pp. 235, 2015

Skills

C#, F#, Java, OCaml, Haskell, C
Good knowledge of algorithms, data structures and design patterns

Languages

English:	Fluent, IELTS 7.5 in 2012
French:	Low Intermediate, B1
Russian:	Mothertongue

Awards

Diplomas of Russian Mathematical Olympiad in 2005, 2006, 2007

Summer Schools

2014	Summer School Marktoberdorf
2012, 2013	Nano-Tera/Artist International Summer School