Configuration Logics: Modelling Architecture Styles

Anastasia Mavridou, Eduard Baranov, Simon Bliudze and Joseph Sifakis

École polytechnique fédérale de Lausanne
Rigorous System Design Laboratory
Station 14, CH-1015, Lausanne, Switzerland
E-mails: firstname.lastname@epfl.ch

Abstract

We study a framework for the specification of architecture styles as families of architectures involving a common set of types of components and coordination mechanisms. The framework combines two logics: 1) interaction logics for the specification of architectures as generic coordination schemes involving a configuration of interactions between typed components; and 2) configuration logics for the specification of architecture styles as sets of interaction configurations. The presented results build on previous work on architecture modelling in BIP. We show how propositional interaction logic can be extended into a corresponding configuration logic by adding new operators on sets of interaction configurations. In addition to the usual set-theoretic operators, configuration logic is equipped with a coalescing operator + to express combination of configuration sets. We provide a complete axiomatization of propositional configuration logic as well as decision procedures for checking that an architecture satisfies given logical specifications. To allow genericity of specifications, we study first-order and second-order extensions of the propositional configuration logic. First-order logic formulas involve quantification over component variables. Second-order logic formulas involve additional quantification over sets of components. We provide several examples illustrating the application of the results to the characterisation of various architecture styles. We also provide an experimental evaluation using the Maude rewriting system to implement the decision procedure for the propositional flavour of the logic.

Keywords: Architecture Styles, Coordination, Configuration Logics, Component Interaction, BIP

1. Introduction

Architectures are common means for organizing coordination between components in order to build complex systems and to make them manageable. They depict generic coordination principles between components and embody design rules that can be understood by all. Architectures allow thinking on a higher plane and avoiding low-level mistakes. They are a means for ensuring global

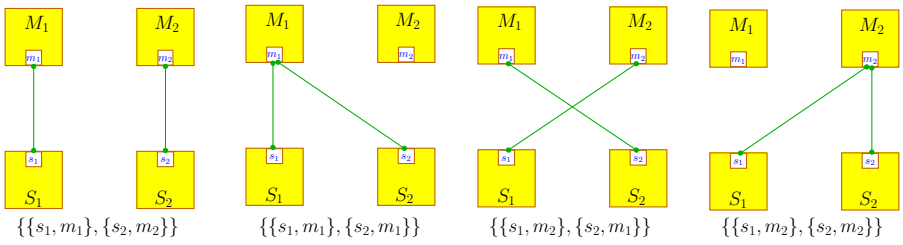


Figure 1: Master/Slave architectures.

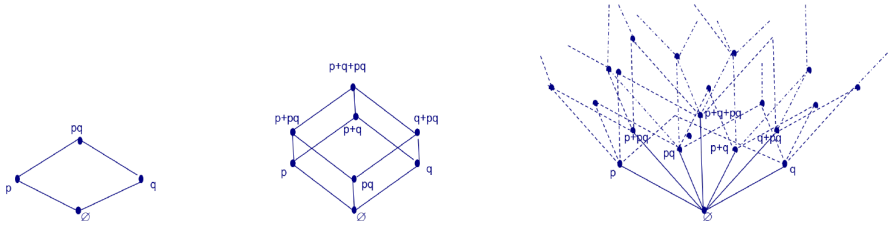
coordination properties between components and thus, achieving correctness by construction [1].

Using architectures largely accounts for our ability to master complexity and develop systems cost-effectively. System developers extensively use reference architectures ensuring both functional and non-functional properties, e.g. fault-tolerant, time-triggered, adaptive, security architectures.

Many languages have been proposed for architecture description such as architecture description languages, e.g. [2], coordination languages, e.g. [3] and configuration languages [4]. All these works rely on the distinction between behaviour of individual components and their coordination in the overall system organization. Informally architectures are characterized 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.

The field of software architecture remains relatively immature [5]. A lot of foundational issues remain open. One is the distinction between architectures and their properties. Architecture styles characterize 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. Simple examples of architecture styles are Pipeline, Ring, Master/Slave, Pipe and Filter. For instance, Master/Slave architectures integrate two types of components, masters and slaves such that each slave can interact only with one master. Figure 1 depicts four Master/Slave architectures involving two master components M_1, M_2 and two slave components S_1, S_2. Their communication ports are, respectively, m_1, m_2 and s_1, s_2. The architectures correspond to interaction configurations: $\{\{s_1, m_1\}, \{s_2, m_2\}\}$, $\{\{s_1, m_1\}, \{s_2, m_1\}\}$, $\{\{s_1, m_2\}, \{s_2, m_1\}\}$ and $\{\{s_1, m_2\}, \{s_2, m_2\}\}$. The set $\{s_i, m_j\}$ denotes an interaction between ports s_i and m_j. A configuration is a non-empty set of interactions. The Master/Slave architecture style characterizes all the Master/Slave architectures for arbitrary numbers of masters and slaves.

The paper studies the relation between architectures and architecture styles. This relation is similar to the relation between programs and their specifications. As program specifications can be expressed by using logics, e.g. temporal logics, architecture styles can be specified by configuration logics characterizing classes of architectures.



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

Figure 2: Lattices of interactions (a), configurations (b) and configuration sets (c) for $P = \{p, q\}$.

First, we propose a propositional configuration logic (PCL) whose formulas represent, for a given set of components, the allowed configuration sets. Then, we introduce first-order and second-order logics as extensions of the propositional logic. These allow genericity of description as they are defined for types of components.

The proposed formalism is declarative and has some similarities with languages used for a feature-oriented analysis of architectures, such as OCL [6]. It differs from formalisms used to describe the possible configurations of a dynamic architecture by using graph grammars [7, 8].

The meaning of a configuration logic formula is a configuration set. A configuration on a set of components represents a particular architecture. Thus, configuration logic formulas describe architecture sets. The definition of configuration logics requires considering three hierarchically structured semantic domains:

The lattice of interactions. An interaction a is a non-empty subset of P, the set of ports of the integrated components. Its execution implies the atomic synchronization of all component actions (at most one action per component) associated with the ports of a.

The lattice of configurations. Configurations are non-empty sets of interactions characterizing architectures.

The lattice of configuration sets. Sets of configurations are properties described by the configuration logic.

We aim at describing systems of interacting components: in each configuration there must be at least one interaction and each of the interactions should involve at least one component. Therefore, we only consider non-empty interactions and configurations.

Figure 2 shows the three lattices for $P = \{p, q\}$. For the lattice of configuration sets, we show only how it is generated.

This work consistently extends results on modelling architectures by using propositional interaction logic [9, 10, 11], which are Boolean algebras on the set

of ports P of the composed components. Their semantics is defined via a satisfaction relation $\models_i$ between interactions and formulas. Each interaction logic formula ϕ represents exactly the set of interactions corresponding to Boolean valuations of P satisfying ϕ.

Configuration logic is a powerset extension of interaction logic. Its formulas are generated from the formulas of the propositional interaction logic by using the operators union, intersection and complementation, as well as a *coalescing operator* $+$. 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 the configuration logic is defined via a satisfaction relation $\models$ between configurations $\gamma = \{a_1, ..., a_n\}$ and formulas. An interaction formula f represents any configuration consisting of interactions satisfying it; that is $\gamma \models f$ if, for all $a \in \gamma$, $a \models_i f$. 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.

Despite its apparent complexity, configuration logic is easy to use because of its stratified construction. From interaction logic it inherits the Boolean connectives of conjunction ($\wedge$), disjunction ($\vee$) and negation ($\bar{}$). It also uses the set-theoretic operations of union ($\sqcup$), complementation ($\neg$) and coalescing ($+$). It can be shown that intersection coincides with conjunction.

The following simple example illustrates the difference between interaction and configuration logic. 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 the configurations built from these interactions, i.e. $\big\{\{p, q\}\big\}$, $\big\{\{p, q, s\}\big\}$ and $\big\{\{p, q\}, \{p, q, s\}\big\}$. The formula $p \wedge q \wedge \overline{r} + true$ characterizes all the configurations of the form $\gamma = \gamma_1 \cup \gamma_2$, where γ_1 satisfies $p \wedge q \wedge \overline{r}$ and γ_2 is an arbitrary configuration.

Formulas of the form $f + true$, denoted $\sim f$, present a particular interest for writing specifications. Their characteristic configuration set is the largest set containing configurations satisfying f. The formula $\sim f$ admits the modal interpretation "possible f". We show that $\neg f = \sim \overline{f}$. This means that the complement of the characteristic configuration set of f is the set "possible $\overline{f}$". Dually, the complement of $\overline{f}$ is "possible f".

We provide a full axiomatization 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 satisfaction of a formula by an architecture model.

To allow genericity of specifications, we study first-order and second-order extensions of the propositional configuration logic. First-order logic formulas involve quantification over component variables. Second-order logic formulas involve additionally quantification over sets of components. For instance, the first-order formula $\forall c\!:\!Filter.\ \exists c'\!:\!Pipe.\ \sim(c.in \wedge c'.out)$ expresses the fact that for any component c of type $Filter$ there exists a component c' of type $Pipe$

such that the port *in* of c interacts with the port *out* of c'. Second-order logic is needed to express some interesting topological properties, e.g. the existence of cycles of interactions.

The paper extends the results of [12] as follows:

- We introduce and study three classes of PCL formulas: the *downward-closed*, *upward-closed* and $\cup$-*closed* formulas.
- We define a *full normal form*, which is unique for any PCL formula.
- We describe two methods for checking satisfaction of formulas.
- Finally, we provide full proofs of all results and additional examples that illustrate the specification of architecture styles.

The paper is structured as follows. Section 2 presents basic facts about interaction logic. Section 3 presents the propositional configuration logic. Section 4 includes basic theorems and properties of the propositional configuration logic, as well as the results about the normal form and the decision method. Section 5 introduces an architecture specification methodology. Section 6 presents first-order and second-order logics and their application to the specification of architecture styles. Section 7 presents the results of an implementation of the decision procedure in the Maude rewriting system. Section 8 presents an analysis of related work. Section 9 concludes the paper and discusses directions for future work.

2. Propositional interaction logic

The propositional interaction logic (PIL), studied in [9, 10], is a Boolean logic used to characterize the interactions between components on a global set of ports P. In this section, we present only the results needed to introduce the propositional configuration logic (Section 3). Below, we assume that the set P is given and finite.

Definition 2.1. An *interaction* is a non-empty set of ports $a \subseteq P$.

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 is defined as usual: $\phi_1 \wedge \phi_2 \stackrel{def}{=} \overline{(\overline{\phi_1} \vee \overline{\phi_2})}$. To simplify the notation, we omit it 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. Let $a \in 2^P$ be an interaction. We define: $a \models_i \phi$ iff ϕ evaluates to *true* for the valuation $p = \mathit{true}$, for all $p \in a$ and $p = \mathit{false}$, for all $p \notin a$. Thus, the semantic domain of PIL is the lattice of configurations $C(P) = 2^{I(P)\setminus\{\emptyset\}}$, where $I(P) = 2^P$ (Figure 2).

The operators meet the usual Boolean axioms and the additional axiom $\bigvee_{p\in P} p = \mathit{true}$ meaning that interactions are non-empty sets of ports.

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 $a' \models_i m_a$ iff $a' = a$.

Example 2.2. Consider a system consisting of three components: a sender with port p and two receivers with ports q and r, respectively. We can express the following interaction patterns:

- *Strong synchronization* between the components is specified by a single interaction involving all components. In PIL it is represented by the monomial pqr.
- *Broadcast* defines weak synchronization among the sender and any number of the receivers: $\big\{\{p\},\{p,q\},\{p,r\},\{p,q,r\}\big\}$, 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* ensures that either all or none of the receivers are involved in the interaction: $\big\{\{p\},\{p,q,r\}\big\}$ and can be characterised by the formula $p\overline{q}\,\overline{r} \vee pqr$.

3. Propositional configuration logic

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

$$f ::= \mathit{true} \mid \phi \mid \neg f \mid f + f \mid f \sqcup f\,, \tag{1}$$

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:

$$f_1 \sqcap f_2 \stackrel{def}{=} \neg(\neg f_1 \sqcup \neg f_2)\,,$$
$$f_1 \Rightarrow f_2 \stackrel{def}{=} \neg f_1 \sqcup f_2\,.$$

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 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:

$$\gamma \models \mathit{true}\,, \qquad \text{always,} \tag{2}$$

$$\gamma \models \phi\,, \qquad \text{if } \forall a \in \gamma, a \models_i \phi, \text{ where } \phi \text{ is an interaction formula and } \models_i \text{ is the satisfaction relation of PIL,} \tag{3}$$

$$\gamma \models f_1 + f_2\,, \qquad \text{if there exist } \gamma_1, \gamma_2 \in C(P) \setminus \{\emptyset\}, \text{ such that } \gamma = \gamma_1 \cup \gamma_2, \ \gamma_1 \models f_1 \text{ and } \gamma_2 \models f_2, \tag{4}$$

$$\gamma \models f_1 \sqcup f_2\,, \qquad \text{if } \gamma \models f_1 \text{ or } \gamma \models f_2, \tag{5}$$

$$\gamma \models \neg\, f\,, \qquad \text{if } \gamma \not\models f \text{ (i.e. } \gamma \models f \text{ does not hold).} \tag{6}$$

In particular, the meaning of an interaction formula ϕ in PCL is the set $2^{I_a} \setminus \{\emptyset\}$, with $I_a = \{a \in I(P) \,|\, a \models_i \phi\}$, 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:

$$\gamma \models f_1 \sqcap f_2\,, \quad \text{if } \gamma \models f_1 \text{ and } \gamma \models f_2, \tag{7}$$

$$\gamma \models f_1 \Rightarrow f_2\,, \quad \text{if } \gamma \not\models f_1 \text{ or } \gamma \models f_2. \tag{8}$$

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

Proposition 3.1. *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 (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 a 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

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

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

Example 3.2. 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, characterizes the four configurations of Figure 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 $i \neq i', j \neq j'$ are monomials defining a binary interaction between ports s_i and m_j, respectively.

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 1.

4. Properties of PCL

In this section, we present some properties of PCL. In particular, we show that PCL is a conservative extension of PIL (Section 4.1). We also present the key properties of PCL operators (Sections 4.2–4.7), which allow us to define a normal form (Section 4.8), a sound and complete axiomatization of PCL (Section 4.9) and decision procedures for the equality and satisfaction of PCL formulas (Subsection 4.10).

4.1. Conservative extension

Notice that from the PCL semantics of interaction formulas, 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 4.1. *$\phi_1 \wedge \phi_2 \equiv \phi_1 \sqcap \phi_2$, 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 (3), $\gamma \models \phi_1 \wedge \phi_2$ iff $\gamma \subseteq \{a \mid a \models_i \phi_1 \wedge \phi_2\} = \{a \mid a \models_i \phi_1 \wedge a \models_i \phi_2\}$. By (7), $\gamma \models \phi_1 \sqcap \phi_2$ iff $\gamma \models \phi_1$ and $\gamma \models \phi_2$, that is $\gamma \subseteq \{a \mid a \models_i \phi_1\} \cap \{a \mid a \models_i \phi_2\} = \{a \mid a \models_i \phi_1 \wedge a \models_i \phi_2\}$. 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 paper, 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,

$$\gamma \models f_1 \vee f_2\,, \quad \text{if } \gamma \models f_1 \sqcup f_2 \sqcup f_1 + f_2. \tag{9}$$

Proposition 4.2. *For any interaction formulas ϕ_1 and ϕ_2 and any $\gamma \in C(P)$ such that $\gamma \neq \emptyset$, we have $\gamma \models \phi_1 \vee \phi_2$ iff $\forall a \in \gamma, a \models_i \phi_1 \vee \phi_2$.*

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 $a \in \gamma$, $a \models_i \phi$. Thus, in all three cases all interactions in γ either satisfy ϕ_1 or ϕ_2 and consequently, for all $a \in \gamma$, $a \models_i \phi_1 \vee \phi_2$. Conversely, if γ consists of interactions a, such that $a \models_i \phi_1 \vee \phi_2$, these interactions can be split into two possibly empty sets γ_1 and γ_2 such that for all $a \in \gamma_j$, where $j \in [1, 2]$, $a \models_i \phi_j$. 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 $\gamma \models \phi_1 \vee \phi_2$. □

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

Proposition 4.3. *The operators $\sqcup$, $\neg$, $\wedge$ satisfy the usual axioms of propositional logic.*

Proof. The proof is immediate from the semantics (5), (6) and (7). □

4.2. The coalescing operator

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

Proposition 4.4. *+ is associative, commutative and has an absorbing element $false \stackrel{def}{=} \neg true$.*

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

Proposition 4.5. *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. □

Associativity of coalescing and union, together with the distributivity of coalescing over union, immediately imply the following generalisation of the extended semantics of disjunction (9).

Corollary 4.6. *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 4.7. 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 4.8. *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 4.9. Let $P = \{p, q\}$ and consider $f = p \sqcup q$, $f_1 = p$ and $f_2 = q$. We then have $(f + f_1) \wedge (f + f_2) = \big((p \sqcup q) + p\big) \wedge \big((p \sqcup q) + q\big)$ and $f + (f_1 \wedge f_2) = (p \sqcup q) + (p \wedge q)$. The configuration $\big\{\{p\}, \{q\}\big\}$ satisfies the former, but not the latter.

Proposition 4.10. *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 $\gamma \models f + f_2$. □

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 4.11.

- A formula f is *downward-closed* iff $\gamma \models f$ implies $\forall \gamma_1 \subseteq \gamma, \gamma_1 \models f$.
- A formula f is *upward-closed* iff $\gamma \models f$ implies $\forall \gamma_1 \supseteq \gamma, \gamma_1 \models f$.
- 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 4.12.

- $p \sqcup q$ 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 4.13. *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, $\gamma_1 \models f \wedge g$. □

Proposition 4.14. *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. □

Proposition 4.15. *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 $\gamma \models f$. □

Proposition 4.16. *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, $\gamma_1 \cup \gamma_2 \models f + g$. □

The following proposition shows that the complement of a downward-closed formula is an upward-closed formula.

Proposition 4.17. *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. □

Proposition 4.18. *A formula is $\cup$-closed and downward-closed iff it is an interaction formula.*

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 $a \models_i \phi$, consequently $\gamma_1 \cup \gamma_2 \models \phi$. This shows that ϕ is $\cup$-closed and downward-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. □

Thus, interaction formulas are represented by formulas that are both downward-closed and $\cup$-closed. Figure 3 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 $P = \{p, q\}$, $f_1 = p\overline{q} \sqcup \overline{p}\, q$ is not $\cup$-closed, since $\{\{p\}\}$ and $\{\{q\}\}$ are models of f_1 but $\{\{p\}, \{q\}\}$ is not a model of f_1. Similarly, $f_2 = p\overline{q} + \overline{p}\, q$ 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 4.19. *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 4.18). 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, $\gamma \models \phi \wedge (f_1 + f_2)$. □

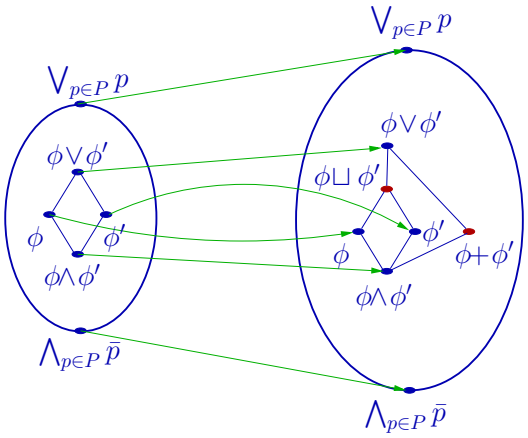


Figure 3: Correspondence between the lattices of PIL and PCL.

Notice that coalescing is not idempotent in general, as it is shown in the following example.

Example 4.20. $(p \sqcup q)+(p \sqcup q) \not\equiv p \sqcup q$. 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 4.21. *$f + f \equiv f$ 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, $\gamma \models f$. □

4.3. The closure operator

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 4.22. For any formula f, the *closure operator* $\sim$ is defined by putting $\sim f \stackrel{def}{=} f + true$. 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 4.23. For $P = \{p, q, r\}$ the formula f characterizing all the 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 4.24. *$\sim\sim f \equiv \sim f$ 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 4.15, 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 4.25. *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. □

The closure operator can be interpreted as a modal operator with existential quantification. The formula $\sim f$ characterizes 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 4.26. *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 4.25, 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 4.17, $\neg f'$ is upward-closed. Hence, by Proposition 4.25, $\sim \neg f \Rightarrow \neg f'$ and, consequently, $f' \Rightarrow \neg \sim \neg f$. □

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 4.27. *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:

$$
\begin{aligned}
\sim(f_1 \sqcup f_2) &\equiv (f_1 \sqcup f_2) + true \equiv f_1 + true \sqcup f_2 + true \equiv \sim f_1 \sqcup \sim f_2\,,\\
\sim(f_1 \vee f_2) &\equiv f_1 + true \sqcup f_2 + true \sqcup f_1 + f_2 + true\\
&\equiv f_1 + true \sqcup f_2 + true \equiv \sim f_1 \sqcup \sim f_2\,,\\
\sim f_1 + \sim f_2 &\equiv f_1 + true + f_2 + true \equiv f_1 + f_2 + true \equiv \sim(f_1 + f_2)\,,\\
\sim f_1 \wedge \sim f_2 &\equiv (f_1 + true) \wedge (f_2 + true) \equiv f_1 + f_2 + true \equiv \sim(f_1 + f_2)\,.
\end{aligned}
$$

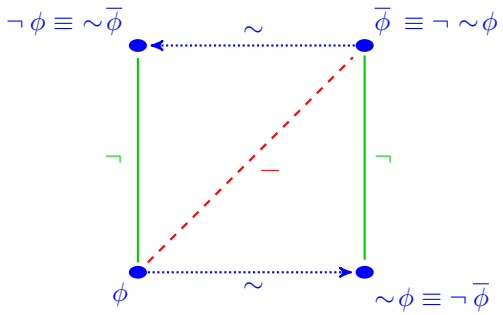


Figure 4: Correspondence between negation and complementation of interaction formulas.

□

4.4. The complementation operator

The following results allow us to address the relation between complementation and negation.

Lemma 4.28. *For any interaction formula ϕ, the following holds:*

$$\phi \sqcup \overline{\phi} \sqcup (\overline{\phi} + \phi) \equiv true\,. \tag{10}$$

Proof. The proof is immediate from Corollary 4.6 and the fact that $\phi \vee \overline{\phi} \equiv true$, for any interaction formula ϕ. □

Notice that the three terms in the left-hand side of (10) are mutually disjoint.

Proposition 4.29. *For any interaction formula ϕ, holds $\neg\, \phi \equiv \sim \overline{\phi}$.*

Proof. By Lemma 4.28, we have $\neg\, \phi \equiv \overline{\phi} \sqcup (\phi + \overline{\phi}) \equiv \overline{\phi} + true \equiv \sim \overline{\phi}$. □

In particular, this means that complementation can also be interpreted as a modality. Proposition 4.29 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 4.29 and, for the latter, Proposition 4.24. Figure 4 depicts the relations between complementation and negation of the interaction formulas.

4.5. Complementation of coalescings of 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 4.30. *For any formulas f, g, we have:*

$$f + g \equiv \sim f \wedge \sim g \wedge (f \vee g)\,.$$

Proof. By (9) and Proposition 4.27, 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 4.31. *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 4.30 $\phi+\psi \equiv \sim\phi \wedge \sim\psi \wedge (\phi \vee \psi)$. 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 4.29 gives $\neg(\phi+\psi) \equiv \overline{\phi} \sqcup \overline{\psi} \sqcup \sim(\overline{\phi} \wedge \overline{\psi})$ □

Proposition 4.31 allows the elimination of complementation as shown in the following example.

Example 4.32. 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 4.30 and Proposition 4.31 can be generalized as follows:

Lemma 4.33. *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 4.34. *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 4.33 and 4.34 are similar to the proofs of Propositions 4.30 and 4.31, respectively.

4.6. Conjunction of coalescings of interaction formulas

Conjunction of coalescings of interaction formulas can be eliminated by using the following distributivity result to push it down within the formula tree.

Proposition 4.35. *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 4.34, 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 4.29:

$$\bigwedge_{\xi\in\Phi\cup\Psi}\neg\overline{\xi} \wedge \neg\Big(\sim\bigwedge_{\phi\in\Phi}\overline{\phi}\Big) \wedge \neg\Big(\sim\bigwedge_{\psi\in\Psi}\overline{\psi}\Big) \equiv \bigwedge_{\xi\in\Phi\cup\Psi}\sim\xi \wedge \overline{\bigwedge_{\phi\in\Phi}\overline{\phi}} \wedge \overline{\bigwedge_{\psi\in\Psi}\overline{\psi}}.$$

By Proposition 4.27 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 (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 $a \models_i \phi\wedge\psi$ and, a fortiori, there exists $\xi \in \Phi\cup\Psi$, such that $a \models_i \xi$. Thus, the closure operator in the first conjunct of this formula can be discarded. Finally, by Proposition 4.19, we have

$$\Big(\sum_{\xi\in\Phi\cup\Psi}\xi\Big) \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi) \equiv \sum_{\xi\in\Phi\cup\Psi}\Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi}(\phi\wedge\psi)\Big).$$

□

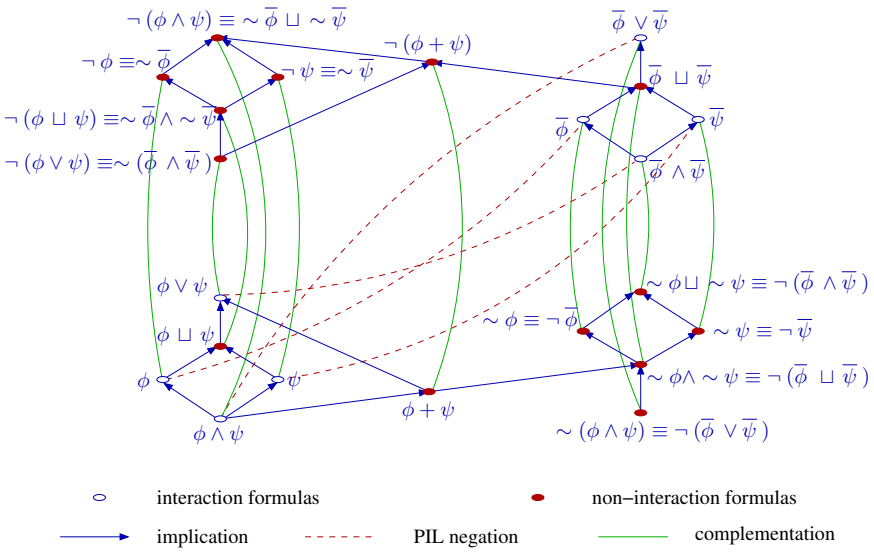


Figure 5: PCL lattice.

Example 4.36. 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 exists 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 one interaction. Thus, the conjunction of these formulas gives the equivalent representation:

$$
\begin{aligned}
f &\equiv (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) \wedge (\phi_1 + \phi_2 + \phi_3 + \phi_4) \\
&= \phi_1 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) + \phi_2 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) \\
&+ \phi_3 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4) + \phi_4 \wedge (\phi_1\phi_3 \vee \phi_1\phi_4 \vee \phi_2\phi_3 \vee \phi_2\phi_4)\,.
\end{aligned}
$$

4.7. The PCL lattice

The PCL lattice is illustrated in Figure 5. 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 4.37. *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

$$\begin{aligned}
\sim f &\equiv \sim \bigsqcup_{i\in I} \Big(\phi_i \wedge \bigwedge_{j\in J_i} \neg\, \phi_j \Big) \\
&\equiv \bigsqcup_{i\in I} \sim \Big(\phi_i \wedge \bigwedge_{j\in J_i} \neg\, \phi_j \Big) && \text{// by Proposition 4.27} \\
&\equiv \bigsqcup_{i\in I} \sim \Big(\phi_i \wedge \sim \Big(\sum_{j\in J_i} \overline{\phi_j} \Big) \Big) && \text{// by Propositions 4.29 and 4.27} \\
&\equiv \bigsqcup_{i\in I} \sim \Big(\phi_i + \sum_{j\in J_i} \big(\phi_i \wedge \overline{\phi_j} \big) \Big) && \text{// by Proposition 4.19} \\
&\equiv \bigsqcup_{i\in I} \Big(\sim \phi_i \wedge \bigwedge_{j\in J_i} \sim \big(\phi_i \wedge \overline{\phi_j} \big) \Big) && \text{// by Proposition 4.27} \\
&\equiv \bigsqcup_{i\in I} \Big(\neg\, \overline{\phi_i} \wedge \bigwedge_{j\in J_i} \neg\, \overline{\phi_i \wedge \overline{\phi_j}} \Big) && \text{// by Proposition 4.29} \\
&\equiv \bigsqcup_{i\in I} \neg \Big(\overline{\phi_i} \sqcup \bigsqcup_{j\in J_i} \overline{\phi_i \wedge \overline{\phi_j}} \Big)\,.
\end{aligned}$$

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. □

4.8. Normal form

To simplify the presentation, we assume in this subsection that disjunction can appear only within interaction formulas, i.e. we do not consider

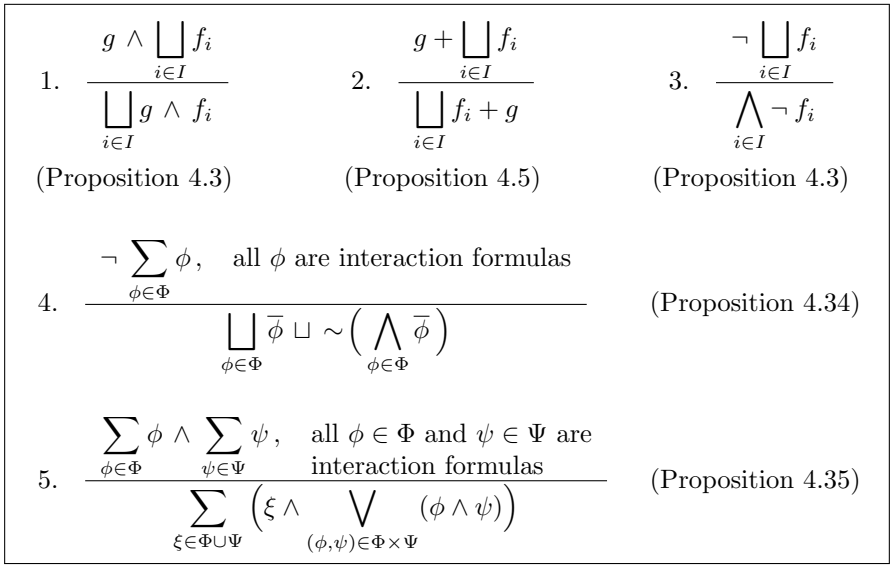


Figure 6: Rewriting system for computing the normal form by the procedure in Figure 7.

the extension (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 using the rewriting system given in Figure 6 and usual Boolean transformations for interaction formulas. Notice that no two rules can be simultaneously applicable to the same node. Normal form of a formula is computed by applying the procedure in Figure 7 to the root of its Abstract Syntax Tree (AST).

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 order of application of the rules from the rewriting system in Figure 6.

Constraint 4.38. 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 4.39. We extend Constraint 4.38 to include usual Boolean transformations. Hence, at every step of the process, all interaction sub-formulas are maintained in Disjunctive Normal Form.

```
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 7: Procedure for computing the normal form using the rewriting system of Figure 6.

Example 4.40. The following example illustrates the normalization process:

$$
\begin{aligned}
&(pq \sqcup r) \wedge (pr + \neg\, q) \equiv (pq \sqcup r) \\
&\qquad \wedge \big(pr + (\overline{q} \;\sqcup\; \overline{q} \;+ \mathit{true})\big) && \text{// rule 4 with } \Phi = \{q\} \\
&\quad \equiv (pq \sqcup r) \wedge (pr + \overline{q} \;+ \mathit{true}) && \text{// absorption laws} \\
&\quad \equiv \big(pq \wedge (pr + \overline{q} \;+ \mathit{true})\big) \\
&\qquad \sqcup \big(r \wedge (pr + \overline{q} \;+ \mathit{true})\big) && \text{// rule 1} \\
&\quad \equiv \big((pq \wedge pr) + (pq \wedge \overline{q}\,) + (pq \wedge \mathit{true})\big) \\
&\qquad \sqcup \big((r \wedge pr) + (r \wedge \overline{q}\,) + (r \wedge \mathit{true})\big) && \text{// rule 5} \\
&\quad \equiv (pqr + \mathit{false} + pq) \sqcup (pr + r\overline{q} \;+ r) && \text{// Boolean laws} \\
&\quad \equiv pr + r\overline{q} \;+ r\,. && \text{// absorption and identity laws}
\end{aligned}
$$

The first step removes the complementation. Then the application of distributivity rules pushes conjunction down in the expression tree 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.

Theorem 4.41. *Under Constraint 4.38, the rewriting system in Figure 6 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 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 the 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.
- 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.
- 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 $p(n') < p(n)$.
- Application of Rule 4 decreases N. It transforms a $\neg$-node into a union of conjunctions and coalescing.
- Application of Rule 5 decreases C and does not change N. It transforms a $\wedge$-node into a coalescing of interaction formulas.
- 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 4.38, guarantees the confluence of the rewriting system.

2. Since all rewriting rules in Figure 6 preserve equality, the formula obtained by application of these rules is necessarily equal to the initial one.

3. Let f be an irreducible formula and let T be an AST of f. Any 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 $n_1 = x \to (m_1, \ldots, m_l)$, then $x \to (n_1, \ldots, n_k)$ can be substituted by $x \to (m_1, \ldots, m_l, n_2, \ldots, n_k)$ 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 coalescing 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 4.39), we conclude that f is in normal form. □

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 $\bigsqcup_{i \in I} \sum_{j \in J} m_{i,j,k}$, where $m_{i,j,k}$ are full monomials. We show that any formula has an equivalent full normal form.

Lemma 4.42. *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 corresponding to the full monomial m_i: $m_i = \bigwedge_{p\in a_i} p \wedge \bigwedge_{p\notin a_i} \overline{p}$.*

Proof. 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 $a_i \models_i m_i$.

$\gamma \models \sum_{i\in I} m_i$ 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 $\gamma \models f$. □

Theorem 4.43. *Any formula f has a unique full normal form.*

Proof. By Theorem 4.41 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 4.6, can be further transformed into a union of coalesced full monomials. The application of Proposition 4.5 leads to the full normal form. Uniqueness is a corollary of Lemma 4.42. □

Example 4.44. Let $P = \{p, q, r\}$ and consider the normal form formula $pr+r\overline{q}$. It can be transformed into full normal form as follows:

$$\begin{aligned} pr + r\overline{q} \;&\equiv (pqr \sqcup p\overline{q}\, r \sqcup pqr + p\overline{q}\, r) + (p\overline{q}\, r \sqcup \overline{p}\;\; \overline{q}\, r \sqcup p\overline{q}\, r + \overline{p}\;\; \overline{q}\, r) \\ &\equiv (pqr+p\overline{q}r) \sqcup (pqr+\overline{p}\,\overline{q}r) \sqcup (pqr+p\overline{q}r+\overline{p}\,\overline{q}r) \sqcup p\overline{q}r \sqcup (p\overline{q}r+\overline{p}\,\overline{q}r) \sqcup (p\overline{q}r+\overline{p}\,\overline{q}r) \\ &\qquad \sqcup (pqr + p\overline{q}\, r) \sqcup (pqr + p\overline{q}\, r + \overline{p}\;\; \overline{q}\, r) \sqcup (pqr + p\overline{q}\, r + \overline{p}\;\; \overline{q}\, r). \end{aligned}$$

4.9. Soundness and completeness

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 sets of interaction formulas Φ and Ψ, holds
$$\sum_{\phi\in\Phi} \phi \wedge \sum_{\psi\in\Psi} \psi \;\equiv \sum_{\xi\in\Phi\cup\Psi} \Big(\xi \wedge \bigvee_{(\phi,\psi)\in\Phi\times\Psi} (\phi \wedge \psi)\Big).$$
6. For any set of interaction formulas Φ, holds
$$\neg\Big(\sum_{\phi\in\Phi} \phi\Big) \equiv \bigsqcup_{\phi\in\Phi} \overline{\phi} \sqcup \sim\Big(\bigwedge_{\phi\in\Phi} \overline{\phi}\Big).$$

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.

Algorithm:	$J' := \emptyset$; $l := 1$; $b := true$;
	while ($l \leq n$ and b) **do**
	$X := \{j \in J \mid a_l \models_i \bigvee_{k \in K_j} m_{j,k}\}$;
	if ($X \neq \emptyset$)
	$J' := J' \cup X$;
	else
	$b := false$;
	endif
	$l := l + 1$;
	od
	return $J' = J$;

Figure 8: Algorithm for checking satisfaction of formulas.

Theorem 4.45. *The above axiomatization is sound and complete for the equality in PCL.*

Proof. Soundness of all the above axioms has been proved in previous sections. Completeness is an immediate consequence of the existence of a unique full normal form. □

4.10. 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 8 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 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 8, 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 4.42). If formulas φ_γ and $\neg f$ have no common models then γ is a model of f. Thus, $\gamma \models f$ iff $\varphi_\gamma \wedge \neg f \equiv false$. This latter equality can be decided by verifying whether all terms of the normal form of $\varphi_\gamma \wedge \neg f$ are equal to $false$. Recall that the terms of a formula in normal form are coalescings of interaction formulas. Therefore, for a term to be equal to $false$, it is sufficient that one of its participating interaction formulas be equal to $false$. Finally, as in Boolean logics, a disjunction of monomials is equal to $false$ iff all monomials contain one of the variables at least twice in opposite (positive and negative) forms.

Example 4.46. Let $P = \{p, q, r\}$ and consider $f = p\,q + r\,\overline{p}$ and $\gamma = \big\{\{p, q, r\}, \{q, r\}\big\}$. In order to decide whether $\gamma \models f$, we first apply the algorithm in Figure 8. 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 $\varphi_\gamma \wedge \neg f = (p\,q\,r + q\,r\,\overline{p}\,) \wedge \neg\,(p\,q + r\,\overline{p}\,) \equiv false$. We have

$$
\begin{aligned}
&(pqr + qr\overline{p}\,) \wedge \neg\,(pq + r\overline{p}\,) \\
&\quad \equiv (pqr + qr\overline{p}\,) \wedge \Big((\overline{p} \vee \overline{q}\,) \sqcup (\overline{r} \vee p) \sqcup \sim\big((\overline{p} \vee \overline{q}\,) \wedge (\overline{r} \vee p)\big)\Big) \\
&\quad \equiv \Big(\big(pqr \wedge (\overline{p} \vee \overline{q}\,)\big) + \big(qr\overline{p} \wedge (\overline{p} \vee \overline{q}\,)\big)\Big) \\
&\qquad \sqcup \Big(\big(pqr \wedge (\overline{r} \vee p)\big) + \big(qr\overline{p} \wedge (\overline{r} \vee p)\big)\Big) \\
&\qquad \sqcup \Big((pqr + qr\overline{p}\,) \wedge \big((\overline{p} \vee \overline{q}\,) \wedge (\overline{r} \vee p) + true\big)\Big) \\
&\quad \equiv (false + \overline{p}\,qr) \sqcup (pqr + false) \\
&\qquad \sqcup \Big((pqr + qr\overline{p}\,) \wedge \big((\overline{p}\;\overline{r} \vee \overline{q}\;\overline{r} \vee p\,\overline{q}\,) + true\big)\Big) \\
&\quad \equiv false \sqcup false \sqcup \Big((pqr + qr\overline{p}\,) \wedge \big((\overline{p}\;\overline{r} \vee \overline{q}\;\overline{r} \vee p\,\overline{q}\,) + true\big)\Big) \\
&\quad \equiv (pqr \vee qr\overline{p}\,) \wedge pqr + (pqr \vee qr\overline{p}\,) \wedge qr\overline{p} \\
&\qquad + (pqr \vee qr\overline{p}\,) \wedge (\overline{p}\;\overline{r} \vee \overline{q}\;\overline{r} \vee p\,\overline{q}\,) + (pqr \vee qr\overline{p}\,) \wedge true \\
&\quad \equiv pqr + qr\overline{p} + false + (pqr \vee qr\overline{p}\,) \equiv false\,.
\end{aligned}
$$

5. Architecture style specification methodology

The methodology for writing architecture style specifications can be conceptually simplified from the fact that an architecture can be considered as a hypergraph whose vertices are ports and edges are interactions. If a is an interaction then its characteristic monomial m_a specifies in PCL a single configuration (hypergraph) that contains only the interaction (edge) a. In some frameworks, the term of connector is used as a synonym of interaction. The formula $\sim m_a$ specifies all the configurations (hypergraphs) that contain the

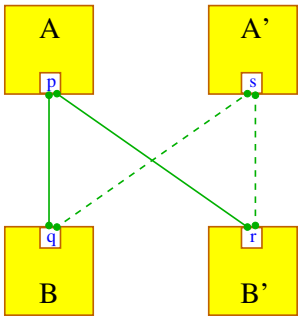


Figure 9: Configurations of Example 5.1.

interaction (edge) a. It can be considered as a predicate on ports expressing their connectivity.

A key idea in writing architecture style specifications is that these can be expressed as logical relations between connectivity formulas of the form $\sim m$ where m is a monomial. This allows simplification through separation of concerns: first configurations are specified as the conjunction of formulas on Boolean variables representing connectivity formulas; then after simplification, the connectivity formulas are replaced. This may require another round of simplifications based on specific properties of PCL. The idea is illustrated in Examples 5.1 and 5.2.

Example 5.1. Consider the four components in Figure 9 with the following connectivity constraints: *1) if A is connected to one of the components B and B', it is also connected to the other and similarly for A'; 2) precisely one of the components A and A' interacts with B and B'.*

First of all, let us assume that $C(X,Y)$ is the predicate *"X is connected to Y"*, where $X \in \{A, A'\}$ and $Y \in \{B, B'\}$. The above constraints can then be formalised by the formula:

$$\begin{aligned}
&\big(C(A,B) \Leftrightarrow\ C(A,B')\big)\ \wedge\ \big(C(A',B) \Leftrightarrow\ C(A',B')\big)\\
&\qquad \wedge\ \big(C(A,B) \Rightarrow \neg C(A',B)\big)\ \wedge\ \big(C(A,B) \sqcup C(A',B)\big)\\
&\equiv \big(C(A,B) \wedge C(A,B') \sqcup \neg\, C(A,B) \wedge \neg\, C(A,B')\big)\\
&\qquad \wedge\ \big(C(A',B) \wedge C(A',B') \sqcup \neg\, C(A',B) \wedge \neg\, C(A',B')\big)\\
&\qquad \wedge\ \big(\neg\, C(A,B) \sqcup \neg C(A',B)\big)\ \wedge\ \big(C(A,B) \sqcup C(A',B)\big)\\
&\equiv \big(C(A,B) \wedge C(A,B') \wedge \neg\, C(A',B) \wedge \neg\, C(A',B')\big) \qquad (11)\\
&\qquad \sqcup \big(\neg\, C(A,B) \wedge \neg\, C(A,B') \wedge C(A',B) \wedge C(A',B')\,.
\end{aligned}$$

Notice now that the predicates $C(A,B)$, $C(A,B')$, $C(A',B)$ and $C(A',B')$ can be expressed, respectively, as $\sim(pq)$, $\sim(pr)$, $\sim(sq)$ and $\sim(sr)$. Substituting

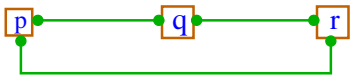


Figure 10: Configuration of Example 5.2.

these into (11), we obtain

$$\begin{aligned}
&\big(\sim(pq) \wedge \sim(pr) \ \wedge \neg \sim(sq) \ \wedge \neg \sim(sr)\big) \\
&\qquad \sqcup \ \big(\neg \sim(pq) \ \wedge \neg \sim(pr) \wedge \sim(sq) \wedge \sim(sr)\big) \\
&\qquad \equiv \big(\sim(pq+pr) \wedge \overline{sq} \ \wedge \overline{sr} \ \big) \sqcup \big(\overline{pq} \ \wedge \overline{pr} \ \wedge \sim(sq+sr)\big) \\
&\qquad \equiv \big(\sim(pq+pr) \wedge (\overline{s} \ \vee \overline{q}\,\overline{r} \,)\big) \sqcup \big((\overline{p} \ \vee \overline{q}\,\overline{r} \,) \wedge \sim(sq+sr)\big)\,. \qquad (12)
\end{aligned}$$

The left-hand choice in (12) imposes that any configuration has the port p connected with both ports q and r without the participation of s; symmetrically for the right-hand choice. Notice that this formula also allows configurations to contain arbitrary unary interactions and interactions among p and s that do not involve q and r, which is consistent with the constraints of the example. Indeed, Figure 9 shows only two among all the possible configurations. For instance, neither the constraints above, nor (12) impose that the interactions be binary. In particular, the configurations $\{pqr\}$ and $\{sqr\}$, consisting of one ternary interaction each, satisfy both the constraints and (12).

Example 5.2. Consider a system with three ports p, q, r and the following connectivity constraint: *If any port is connected to the two others, the latter have to be connected between themselves, i.e. "connected" is an euclidean relation.* Figure 10 represents a configuration corresponding to this constraint.

In order to specify this constraint in PCL, we first define three predicates $X = \sim(pq)$, $Y = \sim(qr)$ and $Z = \sim(pr)$. The above constraint can be formalised by the conjunction of three implications:

$$\begin{aligned}
&(X \wedge Y \Rightarrow Z) \wedge (Y \wedge Z \Rightarrow X) \wedge (Z \wedge X \Rightarrow Y) \\
&\qquad \equiv \neg Z \wedge \neg Y \sqcup \neg Y \wedge \neg X \sqcup \neg X \wedge \neg Z \sqcup X \wedge Y \wedge Z \\
&\qquad \equiv \neg \big((Z \sqcup Y) \wedge (Y \sqcup X) \wedge (X \sqcup Z)\big) \sqcup X \wedge Y \wedge Z \\
&\qquad \equiv (Z \sqcup Y) \wedge (Y \sqcup X) \wedge (X \sqcup Z) \Rightarrow X \wedge Y \wedge Z \\
&\qquad \equiv (X \wedge Y) \sqcup (Y \wedge Z) \sqcup (Z \wedge X) \Rightarrow X \wedge Y \wedge Z\,. \qquad (13)
\end{aligned}$$

Substituting $\sim(pq)$, $\sim(qr)$, $\sim(pr)$ for X, Y, Z, respectively, in (13) we obtain

$$\begin{aligned}
&\big(\sim(pq) \wedge \sim(qr)\big) \sqcup \big(\sim(qr) \wedge \sim(pr)\big) \sqcup \big(\sim(pr) \wedge \sim(pq)\big) \\
&\qquad \Rightarrow \ (\sim(pq) \wedge \sim(qr) \wedge \sim(pr)) \\
&\qquad \equiv \sim(pr+qr) \sqcup \sim(pq+qr) \sqcup \sim(pq+pr) \Rightarrow \sim(pq+qr+pr)\,.
\end{aligned}$$

6. First and second 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. Additionally, we denote C_T the set of all the components of type T. Finally, we assume the existence of the universal component type U, such that any component or component set is of this type. Thus, C_U represents all the components of a model.
- 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.

6.1. First-order configuration logic

Syntax. The language of the formulas of the first-order configuration logic is an extension of the PCL language by allowing Boolean expressions on component variables, existential quantification and a specific coalescing quantifier $\Sigma c{:}T$.

$$F ::= \mathit{true} \mid \phi \mid \exists c{:}T\big(\Phi(c)\big).F \mid \Sigma c{:}T\big(\Phi(c)\big).F \mid F \sqcup F \mid \neg F \mid F + F\,,$$

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:

$$\forall c{:}T\big(\Phi(c)\big).F \stackrel{def}{=} \neg\, \exists c{:}T\big(\Phi(c)\big).\neg\, F.$$

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. We assume that a finite set of component types $\mathcal{T} = \{T_1, \ldots, T_n\}$ is given. 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:

$$\langle B, \gamma\rangle \models \exists c{:}T\left(\Phi(c)\right)\,.\,F\,, \qquad \text{iff} \quad \gamma \models \bigsqcup_{c':T\in B\,\wedge\,\Phi(c')} F[c'/c]\,,$$

$$\langle B, \gamma\rangle \models \Sigma c{:}T\left(\Phi(c)\right)\,.\,F\,,$$
$$\text{iff} \quad \{c' : T \in B \,|\, \Phi(c')\} \neq \emptyset \ \wedge\ \gamma \models \sum_{c':T\in B\,\wedge\,\Phi(c')} F[c'/c]\,,$$

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:

$$\sharp(c_1.p_1, \ldots, c_n.p_n) \stackrel{def}{=} \bigwedge_{i=1}^{n} c_i.p_i \ \wedge \bigwedge_{i=1}^{n} \bigwedge_{p \in c_i.P \setminus \{p_i\}} \overline{c_i.p}$$
$$\wedge \bigwedge_{T \in \mathcal{T}} \left(\forall c{:}T\big(c \notin \{c_1, \ldots, c_n\}\big).\ \bigwedge_{p \in c.P} \overline{c.p} \right) .$$

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:

$$\langle B, \gamma \rangle \models \sharp(c_1.p_1, c_2.p_2, \ldots, c_n.p_n)$$
$$\text{iff } c_1, c_2, \ldots, c_n \in B \text{ and } \gamma = \big\{\{c_1.p_1,\ c_2.p_2, \ldots, c_n.p_n\}\big\} .$$

The following three examples illustrate the specification of simple interactions.

Example 6.1 (Single interaction)**.** Assume that there is only one type of components T with a single port p. We characterize 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 $\gamma = \big\{\{c_1.p,\ c_2.p,\ c_3.p\}\big\}$ 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:

$$c_1.p \wedge c_2.p \wedge \forall c{:}T\big(c \notin \{c_1, c_2\}\big).\ \overline{c.p} \ . \tag{14}$$

This formula is can be equivalently rewritten using the $\sharp$ notation introduced above: $\sharp(c_1.p,\ c_2.p)$.

Example 6.2 (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 6.3 (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.

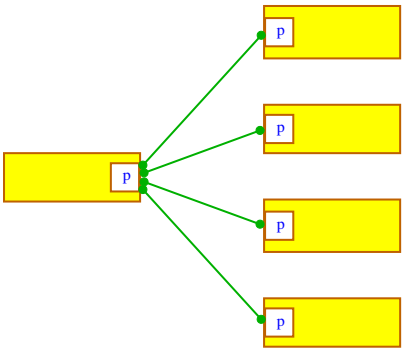


Figure 11: Star architecture.

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:

$$\forall c_1 : T.\ \forall c_2 : T(c_1 \neq c_2).\ \forall c_3 : T\big(c_3 \notin \{c_1, c_2\}\big).(\overline{c_1.p\ c_2.p\ c_3.p}\ ).$$

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 (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 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 6.4. The Star architecture style, illustrated in Figure 11, 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:

$$\exists s{:}T.\ \forall c{:}T(c \neq s).\ \Big(\sim(c.p\ s.p)\ \wedge\ \forall c'{:}T\big(c' \notin \{c, s\}\big).\ (\overline{c'.p\ c.p}\ )\Big) \\ \wedge \neg\big(\exists c{:}T.\ \sim\sharp(c.p)\big)\,. \quad (15)$$

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 (15), $\forall c : T(c \neq s).\ \sim (c.p\ s.p)$, is a conjunction of closure formulas. In this conjunct, the closure operator also allows interactions in addition to the ones explicitly defined.

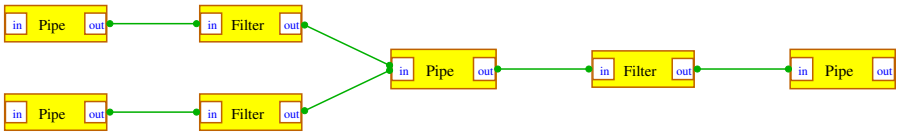


Figure 12: Pipes and Filters architecture.

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:

$$\exists s{:}T.\ \Sigma c{:}T(c \neq s).\ \sharp(c.p,\ s.p)\,. \tag{16}$$

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 (15), we do not need to explicitly forbid unary interactions and connections between non-center components.

Example 6.5. The Pipes and Filters architecture style [13] 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 12.

This style can be specified as follows:

$$\forall f{:}F.\ \exists p{:}P.\ \sim(f.in\ p.out) \wedge \forall p'{:}P(p \neq p').\ \big(\overline{f.in\ p'.out}\,\big) \tag{17}$$

$$\wedge\ \forall f{:}F.\ \exists p{:}P.\ \sim(f.out\ p.in) \wedge \forall p'{:}P(p \neq p').\ \big(\overline{f.out\ p'.in}\,\big) \tag{18}$$

$$\wedge\ \forall p{:}P.\ \exists f{:}F.\ \forall f'{:}F(f \neq f').\ \big(\overline{p.out\ f'.in}\,\big) \tag{19}$$

$$\wedge\ \forall p{:}P.\ \Big(\overline{p.in\,p.out}\ \wedge \forall p'{:}P(p \neq p').\ \big(\overline{p.in\,p'.in}\ \wedge \overline{p.in\,p'.out}\,\big)\Big) \tag{20}$$

$$\wedge\ \forall f{:}F.\ \Big(\overline{f.in\,f.out}\ \wedge \forall f'{:}F(f \neq f').\ \big(\overline{f.in\,f'.in}\ \wedge \overline{f.in\,f'.out}\,\big)\Big)\,, \tag{21}$$

The first conjunct (17) requires that the input of each filter be connected to the output of a single pipe. The second conjunct (18) requires that the output of each filter be connected to the input of a single pipe. The third conjunct (19) requires that the output of a pipe be connected to at most one filter. Finally, the fourth and fifth conjuncts (20) and (21) require that pipes only be connected to filters and vice-versa.

Notice that (17) and (18) in Example 6.5 can be simplified by introducing the additional notation for "exists unique":

$$\exists!c{:}T\big(\Phi(c)\big).\ F(c) \stackrel{def}{=} \exists c{:}T\big(\Phi(c)\big).\ F(c) \wedge \forall c'{:}T\big(c \neq c' \wedge \Phi(c')\big).\ \neg F(c')\,. \tag{22}$$

Using this notation, (17) and (18) can be rewritten, respectively, as

$$\forall f{:}F.\ \exists!p{:}P.\ \sim(f.in\ p.out) \qquad \text{and} \qquad \forall f{:}F.\ \exists!p{:}P.\ \sim(f.out\ p.in)\,.$$

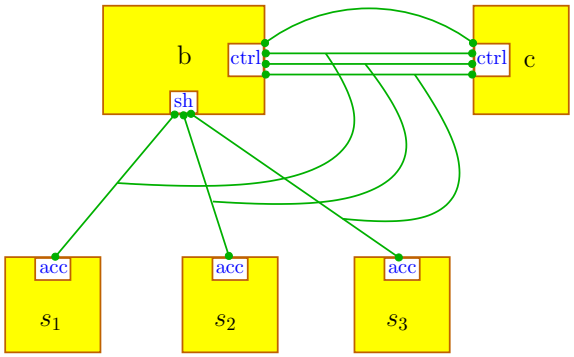


Figure 13: Blackboard architecture.

Example 6.6. In the Blackboard architecture style [14], a blackboard component of type B holds data[1] 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 13 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:

$$b.ctrl \wedge c.ctrl \wedge \sim\Big(\Sigma s{:}S.\ (s.acc\ b.sh)\Big) \\ \wedge \Big(\forall s_1 : S.\ \forall s_2 : S(s_1 \neq s_2).\ (\overline{s_1.acc\ s_2.acc}\ )\Big)\,.$$

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 involving two or more knowledge sources.

Example 6.7. The Request/Response pattern involves Clients and Services. It is defined as follows [15]:

"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."

[1] We omit the data representation in this example, since only the fact that the data is updated is relevant and not the data itself.

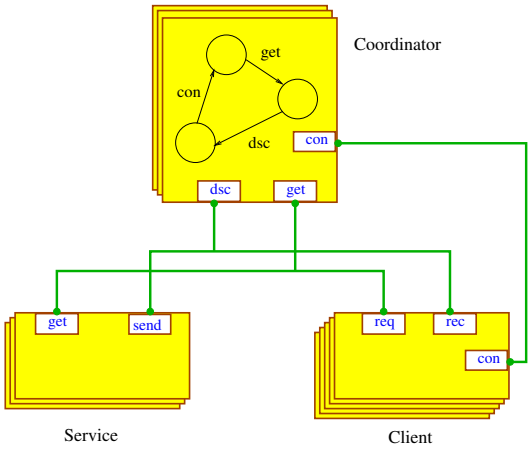


Figure 14: Request/Response architecture.

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 14.

This pattern can be specified as follows:

$$\Sigma cl{:}Cl.\ \Sigma s{:}S.\ \exists c{:}C.\ \Big(\sharp(cl.con,\ c.con) + \sharp(cl.req,\ s.get,\ c.get) + \sharp(cl.rec,\ s.send,\ c.dsc)\Big)$$
$$\wedge\ \Sigma cl{:}Cl.\ \Sigma c{:}C.\ \exists s{:}S.\ \Big(\sharp(cl.con,\ c.con) + \sharp(cl.req,\ s.get,\ c.get) + \sharp(cl.rec,\ s.send,\ c.dsc)\Big).$$

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 con-

nected to a single coordinator; and 2) the second conjunct requires that each coordinator be connected to a single service.

Example 6.8. The Repository architecture style [16] consists of a repository component r with a port p and a set of data-accessor components of type A with ports q. We provide below a list of increasingly strong properties that may be used to characterize this style:

1. The basic property *"there exists a single repository and all interactions involve it"* is specified as follows:

$$\mathit{SingleRepo} \stackrel{def}{=} \exists r\!:\!R.\ (r.p) \wedge \forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r')\,,$$

   where the subterm $\forall r\!:\!R.\ \forall r'\!:\!R.\ (r = r')$ can be expressed in the logic as $\forall r\!:\!R.\ \forall r'\!:\!R(r' \neq r).\ \mathit{false}$.

2. 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:

$$\mathit{DataAccessors} \stackrel{def}{=} \mathit{SingleRepo} \wedge \exists a\!:\!A.\ \mathit{true} \wedge \forall a\!:\!A.\ \exists r\!:\!R.\ \sim\!(r.p\ a.q)\,.$$

3. Finally, the additional property *"there are no components of other types than Repository and Data-accessor"* is enforced by the formula:

$$\mathit{DataAccessors} \wedge\ \ \forall c\!:\!U.\ (c \in C_R \sqcup c \in C_A)\,,$$

   where the subterm $\forall c : U.\ (c \in C_R \sqcup c \in C_A)$ can be expressed as $\forall c : U(c \notin C_R \wedge c \notin C_A).\ \mathit{false}$.

6.2. 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 [17], it is shown that transitive closure, necessary to specify such reachability properties, cannot be expressed in the first-order logic. This motivates the introduction of the 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 second-order configuration logic is defined by:

$$\begin{aligned} S ::= {} & \mathit{true} \mid \phi \mid \exists c\!:\!T\big(\Phi(c)\big).S \mid \Sigma c\!:\!T\big(\Phi(c)\big).S \mid S \sqcup S \mid \neg\, S \mid S + S \\ & \mid \exists C : T\big(\Psi(C)\big).S \mid \Sigma C : T\big(\Psi(C)\big).S\,, \end{aligned}$$

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:

$$\forall C{:}T\big(\Psi(C)\big).S \stackrel{def}{=} \neg\, \exists C{:}T\big(\Psi(C)\big).\neg\, S.$$

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:

$$\langle B,\gamma\rangle \models \exists C{:}T\left(\Psi(C)\right)\,.\,S\,, \qquad \text{iff} \quad \gamma \models \bigsqcup_{C'{:}T\in B\,\wedge\,\Psi(C')} S[C'/C]\,,$$

$$\langle B,\gamma\rangle \models \Sigma C{:}T\left(\Psi(C)\right)\,.\,S\,,$$
$$\text{iff} \quad \{C' : T\in B \,|\, \Psi(C')\} \neq \emptyset \;\wedge\; \gamma \models \sum_{C'{:}T\in B\,\wedge\,\Psi(C')} S[C'/C]\,,$$

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 6.9. The property that the graph, formed by components belonging to a set C and interactions among their ports, is connected can be expressed as follows:

$$\mathit{Connected}(C) \stackrel{def}{=} \forall C'{:}T(C' \subsetneq C).$$
$$\Big(\exists c'{:}T(c'\in C').\ \exists c{:}T(c\in C\setminus C').\ \sim(c.in\ c'.out)\sqcup \sim(c'.in\ c.out)\Big)\,.$$

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 6.10. The component connection graph respects the Ring architecture style (Figure 15) if the following predicate is satisfied:

$$\mathit{Connected}(U) \;\wedge\; \Sigma c{:}T.\ \exists c'{:}T(c\neq c').\ \sharp(c.in,\ c'.out)$$
$$\wedge\ \Sigma c{:}T.\ \exists c'{:}T(c\neq c').\ \sharp(c.out,\ c'.in)\,,$$

The constraint $\mathit{Connected}(U)$ 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.

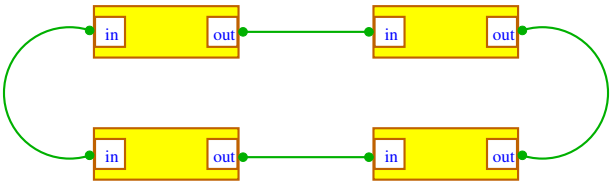


Figure 15: Ring architecture.

Figure 16: Linear architecture.

Example 6.11. The Linear architecture style, illustrated in Figure 16, 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.

$$
\begin{aligned}
&\mathit{Linear}(C, \mathit{out}, \mathit{in}) \stackrel{def}{=} \\
&\quad \mathit{Connected}(C) \ \wedge \ \exists c_1{:}T(c_1 \in C).\ \exists c_2{:}T(c_2 \neq c_1 \wedge c_2 \in C). \\
&\qquad \Big(\Sigma c{:}T(c \neq c_1 \wedge c \in C).\ \exists c'{:}T(c' \notin \{c, c_2\} \wedge c' \in C).\ \sharp(c.\mathit{in},\ c'.\mathit{out}) \\
&\qquad \wedge \ \Sigma c{:}T(c \neq c_2 \wedge c \in C).\ \exists c'{:}T(c' \notin \{c, c_1\} \wedge c' \in C).\ \sharp(c.\mathit{out},\ c'.\mathit{in})\Big)\,.
\end{aligned}
$$

Example 6.12. The Square Grid architecture style, illustrated in Figure 17, 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

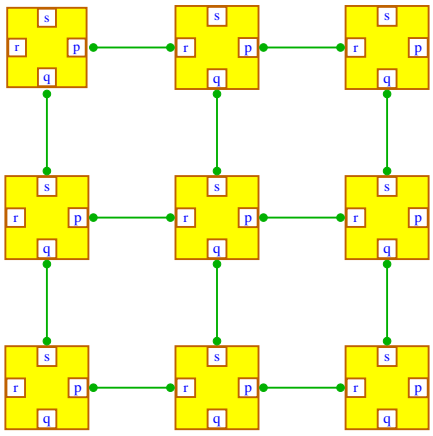


Figure 17: Grid architecture.

through ports q and s in each column. It can be expressed as follows:

$$\Big(\forall c{:}T.\quad \Big(\overline{c.p}\ \sqcup\ \exists c'{:}T(c \neq c').\, \sharp(c.p, c'.r) + \overline{c.p}\Big)$$
$$\wedge\Big(\overline{c.q}\ \sqcup\ \exists c'{:}T(c \neq c').\, \sharp(c.q, c'.s) + \overline{c.q}\Big)$$
$$\wedge\Big(\overline{c.r}\ \sqcup\ \exists c'{:}T(c \neq c').\, \sharp(c.r, c'.p) + \overline{c.r}\Big)$$
$$\wedge\Big(\overline{c.s}\ \sqcup\ \exists c'{:}T(c \neq c').\, \sharp(c.s, c'.q) + \overline{c.s}\Big)\Big)$$

$$\bigwedge\ \Big(\forall c{:}T.\, \exists C{:}T(c \in C).\, \mathit{MaxLinear}(C, p, r)$$
$$\wedge \exists C'{:}T(C' \cap C = \{c\} \wedge |C'| = |C|).\, \mathit{MaxLinear}(C', q, s)\Big)$$

$$\bigwedge\ \Big(\forall c_1 : T.\, \forall c_2 : T(c_1 \neq c_2).\, \forall c_3{:}T\big(c_3 \notin \{c_1, c_2\}\big).$$
$$\sim(c_1.p\, c_2.r + c_1.q\, c_3.s) \Rightarrow \exists c_4{:}T\big(c_4 \notin \{c_1, c_2, c_3\}\big).\ \sim(c_2.q\, c_4.s + c_3.p\, c_4.r)$$
$$\wedge\ \sim(c_1.q\, c_2.s + c_1.r\, c_3.p) \Rightarrow \exists c_4{:}T\big(c_4 \notin \{c_1, c_2, c_3\}\big).\ \sim(c_2.r\, c_4.p + c_3.q\, c_4.s)$$
$$\wedge\ \sim(c_1.r\, c_2.p + c_1.s\, c_3.q) \Rightarrow \exists c_4{:}T\big(c_4 \notin \{c_1, c_2, c_3\}\big).\ \sim(c_2.s\, c_4.q + c_3.r\, c_4.p)$$
$$\wedge\ \sim(c_1.s\, c_2.q + c_1.p\, c_3.r) \Rightarrow \exists c_4{:}T\big(c_4 \notin \{c_1, c_2, c_3\}\big).\ \sim(c_2.p\, c_4.r + c_3.s\, c_4.q)\Big)$$

$$\bigwedge\ \mathit{Connected}(U),$$

where

$$\mathit{MaxLinear}(C, p_1, p_2) \stackrel{def}{=} \mathit{Linear}(C, p_1, p_2) \wedge$$
$$\forall C'{:}T(C \subset C').\neg\, \mathit{Linear}(C', p_1, p_2).$$

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.

7. Implementation of the 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 6 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 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 6 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

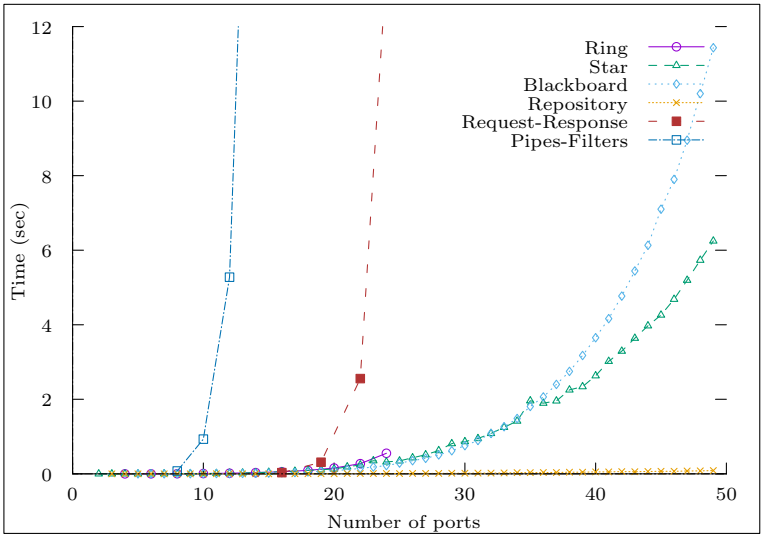


Figure 18: Decision procedure for architecture styles

model, we can apply the implementation of the procedure in Figure 8 and decide for satisfaction. The total number of lines of code of the implementation is approximately 300.

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 6) 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 18 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.

8. Related Work

An architecture style typically specifies a design vocabulary, constraints on how that vocabulary is used and semantic assumptions about that vocabulary [5]. Constraints may be about the allowed interactions between components, e.g. strong synchronization between components. Semantic assumptions concern the behaviour of the involved components, e.g. loss-less channel, server etc.

A plethora of approaches exist for characterizing architecture styles. For instance, patterns are very commonly used for this purpose. Patterns in [15, 18] incorporate explicit constructs for architecture modelling. Nonetheless, they lack formal semantics and they are not amenable to analysis.

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 (cf. the specification (16) of the Star pattern). All connections, other than the ones specified, are excluded. Most ADLs, for instance SOFA [19], Wright [20], XCD [21], 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 (cf. the specification (15) of the Star pattern). In this case specifications are conjunctions of logical formulas.

The proposed framework encompasses both approaches. It allows the description of individual interactions, e.g. by using interaction formulas. It also allows specification of configuration sets, e.g. by using formulas of the form $\sim f$.

A large body of literature, originating in [22, 7], studies the use of graph grammars and transformations [23] to define software architectures. Although this work focuses mainly on dynamic reconfiguration of architectures, e.g. [24, 8, 25], graph grammars can be used to *extensionally* define architecture styles: a style admits all the configurations that can be derived by its defining grammar. The main limitations, outlined already in [7], are the following: 1) the difficulty of understanding the architecture style defined by a grammar; 2) the fact that the restriction to context-free grammars precludes the specification of certain styles (e.g. trees with unbounded number of components or interactions, square grids); 3) the impossibility of combining several styles in a homogeneous manner. To some extent, the latter two are addressed, respectively, by considering synchronised hyperedge replacement [26], context-sensitive grammars [27, 28] and architecture views [29]. Our approach avoids these problems. Combining the extensional and intentional approaches allows intuitive specification of architecture styles. The higher-order extensions of PCL allow imposing global constraints necessary to specify styles that are not expressible by context-free graph grammars. Finally, the combination of several architecture styles is defined by the conjunction of the corresponding PCL formulas.

The proposed framework has similarities, but also differences, with approaches that use Alloy [30] (e.g. ACME [31], Darwin [32]), and with approaches that use OCL [6] (e.g. [33, 34], where OCL is used to extend UML [35]) to *intentionally* define architecture styles. Our approach achieves a strong semantic integration between architectures and architecture styles. By relying on a small set of notions, we emphasize conceptual clarity. Moreover, configuration logics allow a fine characterization of the coordination structure by using n-ary connectivity predicates with no associated behaviour. This strict separation of computation from coordination allows reasoning about the coordination constraints structurally and independently from the behaviour of coordinating components. On the contrary, the connectivity primitives in [31, 32] are binary and cannot tightly characterize 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. Similarly, all used connectivity primitives, UML associations in [34] and UML associations (between UML ports and UML connector-ends) and UML connectors in [33], are binary. UML connectors [36] may have associated behaviour [35] which is not part of the defined architecture styles in [33].

Approaches that use Alloy and OCL are limited to 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 [17]. To the best of our knowledge, the expressiveness relation between a first-order logic extended with Kleene closure and a corresponding second-order logic remains to be established.

9. Discussion

The presented work is a contribution to a long-term research program that we have been pursuing for more than 15 years. The program aims at developing the BIP component framework for rigorous systems design [37]. BIP is a language and a set of supporting tools including code generators, verification and simulation tools. So far the theoretical work has focused on the study of expressive composition frameworks and their algebraic and logical formalization. This led in particular, to the formalization of architectures as generic coordination schemes applied to sets of components in order to enforce a given global property [1].

The presented work nicely complements the existing component framework with logics for the specification of architecture styles. Configuration logic formulas characterize interaction configurations between instances of typed components. It is a powerset extension of interaction logic used to describe architectures. Configuration logic is integrated in a unified semantic framework which is equipped with a decision procedure for checking that a given architecture model meets given style requirements. Quantification over components

and sets of components allows the genericity needed for architecture styles. We have shown through examples that configuration logic allows full expressiveness combined with ease of use.

As part of the future work, we will extend the theoretical results in several directions. From the specification perspective, we are planning to incorporate hierarchically structured interactions, data transfer among the participating ports and mechanisms to express dynamic evolution of architectures. From the analysis perspective, we will study techniques for deciding satisfiability of higher-order extensions of PCL. Finally, from the practical perspective, we also plan to extend to the higher-order logics the Maude implementation of the decision procedures. We will also study sublogics that are practically relevant and for which more efficient decision procedures can be applied.

References

[1] P. Attie, E. Baranov, S. Bliudze, M. Jaber, J. Sifakis, A general framework for architecture composability, in: Proceedings of the 12th International Conference on Software Engineering and Formal Methods (SEFM 2014), no. 8702 in LNCS, Springer, 2014, pp. 128–143.

[2] ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description (2011).

[3] N. Carriero, D. Gelernter, Linda in context, Communications of the ACM 32 (4) (1989) 444–458.

[4] J. 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, IEEE, 1990, pp. 374–384.

[5] D. Garlan, Software architecture: a travelogue, in: Proceedings of the on Future of Software Engineering, ACM, 2014, pp. 29–39.

[6] J. B. Warmer, A. G. Kleppe, The Object Constraint Language: Precise Modeling With UML, Addison-Wesley, 1998.

[7] D. Le Métayer, Describing software architecture styles using graph grammars, IEEE Transactions on Software Engineering 24 (7) (1998) 521–533. doi:10.1109/32.708567.

[8] C. Koehler, A. Lazovik, F. Arbab, Connector rewriting with high-level replacement systems, Electronic Notes in Theoretical Computer Science 194 (4) (2008) 77–92.

[9] S. Bliudze, J. Sifakis, The algebra of connectors—structuring interaction in BIP, IEEE Transactions on Computers 57 (10) (2008) 1315–1330. doi:http://doi.ieeecomputersociety.org/10.1109/TC.2008.26.

[10] S. Bliudze, J. Sifakis, Causal semantics for the algebra of connectors, Formal methods in system design 36 (2) (2010) 167–194.

[11] S. Bliudze, J. Sifakis, Synthesizing glue operators from glue constraints for the construction of component-based systems, in: Software Composition, Vol. 6708 of LNCS, Springer Berlin / Heidelberg, 2011, pp. 51–67. doi:10.1007/978-3-642-22045-6_4.
URL http://dx.doi.org/10.1007/978-3-642-22045-6_4

[12] A. Mavridou, E. Baranov, S. Bliudze, J. Sifakis, Configuration Logics: Modelling architecture styles, in: C. Braga, P. C. Ölveczky (Eds.), Formal Aspects of Component Software — 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers, Vol. 9539 of Lecture Notes in Computer Science, Springer, 2015, pp. 256–274. doi:10.1007/978-3-319-28934-2_14.
URL http://dx.doi.org/10.1007/978-3-319-28934-2_14

[13] D. Garlan, M. Shaw, An introduction to software architecture, in: Advances in Software Engineering and Knowledge Engineering, World Scientific Publishing Company, 1993, pp. 1–39.

[14] D. D. Corkill, Blackboard systems, AI expert 6 (9) (1991) 40–47.

[15] R. Daigneau, Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services, Addison-Wesley, 2011.

[16] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, R. Little, Documenting software architectures: views and beyond, Pearson Education, 2002.

[17] U. 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).

[18] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[19] T. Kalibera, P. Tuma, Distributed component system based on architecture description: The Sofa experience, in: On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, Springer, 2002, pp. 981–994.

[20] R. Allen, D. Garlan, Formalizing architectural connection, in: Proceedings of the 16th international conference on Software engineering, IEEE Computer Society Press, 1994, pp. 71–80.

[21] M. Ozkaya, C. Kloukinas, Design-by-contract for reusable components and realizable architectures, in: Proceedings of the 17th international ACM Sigsoft symposium on Component-based software engineering, ACM, 2014, pp. 129–138.

[22] D. Hirsch, P. Inverardi, U. Montanari, Modeling software architectures and styles with graph grammars and constraint solving, in: P. Donohoe (Ed.), Software Architecture, Vol. 12 of IFIP, Springer, 1999, pp. 127–143. doi:10.1007/978-0-387-35563-4_8.

[23] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1997.
URL https://books.google.ch/books?id=P3r82gfRf8MC

[24] R. Bruni, A. Lluch-Lafuente, U. Montanari, E. Tuosto, Style-based architectural reconfigurations, Bulletin of the EATCS 94 (2008) 161–180.

[25] C. Krause, Z. Maraikar, A. Lazovik, F. Arbab, Modeling dynamic reconfigurations in Reo using high-level replacement systems, Sci. of Comp. Prog. 76 (1) (2011) 23–36.

[26] G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, E. Tuosto, Synchronised hyperedge replacement as a model for service oriented computing, in: Formal Methods for Components and Objects, Springer, 2006, pp. 22–43.

[27] H. Ehrig, B. Konig, Deriving bisimulation congruences in the DPO approach to graph rewriting, in: FoSSaCS, Vol. 2987 of LNCS, Springer, 2004, pp. 151–166.

[28] D.-Q. Zhang, K. Zhang, J. Cao, A context-sensitive graph grammar formalism for the specification of visual languages, The Computer Journal 44 (3) (2001) 186–200.

[29] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, ACM SIGSOFT Software Engineering Notes 17 (4) (1992) 40–52.

[30] D. Jackson, Alloy: A lightweight object modelling notation, ACM Trans. Softw. Eng. Methodol. 11 (2) (2002) 256–290. doi:10.1145/505145.505149.

[31] J. S. Kim, D. Garlan, Analyzing architectural styles, Journal of Systems and Software 83 (7) (2010) 1216–1235.

[32] I. Georgiadis, J. Magee, J. Kramer, Self-organising software architectures for distributed systems, in: Proceedings of the first workshop on Self-healing systems, ACM, 2002, pp. 33–38.

[33] U. Zdun, P. Avgeriou, A catalog of architectural primitives for modeling architectural patterns, Information and Software Technology 50 (9) (2008) 1003–1034.

[34] L. Baresi, R. Heckel, S. Thöne, D. Varró, Modeling and validation of service-oriented architectures: application vs. style, ACM SIGSOFT Software Engineering Notes 28 (5) (2003) 68–77.

[35] OMG Unified Modeling Language (OMG UML) specification, Version 2.5, http://www.omg.org/spec/UML/2.5/, (Accessed on 17/04/2016).

[36] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, J. R. Silva, Documenting component and connector views with UML 2.0, Tech. rep., DTIC Document (2004).

[37] J. Sifakis, Rigorous system design, Foundations and Trends in Electronic Design Automation 6 (2012) 293–362.