Scheduled protocol programming

Dokter, K.P.C.

Part I

Coordination

Chapter 2

Coordination Languages

Software scheduling requires detailed information on the interaction and workload of the tasks in the given application[1]. For software written in general-purpose programming languages, such information is generally not available, because the interaction protocol is implemented with basic primitives such as locks, semaphores, and (a)synchronous message passing. With such primitives, the code of the interaction protocol gets easily mixed with the application code, which renders the analysis, optimization and reusability of the implemented protocol impossible.

Exogenous coordination languages, like BIP [BBS06, BS07] and Reo [Arb04, Arb11], make the interaction protocol explicit by separating coordination of interactions from computation in processes [PA01]. This enables designers to control interaction using language constructs, making coordination visible to tools like model checkers, compilers and schedulers.

In BIP, a concurrent system consists of a superposition of three layers: behavior, interaction and priorities. The behavior layer contains the processes that need to be coordinated. The interaction layer explicitly specifies which interactions are possible, which gives full control over the interactions in the system. Mutually exclusive execution of these interactions ensures that overlapping interactions do not cause a conflict. If multiple interactions are possible, then the priority layer selects a preferred one.

In Reo, processes interact by means of a coordination protocol. A protocol consists of a graph-like structure, called a connector, that models the synchronization and dataflow among the processes. Reo connectors may compose together to form more complex connectors, allowing reusability and compositional construction of coordination protocols.

Although BIP and Reo address the same coordination problem, their underlying design principles and toolchains (containing tools for editing, code generation and model checking [bip16, reo16, Arb11]) differ significantly. By combining their principles and tools, we would conquer new terrain in the field of concurrent languages. However, some principles(visible in the formal definitions of each language) may be conflicting, and prevent such a complete unification. A formal relation between BIP and Reo is necessary to identify these conflicts.

[1]The work in this chapter is based on [DJAB17, DJAB15]

In this chapter, we provide such a formal relation between BIP and Reo by relating their semantic models. We consider two kinds of semantic models for BIP and Reo: data-agnostic and data-sensitive. In the data-agnostic domain, we relate port automata as semantics of Reo and BIP architectures [JA12, ABB+14]. We show that connectors in BIP and Reo coincide modulo internal transitions and independent progress of transitions. In the data-sensitive domain, we relate stateless constraint automata as semantics of Reo with BIP interaction models [JA12, BSBJ14]. The restriction to stateless constraint automata arises from the fact that BIP interaction models are stateless. We show that stateless constraint automata and BIP interaction models have the same observable behavior.

Stateful data-sensitive Reo connectors require stateful constraint automata for their semantics, which informally correspond to data-sensitive BIP architectures. A data-sensitive BIP architecture consists of a (data-sensitive) BIP interaction model together with a set of coordinating components. However, current literature on BIP does not provide definitions that allow composition of data-sensitive BIP architectures. Indeed, only hierarchical composition of interaction models is defined in [BSBJ14], which is insufficient to define a full composition of data-sensitive BIP architectures.

We address this problem by using our formal translations to propose a composition operator for data-sensitive BIP architectures. In addition, we show that it is possible to relate (stateful) constraint automata and data-sensitive BIP architectures.

Although BIP's notion of priority is equally applicable to the constraint automata semantics of Reo, Reo provides no syntax to specify such global priority preferences.[2] Therefore, in this chapter, "BIP" generally refers to "BI(P)", a name that others have already used to designate BIP without its priority layer.

The rest of this chapter is organized as follows: In Section 2.1, we recall the semantic models of BI(P) and Reo. In Section 2.2, we relate port automata in Reo and BIP architectures. In Section 2.3, we relate BIP interaction models with stateless constraint automata in Reo. In Section 2.4, we propose an extension of data-agnostic BIP architectures to the data-sensitive domain, and show how this enables incremental translation from stateful constraint automata to data-sensitive BIP architectures. In Section 2.5, we discuss related work. In Section 2.6, we conclude and point out future work.

2.1 Overview of BIP and Reo

2.1.1 BIP

A BIP system consists of a superposition of three layers: Behavior, Interaction, and Priority. The behavior layer encapsulates all computation, consisting of *atomic components* processing sequential code. *Ports* form the interface of a component through which it interacts with other components. BIP represents these atomic

[2] Reo does have a weaker priority mechanism to specify local preferences, called *context-sensitivity*. A premier example in the Reo literature is the context-sensitive channel LossySync, which prefers locally maximal dataflow. Clarke et al. first studied context-sensitivity through a special context-sensitive semantic model for Reo [CCA07]; later, Jongmans et al. showed how to encode context-sensitivity in non-context-sensitive models [JKA11].

components as *Labelled Transition Systems* (LTS) having transitions labelled with ports and extended with data stored in local variables. The second layer defines component coordination by means of *BIP interaction models* [BSBJ14]. For each *interaction* among components in a BIP system, the interaction model of that system specifies the set of ports synchronized by that interaction and the way data is retrieved, filtered and updated in each of the participating components. In the third layer, priorities impose scheduling constraints to resolve conflicts in case alternative interactions are possible.

In the rest of this chapter, we disregard priorities and focus mainly on interaction models (cf. footnote 2).

Data-agnostic semantics We first introduce a data-agnostic semantics for BIP.

Definition 2.1.1 (BIP component [ABB+14])**.** A *BIP component* C over a set of ports P_C is a labelled transition system $(Q, q^0, P_C, \to)$ over the alphabet 2^{P_C}. If $\mathcal{C}$ is a set of components, we say that $\mathcal{C}$ is *disconnected* iff $P_C \cap P_{C'} = \emptyset$ for all distinct $C, C' \in \mathcal{C}$. Furthermore, we define $P_{\mathcal{C}} = \bigcup_{C \in \mathcal{C}} P_C$.

Then, BIP defines an *interaction model* over a set of ports P to be a set of subsets of P. Interaction models are used to define synchronizations among components, which can be intuitively described as follows. Given a disconnected set of BIP components $\mathcal{C}$ and an interaction model γ over $P_{\mathcal{C}}$, the state space of the corresponding *composite component* $\gamma(\mathcal{C})$ is the cross product of the state spaces of the components in $\mathcal{C}$; $\gamma(\mathcal{C})$ can make a transition labelled by an interaction $N \in \gamma$ iff all the involved components (those that have ports in N) can make the corresponding transitions. A straightforward formal presentation can be found in [BS07] (cf. Definition 2.1.3 below). Thus, BIP interaction models are *stateless*: every interaction in γ is always allowed; it is enabled if all ports in the interaction are ready. However, [ABB+14] shows the need for stateful interaction, which motivates *BIP architectures*.

Definition 2.1.2 (BIP architecture [ABB+14])**.** A *BIP architecture* is a tuple $A = (\mathcal{C}, P_A, \gamma)$, where $\mathcal{C}$ is a finite disconnected set of *coordinating* BIP components, P_A is a set of ports, such that $P_{\mathcal{C}} = \bigcup_{C \in \mathcal{C}} P_C \subseteq P_A$, and $\gamma \subseteq 2^{P_A}$ is a *data-agnostic interaction model*. We call ports in $P_A \setminus P_{\mathcal{C}}$ *dangling ports* of A.

Essentially, a BIP architecture is a structured way of combining an interaction model γ with a set of distinguished components, whose only purpose is to control which interactions in γ are applicable at which point in time (which depends on the states of the coordinating components).

Definition 2.1.3 (BIP architecture application [ABB+14])**.** Let $A = (\mathcal{C}, P_A, \gamma)$ be a BIP architecture, and $\mathcal{B}$ a set of components, such that $\mathcal{B} \cup \mathcal{C}$ is finite and disconnected, and that $P_A \subseteq P_{\mathcal{B}} \cup P_{\mathcal{C}}$. Write $\mathcal{B} \cup \mathcal{C} = \{B_i \mid i \in I\}$, with $B_i = (Q_i, q_i^0, P_i, \to_i)$. Then, the *application* $A(\mathcal{B})$ *of* A *to* $\mathcal{B}$ is the BIP component $(\prod_{i \in I} Q_i, (q_i^0)_{i \in I}, P_{\mathcal{B}} \cup P_{\mathcal{C}}, \to)$, where $\to$ is the smallest relation satisfying: $(q_i)_{i \in I} \xrightarrow{N} (q_i')_{i \in I}$ whenever

1. $N = \emptyset$, and there exists an $i \in I$ such that $q_i \xrightarrow{\emptyset}_i q_i'$ and $q_j' = q_j$ for all $j \in I \setminus \{i\}$; or

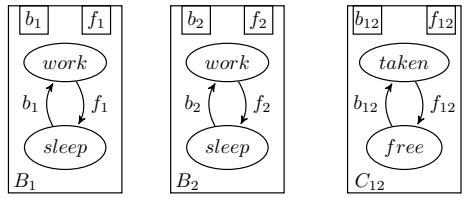


Figure 2.1: BIP components (a); the coordinating component (b) of the BIP architecture A_{12}.

2. $N \cap P_A \in \gamma$, and for all $i \in I$ we have $N \cap P_i \neq \emptyset$ implies $q_i \xrightarrow{N \cap P_i}_i q_i'$, and $N \cap P_i = \emptyset$ implies $q_i' = q_i$.

The application $A(\mathcal{B})$, of a BIP architecture A to a set of BIP components $\mathcal{B}$, enforces coordination constraints specified by that architecture on those components [ABB+14]. The *interface* P_A of A contains all ports $P_{\mathcal{C}}$ of the coordinating components $\mathcal{C}$ and some additional ports, which must belong to the components in $\mathcal{B}$. In the application $A(\mathcal{B})$, the ports belonging to P_A can participate only in interactions defined by the interaction model γ of A. Ports that do not belong to P_A are not restricted and can participate in any interaction.

Intuitively, an architecture can also be viewed as an incomplete system: the application of an architecture consists in "attaching" its dangling ports to the operand components. The operational semantics is that of composing all components (operands and coordinators) with the interaction model as described in the previous paragraph. The intuition behind transitions labelled by $\emptyset$ is that they represent *observable idling* (as opposed to internal transitions). This allows us to "desynchronize" combined architectures (see Definition 2.1.4) in a simple manner, since coordinators of one architecture can idle, while those of another performs a transition. Note that, if $N = \emptyset$, in item 2 of Definition 2.1.3, $N \cap P_i = \emptyset$, hence also, $q_i' = q_i$, for all i. Thus, intuitively, one can say that none of the components moves. Item 1, however, does allow one component to make a real move labelled by $\emptyset$, if such a move exists. Thus, the transitions labelled by $\emptyset$ interleave, reflecting the idea that in BIP synchronization can happen only through ports.

Example 2.1.1 (Mutual exclusion [ABB+14])**.** Consider the components B_1 and B_2 in Figure 2.1(a). In order to ensure mutual exclusion of their `work` states, we apply the BIP architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$ with C_{12} from Figure 2.1(b), $P_{12} = \{b_1, b_2, b_{12}, f_1, f_2, f_{12}\}$ and $\gamma_{12} = \big\{\emptyset, \{b_1, b_{12}\}, \{b_2, b_{12}\}, \{f_1, f_{12}\}, \{f_2, f_{12}\}\big\}$. The interface P_{12} of A_{12} covers all ports of B_1, B_2 and C_{12}. Hence, the only possible interactions are those that explicitly belong to γ_{12}. Assuming that the initial states of B_1 and B_2 are `sleep`, and that of C_{12} is `free`, neither of the two states (`free`, `work`, `work`) and (`taken`, `work`, `work`) is reachable, i.e. the mutual exclusion property $(q_1 \neq \texttt{work}) \vee (q_2 \neq \texttt{work})$—where q_1 and q_2 are state variables of B_1 and B_2 respectively—holds in $A_{12}(B_1, B_2)$. ◊

Definition 2.1.4 (Composition of BIP architectures [ABB+14])**.** Let $A_1 = (\mathcal{C}_1, P_1, \gamma_1)$ and $A_2 = (\mathcal{C}_2, P_2, \gamma_2)$ be two BIP architectures. Recall that $P_{\mathcal{C}_i} = \bigcup_{C \in \mathcal{C}_i} P_C$, for

$i = 1, 2$. If $P_{\mathcal{C}_1} \cap P_{\mathcal{C}_2} = \emptyset$, then $A_1 \oplus A_2$ is given by $(\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_{12})$, where $\gamma_{12} = \{N \subseteq P_1 \cup P_2 \mid N \cap P_i \in \gamma_i, \text{ for } i = 1, 2\}$. In other words, γ_{12} is the interaction model defined by the conjunction of the characteristic predicates of γ_1 and γ_2.

Data-sensitive semantics Recently, the data-agnostic formalization of BIP interaction models was extended with data transfer, using the notion of *interaction expressions* [BSBJ14].

Let $\mathcal{P}$ be a global set of ports. For each port $p \in \mathcal{P}$, let $x_p : \mathsf{D}_p$ be a typed variable used for the data exchange at that port. For a set of ports $P \subseteq \mathcal{P}$, let $X_P = (x_p)_{p \in P}$. An interaction expression models the effect of an interaction among ports in terms of the data exchanged through their corresponding variables.

Definition 2.1.5 (Interaction expression [BSBJ14])**.** An *interaction expression* is an expression of the form

$$(P \leftarrow Q).[g(X_Q, X_L) : \ (X_P, X_L) := up(X_Q, X_L) \,//\, (X_Q, X_L) := dn(X_P, X_L)]$$

where $P, Q \subseteq \mathcal{P}$ are *top* and *bottom* sets of ports; $L \subseteq \mathcal{P}$ is a set of *local* variables; $g(X_Q, X_L)$ is the boolean *guard*; $up(X_Q, X_L)$ and $dn(X_P, X_L)$ are respectively the *up-* and *downward data transfer* expressions.

For an interaction expression α as above, we define by $top(\alpha) = P$, $bot(\alpha) = Q$ and $supp(\alpha) = P \cup Q$ the sets of top, bottom and all ports in α, respectively. We denote g_α, up_α and dn_α the guard, upward and downward transfer corresponding expressions in α.

The first part of an interaction expression, $(P \leftarrow Q)$, describes the control flow as a dependency relation between the bottom and the top ports. The expression in the brackets describes the data flow, first "upward"—from bottom to top ports—and then "downward". The guard $g(X_Q, X_L)$ relates these two parts: interaction is enabled only when the values of the local variables together with those of variables associated to the bottom ports satisfy a boolean condition. As a side effect, an interaction expression may also modify local variables in X_L. Intuitively, such an interaction expression can *fire* only if its guard is true. When it fires, its upstream transfer is computed first using the values offered by its participating BIP components. Then, the downstream transfer modifies all of its port variables with updated values. These upstream and downstream data transfers execute atomically, which means that an interaction expression behaves as a stateless connector.

Definition 2.1.6 (BIP interaction models [BSBJ14])**.** A *(data-sensitive) BIP interaction model* is a set Γ of *simple BIP connectors* α that are BIP interaction expressions of the form

$$(\{w\} \leftarrow A).[g(X_A) : \ (x_w, X_L) := up(X_A) \,//\, X_A := dn(x_w, X_L)],$$

where $w \in P$ is a single top port, $A \subseteq P$ is a set of ports, such that $w \notin A$, and neither *up* nor g involves local variables.

Example 2.1.2 (Maximum)**.** Let $\mathcal{P} = \{a, b, w, l\}$ be a set of ports of type integer, i.e., $x_p : \mathsf{D}_p = \mathbb{Z}$, for all $p \in \mathcal{P}$, and consider the interaction expression (simple BIP connector)

$$\alpha_{\max} = (\{w\} \leftarrow \{a, b\}).[\mathtt{tt} : \ x_l := \max(x_a, x_b) \,//\, x_a, x_b := x_l],$$

where $\mathtt{tt}$ is true. First, the connector takes the values presented at ports a and b. Then, the simple BIP connector $\alpha_{\max}$ computes atomically the maximum of x_a and x_b and assigns it to its local variable x_l. Finally, $\alpha_{\max}$ assigns atomically the value of x_l to both x_a and x_b. ◊

BIP interaction expressions capture complete information about all aspects of component interaction—i.e., synchronization and data transfer possibilities—in a structured and concise manner. Thus, by examining interaction expressions, one can easily understand, on the one hand, the interaction model used to compose components and, on the other hand, how the valuations of data variables affect the enabledness of the interactions and how these valuations are modified. Furthermore, a formal definition of a composition operator on interaction expressions is provided in [BSBJ14], which allows combining such expressions hierarchically to manage the complexity of systems under design. Since any BIP system can be flattened, this hierarchical composition of interaction expressions is not relevant for the semantic comparison of BIP and Reo in this chapter. Nevertheless, the possibility of concisely capturing all aspects of component interaction in one place is rather convenient.

2.1.2 Reo

We briefly recall the basics of the Reo language and refer to [Arb04] and [Arb11] for further details. Reo is a coordination language wherein graph like structures express concurrency constraints (e.g., synchronization, exclusion, ordering, etc.) among multiple components. A Reo program, called a *connector*, is a graph-like structure whose edges consist of *channels* that enable synchronous and asynchronous data flow and whose vertices consist of *nodes* that synchronously route data among multiple channels.

A channel in Reo has exactly two *ends*, and each end either accepts data items, if it is a *source end*, or offers data items, if it is a *sink end*. The *type* of a channel is a formal constraint on the dataflow through its two ends that completely defines the behavior of the channel. Beside the established channel types (Figure 2.2 contains some of them) Reo allows arbitrary user-defined channel types.

Reo is agnostic regarding the semantics that expresses the behavior of its channel types, so long as the semantics preserves Reo's compositional construction principle (i.e., the behavior of a connector is computed by composing the behaviors of all channels and nodes). Jongmans [JA12] provides an overview of thirty alternative semantics for Reo channels. Its abstract definition of channels and its notion of channel types make Reo an extensible programming language.

Multiple ends may glue together into *nodes* with a fixed *merge-replicate* behavior: a data item out of a single sink end coincident on a node, atomically propagates to all source ends coincident on that node. This propagation happens only if all their respective channels allow the data exchange. A node is called a *source node*

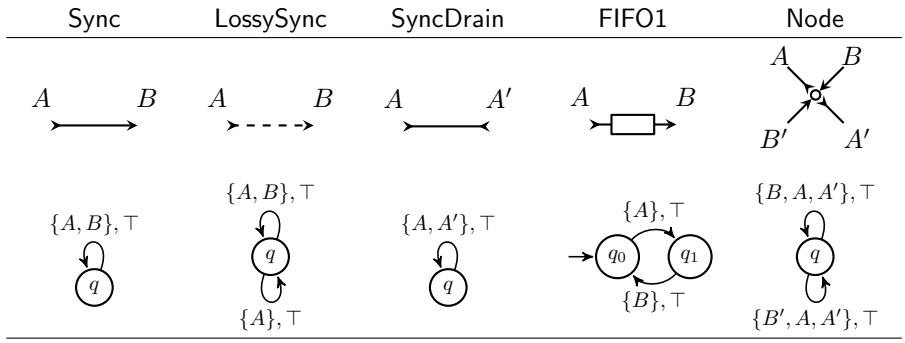


Figure 2.2: Some primitives in the Reo language with CA semantics over a singleton data domain $\mathcal{D}$.

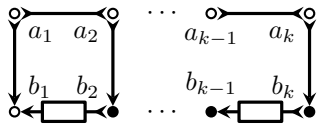


Figure 2.3: Construction of the $\mathsf{Alternator}_k$ Reo connector, for $k \geq 2$.

if it consists of source ends, a *sink node* if it consists of sink ends, and a *mixed node* otherwise. Together, the source and sink nodes of a connector constitute its set of *boundary nodes*.

Example 2.1.3 (Primitive channels)**.** Figure 2.2 shows some typical primitive Reo channels and an example of how these channels can compose at nodes.

A Sync channel accepts a datum from its source end A, when its simultaneous offer of this datum at its sink end B succeeds.

A SyncDrain channel simultaneously accepts a datum from both its source ends A and B and loses this datum.

An empty FIFO_1 accepts data from its source end A and becomes a full FIFO_1. A full FIFO_1 offers its stored data at its sink end B and, when its offer succeeds, it becomes an empty FIFO_1 again.

A Reo node accepts a datum from one of its coincident sink ends (B or B'), when its simultaneous offer to dispense a copy of this datum through every one of its coincident source ends (A and A') succeeds. ◊

The key concept in Reo is composition, which allows a programmer to build complex connectors out of simpler ones.

Example 2.1.4 (Alternator)**.** Using the channels in Figure 2.2, we can construct the $\mathsf{Alternator}_k$ connector, for $k \geq 2$, as shown in Figure 2.3. For $k = 2$, the $\mathsf{Alternator}_2$ consists of four nodes (a_1, a_2, b_1, and b_2) and four channels, namely a SyncDrain channel (between a_1 and a_2), two Sync channels (from a_1 to b_1, and from a_2 to b_2), and a FIFO_1 channel (from b_2 to b_1).

The behavior of the $\mathsf{Alternator}_2$ connector is as follows. Suppose that the environment is ready to offer a datum at each of the nodes a_1 and a_2, and ready

to accept a datum from node b_1. According to Example 2.1.3, nodes a_1 and a_2 both offer a copy of their received datum to the SyncDrain channel. The SyncDrain channel ensures that nodes a_1 and a_2 accept data from the environment only simultaneously. The Sync channel from a_1 to b_1 ensures that node b_1 simultaneously obtains a copy of the datum offered at a_1. By definition, node b_1 either accepts a datum from the connected Sync channel or it accepts a datum from the FIFO_1 channel (but not from both simultaneously), and offers this datum immediately to the environment. Because the FIFO_1 is initially empty, b_1 has no choice but to accept and dispense the datum from a_1. Simultaneously, the Sync channel from a_2 to b_2 ensures that the value offered at a_2 is stored in the FIFO_1 buffer. In the next step, the environment at node b_1 has no choice but to retrieve the datum in the buffer, after which the behavior repeats. ◊

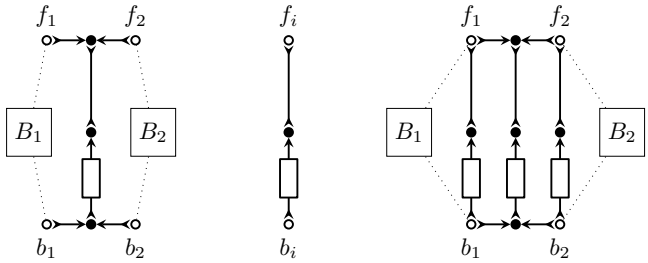
Example 2.1.5. Figure 2.4(a) shows a Reo connector that achieves mutual exclusion of components B_1 and B_2, exactly as the BIP system shown in Figure 2.1 does. This connector consists of a composition of channels and nodes in Figure 2.2. The Reo connector atomically accepts data from either b_1 or b_2 and puts it into the FIFO_1 channel, a buffer of size one. A full FIFO_1 channel means that B_1 or B_2 holds the lock. If one of the components writes to f_1 or f_2, the SyncDrain channel flushes the buffer, and the lock is released, returning the connector to its initial configuration, where B_1 and B_2 can again compete for exclusive access by attempting to write to b_1 or b_2.

The connector in Figure 2.4(a) is not fool-proof. Even if B_1 takes the lock, B_2 may release it, and vice versa. Hence, exactly as the BIP architecture in Figure 2.1, the Reo connector in Figure 2.4(a) relies on the conformance of the coordinated components B_1 and B_2. The expected behavior of B_i, $i = 1, 2$, is that it alternates writes on the b_i and f_i, and that every write on f_i comes after a write on b_i. Depending on such assumptions may not be ideal. The connector, shown in Figure 2.4(b), makes this expected behavior explicit. By composing two such connectors with the connector in Figure 2.4(a), we obtain a fool-proof mutual exclusion protocol, as shown in Figure 2.4(c). Figure 2.6(c) shows the constraint automaton semantics of the connector in Figure 2.4(c). Like the case of the connector in Figure 2.4(a) or the BIP architecture in Figure 2.1, noncompliant writes to b_i or f_i nodes of the connector in Figure 2.4(c) will *block* a renegade component B_i that attempts such writes. However, contrary to the case of the connector in Figure 2.4(a) or the BIP architecture in Figure 2.1, such a renegade component cannot *break* the mutual exclusion protocol that the connector in Figure 2.4(c) implements, as it allows the other component to run undisturbed. ◊

Formal semantics of Reo Reo has a variety of formal semantics [Arb11, JA12]. In this chapter we use its operational *constraint automaton* (CA) semantics [BSAR06].

Definition 2.1.7 (Constraint automata [BSAR06])**.** Let $\mathcal{N}$ be a set of ports and $\mathcal{D}$ a set of data items. A data constraint is a first-order formula g with constants $v \in \mathcal{D}$ and variables d_p, for $p \in \mathcal{N}$, that represent the datum observed at (i.e., exchanged through) port p. More formally, g is defined by the grammar

$$g \ ::= \ \top \mid \neg g \mid g \wedge g \mid \exists d_p(g) \mid d_p = v, \quad \text{with } p \in \mathcal{N}, v \in \mathcal{D},$$



(a) BIP-like mutex (b) (c) Fool-proof mutex

Figure 2.4: Fool-proof (c) mutual exclusion protocol in Reo, composed from a BIP-like (a) mutual exclusion connector and an alternator connector (b).

where $\top$, $\neg$, $\wedge$, $\exists$ and $=$ are respectively tautology, negation, conjunction, existential quantification and equality. Write $DC(\mathcal{N}, \mathcal{D})$ for the set of all data constraints over $\mathcal{N}$, and let $\models$ denote the usual satisfaction relation between data assignments $\delta : N \to \mathcal{D}$, with $N \subseteq \mathcal{N}$, and data constraints $g \in DC(\mathcal{N}, \mathcal{D})$. A constraint automaton (over data domain $\mathcal{D}$) is a tuple $\mathcal{A} = (Q, \mathcal{N}, \to, q_0)$ where Q is a set of states, $\mathcal{N}$ is a finite set of ports, $q_0 \in Q$ is the initial state, and $\to \subseteq Q \times 2^{\mathcal{N}} \times DC(\mathcal{N}, \mathcal{D}) \times Q$ is a transition relation, such that, for any transition $q \xrightarrow{N,g} q'$, we have $g \in DC(N, \mathcal{D})$.[3]

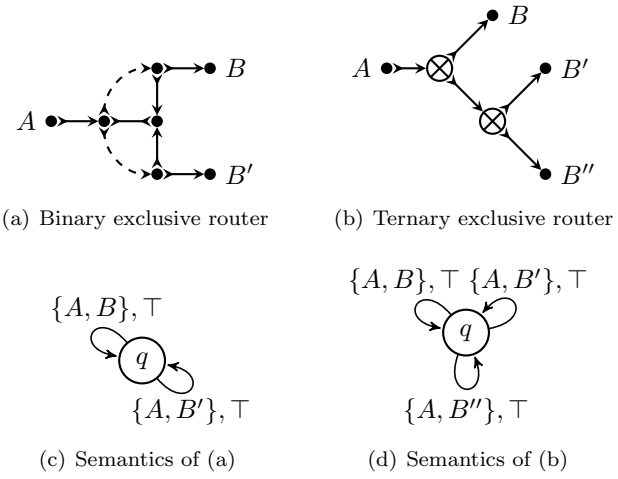
If a constraint automaton $\mathcal{A}$ has only one state, $\mathcal{A}$ is called *stateless*. If the data domain $\mathcal{D}$ of $\mathcal{A}$ is a singleton, $\mathcal{A}$ is called a *port automaton* [KC09]. In that case, we omit data constraints, because all satisfiable constraints reduce to $\top$.

In this chapter, we consider only finite data domains, although most of our results generalize to infinite data domains. Over a finite data domain, the data constraint language $DC(\mathcal{N}, \mathcal{D})$ is expressive enough to define any data assignment. For notational convenience, we relax, in this chapter, the definition of data constraints and allow the use of set-membership and functions in the data constraints (compare the definition of $g(\alpha)$ in Section 2.3.3). However, we preserve the intention that a data constraint describes a set of data assignments.

Example 2.1.6 (CA semantics of Reo primitives)**.** Figure 2.2 shows the CA semantics of some typical Reo primitives. Since constraint automata do not model the direction of dataflow, the CA semantics of Sync and SyncDrain coincides. $\Diamond$

Example 2.1.7 (Exclusive router)**.** The fixed merge-replicate behavior of a Reo node propagates an input datum to all of its output ports (i.e., source ends coincident on that node). An *exclusive router* is a connector that propagates an input datum to one of its, non-deterministically selected, output ports. Figure 2.5(a) shows the construction of a binary exclusive router from the primitive channels Sync, SyncDrain, and LossySync. Figure 2.5(b) shows the construction of a ternary

[3]The original definition of constraint automata excludes internal transitions with $\emptyset, \top$ labels [BSAR06]. If necessary, all internal transitions may be removed modulo (weak) language equivalence of constraint automata by merging any state q with every state q' that is reachable from q by a sequence of internal transitions.



(a) Binary exclusive router (b) Ternary exclusive router

(c) Semantics of (a) (d) Semantics of (b)

Figure 2.5: Construction of a binary exclusive router (a); construction of a ternary exclusive router (b) from binary exclusive routers; and the CA semantics (c) and (d) of the exclusive routers in (a) and (b), respectively.

exclusive router by composing two binary exclusive routers, where we abbreviate a binary exclusive router as a crossed node. Figures 2.5(c) and 2.5(c) show the CA semantics of the binary and ternary exclusive router, respectively. ◊

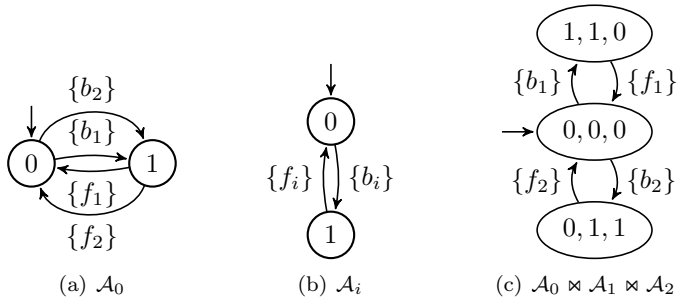
The CA semantics of every Reo connector can be derived as a composition of the constraint automata of its primitives, using the CA product operation in Definition 2.1.8.

The CA semantics for Reo connectors assigns a constraint automaton to every Reo connector. In the other direction, Baier et al. have shown that it is possible to translate every constraint automaton (over a finite data domain) back into a Reo connector [BKK14]. For example, Figure 2.8(c) shows the Reo connector that is generated from the constraint automaton $\mathsf{reo}_1(A_{12})$ in Figure 2.8(b). We refer to Example 2.2.1 for more details. Because of this correspondence, we consider Reo and CA as equivalent and focus on constraint automata only.

Definition 2.1.8 (Product of CA [BSAR06])**.** Let $\mathcal{A}_i = (Q_i, \mathcal{N}_i, \rightarrow_i, q_{0,i})$ be a constraint automaton, for $i = 1, 2$. Then the product $\mathcal{A}_1 \bowtie \mathcal{A}_2$ of these automata is the automaton $(Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \rightarrow, (q_{0,1}, q_{0,2}))$, whose transition relation is the smallest relation obtained by the rule: $(q_1, q_2) \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} (q_1', q_2')$ whenever

1. $q_1 \xrightarrow{N_1, g_1}_1 q_1'$, $q_2 \xrightarrow{N_2, g_2}_2 q_2'$, and $N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1$, or
2. $q_i \xrightarrow{N_i, g_i}_i q_i'$, $N_j = \emptyset$, $g_j = \top$, $q_j' = q_j$, and $N_i \cap \mathcal{N}_j = \emptyset$ with $j \in \{1, 2\} \setminus \{i\}$.

It is not hard to see that constraint automata product operator is associative and commutative modulo equivalence of state names and data constraints (e.g., $d_p = v \wedge d_q = w$ is equivalent to $d_q = w \wedge d_p = v$, for $p, q \in \mathcal{N}$ and $v, w \in \mathcal{D}$).



(a) $\mathcal{A}_0$ (b) $\mathcal{A}_i$ (c) $\mathcal{A}_0 \bowtie \mathcal{A}_1 \bowtie \mathcal{A}_2$

Figure 2.6: CA semantics (a), (b), and (c) of Reo connectors in Figures 2.4(a), 2.4(b), and 2.4(c), respectively.

Definition 2.1.9 (Hiding in CA [BSAR06])**.** Let $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0)$ be a constraint automaton, and $P = \{p_1, \ldots, p_n\}$ a set of ports. Then, hiding ports P of $\mathcal{A}$ yields an automaton $\exists P(\mathcal{A}) = (Q, \mathcal{N} \setminus P, \rightarrow_\exists, q_0)$, where $\rightarrow_\exists$ is given by $\{(q, N \setminus P, \exists d_{p_1} \cdots \exists d_{p_n}(g), q') \mid (q, N, g, q') \in \rightarrow\}$.

In addition to removing ports in P from the transition labels, the original definition of hiding merges any two states that become reachable by a sequence of internal $\emptyset$-labelled transitions (Definition 4.3 in [BSAR06] and Footnote 3). Since we allow these internal transitions, we do not bother to remove the internal transitions produced by the hiding operation in Definition 2.1.9. A constraint automaton obtained using our hiding operator is (weak) language equivalent to a constraint automaton obtained using the original hiding operator of [BSAR06].

As hiding of non-shared ports distributes over product, hiding of non-shared ports commutes with constraint automata product.

Example 2.1.8. Figures 2.6(a) and 2.6(b) show the constraint automaton semantics $\mathcal{A}_0$ and $\mathcal{A}_i$, for $i \in \{1, 2\}$, of the Reo connectors in Figures 2.4(a) and (two copies of) 2.4(b). Example 2.1.5 indicates that the fool-proof mutual exclusion protocol in Figure 2.4(c) can be obtain by composing the Reo connectors in Figures 2.6(a) and 2.6(b). Indeed, the constraint automaton semantics of the fool-proof mutual exclusion protocol in Figure 2.4(c) is given by $\mathcal{A} = \mathcal{A}_0 \bowtie \mathcal{A}_1 \bowtie \mathcal{A}_2$. The part of $\mathcal{A}$ that is reachable from initial state $(0, 0, 0)$ is shown in Figure 2.6(c). $\Diamond$

2.2 Port automata and BIP architectures

To study the relation between BIP and Reo with respect to synchronization, we start by defining a correspondence between them in the data-agnostic domain. This correspondence consists of a pair of mappings between the sets containing semantic models of BIP and Reo connectors. For the data independent semantic model of Reo connectors we choose port automata: a restriction of constraint automata over a singleton set as data domain. We model BIP connectors by BIP architectures introduced in [ABB+14]. In order to compare the behavior of BIP and Reo connectors we interpret them as labelled transition systems. We define a mapping reo_1 that transforms BIP architectures into port automata, and a mapping bip_1

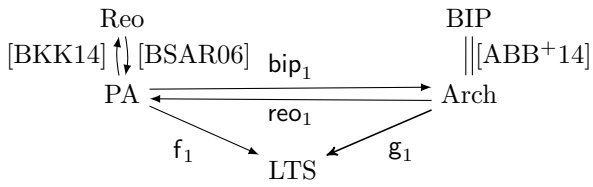


Figure 2.7: Translations and interpretations in the data-agnostic domain.

that transforms port automata into BIP architectures. We then show that these mappings preserve(1) properties closed under bisimulation, and (2) composition structure modulo semantic equivalence.

2.2.1 Interpretation of BIP and Reo

To compare the behavior of BIP and Reo connectors, we interpret all connectors as labelled transitions systems with one initial state and an alphabet 2^P, for a set of ports P. We write LTS for the class of all such labelled transition systems.

Figure 2.7 shows our translations and interpretations. The objects PA and Arch are the classes of port automata and BIP architectures, respectively. The mappings bip_1, reo_1, f_1 and g_1, respectively, translate Reo to BIP, BIP to Reo, Reo to LTS, and BIP to LTS.

We first consider the semantics of connectors in Reo and BIP. Since BIP connectors differ internally from Reo connectors, we restrict our interpretation to their observable behavior. This means that we hide the ports of the coordinating components in BIP architectures. For port automata this means that for our comparison, we implicitly assume that all ports correspond to boundary nodes only.

Interpretation of PA We define the interpretation of a port automaton as

$$\mathsf{f}_1((Q, \mathcal{N}, \rightarrow, q_0)) = (Q, 2^{\mathcal{N}}, \rightarrow, q_0). \tag{2.1}$$

Hence f_1 acts essentially as an identity function, justifying our choice of interpretation.

Interpretation of Arch We define the interpretation of BIP architectures using their operational semantics obtained by applying them on dummy components and hiding all internal ports. Let $A = (\mathcal{C}, P, \gamma)$ be a BIP architecture with coordinating components $\mathcal{C} = \{C_1, \ldots, C_n\}$, $n \geq 0$, and $C_i = (Q_i, q_i^0, P_i, \rightarrow_i)$. Recall that $P_{\mathcal{C}} = \bigcup_i P_i$ is the set of internal ports in A. Define $D = (\{q_D\}, q_D, P, \{(q_D, N, q_D) \mid \emptyset \neq N \subseteq P \setminus P_{\mathcal{C}}\})$ as a dummy component relative to the BIP architecture A. Using Definition 2.1.3, we compute the BIP architecture application $A(\{D\}) = ((\prod_{i=1}^n Q_i) \times \{q_D\}, (\mathbf{q}^0, q_D), P, \rightarrow_s)$ of A to its dummy component D. Then,

$$\mathsf{g}_1(A) = ((\textstyle\prod_{i=1}^n Q_i) \times \{q_D\}, 2^{P \setminus P_{\mathcal{C}}}, \rightarrow, (\mathbf{q}^0, q_D)) \tag{2.2}$$

where $\rightarrow = \{((\mathbf{q}, q_D), N \setminus P_{\mathcal{C}}, (\mathbf{q}', q_D)) \mid (\mathbf{q}, q_D) \xrightarrow{N}_s (\mathbf{q}', q_D)\}$. In other words, $\mathsf{g}_1(A)$ equals $A(\{D\})$ after hiding all internal ports $P_{\mathcal{C}}$.

Note that we based our interpretation g_1 on the operational semantics of BIP architectures, i.e., BIP architecture application. This justifies the definition of interpretation of architectures.

With a common semantics for BIP and Reo, we can define the notion of preservation of properties expressible in this common semantics. Recall that a property of labelled transition systems corresponds to the subset of labelled transition systems satisfying that property.

Definition 2.2.1. Let $P \subseteq \text{LTS}$ be a property. Then, bip_1 *preserves* P iff $\mathsf{f}_1(\mathcal{A}) \in P \Leftrightarrow \mathsf{g}_1(\mathsf{bip}_1(\mathcal{A})) \in P$ for all $\mathcal{A} \in \text{PA}$. Similarly, reo_1 *preserves* P iff $\mathsf{g}_1(A) \in P \Leftrightarrow \mathsf{f}_1(\mathsf{reo}_1(A)) \in P$ for all $A \in \text{Arch}$.

2.2.2 BIP to Reo

To translate BIP connectors to Reo connectors, we first determine what elements of BIP architectures correspond to Reo connectors. Our interpretations of port automata and BIP architectures show that dangling ports in BIP architectures correspond to boundary port names in port automata. Furthermore, the mutual exclusion of the interactions in an interaction model in a BIP architecture simulates mutually exclusive firing of transitions in port automata. The definition of a coordinating component in a BIP architecture is almost identical to that of a port automaton, yielding an obvious translation.

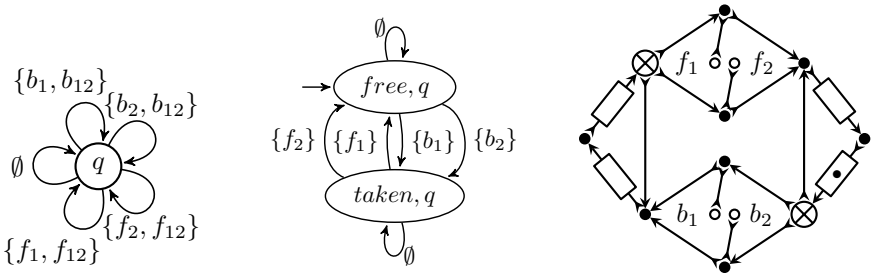
Let $A = (\mathcal{C}, P, \gamma)$ be a BIP architecture, with $\mathcal{C} = \{C_1, \ldots, C_n\}$. Each C_i corresponds trivially to a port automaton C_i^*. Let $\mathcal{A}_\gamma = (\{q\}, P, \to, q)$ be the stateless port automaton over P with transition relation $\to$ defined by $\{(q, N, q) \mid N \in \gamma\}$. Then $\mathcal{A}_\gamma$ can be seen as the port automata encoding of the interaction model γ. Recall that $P_\mathcal{C} = \bigcup_{C \in \mathcal{C}} P_C$. The corresponding port automaton of A is given by

$$\mathsf{reo}_1(A) = \exists P_\mathcal{C}(C_1^* \bowtie \cdots C_n^* \bowtie \mathcal{A}_\gamma). \tag{2.3}$$

Example 2.2.1. We translate the BIP architecture $A_{12} = (\{C_{12}\}, P_{12}, \gamma_{12})$ from Example 2.1.1 using reo_1 defined in Equation (2.3). First, we transform γ_{12} into a port automaton $\mathcal{A}_{\gamma_{12}}$, which is shown in Figure 2.8(a). Then, interpret the coordinating component C_{12} as a port automaton C_{12}^*. Finally, we compute the product of $\mathcal{A}_{\gamma_{12}}$ with the coordinating component C_{12}^* and hide the ports $\{b_{12}, f_{12}\}$ of C_{12}. Figure 2.8(b) shows the resulting port automaton.

As mentioned in Section 2.1.2, we can transform the port automaton in Figure 2.8(b) into a Reo connector, using the method described in [BKK14]. This mechanical translation yields the Reo connector in Figure 2.8(c)[4]. Intuitively, each state is represented by a FIFO buffer, and the current state is indicated by the presence of a token. A transition is represented by synchronous channels that move the token from one buffer to another. The transition is selected by an ternary exclusive router, represented as a crossed node (cf. Example 2.1.7). Note that the port automaton semantics of the connector in Figure 2.4(a) (see Figure 2.6(a)) is similar to the automaton in Figure 2.8(b), up to empty transitions. ◊

[4]For simplicity, we use two FIFO_1 buffers instead of simultaneous FIFO_1 buffers used in [BKK14].



(a) $\mathcal{A}_{\gamma_{12}}$ (b) $\mathsf{reo}_1(A_{12})$ (c) Generated mutex

Figure 2.8: Translation of the interaction model γ_{12} (a) and BIP architecture A_{12} (b) from Figure 2.1, and the Reo connector (c) generated from $\mathsf{reo}_1(A_{12})$.

2.2.3 Reo to BIP

In BIP, interaction is memoryless. This means that a stateful channel in Reo must translate to a coordinating component. In fact, we may encode an entire generic Reo connector as one such component.

The most natural way to translate a port automaton $\mathcal{A}$ into a BIP architecture A is by interpreting $\mathcal{A}$ as the coordinating component of A. However, BIP requires atomic components to synchronize via interactions, rather than directly on shared ports. Indeed, a BIP architecture excludes any two coordinating components to share a port (see Definition 2.1.2).

Since we want a compositional translation of port automata to BIP architectures, we need to interpret each port $p \in \mathcal{N}$ in the interface of $\mathcal{A}$ as a dangling port of A (see Definition 2.1.2). To this end, we rename every port $p \in \mathcal{N}$ in the interface of $\mathcal{A}$ to p', and synchronize p and p' by means of a BIP interaction.

Let $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0)$ be a port automaton. We construct a corresponding BIP architecture for $\mathcal{A}$. Duplicate all ports in $\mathcal{N}$ by defining $\mathcal{N}' = \{n' \mid n \in \mathcal{N}\}$. We do not use a port n', for $n \in \mathcal{N}$, for composition with other BIP architectures. Therefore, the exact names of ports in an $\mathcal{N}'$ are not important, instead only their relation to their dangling siblings $n \in \mathcal{N}$ matters. For every $N \subseteq \mathcal{N}$, define $N' = \{n' \in \mathcal{N} \mid n \in N\}$. Trivially, $\overline{\mathcal{A}} = (Q, q_0, \mathcal{N}', \rightarrow_c)$, with $\rightarrow_c = \{(q, N', q') \mid (q, N, q') \in \rightarrow\}$, is a BIP component (cf. Definition 2.1.1). Essentially, $\mathcal{A}$ and $\overline{\mathcal{A}}$ are the same labelled transition system. Now we define bip_1 as follows:

$$\mathsf{bip}_1(\mathcal{A}) = (\{\overline{\mathcal{A}}\}, \mathcal{N} \cup \mathcal{N}', \{N \cup N' \mid N \subseteq \mathcal{N}\}). \tag{2.4}$$

Thus, bip_1 uses the port automaton as the coordinating component of the generated BIP architecture.

Example 2.2.2. We determine $\mathsf{bip}_1(\mathcal{A})$, where $\mathcal{A}$ is the port automaton in Figure 2.6(b) over the name set $\mathcal{N} = \{b_i, f_i\}$. Obtain $\overline{\mathcal{A}}$ by adding a prime to each port in $\mathcal{A}$. The interaction model of $\mathsf{bip}_1(\mathcal{A})$ consists of $\{N \cup N' \mid N \subseteq \mathcal{N}\} = \big\{\emptyset, \{b_i, b_i'\}, \{f_i, f_i'\}, \{b_i, b_i', f_i, f_i'\}\big\}$. Hence, $\mathsf{bip}_1(\mathcal{A})$ is given by the BIP architecture $(\{\overline{\mathcal{A}}\}, \{b_i, f_i, b_i', f_i'\}, \big\{\emptyset, \{b_i, b_i'\}, \{f_i, f_i'\}, \{b_i, b_i', f_i, f_i'\}\big\})$. ◊

2.2.4 Preservation of properties

To show that translations reo_1 and bip_1 preserve properties, we need to show that the diagram in Figure 2.7 commutes, i.e., $\mathsf{f}_1(\mathsf{reo}_1(A))$ is equivalent to $\mathsf{g}_1(A)$ and $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ is equivalent to $\mathsf{f}_1(\mathcal{A})$, for all $A \in \text{Arch}$ and $\mathcal{A} \in \text{PA}$.

The following examples show that this equivalence cannot be interpreted as equality or (strong) bisimulation.

Example 2.2.3. Consider the port automaton $\mathcal{A} = (\{q_0\}, \{a\}, \{(q_0, \{a\}, q_0)\}, q_0)$. The translation $\mathsf{bip}_1(\mathcal{A})$ of $\mathcal{A}$ into a BIP architecture is $(\{\overline{A}\}, \{a, a'\}, \{\emptyset, \{a, a'\}\})$, with coordinating component $\overline{A} = (\{q_0\}, q_0, \{a'\}, \{(q_0, \{a'\}, q_0)\})$. Since the interaction model of $\mathsf{bip}_1(\mathcal{A})$ contains the empty set, we find that the semantics $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ of $\mathsf{bip}_1(\mathcal{A})$ is given by $(\{q_0\}, 2^{\{a\}}, \{(q_0, \{a\}, q_0), (q_0, \emptyset, q_0)\}, q_0)$. On the other hand, the semantics $\mathsf{f}_1(\mathcal{A})$ of $\mathcal{A}$ does not admit an internal transition $(q_0, \emptyset, q_0)$, which shows that $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ and $\mathsf{f}_1(\mathcal{A})$ are not strongly bisimilar. $\Diamond$

Example 2.2.4. Consider the BIP architecture $A = (\{C_1, C_2\}, \emptyset, \emptyset)$ with coordinating components $C_i = (\{q_i, q_i'\}, q_i, \emptyset, \{(q_i, \emptyset, q_i')\})$, for $i = 1, 2$. Since the interaction model of A is empty, its translation $\mathcal{A}_\emptyset$ to a port automaton equals $(\{q_I\}, \emptyset, \emptyset, q_I)$. In addition, $P_{\{C_1,C_2\}} = \emptyset$, which shows that the translation of A to a port automaton equals $\mathsf{reo}_1(A) = \exists P_{\{C_1,C_2\}}(C_1^* \bowtie C_2^* \bowtie \mathcal{A}_\emptyset) = C_1^* \bowtie C_2^*$. Definition 2.1.8 shows that the semantics $\mathsf{f}_1(\mathsf{reo}_1(A))$ of $\mathsf{reo}_1(A)$ contains a transition $((q_1, q_2, q_I), \emptyset, (q_1', q_2', q_I))$.

Let $D = (\{q_D\}, q_D, \emptyset, \emptyset)$ be a dummy component relative to the BIP architecture A. Since BIP architecture application in Definition 2.1.3 requires state-changing internal (i.e., $\emptyset$-labelled) transitions to execute in isolation, we conclude that $A(\{D\})$ does not admit a transition $((q_1, q_2, q_D), \emptyset, (q_1', q_2', q_D))$. This shows that the semantics $\mathsf{g}_1(A)$ of A and $\mathsf{f}_1(\mathsf{reo}_1(A))$ are not strongly bisimilar. $\Diamond$

Since equality or (strong) bisimulation is a too strong semantic equivalence, we use the slightly weaker notion of equivalence called weak bisimulation [Mil89].

Definition 2.2.2 (Weak bisimulation [Mil89])**.** If $L_i = (Q_i, 2^{P_i}, \to_i, q_i^0) \in \text{LTS}$, $i = 1, 2$, then L_1 and L_2 are *weakly bisimilar* ($L_1 \cong L_2$) iff $P_1 = P_2$ and there exists $R \subseteq Q_1 \times Q_2$ such that $(q_1^0, q_2^0) \in R$ and$(q_1, q_2) \in R$ implies for all $N \in 2^{P_0} = 2^{P_1}$ and all $i, j \in \{1, 2\}$ with $i \neq j$, that

1. if $q_i \xrightarrow{\emptyset}_i q_i'$, then $q_j \ (\xrightarrow{\emptyset}_j)^* \ q_j'$ and $(q_1', q_2') \in R$, for some q_j'; and
2. if $q_i \xrightarrow{N}_i q_i'$ and $N \neq \emptyset$, then $q_j \ (\xrightarrow{\emptyset}_j)^* \xrightarrow{N}_j (\xrightarrow{\emptyset}_j)^* \ q_j'$ and $(q_1', q_2') \in R$, for some q_j'.

Definition 2.2.3 (Semantic equivalence)**.** Port automata $\mathcal{A}$ and $\mathcal{B}$ are *semantically equivalent* ($\mathcal{A} \sim \mathcal{B}$) iff $\mathsf{f}_1(\mathcal{A}) \cong \mathsf{f}_1(\mathcal{B})$. BIP architectures A and B are *semantically equivalent* ($A \sim B$) iff $\mathsf{g}_1(A) \cong \mathsf{g}_1(B)$.

Lemma 2.2.1. *Semantic equivalence of port automata satisfies the following properties: for all $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2 \in \text{PA}$ we have*

1. *associativity:* $\mathcal{A}_0 \bowtie (\mathcal{A}_1 \bowtie \mathcal{A}_2) \sim (\mathcal{A}_0 \bowtie \mathcal{A}_1) \bowtie \mathcal{A}_2$
2. *commutativity:* $\mathcal{A}_0 \bowtie \mathcal{A}_1 \sim \mathcal{A}_1 \bowtie \mathcal{A}_0$

3. congruence: $\mathcal{A}_0 \sim \mathcal{A}_1$ *implies* $\mathcal{A}_0 \bowtie \mathcal{A}_2 \sim \mathcal{A}_1 \bowtie \mathcal{A}_2$.

Proof. Consider (strong) bisimulation of port automata(i.e., constraint automata all of whose data constraints are $\top$) as defined in [BSAR06]. Composition of port automata is commutative and associative up to bisimulation [BSAR06]. Since f_1 acts like the identity and every (strong) bisimulation is also a weak bisimulation, we conclude that composition of port automata is commutative and associative modulo semantic equivalence.

Since f_1 acts as the identity and every (strong) bisimulation is also a weak bisimulation, we conclude that semantic equivalence of port automata corresponds to weak bisimulation of port automata. Let Q_0, Q_1 and Q_2 be the state spaces of $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Suppose that $R \subseteq Q_0 \times Q_1$ is a weak bisimulation between $\mathcal{A}_0$ and $\mathcal{A}_1$. Using Definition 2.1.8, it follows that $R' = \{((q_0, q_2), (q_1, q_2')) \mid (q_0, q_1) \in R \text{ and } q_2 = q_2'\} \subseteq (Q_0 \times Q_2) \times (Q_1 \times Q_2)$ is a weak bisimulation between $\mathcal{A}_0 \bowtie \mathcal{A}_2$ and $\mathcal{A}_1 \bowtie \mathcal{A}_2$. □

Theorem 2.2.2. *For all* $\mathcal{A} \in \mathrm{PA}$ *and* $A \in \mathrm{Arch}$ *we have* $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A})) \cong \mathsf{f}_1(\mathcal{A})$ *and* $\mathsf{f}_1(\mathsf{reo}_1(A)) \cong \mathsf{g}_1(A)$.

Proof. First, we show that $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A})) \cong \mathsf{f}_1(\mathcal{A})$ for all port automata $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0) \in \mathrm{PA}$. The state space of $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ is $Q \times \{q_D\}$, where q_D is the state of the dummy component, and the state space of $\mathsf{f}_1(\mathcal{A})$ is Q. We show that $\sim$ given by $(q, q_D) \sim q$ for all $q \in Q$ is a weak bisimulation.

Trivially, $(q_0, q_D) \sim q_0$. Suppose that $((q, q_D), N, (q', q_D))$ is a transition in $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$. We show that either $N = \emptyset$ and $q' = q$, or there exists a transition (q, N, q') in $\mathsf{f}_1(\mathcal{A})$ with $(q', q_D) \sim q'$. Using the shape of the interaction model γ, we obtain a transition $((q, q_D), N \cup N', (q', q_D))$ in $\mathsf{bip}_1(\mathcal{A})(\{D\})$, with $N' = \{n' \mid n \in N\}$. Definition 2.1.3, with $\mathcal{C} = \{\overline{\mathcal{A}}\}$ and $\mathcal{B} = \{D\}$, shows that either

1a) $N \cup N' = \emptyset$, $(q, \emptyset, q')$ is a transition in $\overline{\mathcal{A}}$, and $q_D = q_D$; or

1b) $N \cup N' = \emptyset$, $(q_D, \emptyset, q_D)$ is a transition in D, and $q' = q$; or

2) $N \cup N' \in \gamma_{\mathsf{bip}_1(\mathcal{A})}$, and if $N' \neq \emptyset$ then (q, N', q') is a transition in $\overline{\mathcal{A}}$, and if $N' = \emptyset$ then $q' = q$, and if $N \neq \emptyset$ then (q_D, N, q_D) is a transition in D, and if $N = \emptyset$ then $q_D = q_D$.

If (1a) holds, then $N = \emptyset$, and by the definition of f_1 we find a transition (q, N, q') in $\mathsf{f}_1(\mathcal{A})$. Trivially, $(q', q_D) \sim q'$. Case (1b) is impossible, since dummy component D does not have an empty transition. Suppose that (2) holds. If $N = \emptyset$, then we have $q' = q$. If $N \neq \emptyset$, then the definition of f_1 gives a (q, N, q') in $\mathsf{f}_1(\mathcal{A})$, and trivially we have $(q', q_D) \sim q'$. Thus, in each case, either $N = \emptyset$ and $q' = q$, or there exists a transition (q, N, q') in $\mathsf{f}_1(\mathcal{A})$ with $(q', q_D) \sim q'$.

On the other hand, let (q, N, q') be a transition in $\mathsf{f}_1(\mathcal{A})$. We show that there exists a transition $((q, q_D), N, (q', q_D))$ in $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$. Using the definition of f_1, we find that (q, N', q') is a transition in $\overline{\mathcal{A}}$, with $N' = \{n' \mid n \in N\}$. If $N = \emptyset$, then the first rule of Definition 2.1.3 implies that $((q, q_D), N \cup N', (q', q_D))$ is a transition in $\mathsf{bip}_1(\mathcal{A})(\{D\})$. If $N \neq \emptyset$, then we have that (q_D, N, q_D) is a transition in the dummy component D of the BIP architecture application $\mathsf{bip}_1(\mathcal{A})(\{D\})$. The second rule of Definition 2.1.3 implies that $((q, q_D), N \cup N', (q', q_D))$ is a transition

in $\mathsf{bip}_1(\mathcal{A})(\{D\})$. In either case, we find that $((q, q_D), N, (q', q_D))$ is a transition in $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ and trivially that $(q', q_D) \sim q'$. Thus, $\sim$ is a weak bisimulation between $\mathsf{g}_1(\mathsf{bip}_1(\mathcal{A}))$ and $\mathsf{f}_1(\mathcal{A})$.

Second, We show that $\mathsf{f}_1(\mathsf{reo}_1(A)) \cong \mathsf{g}_1(A)$ for any BIP architecture $A = (\{C_i\}_{i\in I}, P, \gamma)$ with components given by $C_i = (Q_i, q_i^0, P_i, \rightarrow_i)$, for all $i \in I$. The state space of $\mathsf{f}_1(\mathsf{reo}_1(A))$ is $(\prod_{i\in I} Q_i) \times \{q_I\}$, where q_I is the state of the port automaton of the interaction model of A. The state space of $\mathsf{g}_1(A)$ is $(\prod_{i\in I} Q_i)\times\{q_D\}$, where q_D is the state of the dummy component. We show that $\sim$ given by $(\mathbf{q}, q_I) \sim (\mathbf{q}, q_D)$ for all $\mathbf{q} = (q_i)_{i\in I} \in \prod_{i\in I} Q_i$, is a weak bisimulation.

Trivially, $(\mathbf{q}^0, q_I) \sim (\mathbf{q}^0, q_D)$. Let $((\mathbf{q}, q_D), N, (\mathbf{q}', q_D))$ be a transition in $\mathsf{g}_1(A)$, for some $N \subseteq P\backslash P_{\mathcal{C}}$. We show that $((\mathbf{q}, q_I), N, (\mathbf{q}', q_I))$ is a transition in $\mathsf{f}_1(\mathsf{reo}_1(A))$. The definition of g_1 shows that there exists some $M \subseteq P$, with $M \setminus P_{\mathcal{C}} = N$, such that $((\mathbf{q}, q_D), M, (\mathbf{q}', q_D))$ is a transition in $A(\{D\})$, where D is the dummy component of A. Definition 2.1.3 implies that either

1a) $M = \emptyset$, $(q_i, \emptyset, q_i') \in \rightarrow_i$ and $q_j' = q_j$, for some $i \in I$ and all $j \in I \setminus \{i\}$; or

1b) $M = \emptyset$, $(q_D, \emptyset, q_D)$ is a transition in D, and $q_j' = q_j$ for all $j \in I$; or

2) $M \in \gamma$, and if $M \cap P_i \neq \emptyset$ then $(q_i, M \cap P_i, q_i') \in \rightarrow_i$, and if $M \cap P_i = \emptyset$ then $q_i' = q_i$, for all $i \in I$.

If (1a), then $(q_i, \emptyset, q_i')$ is a transition in C_i^*. Hence, the second item in Definition 2.1.8 gives a transition $((\mathbf{q}, q_I), N, (\mathbf{q}', q_I))$ in $\mathsf{f}_1(\mathsf{reo}_1(A))$, with $N \subseteq M = \emptyset$. Case (1b) is impossible, since dummy component D does not have an empty transition. If (2), then $M \in \gamma$ implies $(q_I, M, q_I) \in \mathcal{A}_\gamma$. Using Definition 2.1.8 and $M \setminus P_{\mathcal{C}} = N$, we find a transition $((\mathbf{q}, q_I), N, (\mathbf{q}', q_I))$ in $\mathsf{f}_1(\mathsf{reo}_1(A))$.

Let $((\mathbf{q}, q_I), N, (\mathbf{q}', q_I))$ be a transition in $\mathsf{f}_1(\mathsf{reo}_1(A))$, for some $N \subseteq P \setminus P_{\mathcal{C}}$. We show that there exist a sequence of transitions $(\mathbf{q}, q_I)$ $(\xrightarrow{\emptyset})^*$ $\xrightarrow{N}$ $(\mathbf{q}', q_I)$ in $\mathsf{g}_1(A)$. The definition of reo_1 shows that there exists some $M \subseteq P$ such that $M \setminus P_{\mathcal{C}} = N$ and $((\mathbf{q}, q_I), M, (\mathbf{q}', q_I))$ is a transition in $C_1^* \bowtie \cdots C_n^* \bowtie \mathcal{A}_\gamma$. According to Definition 2.1.8, we find that either

1) $(\mathbf{q}, M, \mathbf{q}')$ and (q_I, M, q_I) are transitions in $C_1^* \bowtie \cdots \bowtie C_n^*$ resp. $\mathcal{A}_\gamma$; or

2a) $(\mathbf{q}, M, \mathbf{q}')$ is a transition in $C_1^* \bowtie \cdots \bowtie C_n^*$ and $M \cap P = \emptyset$; or

2b) (q_I, M, q_I) is a transition in $\mathcal{A}_\gamma$, $M \cap P_{\mathcal{C}} = \emptyset$ and $\mathbf{q}' = \mathbf{q}$.

If (1) holds, then $M \in \gamma$, and, for each $i \in I$, we have either $M \cap P_i = \emptyset$ and $q_i' = q_i$ or we find a transition $(q_i, M \cap P_i, q_i')$ in C_i^*. Definition 2.1.3 requires a transition $(q_i, M \cap P_i, q_i')$ in C_i^* that satisfies both $M \cap P_i = \emptyset$ and $q_i' \neq q_i$ to execute in isolation. Therefore, Definition 2.1.3 yields a sequence of transitions $(\mathbf{q}, q_I)$ $(\xrightarrow{\emptyset})^*$ $(\overline{\mathbf{q}}, q_I)$ $\xrightarrow{N}$ $(\mathbf{q}', q_I)$ in $\mathsf{g}_1(A)$, where $\overline{q_i} = q_i'$, if $M \cap P_i = \emptyset$ and $q_i' \neq q_i$, and $\overline{q_i} = q_i$ otherwise. If (2a) holds, then $N \subseteq M = M \cap P = \emptyset$ and, by Definition 2.1.8, we have for some $i \in I$ that $(q_i, \emptyset, q_i')$ is a transition in C_i^*. Similar to case(1), we obtain a non-empty sequence of transitions $(\mathbf{q}, q_I)$ $(\xrightarrow{\emptyset})^+$ $(\mathbf{q}', q_I)$ in $\mathsf{g}_1(A)$. If (2b) holds, then we have $N = M \in \gamma$, and Definition 2.1.3 shows that there exist a transition $(\mathbf{q}, q_I)$ $\xrightarrow{N}$ $(\mathbf{q}', q_I)$ in $\mathsf{g}_1(A)$. In each case, we found a sequence of

transitions $(\mathbf{q}, q_I)$ $(\xrightarrow{\emptyset})^*$ $\xrightarrow{N}$ $(\mathbf{q}', q_I)$ in $\mathsf{g}_1(A)$, and $(\mathbf{q}', q_I) \sim (\mathbf{q}', q_D)$. Thus, $\sim$ is a weak bisimulation between $\mathsf{f}_1(\mathsf{reo}_1(A))$ and $\mathsf{g}_1(A)$. □

Corollary. bip_1 *and* reo_1 *preserve all properties closed under weak bisimulation, i.e., for all* $P \subseteq$ LTS, $\mathcal{A} \in$ PA *and* $A \in$ Arch *we have* $\mathsf{f}_1(\mathcal{A}) \in P \Leftrightarrow \mathsf{g}_1(\mathsf{bip}_1(\mathcal{A})) \in P$ *and* $\mathsf{g}_1(A) \in P \Leftrightarrow \mathsf{f}_1(\mathsf{reo}_1(A)) \in P$*, whenever* $L \in P$ *and* $L' \cong L$ *implies* $L' \in P$*, for all* $L, L' \in$ LTS.

Section 2.2.4 allows model checking of BIP architectures with Reo model checkers, and vice versa. This is particularly interesting, since tools for BIP and Reo employ different model checking techniques. For example, the D-Finder tool allows for compositional deadlock detection and verification of BIP systems [bip16], while Vereofy allows for linear and branching time model checking of Reo systems [reo16].

Example 2.2.5. Consider the following safety property φ satisfied by the Reo connector in Figure 2.4(c): "if b_1 fires, then b_2 fires only after f_1 fires". The automaton $\mathcal{A}$ in Figure 2.6(c) clearly satisfies this property. Using Section 2.2.4, we conclude that the BIP architecture $\mathsf{bip}_1(\mathcal{A})$ satisfies φ also. ◊

2.2.5 Compatibility with composition

BIP architectures and port automata have their own notions of composition. We show that, under some mild conditions, our translations preserve composition modulo semantic equivalence.

Recall the port automaton representation of the interaction model from Section 2.2.2. The following lemma provides a decomposition of the port automaton representation of the interaction model of a composed BIP architecture.

Lemma 2.2.3. *Let* $A_i = (\mathcal{C}_i, P_i, \gamma_i) \in$ Arch*,* $i = 1, 2$*, with* $P_{\mathcal{C}_1} \cap P_{\mathcal{C}_2} = \emptyset$ *and* $\emptyset \in \gamma_1 \cap \gamma_2$*. Then, we have that* $\mathcal{A}_{\gamma_{12}} \sim \mathcal{A}_{\gamma_1} \bowtie \mathcal{A}_{\gamma_2}$*, where* γ_{12} *is the interaction model of* $A_1 \oplus A_2$*.*

Proof. Let (q, N, q) be a transition in $\mathcal{A}_{\gamma_{12}}$. By definition, $N \in \gamma_{12}$, and from Definition 2.1.4 we deduce $N \cap P_i \in \gamma_i$, $i = 1, 2$. Therefore $(q, N \cap P_i, q)$ is a transition in $\mathcal{A}_{\gamma_i}$. Then, Definition 2.1.8, implies that $((q, q), N, (q, q))$ in $\mathcal{A}_{\gamma_1} \bowtie \mathcal{A}_{\gamma_2}$. On the other hand, suppose that $((q, q), N, (q, q))$ is a transition in $\mathcal{A}_{\gamma_1} \bowtie \mathcal{A}_{\gamma_2}$. Then, Definition 2.1.8 gives either that (1) for $i = 1, 2$, $(q, N \cap P_i, q)$ is a transition in $\mathcal{A}_{\gamma_i}$, or (2) for $i, j \in \{1, 2\}$, $i \neq j$, $(q, N \cap P_i, q)$ is a transition in $\mathcal{A}_{\gamma_i}$ and $N \cap P_j = \emptyset$. In the first case, we conclude that $N \cap P_i \in \gamma_i$, for $i = 1, 2$. Hence, Definition 2.1.4 implies $N \in \gamma_{12}$. In the second case, we see that $N \cap P_i \in \gamma_i$ and $N \cap P_j = \emptyset \in \gamma_j$, since $\emptyset \in \gamma_1 \cap \gamma_2$. Thus, Definition 2.1.4 implies $N \in \gamma_{12}$. In both cases we find $N \in \gamma_{12}$, and we conclude that (q, N, q) is a transition of $\mathcal{A}_{\gamma_{12}}$. □

For any two BIP architectures $A_1, A_2 \in$ Arch, consider the equation

$$\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2), \tag{2.5}$$

Recall that reo_1 hides all internal ports $P_{\mathcal{C}_1 \cup \mathcal{C}_2}$ of $A_1 \oplus A_2$, where, for $i \in \{1, 2\}$, $\mathcal{C}_i$ is the set of coordinating components of A_i. This means that internal ports $P_{\mathcal{C}_1 \cup \mathcal{C}_2}$ in $A_1 \oplus A_2$ cannot be used for composition in the right hand side of equation

Equation (2.5). In particular, the BIP architectures cannot share any internal port in $P_{\mathcal{C}_1 \cup \mathcal{C}_2} = P_{\mathcal{C}_1} \cup P_{\mathcal{C}_2}$. Therefore, we need to assume that $P_{\mathcal{C}_1} \cap P_2 = P_{\mathcal{C}_2} \cap P_1 = \emptyset$, where, for $i \in \{1, 2\}$, P_i is the interface of A_i.

Note that shared internal ports can be transformed into shared dangling ports. Let $p \in P_{\mathcal{C}_1} \cap P_2$ be a dangling port of P_2 that is connected to a component in A_1. Change A_1 to A_1' by adding a (dangling) port x to A_1 and synchronizing p with x by changing the BIP interaction model γ_1 of A_1 to $\gamma_1' = \{N \cup \{x\} \mid p \in N \in \gamma_1\} \cup \{N \mid p \notin N \in \gamma_1\}$. Change A_2 to A_2' by renaming p to x in A_2. The resulting architectures A_1' and A_2' satisfy the assumption. This construction shows that $P_{\mathcal{C}_1} \cap P_2 = P_{\mathcal{C}_2} \cap P_1 = \emptyset$ is only a mild assumption.

Theorem 2.2.4. $\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$ *for all* $A_i = (\mathcal{C}_i, P_i, \gamma_i) \in$ Arch*, with* $P_{\mathcal{C}_1} \cap P_2 = P_{\mathcal{C}_2} \cap P_1 = \emptyset$ *and* $\emptyset \in \gamma_1 \cap \gamma_2$*.*

Proof. Let $\mathcal{C}_1 \cup \mathcal{C}_2 = \{C_1, \ldots, C_n, \ldots, C_m\}$, with $C_i \in \mathcal{C}_1$ iff $i \leq n$, be the set of coordinating components of A_1 and A_2. By definition, we have $\mathsf{reo}_1(A_1 \oplus A_2) = \exists P_{\mathcal{C}_1 \cup \mathcal{C}_2}(C_1^* \bowtie \cdots C_n^* \bowtie C_{n+1}^* \bowtie \cdots C_m^* \bowtie \mathcal{A}_{\gamma_{12}})$. Using Lemmas 2.2.1 and 2.2.3, we obtain $\mathsf{reo}_1(A_1 \oplus A_2) \sim \exists P_{\mathcal{C}_1} \exists P_{\mathcal{C}_2}(C_1^* \bowtie \cdots C_n^* \bowtie \mathcal{A}_{\gamma_1} \bowtie C_{n+1}^* \bowtie \cdots C_m^* \bowtie \mathcal{A}_{\gamma_2})$. From $P_{\mathcal{C}_1} \cap P_2 = P_{\mathcal{C}_2} \cap P_1 = \emptyset$, we conclude that the port automata $C_1^*, \ldots, C_n^*$ and $\mathcal{A}_{\gamma_1}$ do not use ports from $P_{\mathcal{C}_2}$. Since hiding of non-shared ports distributes over composition of port automata, we find that

$$\mathsf{reo}_1(A_1 \oplus A_2) \sim \exists P_{\mathcal{C}_1}(C_1^* \bowtie \cdots C_n^* \bowtie \mathcal{A}_{\gamma_1}) \bowtie \exists P_{\mathcal{C}_2}(C_{n+1}^* \bowtie \cdots C_m^* \bowtie \mathcal{A}_{\gamma_2}).$$

Hence, we conclude that $\mathsf{reo}_1(A_1 \oplus A_2) \sim \mathsf{reo}_1(A_1) \bowtie \mathsf{reo}_1(A_2)$. □

Theorem 2.2.5. $\mathsf{bip}_1(\mathcal{A}_1 \bowtie \mathcal{A}_2) \sim \mathsf{bip}_1(\mathcal{A}_1) \oplus \mathsf{bip}_1(\mathcal{A}_2)$ *for all* $\mathcal{A}_i \in$ PA*.*

Proof. Applying Theorem 2.2.4, with $A_1 = \mathsf{bip}_1(\mathcal{A}_1)$ and $A_2 = \mathsf{bip}_1(\mathcal{A}_2)$, gives that $\mathsf{reo}_1(\mathsf{bip}_1(\mathcal{A}_1) \oplus \mathsf{bip}_1(\mathcal{A}_2)) \sim \mathsf{reo}_1(\mathsf{bip}_1(\mathcal{A}_1)) \bowtie \mathsf{reo}_1(\mathsf{bip}_1(\mathcal{A}_2))$. Using Theorem 2.2.2, we find, for any $\mathcal{B} \in$ PA, that $\mathsf{f}_1(\mathsf{reo}_1(\mathsf{bip}_1(\mathcal{B}))) \cong \mathsf{g}_1(\mathsf{bip}_1(\mathcal{B})) \cong \mathsf{f}_1(\mathcal{B})$ and $\mathsf{reo}_1(\mathsf{bip}_1(\mathcal{B})) \sim \mathcal{B}$. Since semantic equivalence is a congruence by Lemma 2.2.1, we find that $\mathsf{reo}_1(\mathsf{bip}_1(\mathcal{A}_1) \oplus \mathsf{bip}_1(\mathcal{A}_2)) \sim \mathcal{A}_1 \bowtie \mathcal{A}_2 \sim \mathsf{reo}_1(\mathsf{bip}_1(\mathcal{A}_1 \bowtie \mathcal{A}_2))$. By Theorem 2.2.2, we conclude that $\mathsf{bip}_1(\mathcal{A}_1) \oplus \mathsf{bip}_1(\mathcal{A}_2) \sim \mathsf{bip}_1(\mathcal{A}_1 \bowtie \mathcal{A}_2)$ □

Example 2.2.6. For any two ports x and y, let $\mathcal{A}_{\{x,y\}}$ be the port automaton of a synchronous channel (cf. Figure 2.2), and let $C_{\{x,y\}}$ be its corresponding BIP component. Suppose we need to translate $\mathcal{A}_{\{a,b\}} \bowtie \mathcal{A}_{\{b,c\}}$ to a BIP architecture. Then, we compute $\mathsf{bip}_1(\mathcal{A}_{\{a,b\}}) = (\{C_{\{a',b'\}}\}, \{a, a', b, b'\}, \gamma_{\{a,b\}})$, with

$$\gamma_{\{a,b\}} = \{\emptyset, \{a, a'\}, \{b, b'\}, \{a, a', b, b'\}\}.$$

Next, we compute $\mathsf{bip}_1(\mathcal{A}_{\{b,c\}}) = (\{C_{\{b'',c''\}}\}, \{b, b'', c, c''\}, \gamma_{\{b,c\}})$, with

$$\gamma_{\{b,c\}} = \{\emptyset, \{b, b''\}, \{c, c''\}, \{b, b'', c, c''\}\}.$$

Note that we need to use double primes now, because otherwise b' would be a shared port of $C_{\{a',b'\}}$ and $C_{\{b'',c''\}}$. Using Theorem 2.2.5, we find that $\mathsf{bip}_1(\mathcal{A}_{\{a,b\}} \bowtie \mathcal{A}_{\{b,c\}}) = \mathsf{bip}_1(\mathcal{A}_{\{a,b\}}) \oplus \mathsf{bip}_1(\mathcal{A}_{\{b,c\}})$. Therefore, $\mathcal{A}_{\{a,b\}} \bowtie \mathcal{A}_{\{b,c\}}$ translates to

$$(\{C_{\{a',b'\}}, C_{\{b'',c''\}}\}, \{a, a', b, b', b'', c, c''\}, \gamma_{\{a,b,c\}}),$$

where $\gamma_{\{a,b,c\}}$ is the composition of $\gamma_{\{a,b\}}$ and $\gamma_{\{b,c\}}$. ◊

Example 2.2.7. Consider the port automaton $\mathcal{A}$ from Figure 2.6(c). If we translate $\mathcal{A}$ to BIP, we obtain a BIP architecture $B_1 = \mathsf{bip}_1(\mathcal{A})$, which has only a single coordinating component. From Example 2.1.8, we see that $\mathcal{A} \cong \mathcal{A}_0 \bowtie \mathcal{A}_1 \bowtie \mathcal{A}_2$, where $\mathcal{A}_0$ is the port automaton in Figure 2.6(a), and $\mathcal{A}_i$ is the port automaton in Figure 2.6(b), for $i = 1, 2$. Now consider $B_3 = \mathsf{bip}_1(\mathcal{A}_0) \oplus \mathsf{bip}_1(\mathcal{A}_1) \oplus \mathsf{bip}_1(\mathcal{A}_2)$. Using Definition 2.1.4, we see that B_3 has three coordinating components. Nevertheless, Theorem 2.2.5 shows that B_3 is semantically equivalent to B. Therefore, Theorem 2.2.5 allows to compute translations compositionally. ◊

2.3 Stateless CA's and interaction models

In Section 2.2, we established a correspondence between port automata and BIP architectures. Here, we offer translations between data-sensitive connector models in BIP and Reo.

For BIP connectors we use BIP interaction models, which are tuples consisting of an interface P and a set Γ of interaction expressions α that have:

1. a single top port that is not a bottom port,
2. bottom ports included in their interface P, and
3. guard and up functions that are independent of local variables (Definition 2.1.5).

We assume that every top port occurs only in one interaction expression per BIP interaction model. We denote the class of such BIP interaction models by IM.

For the semantics of Reo connectors, we take a pair consisting of a constraint automaton and a partition of its interface into *input* ports $\mathcal{N}_{in}$ and *output* ports $\mathcal{N}_{out}$[5]. We call such pairs *constraint automata with polarity*. The reason we explicitly distinguish CA port types in this semantics is to give direction to dataflow, similar to BIP connectors. Usually such port type distinctions are implicit within the semantics of Reo connectors, but for preciseness we encode them here as a partition.

A full correspondence of BIP interaction models and constraint automata with polarity in Reo is not possible. Firstly, BIP interaction models are stateless, we need to restrict ourselves here to only stateless constraint automata with polarity [ABB+14, BSBJ14]. Secondly, ports of a BIP interaction expression are *bidirectional* in the sense that input and output through a port happen simultaneously in a single execution step. Ports in a Reo connector are *unidirectional* in the sense that each port is either an input port or an output port. To accommodate this distinction, we split every bidirectional port p in a BIP interaction expression into an input port $p!$, providing write operations to the user of the connector, and an output port $p?$, providing read operations to the user of the connector. Therefore, we consider the class $\mathrm{CA}^{\pm}$ of all stateless constraint automata with polarity, such that, for some set of BIP ports P, we have the set of Reo ports $\mathcal{N}_{in} = \{p! \mid p \in P\}$, $\mathcal{N}_{out} = \{p? \mid p \in P\}$, and, for every $p \in P$, ports $p!$ and $p?$ synchronize (i.e., $p! \in N$ if and only if $p? \in N$ for every transition $(q, N, g, q') \in \rightarrow$).

[5]To simplify notation, we deviate from [DJAB15] by excluding internal ports.

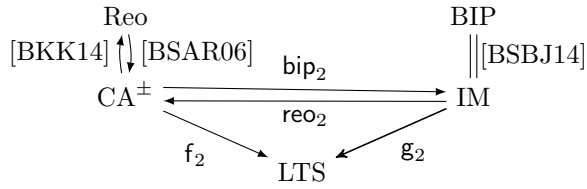


Figure 2.9: Translations and interpretations in the data-sensitive domain.

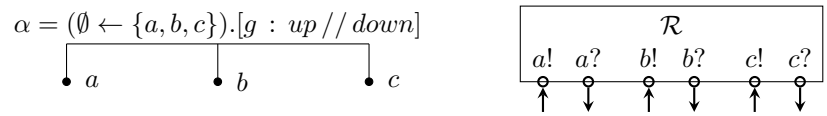


Figure 2.10: Simulating bidirectional ports in BIP with unidirectional ports in Reo.

As in Section 2.2, we interpret all connectors as labelled transition systems. Then, we define translations between Reo connectors ($\mathrm{CA}^{\pm}$) and BIP connectors (IM), and show that they preserve properties.

2.3.1 Interpretation of BIP and Reo

Consider the diagram in Figure 2.9. Classes $\mathrm{CA}^{\pm}$ and IM consist of constraint automata with polarity and BIP interaction models. Morphisms bip_2 and reo_2 are translations of those classes and f_2 and g_2 are interpretations in a common LTS semantics. We do not intend to redefine the semantics of constraint automata with polarity and of BIP interaction models in this section. Hence, we interpret them using their definitions from [BSAR06, BSBJ14].

The class LTS in Figure 2.9 is the class of all labelled transition systems over an alphabet $(D+1)^{2P}$, where D is a set of data items; $1 = \{0\}$, where 0 represents the absence of data (similar to *void* or *null*); and $2P = \{p!, p? \mid p \in P\}$ is the *duplicated (unidirectional) port set* of a set of (bidirectional) ports P. If the environment writes a datum d to bidirectional port p of a connector, then we represent this by an assignment of d to the unidirectional port $p!$. If the environment reads a datum d from a bidirectional port p of a connector, then we represent this by an assignment of d to the unidirectional port $p?$.

Example 2.3.1. Figure 2.10 shows an example of this port duplication. First, the upward data transfer expression in α takes data from the bottom ports a, b and c. In the Reo connector $\mathcal{R}$, this corresponds to taking data from ports $a!$, $b!$ and $c!$. Finally, the downward data transfer expression in the BIP interaction expression α offers data to the bottom ports, which corresponds in Reo connector $\mathcal{R}$ to offering data to ports $a?$, $b?$ and $c?$. ◊

Interpretation of IM We first define the interpretation $\mathsf{g}_2(\Gamma) \in \mathrm{LTS}$ of a BIP interaction model Γ. We define the interface of $\mathsf{g}_2(\Gamma)$ to be $2P = \{p!, p? \mid p \in P\}$, where P is the interface of Γ. We define the data domain of $\mathsf{g}_2(\Gamma)$ to be $\mathcal{D} = \bigcup_{p \in P} \mathsf{D}_p$, where D_p is the data type of port p (cf. Section 2.1.1). We associate to

every interaction expression $\alpha \in \Gamma$ a set $\Delta(\alpha) \subseteq (\mathcal{D}+1)^{2P}$ of data assignments $\delta : 2P \to \mathcal{D}+1$, and we add, for every $\alpha \in \Gamma$ and $\delta \in \Delta(\alpha)$, a transition (q, δ, q) to the stateless labelled transition system $\mathsf{g}_2(\Gamma)$.

We introduce some notation to define the set of data assignments $\Delta(\alpha)$. For every BIP interaction expression α, we write P_α for its bottom ports, g_α for its guard, up^α_w and up^α_L for the restriction of the up function to its top port and its local variables, respectively, and dn^α_{bot} for the restriction of the down function to its bottom ports. For every data assignment $\delta : 2P \to \mathcal{D}+1$, we define $\delta_{up}(p) = \delta(p!)$ and $\delta_{dn}(p) = \delta(p?)$, for all $p \in P_\alpha$.

In this notation, we define

$$\mathsf{g}_2(\Gamma) = (\{q\}, (\mathcal{D}+1)^{2P}, \{(q, \delta, q) \mid \alpha \in \Gamma, \delta \in \Delta(\alpha)\}), \tag{2.6}$$

where $\delta \in \Delta(\alpha)$ iff $\delta(2P \setminus 2P_\alpha) = \{0\}$, $\delta_{dn} = dn^\alpha_{bot}(up^\alpha_w(\delta_{up}), up^\alpha_L(\delta_{up}))$, and $g_\alpha(\delta_{up}) = \mathtt{tt}$. Note that we use the value of $up^\alpha_w(\delta_{up})$ as a local variable, since we consider only non-hierarchical BIP interaction models.

In [BSBJ14], Bliudze et al. encode BIP interaction models in *Top/Bottom (T/B) components*, i.e., an automaton over interaction expressions together with local variables. Furthermore, they define a semantics for T/B components, which indirectly defines an interpretation of interaction models. Equation (2.6) imitates this interpretation without using T/B components explicitly.

Interpretation of $\mathrm{CA}^\pm$ We now define the interpretation of a stateless constraint automaton with polarity $\mathcal{A} = (\{q\}, \mathcal{N}_{in}, \mathcal{N}_{out}, \to, q) \in \mathrm{CA}^\pm$ over a data domain $\mathcal{D}$. By definition, we find a set of unidirectional ports P, such that $\mathcal{N}_{in} = \{p! \mid p \in P\}$, $\mathcal{N}_{out} = \{p? \mid p \in P\}$, and, for every $p \in P$, ports $p!$ and $p?$ synchronize. We use $2P$ as the port names of $\mathsf{f}_2(\mathcal{A})$. We obtain the transitions of $\mathsf{f}_2(\mathcal{A})$ by replacing every transition labelled with N, g in $\mathcal{A}$ with a set of transitions labelled with $\delta \in \Delta(N, g) = \{\delta : 2P \to \mathcal{D}+1 \mid \delta(2P \setminus N) = \{0\}, \delta \models g\}$, where $\Delta(N, g)$ contains all data assignments $\delta : 2P \to \mathcal{D}+1$ that satisfy the synchronization constraint N and data constraint g. Now, define

$$\mathsf{f}_2(\mathcal{A}) = (\{q\}, (\mathcal{D}+1)^{2P}, \{(q, \delta, q) \mid q \xrightarrow{N,g} q, \delta \in \Delta(N, g)\}). \tag{2.7}$$

2.3.2 Reo to BIP

Since BIP interaction models are stateless, we cannot translate an arbitrary constraint automaton (i.e., Reo connector) into BIP. Interaction models in BIP preclude keeping track of the state of a Reo connector. Hence, the translation of the interaction model of a BIP architecture into a port automaton in Section 2.2.2 inspires us for our translation bip_2.

First, we describe intuitively how we translate a stateless constraint automaton $\mathcal{A}$ over a data domain $\mathcal{D}$ to a BIP interaction model. We transform every transition in $\mathcal{A}$ with label N, g into a simple BIP connector with N as its bottom ports, together with a guard, an up and a down function that mimic the data constraint g. We define the corresponding set $\mathsf{bip}_2(\mathcal{A})$ of BIP interaction expressions by the set of all transformed transitions from $\mathcal{A}$.

We now construct an interaction expression for any transition labelled N, g in automaton $\mathcal{A}$ as follows:

$$\alpha(N,g) = (\{w_{N,g}\} \leftarrow P_N).[g_{in}(X_{P_N}) \,:\, Y_{P_N} := \mathrm{solve}(g, X_{P_N}) \,//\, X_{P_N} := Y_{P_N}],$$

where P_N satisfies $2P_N = \{p!, p? \mid p \in P_N\} = N$; the variables $X_{P_N} = \{x_p \mid p \in P_N\}$ model the values assigned to bottom ports; the variables $Y_{P_N} = \{y_p \mid p \in P_N\}$ model some fresh local variables; the guard g_{in} is any quantifier free formula equivalent to $\exists O_N : g(I_N, O_N)$, with input variables $I_N = \{d_{p!} \mid p! \in N\}$ and output variables $O_N = \{d_{p?} \mid p? \in N\}$; and function $\mathrm{solve}(g, X_{P_N})$ returns any vector Y_{P_N} satisfying $g(X_{P_N}, Y_{P_N})$. All variables have data type $\mathcal{D}$ (the data domain of $\mathcal{A}$), i.e., $x_p{:}\mathcal{D}$ for all $p \in \mathcal{N}$.

Let P be the interface of $\mathcal{A}$. Define bip_2 as follows:

$$\mathsf{bip}_2(\mathcal{A}) = (P, \{\alpha(N,g) \mid (q, N, g, q) \in \rightarrow\}). \tag{2.8}$$

Intuitively, the solve function in $\alpha(N,g)$ computes a solution of the guard g, given all input values $d_{p!}$, with $p! \in N$. Note that the solve function in $\alpha(N,g)$ is not deterministic. However, comparing the solve function to the random function in Figure 4 in [BSBJ14], we see that this generality is justified.

Example 2.3.2. Consider a Sync channel from port a to b. To model this channel as a constraint automaton $\mathcal{A} \in \mathrm{CA}^{\pm}$, we duplicate the ports and obtain the interface $P = \{a!, a?, b!, b?\}$. In view of Figure 2.2, we model a Sync channel as $\mathcal{A} = (\{q\}, P, \{(q, P, g, q)\}, q)$, with $g \equiv d_{a!} = d_{b?}$. The translation of $\mathcal{A}$ to a BIP interaction model consist of a single BIP interaction expression

$$\alpha(P,g) = (\{w\} \leftarrow \{a,b\}).[\mathtt{tt} \,:\, (y_a, y_b) := (x_a, x_b) \,//\, (x_a, x_b) := (y_a, y_b)],$$

because $\mathtt{tt} \equiv \exists d_{a?} \exists d_{b?} (d_{a!} = d_{b?})$, for any given $d_{a!}, d_{b!} \in \mathcal{D}$, and the solve function $\mathrm{solve}(g, x_a, x_b) = (x_a, x_b)$ acts as the identity. ◊

2.3.3 BIP to Reo

The correspondence between BIP interaction expressions and automata transitions from Section 2.3.2, provides the main idea for the translation of interaction models into stateless constraint automata. If Γ is a set of simple BIP connectors, we assign to every $\alpha \in \Gamma$ a transition τ_α labelled with $N(\alpha), g(\alpha)$, and subsequently construct the stateless constraint automaton consisting of all such τ_α transitions.

Let α be a simple BIP interaction expression. Define $N(\alpha) = 2P_\alpha = \{p?, p! \mid p \in P_\alpha\}$. Furthermore, let $D? = (d_{p?})_{p \in P}$, $D! = (d_{p!})_{p \in P}$, and define

$$g(\alpha) \;=\; \textstyle\bigwedge_{p \in P} d_{p!}, d_{p?} \in \mathsf{D}_p \;\wedge\; g_\alpha(D!) \;\wedge\; D? = dn^\alpha_{bot}(up^\alpha_w(D!), up^\alpha_L(D!)),$$

where we use our relaxation on the data constraint language from Section 2.1.2 and our notation regarding a BIP interaction expression α from Section 2.3.1. Note that $g(\alpha)$ is independent of the top port w, because we consider only non-hierarchical connectors.

Let Γ be a set of simple BIP connectors with interface P. Recall that $\mathcal{D} = \bigcup_{p \in P} \mathsf{D}_p$. Define the constraint automaton $\mathsf{reo}_2(\Gamma)$ over $\mathcal{D}$ by

$$\mathsf{reo}_2(\Gamma) = (\{q\}, P! \cup P?, \{(q, N(\alpha), g(\alpha), q) \mid \alpha \in \Gamma\}, q). \tag{2.9}$$

Example 2.3.3. Consider the interaction expression $\alpha_{\max}$ from Example 2.1.2, with data domain restricted to $\mathcal{D} = \{0, \ldots, 2^{32} - 1\}$. We translate the interaction model $\Gamma = \{\alpha_{\max}\}$ using Equation (2.9), i.e., we compute $\mathcal{A} = \mathsf{reo}_2(\Gamma)$. Trivially, $\mathcal{A}$ is stateless. Its set of input ports equals $P! = \{a!, b!\}$, and its set of output ports equals $P? = \{a?, b?\}$. $\mathcal{A}$ has a single transition (q, N, g, q), with guard $g \equiv \bigvee_{x,y,z \in \mathcal{D} \,:\, z = \max(x,y)} (d_{a!} = x \wedge d_{b!} = y \wedge d_{a?} = z \wedge d_{b?} = z)$ and synchronization constraint $N = \{a!, b!, a?, b?\}$. ◊

2.3.4 Preservation of properties

To show the faithfulness of translations bip_2 and reo_2, we show that interpretations f_2 and g_2 commute with translations bip_2 and reo_2 in Figure 2.9.

Theorem 2.3.1. *For all* $\mathcal{A} \in \mathrm{CA}^{\pm}$ *and all* $\Gamma \in \mathrm{IM}$ *we have* $\mathsf{g}_2(\mathsf{bip}_2(\mathcal{A})) = \mathsf{f}_2(\mathcal{A})$ *and* $\mathsf{f}_2(\mathsf{reo}_2(\Gamma)) = \mathsf{g}_2(\Gamma)$.

Proof. (Sketch) Let $\mathcal{A} \in \mathrm{CA}^{\pm}$ be a constraint automaton with polarity with interface P, let (q, N, g, q) be a transition in $\mathcal{A}$, and let $\delta : 2P \to \mathcal{D} + 1$ be a data assignment. By definition, we have $\delta \in \Delta(\alpha(N, g))$ if and only if $\delta(2P \setminus 2P_\alpha) = \{0\}$, $\delta_{dn} = dn^{\alpha}_{bot}(up^{\alpha}_{w}(\delta_{up}), up^{\alpha}_{L}(\delta_{up}))$, and $g_\alpha(\delta_{up}) = \mathtt{tt}$, where $\alpha = \alpha(N, g)$. Using the definition of $\alpha(N, g)$, it follows that $\delta \in \Delta(\alpha(N, g))$ if and only if $\delta(2P \setminus N) = \{0\}$ and δ satisfies g. Thus, $\delta \in \Delta(\alpha(N, g))$ if and only if $\delta \in \Delta(N, g)$. Using the definitions of f_2 and g_2, we find that $\mathsf{g}_2(\mathsf{bip}_2(\mathcal{A}))) = \mathsf{f}_2(\mathcal{A})$.

Let $\Gamma \in \mathrm{IM}$ be a BIP interaction model with interface P, let $\alpha \in \Gamma$ be a BIP interaction expression, and let $\delta : 2P \to \mathcal{D}+1$ be a data assignment. By definition, we have $\delta \in \Delta(N(\alpha), g(\alpha))$ if and only if $\delta(2P \setminus N(\alpha)) = \{0\}$ and δ satisfies $g(\alpha)$. Using the definition of $N(\alpha) = 2P_\alpha$ and $g(\alpha)$, it follows $\delta \in \Delta(N(\alpha), g(\alpha))$ if and only if $\delta(2P \setminus 2P_\alpha) = \{0\}$ and $\delta_{dn} = dn^{\alpha}_{bot}(up^{\alpha}_{w}(\delta_{up}), up^{\alpha}_{L}(\delta_{up}))$, and $g_\alpha(\delta_{up}) = \mathtt{tt}$. Thus, $\delta \in \Delta(N(\alpha), g(\alpha))$ if and only if $\Delta(\alpha)$. Using the definitions of f_2 and g_2, we find that $\mathsf{f}_2(\mathsf{reo}_2(\Gamma)) = \mathsf{g}_2(\Gamma)$. □

Corollary. *The translations* bip_2 *and* reo_2 *preserve all properties expressible in* LTS*, i.e.,* $\mathsf{f}_2(\mathcal{A}) \in P \Leftrightarrow \mathsf{g}_2(\mathsf{bip}_2(\mathcal{A})) \in P$ *and* $\mathsf{g}_2(\Gamma) \in P \Leftrightarrow \mathsf{f}_2(\mathsf{reo}_2(\Gamma)) \in P$ *for all* $P \subseteq$ LTS*,* $\mathcal{A} \in \mathrm{CA}^{\pm}$ *and* $\Gamma \in \mathrm{IM}$*.*

Example 2.3.4. Consider the following safety property φ for the interaction expression $\alpha_{\max}$ from Example 2.1.2: "the value retrieved from port a equals zero". Clearly, this safety property does not hold, whenever a or b offers a non-zero integer. Note that φ depends solely on the interpretation of the interaction model $\Gamma = \{\alpha_{\max}\}$ in LTS, and hence φ is expressible in LTS. Using Section 2.3.4 we conclude that φ is false also for $\mathcal{A}_{\max} = \mathsf{reo}_2(\{\alpha_{\max}\})$. Thus, we know any executable code generated from the constraint automaton $\mathcal{A}_{\max}$ does not satisfy φ. More generally, Section 2.3.4 allows us to use the Reo compiler to generate correct code for a BIP interaction model. ◊

2.4 Data-sensitive BIP architectures

Due to the absence[6] of a data-sensitive equivalent of a BIP architecture, our data-sensitive translation presented in Section 2.2 appears restricted in comparison with our data-agnostic translation in Section 2.3. It seems straightforward to extend BIP architectures to the data-sensitive domain by adding coordinating components and replacing the interaction model with a data-sensitive interaction model. However, this extension requires also a composition operator for interaction models, which is not present in the current literature [BSBJ14]. In this section, we propose a data-sensitive extension to BIP architectures and their composition, and we show how this extension relates to Reo connectors.

2.4.1 Composition of BIP interaction expressions

BIP architecture composition in Definition 2.1.4 consists of two parts: it merges the coordinating components into a single set of coordinators, and it composes the BIP interaction models by gluing interactions together. This gluing has not yet been defined for data-sensitive BIP interaction expressions [BSBJ14]. We now propose a possible definition for this gluing of data-sensitive BIP interactions.

Let α_1 and α_2 be two BIP interaction expressions. Intuitively, their composition $\alpha_1 * \alpha_2$ synchronizes α_1 and α_2. That is, both interactions fire in a single atomic step. This means that the composition should evaluate both guards and synchronously execute the upward and downward dataflow of both interaction expressions whenever both guards are satisfied.

Suppose α_1 and α_2 do not share local variables. In that case, we can simulate synchronous execution of the upward data transfer expressions of α_1 and α_2 by sequentially executing both expressions. However, since α_1 and α_2 may share bottom ports, the downward data transfer expressions may write different values to the shared bottom ports. Hence, we cannot simply execute both downward data transfer expressions sequentially.

Generally, the downward data transfer expression of a BIP interaction expression α may depend on the top ports of α. When this is the case, the value produced by the downward data expression becomes known only after hierarchical composition. Thus, at design time we can neither check nor avoid that the downward data transfer expressions of α_1 and α_2 disagree on their shared bottom ports.

Example 2.4.1. Consider the BIP interaction expression

$$\alpha'_{\max} = (\{w\} \leftarrow \{a, b\}).[\mathtt{tt} : \ x_w := \max(x_a, x_b) \,//\, x_a, x_b := x_w],$$

where each port in $\mathcal{P} = \{a, b, w, l\}$ is of type integer, i.e., $x_p : \mathsf{D}_p = \mathbb{Z}$, for all $p \in \mathcal{P}$, and $\mathtt{tt}$ is true. The value of the downward data transfer expression in $\alpha'_{\max}$ depends on the value x_w of its top port w. ♢

When two BIP interaction expressions α_1 and α_2 do not depend on their top ports, we can determine whether α_1 and α_2 agree on shared bottom ports. Indeed, we know the relationship between the values presented to the upward data transfer expression and the values computed by the downward data transfer expression.

[6] This text was written before [BHM19]

This allows us to force agreement already in the guard of the composed BIP interaction expression $\alpha_1 * \alpha_2$. In this way, we can safely execute both downward data transfer expressions sequentially.

Definition 2.4.1 (Composition of interaction expressions)**.** Let α_1 and α_2 be two interaction expressions without shared local variables and for which the downward data transfer expression does not depend on top ports. We define the composition $\alpha_1 * \alpha_2$ of α_1 and α_2 as follows: $\text{top}(\alpha_1 * \alpha_2) = \emptyset$, $\text{bot}(\alpha_1 * \alpha_2) = \text{bot}(\alpha_1) \cup \text{bot}(\alpha_2)$, $up_{\alpha_1 * \alpha_2} = (up_{\alpha_1}, up_{\alpha_2})$, $dn_{\alpha_1 * \alpha_2} = (dn_{\alpha_1}, dn_{\alpha_2})$,

$$g_{\alpha_1 * \alpha_2} = g_{\alpha_1} \wedge g_{\alpha_2} \wedge \Big[dn_{\alpha_1}|_S(up_{\alpha_1}(X^1_Q, X^1_L)) = dn_{\alpha_2}|_S(up_{\alpha_2}(X^2_Q, X^2_L)) \Big],$$

where $dn_{\alpha_i}|_S$ is the restriction of dn_{α_i} to the shared variables X_S over $S = \text{bot}(\alpha_1) \cap \text{bot}(\alpha_2)$, X^i_Q are the variables over $\text{bot}(\alpha_i)$, and X^i_L are the local variables of α_i. The local variables of $\alpha_1 * \alpha_2$ are $X^1_L \cup X^2_L$.

Example 2.4.2. Consider the following BIP interaction expressions $\alpha_1 = (\emptyset \leftarrow \{a, b\}).[\texttt{tt} : x_k := x_a \,//\, x_b := x_k]$, and $\alpha_2 = (\emptyset \leftarrow \{b, c\}).[\texttt{tt} : x_l := x_b \,//\, x_c := x_l]$, which simulate two Sync channels over a, b and b, c respectively (See Figure 2.2). Then, their composition $\alpha_1 * \alpha_2$ is given by $(\emptyset \leftarrow \{a, b, c\}).[\texttt{tt} : x_k := x_a; x_l := x_b \,//\, x_b := x_k; x_c := x_l)$.

This composition merely synchronizes ports a and c, while there is no data exchange between them. On the other hand, the composition of the two Sync channels does transfer data from source a to sink c. Hence, composition of interaction expressions does not correspond directly to composition of Reo channels. ◊

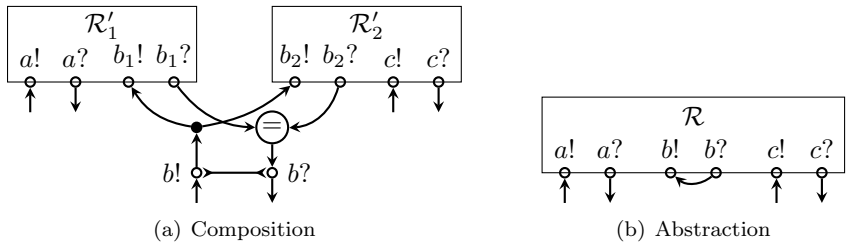
Example 2.4.3. Consider the following BIP interaction expressions $\alpha_1 = (\emptyset \leftarrow \{a, b\}).[\texttt{tt} : x_k := \max(x_a, x_b) \,//\, x_a, x_b := x_k]$, and $\alpha_2 = (\emptyset \leftarrow \{b, c\}).[\texttt{tt} : x_l := \max(x_b, x_c) \,//\, x_b, x_c := x_l]$, which are similar to the BIP interaction expression $\alpha_{\max}$ from Example 2.1.2 (except that we omitted the top port). Intuitively, perhaps, combining $\max(x_a, x_b)$ and $\max(x_b, x_c)$ yields $\max(x_a, x_b, x_c)$. However, the restriction that downward data transfer expressions of α_1 and α_2 must agree on their shared bottom port b, implies that the composition $\alpha_1 * \alpha_2$ takes the following form:

$$\begin{aligned} \alpha_1 * \alpha_2 \quad = \quad & (\emptyset \leftarrow \{a, b, c\}).[\max(x_a, x_b) = \max(x_b, x_c) : \\ & x_k := \max(x_a, x_b); x_l := \max(x_b, x_c) \,//\, x_a, x_b := x_k; x_c := x_l]. \end{aligned}$$

The upward and downward data transfer expressions are composed sequentially. Note that since the downward data transfer does not depend on top ports, the sequential order in this composition is irrelevant. The guard consists of the conjunction of the guards of α_1 and α_2, together with the statement that the downward data transfer expressions agree on the value of x_b. ◊

2.4.2 Abstraction on BIP interaction expressions

Example 2.4.2 shows that the composition of interaction expressions does not correspond directly to composition of Reo connectors. We now investigate the reason for this incompatibility and show that it is possible to simulate composition of Reo connectors by means of an abstraction operator on BIP interaction expressions.



(a) Composition (b) Abstraction

Figure 2.11: Composition (a) and abstraction (b) for interaction expressions.

Consider a Sync channel $\mathcal{R}_1$ over a and b and a Sync channel $\mathcal{R}_2$ over b and c (cf. Figure 2.2). In order to comply with the notation from Section 2.3, we rename every channel end p to $p!$, if it is a source end, or $p?$, if it is a sink end. In this way, we obtain two Reo connectors $\mathcal{R}'_1$ and $\mathcal{R}'_2$ that are Sync channels over $a!$, $b?$ and $b!$, $c?$ respectively.

This renaming splits node b into an output port $b?$ and an input port $b!$. To preserve the intention of composition in Reo, we need to add a Sync channel from $p?$ to $p!$, for every internal port p of the connector. For boundary nodes, there is no need to add a Sync channel.

Using the translation discussed in Section 2.3.2, we obtain from $\mathcal{R}'_1$ a BIP interaction expression α_1 over a and b. Similarly, we find from $\mathcal{R}'_2$ a BIP interaction expression α_2 over b and c. The composition $\alpha_1 * \alpha_2$ of α_1 and α_2 yields a BIP interaction expression over a, b and c.

The composition of BIP interaction expressions may also be described in terms of the Reo connectors $\mathcal{R}'_1$ and $\mathcal{R}'_2$. Figure 2.11(a) shows the construction that simulates this composition. First, we split $\mathcal{R}_1$ and $\mathcal{R}_2$ by renaming their shared ports $b!$ and $b?$ to $b_1!$, $b_2!$ and $b_1?$, $b_2?$ respectively, and we add two fresh ports $b!$ and $b?$. We replicate the data that we observe at $b!$ to both $b_1!$ and $b_2!$. We check the data retrieved from $b_1?$ and $b_2?$ for equality and pass it to $b?$. The node with the equality sign is responsible for this equality check. This node is a Reo component that takes two identical data items from its input and synchronously transfers one of these items to its output. Finally, we synchronize $\mathcal{R}_1$ and $\mathcal{R}_2$ by adding a SyncDrain between $b!$ and $b?$ (cf. Figure 2.2).

As in Example 2.4.2, we see that the BIP interaction expression composition $\mathcal{R}$ of $\mathcal{R}'_1$ and $\mathcal{R}'_2$ yields no dataflow from a to c. Indeed, the depicted composition merely synchronizes $b?$ and $b!$ using a SyncDrain channel. However, the renaming of $\mathcal{R}_1$ and $\mathcal{R}_2$ to $\mathcal{R}'_1$ and $\mathcal{R}'_2$ required an additional Sync channel from $b?$ to $b!$. Hence, in order to simulate composition of Reo connectors, we need to add this Sync channel. We model this addition of the Sync channel by an operation called *abstraction*. Figure 2.11(b) shows the effect of abstraction on the composed Reo connector $\mathcal{R}$.

In terms of Reo connectors, the effect of abstraction is clear. Now, we formulate this abstraction operator in terms of interaction expressions. Consider the interaction expression in Figure 2.11(b). The addition of the Sync channel imposes a restriction on the observed dataflow at b: the data presented as input for the upward data transfer equals the output retrieved from the downward data transfer

expression. This means that the abstraction of b requires us to find a fixed point of the composition of the upward and downward data transfer expressions. Moreover, this fixed point needs to satisfy the guard of the interaction expression. Once we have computed this fixed point, we just use it as input to the interaction.

Since we use our own input at b instead of input obtained from a BIP component, we must hide b from the interface of the interaction. This explains why we call this operation abstraction.

Definition 2.4.2 (Abstraction on interaction models)**.** Let α be the BIP interaction expression $(\emptyset \leftarrow Q).[g : X_L := up(X_Q) \,//\, X_Q := dn(X_L)]$, and let $p \in Q$ be a bottom port of α. Let $ud_p(X_Q) = dn(up(X_Q))|_{x_p}$ be the restriction to x_p of the composition of up and dn. Denote the set of fixed points of the function $x_p \mapsto ud_p(x_p, X_{Q\setminus\{p\}})$ by F. Let $fp(X_{Q\setminus\{p\}}) \in F$ be any partial function that returns, when possible, any fixed point from F such that $g(x_p, X_{Q\setminus\{p\}})$. We call fp a *fixed point function* of α with respect to p. Then, we define the *abstraction* $\alpha \setminus p$ of α with respect to p as

$$(\emptyset \leftarrow Q \setminus \{p\}).[\exists x_p \in F.\, g : X_L := up(X_{Q\setminus\{p\}}, fp(X_{Q\setminus\{p\}})) \,//\, X_{Q\setminus\{p\}} := dn(X_L)]\,.$$

For convenience, we assume that a fixed point function is a random function. However, in practice we care only about the fact that this function returns a fixed point from F that satisfies the guard.

Example 2.4.4. Consider the BIP interaction expressions α_1 and α_2 from Example 2.4.2, and their shared bottom port b. We compute the abstraction $\alpha = (\alpha_1 * \alpha_2) \setminus b$. The mapping $ud_b : x_b \mapsto x_a$ gives the restriction to x_b of the composition of the upward and downward data transfer expressions. The set of fixed points of ud_b consists of $F = \{x_a\}$. Trivially, the guard of α equals $g_\alpha = \texttt{tt}$. Hence, the fixed point function of α is given by $fp(x_a, x_c) = x_a$. Therefore, we find that $\alpha = (\emptyset \leftarrow \{a, c\}).[\texttt{tt} : x_k := x_a; x_l := x_a \,//\, x_c := x_l]$.

We see that the value of x_a flows via x_b to x_c, which simulates the dataflow in the composition of the two Sync channels in Example 2.4.2. ◊

Example 2.4.5. Consider the composed BIP interaction expression $\alpha_1 * \alpha_2$ from Example 2.4.3 and its bottom port b. We compute the abstraction $\alpha = (\alpha_1 * \alpha_2) \setminus b$. The restriction to x_b of the composition of the upward and downward data transfer expressions is given by the mapping $ud_b : x_b \mapsto \max(x_a, x_b)$. The set of fixed points of ud_b is given by $F = \{v \mid v \geq x_a\}$. Since any $x_b \geq x_a, x_c$ can serve as a witness, the guard of α simplifies to $g_\alpha \equiv \exists x_b \geq x_a.(x_b \geq x_c) \vee (x_c \geq x_b \wedge x_b = x_c) \equiv \texttt{tt}$. Thus, the fixed point function $fp(x_a, x_c) = \text{rnd}(\{y \mid y \geq x_a, x_c\})$ may return any value greater than or equal to both x_a and x_c. Finally, we get that $(\alpha_1 * \alpha_2) \setminus b$ is given by

$$(\emptyset \leftarrow \{a, c\}).[\texttt{tt} : x_k := \max(x_a, r);\, x_l := \max(r, x_c) \,//\, x_a := x_k; x_c := x_l],$$

where $r = \text{rnd}(\{v \mid v \geq x_a, x_c\})$. Hence, since r is random, $(\alpha_1 * \alpha_2) \setminus b$ returns the value $\max(x_a, x_c) + C$, where $C \geq 0$ is an arbitrary positive number. ◊

2.4.3 Data-sensitive BIP architectures

The extension of BIP architectures to the data-sensitive domain requires us to combine data-agnostic BIP architectures with interaction expressions that are data-sensitive [ABB+14, BSBJ14].

First, we need to generalize the coordinating components in a BIP architecture. For this, we use a restricted type of constraint automata with polarity.

Definition 2.4.3 (Atomic BIP components)**.** An atomic BIP component is a constraint automaton $\mathcal{A}$ such that every transition $(q, N, g, q') \in \rightarrow$ synchronizes at most one bidirectional port, i.e., $N \in \{\emptyset, \{p!, p?\}\}$, for some bidirectional port p.

Coordinating components in data-agnostic BIP architectures are disconnected (cf. Definition 2.1.1). This notion lifts trivially to sets of atomic BIP components.

Next, we generalize the data-agnostic interaction model γ to a data-sensitive interaction model Γ. Every data-sensitive BIP interaction expression $\alpha \in \Gamma$ reduces to a data-agnostic interaction $N = \mathrm{bot}(\alpha) \in \gamma$.

Definition 2.4.4. A *data-sensitive BIP architecture* is a triple $A = (\mathcal{C}, P, \Gamma)$ consisting of a finite disconnected set $\mathcal{C}$ of atomic BIP components, a finite set P of ports, and an interaction model Γ over P (cf. Definition 2.1.1 and 2.1.6).

Using the operational semantics of atomic components, provided in [BSBJ14, Definition 3.2], and the interpretation $\mathtt{g}_2$ of a data-sensitive interaction model, defined in Section 2.3.1, we define the following semantics for data-sensitive BIP architectures:

Definition 2.4.5 (Semantics of data-sensitive BIP architecture)**.** Consider a data-sensitive BIP architecture $A = (\{C_1, \ldots, C_n\}, P, \Gamma)$. The semantics $\mathtt{g}_3(A)$ of A is given by the labelled transition system $(\prod_{i=1}^{n} Q_i, (\mathcal{D}+1)^{2P}, \rightarrow)$, where Q_i is the state space of atomic component C_i, and $\rightarrow$ is the smallest relation that satisfies the following rule: if $\delta : 2P \rightarrow \mathcal{D} + 1$ is a data assignment such that (q, δ, q) is a transition in $\mathtt{g}_2(\Gamma)$, and for all components C_i we have either

1. $q_i' = q_i$ and $\mathrm{dom}(\delta) \cap P_i = \emptyset$; or
2. (q_i, N, g, q_i') is a transition in C_i, $\mathrm{dom}(\delta) \cap P_i = N$, and $\delta \models g$,

then $(q_i)_{i=1}^n \xrightarrow{\delta} (q_i')_{i=1}^n$.

2.4.4 Composition of data-sensitive BIP architectures

Using the concepts introduced in Sections 2.4.1 and 2.4.2, we lift the composition operator of data-agnostic BIP architectures to data-sensitive BIP architectures.

Because the composition of coordinating components consists of set-union, its extension to data-sensitive BIP architectures is trivial. The composition of data-sensitive interaction models is less straightforward. Given two data-sensitive BIP interaction models Γ_1 and Γ_2, the composed data-sensitive interaction model Γ should intuitively consists of composed BIP interaction expressions $\alpha_1 * \alpha_2$, with $\alpha_i \in \Gamma_i$ for both i. However, we cannot allow every combination of α_1 and α_2, because they may synchronize on different shared ports.

Every BIP interaction expression α in the data-sensitive domain, reduces to a BIP interaction $\mathrm{bot}(\alpha)$ in the data-agnostic domain, where $\mathrm{bot}(\alpha)$ are the bottom ports of α. In this way, a BIP interaction model Γ reduces to a data-agnostic interaction model $\gamma = \{\mathrm{bot}(\alpha) \mid \alpha \in \Gamma\}$.

Let γ_1 and γ_2 be the reduced BIP interaction models derived from Γ_1 and Γ_2, and consider the BIP interactions $\mathrm{bot}(\alpha_1)$ and $\mathrm{bot}(\alpha_2)$ in γ_1 and γ_2. Let γ be the composition of γ_1 and γ_2. According to Definition 2.1.4, we have that $N = \mathrm{bot}(\alpha_1 * \alpha_2) \in \gamma$ if and only if $N \cap P_1 \in \gamma_1$ and $N \cap P_2 \in \gamma_2$. It is not hard to see that, in order to ensure that $\mathrm{bot}(\alpha_1 * \alpha_2) \in \gamma$, it suffices to assume that $\mathrm{bot}(\alpha_1) \cap P_2 = \mathrm{bot}(\alpha_2) \cap P_1$.

Definition 2.4.6 (Composition of data-sensitive BIP interaction models)**.** Let Γ_1 and Γ_2 be two interaction models with interfaces P_1 and P_2, respectively, such that no BIP interaction expression has top ports and no local variable is shared. We define the composition of Γ_1 and Γ_2 as $\Gamma_1 * \Gamma_2 = \{\alpha_1 * \alpha_2 \mid \alpha_i \in \Gamma_i, \mathrm{bot}(\alpha_1) \cap P_2 = \mathrm{bot}(\alpha_2) \cap P_1\}$.

Notice that the restriction to interaction expressions that do not have top ports implies that the condition in Definition 2.4.1, which requires that the downward data transfer do not depend on top ports, is trivially satisfied. Hence, the composition operator on data-sensitive BIP interaction models is well-defined.

Moreover, notice that it does not make sense to weaken the condition $\mathrm{bot}(\alpha_1) \cap P_2 = \mathrm{bot}(\alpha_2) \cap P_1$ any further. Suppose that α_1 and α_2 satisfy only $\mathrm{bot}(\alpha_1 * \alpha_2) \cap P_i \in \gamma_i$, for $i = 1, 2$. Then we find $\alpha_1' \in \Gamma_1$ and $\alpha_2' \in \Gamma_2$ such that $\mathrm{bot}(\alpha_1' * \alpha_2') = \mathrm{bot}(\alpha_1 * \alpha_2)$. Although, $\alpha_1' * \alpha_2'$ and $\alpha_1 * \alpha_2$ extend the same data-agnostic interaction, they may behave very differently with respect to data.

Now, Definition 2.4.6 allows us to define our desired composition operator for data-sensitive BIP architectures.

Definition 2.4.7 (Composition of data-sensitive BIP architectures)**.** Let $A_1 = (\mathcal{C}_1, P_1, \Gamma_1)$ and $A_2 = (\mathcal{C}_2, P_2, \Gamma_2)$ be two data sensitive BIP architectures such that $\mathcal{C}_1 \cup \mathcal{C}_2$ is disconnected and no BIP interaction expression has top ports and A_1 and A_2 share no local variables. Then, we define the composition $A_1 \oplus A_2$ as $(\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \Gamma_1 * \Gamma_2)$.

The composition of data-sensitive BIP interaction models in Definition 2.4.6 can cause an interaction-space explosion. Such an explosion can never occur using hierarchical composition only [BSBJ14]. This makes the data-sensitive BIP architecture composition more expressive than hierarchical composition.

Example 2.4.6. Consider a Reo connector that consist of N parallel Sync channels, i.e., we have a Sync channel $\mathcal{R}_{a_i,b_i}$ from a_i to b_i, for each $i \in \{1, \ldots, N\}$. Since any combination of Sync channels can fire, the associated constraint automaton exhibits 2^N transitions. The direct translation from Section 2.3 requires us to translate every transition into a corresponding BIP interaction expression.

Using BIP architecture composition from Definition 2.4.7, it suffices to translate each Sync channel $\mathcal{R}_{a_i,b_i}$ into a BIP architecture $A_{a_i,b_i} = (\emptyset, \{a_i, b_i\}, \{\alpha_{a_i \to b_i}, \alpha_\emptyset\})$, where $\alpha_{a_i \to b_i} = (\emptyset \leftarrow \{a_i, b_i\}).[\mathtt{tt} : x_l := x_{a_i} \,//\, x_{b_i} := x_l]$ models the Sync channel and $\alpha_\emptyset = (\emptyset \leftarrow \emptyset).[\mathtt{tt} : - \,//\, -]$ models the empty transition. This empty interaction allows the other BIP architectures to proceed independently of this

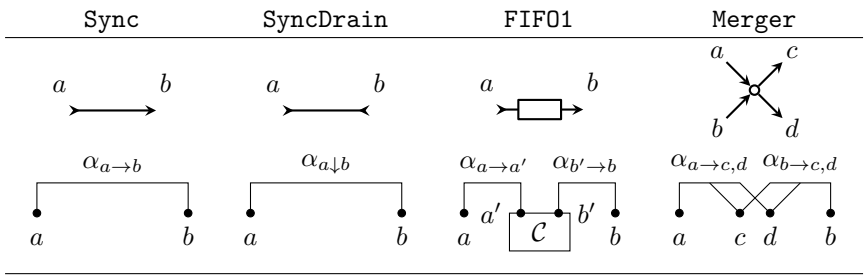


Figure 2.12: Translation of Reo channels and nodes to data-sensitive BIP architectures. The BIP interaction expressions are given by $\alpha_{a\to b,c} = (\emptyset \leftarrow \{a, b, c\}).[\texttt{tt} : x_l := x_a \,//\, x_b, x_c := x_l]$, $\alpha_{a\to b} = \alpha_{a\to b,b}$, and $\alpha_{a\downarrow b} = (\emptyset \leftarrow \{a, b\}).[\texttt{tt} : - \,//\, -]$. The atomic BIP component $\mathcal{C}$ models the behavior of the FIFO_1 channel.

BIP architecture. Hence, Definition 2.4.7 enables us to translate only N channels instead of 2^N transitions. ◊

Definition 2.4.8 (Abstraction of data-sensitive BIP architectures)**.** Let $A = (\mathcal{C}, P, \Gamma)$ be a data-sensitive BIP architecture, and $p \in P$ a dangling port (i.e., $p \notin P_C$, for all $C \in \mathcal{C}$). Then, we define the abstraction $A \setminus p$ as $(\mathcal{C}, P \setminus \{p\}, \{\alpha \setminus p \mid \alpha \in \Gamma\})$.

2.4.5 Incremental translation

The proposed composition operator from Definition 2.4.7 together with the abstraction operator from Definition 2.4.2 allow us to incrementally translate constraint automata to data-sensitive BIP architectures and vice versa. We formalize this by defining two translations, and show that they both preserve the semantics of translated entities.

Reo to BIP Consider a Reo circuit $\mathcal{R}$, and associate to each channel and node in $\mathcal{R}$ its constraint automaton (see Figure 2.2). Rename every input port p of any channel or node in $\mathcal{R}$ to $p!$, and every output port of any channel or node in $\mathcal{R}$ to $p?$. This procedure splits every shared port p into two ports $p!$ and $p?$, which essentially disconnects all channels and nodes. Write $X = \{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ for the obtained set of constraint automata with polarity. Our goal is to translate each $\mathcal{A}_i \in X$ individually to a data-sensitive BIP architecture, and then compose them using Definitions 2.4.2 and 2.4.7. To this end, we define the translation $\mathsf{bip}_3(\mathcal{A})$ of a BIP-friendly constraint automaton with polarity $\mathcal{A}$.

Let $\mathcal{A}$ be a constraint automaton with polarity over P, which means that $\mathcal{A}$ uses names from $2P = \{p!, p? \mid p \in P\}$. Since atomic components are not allowed to synchronize their ports and since interaction in BIP is stateless, we need to assume that $\mathcal{A}$ is *BIP-friendly*: $\mathcal{A}$ is either stateless (i.e., $Q_{\mathcal{A}} = \{q\}$) or does not synchronize any of its ports (i.e., for every transition (q, N, g, q') we have $N = \{p!, p?\}$ for some $p \in P$). Figure 2.2 shows some examples of BIP-friendly automata.

When $\mathcal{A}$ is stateless, we can translate $\mathcal{A}$ into an interaction model $\mathsf{bip}_2(\mathcal{A})$. We now simply define $\mathsf{bip}_3(\mathcal{A}) = (\emptyset, P, \mathsf{bip}_2(\mathcal{A}))$. See Figure 2.12 for an example. When

$\mathcal{A}$ does not synchronize any of its ports, we can interpret $\mathcal{A}$ as an atomic component $\mathcal{A}'$, where we rename every port $p \in P$ to a port $p' \in P'$. The prime is used only to construct a fresh port name. Now, we interpret every $p \in P$ as a dangling port of the translated data-sensitive BIP architecture and connect p with p' using the interaction $\alpha_{p,p'} = (\emptyset \leftarrow \{p, p'\}).[\mathtt{tt} : x_k := x_p; x_l := x_{p'} \,/\!/\, x_p := x_l; x_{p'} := x_k]$. Thus, we define

$$\mathsf{bip}_3(\mathcal{A}) = \begin{cases} (\emptyset, P, \mathsf{bip}_2(\mathcal{A})) & \text{if } \mathcal{A} \text{ is stateless} \\ (\{\mathcal{A}'\}, P \cup P', \{\alpha_{p,p'} \mid p \in P\}) & \text{if } \mathcal{A} \text{ is non-synchronizing} \end{cases} \tag{2.10}$$

The restriction that the automaton $\mathcal{A}$ should be either stateless or non-synchronizing is not problematic. Every synchronizing stateful automaton $\mathcal{A}$ can be decomposed into a set $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ of stateless and non-synchronizing automata [BKK14]. Indeed, each automaton in the decomposition is the CA representation of a stateless Reo channel or a FIFO_1 buffer.

Using the translation bip_3, we can now translate the Reo circuit $\mathcal{R}$ incrementally. Let $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ be a set of BIP-friendly constraint automata with polarity and $S = \{p \mid \{p!, p?\} \cap \mathcal{N}_{\mathcal{A}_i} \cap \mathcal{N}_{\mathcal{A}_j} \neq \emptyset \text{ for some distinct } i, j\}$ be the set of shared/internal ports of this system of automata. The following diagram illustrates the working of the incremental translation from Reo to BIP:

$$\begin{array}{ccccc} \{\mathcal{A}_1, \ldots, \mathcal{A}_m\} & & \xrightarrow{\quad\mathsf{bip}_3\quad} & & \{\mathsf{bip}_3(\mathcal{A}_1), \ldots, \mathsf{bip}_3(\mathcal{A}_m)\} \\ \downarrow & & & & \downarrow \\ \exists 2S(\mathcal{A}_1 \bowtie \cdots \mathcal{A}_m \bowtie \mathcal{G}) & \xrightarrow{\mathsf{f}_3} & L & \xleftarrow{\mathsf{g}_3} & (\mathsf{bip}_3(\mathcal{A}_1) \oplus \cdots \oplus \mathsf{bip}_3(\mathcal{A}_m)) \setminus S \end{array} \tag{2.11}$$

Here, f_3 is the canonical extension of f_2 defined in equation Equation (2.7), $- \setminus S$ is the abstraction operator defined in Definition 2.4.8, and $\mathcal{G}$ is a stateless gluing automaton that for every subset $P \subseteq S$ of internal ports, has a transition with synchronization constraint $N = \{p!, p? \mid p \in P\}$ and data constraint $g \equiv \bigwedge_{p \in P} d_{p!} = d_{p?}$. Observe that $\mathcal{G}$ essentially models all Sync channels from $p?$ to $p!$ for every $p \in S$. In this way, we reconnect the nodes that were split by our encoding of polarity.

Example 2.4.7. Let $\mathcal{R}$ be the sequential composition of two Sync channels, i.e., $\mathcal{R} = \mathcal{R}_{a,b} \bowtie \mathcal{R}_{b,c}$ where $\mathcal{R}_{x,y}$ is a Sync channel from x to y. First, we associate to $\mathcal{R}_{x,y}$ its constraint automaton with polarity

$$\mathcal{A}_{x,y} = (\{q\}, \{x!, x?, y!, y?\}, \{(q, \{x!, x?, y!, y?\}, d_{x!} = d_{y?}, q)\}, q).$$

Thus, we represent $\mathcal{R}$ by $\{\mathcal{A}_{a,b}, \mathcal{A}_{b,c}\}$. To reconnect the channel ends $b!$ and $b?$, we add a stateless gluing automaton $\mathcal{G}$ with a single transition that has a synchronization constraint $N = \{b?, b!\}$ and data-constraint $g \equiv d_{b?} = d_{b!}$. So now, the semantics of $\mathcal{R}$ is given by $\mathsf{f}_3(\exists b! \exists b?(\mathcal{A}_{a,b} \bowtie \mathcal{A}_{b,c} \bowtie \mathcal{G}))$ and consists of a stateless labelled transition system that encodes that for every observed $\delta : 2\{a, c\} \to \mathcal{D}$, we have $\delta(a!) = \delta(c?)$.

Using the incremental translation from Diagram 2.11 and α_1 and α_2 from Example 2.4.2, we obtain data-sensitive BIP architectures $\mathsf{bip}_3(\mathcal{A}_{a,b})$ and $\mathsf{bip}_3(\mathcal{A}_{b,c})$

given by $(\emptyset, \{a, b\}, \{\alpha_1\})$ and $(\emptyset, \{b, c\}, \{\alpha_2\})$, respectively. Note that b is the only internal node in $\mathcal{R}$, hence $S = \{b\}$. Now, Example 2.4.4 shows that the system $\{\mathsf{bip}_3(\mathcal{A}_{a,b}), \mathsf{bip}_3(\mathcal{A}_{b,c})\}$ composes into a single BIP architecture A given by $(\emptyset, \{a, c\}, \{(\alpha_1 * \alpha_2) \setminus b\})$. It is now easy to see that $\mathsf{f}_3(\exists b! \exists b?(\mathcal{A}_{a,b} \bowtie \mathcal{A}_{b,c} \bowtie \mathcal{G}))$ and $\mathsf{g}_3(A)$ are bisimilar. $\Diamond$

In the previous example, we stated that the incremental translation from Diagram 2.11 preserves bisimilarity, but in fact, it preserves even a stronger equivalence: isomorphism. Informally, labelled transition systems are isomorphic if their transition relations are identical modulo state renaming. Consequently, isomorphism implies bisimilarity.

Definition 2.4.9 (Isomorphism)**.** If $L_i = (Q_i, (\mathcal{D}+1)^{2P_i}, \rightarrow_i, q_i^0) \in \text{LTS}$, $i = 1, 2$, then L_1 and L_2 are *isomorphic* iff $P_1 = P_2$ and there exists a bijective function f mapping states from Q_0 to Q_1 such that $f(q_0^0) = q_1^0$ and $q_0 \xrightarrow{\delta}_0 q_0'$, for some $q_0, q_0' \in Q_0$, if and only if $f(q_0) \xrightarrow{\delta}_1 f(q_0')$.

Theorem 2.4.1. *Translation* bip_3 *is correct and compositional, i.e., Diagram 2.11 commutes modulo isomorphism of labelled transition systems.*

Proof. Let $\mathcal{A}_i = (Q_i, \mathcal{N}_i, \rightarrow_i, q_{0i})$, for $i \in \{1, \ldots, m\}$, be BIP-friendly constraint automata with polarity, and let $S = \{p \mid \{p!, p?\} \cap \mathcal{N}_i \cap \mathcal{N}_j \neq \emptyset, \text{ with } i \neq j\}$ be the set of shared ports. The state space of $\mathsf{f}_3(\exists 2S(\mathcal{A}_1 \bowtie \cdots \bowtie \mathcal{A}_m \bowtie \mathcal{G}))$ equals $Q_1 \times \cdots \times Q_m \times \{q_\mathcal{G}\}$, and the state space of $\mathsf{g}_3((\mathsf{bip}_3(\mathcal{A}_1) \oplus \ldots \oplus \mathsf{bip}_3(\mathcal{A}_m)) \setminus S)$ equals $\prod_{j \in J} Q_j$, where $J \subseteq \{1, \ldots, m\}$ is the set of indices of the BIP-friendly components that are non-synchronizing. We show that the mapping $(q_1, \ldots, q_m, q_\mathcal{G}) \mapsto (q_i)_{i \in J}$ constitutes an isomorphism between $K = \mathsf{f}_3(\exists 2S(\mathcal{A}_1 \bowtie \cdots \bowtie \mathcal{A}_m \bowtie \mathcal{G}))$ and $L = \mathsf{g}_3((\mathsf{bip}_3(\mathcal{A}_1) \oplus \ldots \oplus \mathsf{bip}_3(\mathcal{A}_m)) \setminus S)$.

Let $\tau = ((q_1, \ldots, q_m, q_\mathcal{G}), \delta, (q_1', \ldots, q_m', q_\mathcal{G}))$ be a transition in K. Using Definition 2.1.9, if follows that τ is in K if and only if there exists an extension $\delta' : \bigcup_i 2\mathcal{N}_i \to \mathcal{D} + 1$ of δ with $\delta'(p) = \delta(p)$ for all $p \in (\bigcup_i 2\mathcal{N}_i) \setminus 2S$ such that $((q_1, \ldots, q_m, q_\mathcal{G}), \delta', (q_1', \ldots, q_m', q_\mathcal{G}))$ is a transition in $\mathsf{f}_3(\mathcal{A}_1 \bowtie \cdots \bowtie \mathcal{A}_m \bowtie \mathcal{G})$. Write $\delta'|_{2\mathcal{N}_i}$ for the restriction of δ' to $2\mathcal{N}_i$. Using Definition 2.1.8, it follows that τ is in K if and only if $\tau_i = (q_i, \delta'|_{2\mathcal{N}_i}, q_i')$ is a transition in $\mathsf{f}_3(\mathcal{A}_i)$ or $\text{dom}(\delta') \cap 2\mathcal{N}_i = \emptyset$ and $q_i' = q_i$, for all $i \in \{1, \ldots, m\}$, and $\delta'(p!) = \delta'(p?)$, for all $p \in S$, due to the gluing automaton $\mathcal{G}$. Using equations Equation (2.10) and Equation (2.8), we have that τ is in K if and only if $\mathsf{g}_3(\mathsf{bip}_3(\mathcal{A}_i))$ has a transition τ_i or $\text{dom}(\delta') \cap \mathcal{N}_i = \emptyset$ and $q_i' = q_i$, for all $i \in \{1, \ldots, m\}$, and $\delta'(p!) = \delta'(p?)$, for all $p \in S$. By the definition of the composition operator on data-sensitive BIP architectures in Definition 2.4.7 and the definition of g_3 in Definition 2.4.5, it follows that τ is in K if and only if $((q_i)_{i \in J}, \delta', (q_i')_{i \in J})$ is a transition in $\mathsf{g}_3(\mathsf{bip}_3(\mathcal{A}_1) \oplus \ldots \oplus \mathsf{bip}_3(\mathcal{A}_m))$ and $\delta'(p!) = \delta'(p?)$, for all $p \in S$. Using the abstraction operator in Definition 2.4.2, it follows that τ is in K if and only if $((q_i)_{i \in J}, \delta, (q_i')_{i \in J})$ is a transition in L. Since $\mapsto$ trivially preserves initial states, we conclude that $\mapsto$ is an isomorphism which proves the theorem. $\square$

Applying Theorem 2.4.1 for $m = 1$, we obtain, since $S = \emptyset$, correctness of bip_3.

Corollary. $\mathsf{g}_3(\mathsf{bip}_3(\mathcal{A})) \cong \mathsf{f}_3(\mathcal{A})$*, for all CA with polarity* $\mathcal{A}$*.*

BIP to Reo Let $\{A_1, \ldots, A_n\}$ be a set of data-sensitive BIP architectures, and assume no two atomic components share a port. Our goal is to translate the composition $A_1 \oplus \cdots \oplus A_n$ to a constraint automaton with polarity by translating each BIP architecture A_i individually. To this end, we extend the translation reo_2 to data-sensitive BIP architectures.

Let $A = (\{C_1, \ldots, C_n\}, P, \Gamma)$ be a data-sensitive BIP architecture. Trivially, every atomic component C_i constitutes a constraint automaton with polarity. By reusing our translation reo_2, we define

$$\mathsf{reo}_3(A) = \mathsf{reo}_2(\Gamma) \bowtie \prod_{i=1}^{n} C_i. \tag{2.12}$$

Let $\{A_1, \ldots, A_n\}$ be a set of data-sensitive BIP architectures, and assume no two atomic components share a port. The following diagram illustrates the working of the incremental translation from BIP to Reo:

$$\begin{array}{ccccc} \{A_1, \ldots, A_n\} & & \xrightarrow{\quad\mathsf{reo}_3\quad} & & \{\mathsf{reo}_3(A_1), \ldots, \mathsf{reo}_3(A_n)\} \\ \downarrow & & & & \downarrow \\ A_1 \oplus \cdots \oplus A_n & \xrightarrow{\mathsf{g}_3} & L & \xleftarrow{\mathsf{f}_3} & \mathsf{reo}_3(A_1) \bowtie \cdots \bowtie \mathsf{reo}_3(A_n) \end{array} \tag{2.13}$$

Example 2.4.8. Consider the atomic component $C_{42} = (\{q\}, \{b!, b?\}, \rightarrow, q)$, with $\rightarrow = \{(q, \{b!, b?\}, d_{b!} = 42, q)\}$, and let α_1 and α_2 be the BIP interaction expressions from Example 2.4.3. Now, consider the data-sensitive BIP architectures $A_1 = (\{C_{42}\}, \{a, b\}, \{\alpha_1\})$ and $A_2 = (\emptyset, \{b, c\}, \{\alpha_2\})$ over the data domain $\mathcal{D} = \{0, \ldots, 2^{32} - 1\}$. Then, $\mathsf{g}_3(A_1 \oplus A_2)$ is given by a stateless labelled transition system that encodes that for every observed $\delta : 2\{a, b, c\} \to \mathcal{D}$ we have $\delta(a?) = \max(\delta(a!), \delta(b!))$, $\delta(c?) = \max(\delta(b!), \delta(c!))$, $\delta(a?) = \delta(b?) = \delta(c?)$, and $\delta(b!) = 42$. Using Example 2.3.3, it follows that $\mathsf{f}_3(\mathsf{reo}_3(A_1) \bowtie \mathsf{reo}_3(A_2))$, which is equal to $\mathsf{f}_3(\mathsf{reo}_2(\{\alpha_1\}) \bowtie C_{42} \bowtie \mathsf{reo}_2(\{\alpha_2\}))$, amounts to a labelled transition system that is bisimilar to $\mathsf{g}_3(A_1 \oplus A_2)$. ◊

Theorem 2.4.2. *Translation* reo_3 *is correct and compositional, i.e., Diagram 2.13 commutes modulo isomorphism of labelled transition systems.*

Proof. Let $\{A_1, \ldots, A_n\}$ be a set of data-sensitive BIP architectures such that no two atomic components share a port. The state space of $\mathsf{g}_3(A_1 \oplus \cdots \oplus A_n)$ equals $\prod_{C \in \mathcal{C}} Q_C$, where $\mathcal{C} = \bigcup_i \mathcal{C}_{A_i}$ are the atomic components of $A_1 \oplus \cdots \oplus A_n$. The state space of $\mathsf{f}_3(\mathsf{reo}_3(A_1) \bowtie \ldots \bowtie \mathsf{reo}_3(A_n))$ equals $\{q\} \times \prod_{i=1}^{n} \prod_{C \in \mathcal{C}_{A_i}} Q_C$, where $\mathcal{C}_{A_i}$ is the set of atomic components of A_i. We show that the mapping $(q_C)_{C \in \mathcal{C}} \mapsto (q, (q_C)_{C \in \mathcal{C}_{A_i}})_{i=1}^{n}$ constitutes an isomorphism between $K = \mathsf{g}_3(A_1 \oplus \cdots \oplus A_n)$ and $L = \mathsf{f}_3(\mathsf{reo}_3(A_1) \bowtie \ldots \bowtie \mathsf{reo}_3(A_n))$.

Let $\tau = ((q_C)_{C \in \mathcal{C}}, \delta, (q'_C)_{C \in \mathcal{C}})$ be a transition in K. By definition of g_3 in Definition 2.4.5, it follows that τ is in K if and only if δ is accepted by the composed BIP interaction model Γ and $(q_C, \delta|_{P_C}, q'_C)$ in $\mathsf{f}_3(C)$ or $\mathrm{dom}(\delta) \cap P_C = \emptyset$ and $q_C = q'_C$ for all atomic components $C \in \mathcal{C}$. By definition of the composition operator on data-sensitive BIP architectures in Definition 2.4.7, it follows that τ is in K if and only if, for all $i \in \{1, \ldots, n\}$, the following conditions are satisfied: $(q, \delta|_{P_{A_i}}, q)$ is a transition in $\mathsf{g}_2(\Gamma_i)$, with Γ_i the BIP interaction model of A_i,

and $(q_C, \delta|_{P_C}, q'_C)$ in $\mathsf{f}_3(C)$ or $\mathrm{dom}(\delta) \cap P_C = \emptyset$ and $q_C = q'_C$, for all atomic components $C \in \mathcal{C}_{A_i}$. Since $\mathsf{g}_2(\Gamma_i) \cong \mathsf{f}_2(\mathsf{reo}_2(\Gamma_i))$ by Theorem 2.3.1, we conclude that τ is in K if and only if $((q, (q_C)_{C\in\mathcal{C}_{A_i}}), \delta|_{P_{A_i}}, (q, (q'_C)_{C\in\mathcal{C}_{A_i}}))$ is a transition in $\mathsf{f}_3(\mathsf{reo}_3(A_i))$. Using Definition 2.1.8, it follows that that τ is in K if and only if $((q, (q_C)_{C\in\mathcal{C}_{A_i}})_{i=1}^n, \delta, (q, (q'_C)_{C\in\mathcal{C}_{A_i}})_{i=1}^n)$ is a transition in L. Since $\mapsto$ trivially preserves initial states, we conclude that $\mapsto$ is an isomorphism, which proves the theorem. □

By applying Theorem 2.4.2 for $n = 1$, we obtain correctness of reo_3.

Corollary. $\mathsf{f}_3(\mathsf{reo}_3(A)) \cong \mathsf{g}_3(A)$*, for all data-sensitive BIP architectures* A*.*

Thus, Theorems 2.4.1 and 2.4.2 show how our proposed composition operator of Definition 2.4.7 enables us to translate between Reo connectors, modeled by constraint automata with polarity, and data-sensitive BIP architectures.

2.5 Related work

Instead of using labelled transition systems as common semantics (Figures 2.7 and 2.9), we may also choose another model for concurrent systems. The Tile Model offers such an alternative semantics for concurrent systems [GM00]. The basic idea is to associate an m-tuple of terms in n variables $(s_i(x_1, \ldots, x_n))_{i=1}^m$ over the term algebra with signature Σ to an arrow $s : \underline{n} \to \underline{m}$ in the graph with nodes from $\underline{\mathbb{N}}$. Every function symbol $s \in \Sigma$ with arity n is interpreted as an arrow $s : \underline{n} \to \underline{1}$. As Plotkin's structural operational semantics uses terms in an algebra to represent the state of a system, the Tile Model uses the arrows $s : \underline{n} \to \underline{m}$ to describe the configuration of a concurrent system. Transitions from one configuration to another are formulated by means of tiles. A tile α (denoted by $\alpha : s \overset{a}{\underset{b}{\to}} t$) is a diagram

$$\begin{array}{ccc} \underline{n} & \xrightarrow{\ s\ } & \underline{m} \\ a\downarrow & \alpha & \downarrow b \\ \underline{p} & \xrightarrow[\ t\ ]{} & \underline{q} \end{array} \tag{2.14}$$

that represents a rewriting rule that states that trigger a can transform initial configuration s into the final configuration t and produce effect b. The trigger a and effect b are called the observations of α. Tiles may be composed horizontally, vertically, and in parallel, using the monoidal operator $\otimes$ on $\underline{\mathbb{N}}$ given by $\underline{n} \otimes \underline{m} = \underline{n + m}$.

A configuration can be seen as a connector. In this view, the source $\underline{n}$ and target $\underline{m}$ of a configuration $s : \underline{n} \to \underline{m}$ correspond to the interface of the connector. Since the interfaces $\underline{p}$ and $\underline{q}$ in diagram Equation (2.14) may differ from $\underline{n}$ and $\underline{m}$, the Tile Model provides a natural semantics for dynamic reconfiguration in Reo [ABC+08].

Bruni et al. show that Petri nets with boundaries are equally expressive as BIP without priorities [BMM11]. They showed that this formal correspondence indirectly relates BIP to the Tile Model, which resulted in the definition of the Petri calculus. Since boundaries are mainly used for composition, the monolithic

translation by Bruni et al. encodes BIP without priorities into Petri nets without boundaries. A similar encoding exists for Reo, which translates port automata into Petri nets [Kra09].

An indirect comparison of BIP and Reo, in the data-agnostic domain, through their respective comparisons with other models, e.g., Petri nets, is certainly possible. Nevertheless, the direct and formal translations we present in this chapter allow direct translation tools between BIP and Reo, that are otherwise difficult, if not impossible, to construct based on such indirect comparisons.

Beside BIP and Reo, there are many other examples of coordination languages [PA98]. Their relations with BIP and Reo have been studied by others. For instance, Proença and Clarke provide a detailed comparison between Orc and Reo [PC08], Chkouri et al. present a translation of AADL into BIP [CRBS08], and Talcott et al. connect both ARC and PBRD to Reo by providing mappings between their semantic models [TSR11].

2.6 Discussion

In the data-agnostic domain, we showed that BIP architectures and port automata coincide modulo internal transitions, witnessed by the weak simulation in Theorem 2.2.2, and independent progress, witnessed by the condition $\emptyset \in \gamma_1 \cup \gamma_2$ in Theorem 2.2.4. In the data-sensitive domain, we showed by Theorem 2.3.1 that the observable behavior of BIP interaction models and stateless constraint automata is identical. We extended the notion of a data-agnostic BIP architecture to the data-sensitive domain (Definition 2.4.4), and showed that these data-sensitive BIP architectures correspond to constraint automata with polarity (Corollaries 2.4.5 and 2.4.5).

Our formal correspondences between BIP and Reo reveal differences and similarities of their fundamental design principles. One similarity is that both BIP and Reo provide constructs that allow high-level specification of multiparty synchronization, such as a barrier synchronization. Although multiparty synchronization is used in several approaches, such as the bulk-synchronous parallel (BSP) model [Val90] or the Parameterized Networks of Synchronized Automata (pNets) [BAC$^+$09], most of the process algebras lack this feature, expressing multiparty synchronization by a cluttered composition of binary synchronizations. Exceptions include Winskel's synchronization algebra [WN95] and Bergstra & Klop's algebra of communicating processes (ACP) [BK85]. Controlling and constraining multiparty synchronization is, however, more complex in ACP than it is in BIP and Reo (because additional operators, communication and block, need to be used beside parallel composition to specify admissible synchronizations). This is illustrated in work by Krause et al. [KKdV12], who encoded Reo's semantics (i.e., Reo's composition operator and a number of primitives) in mCRL2 [CGK$^+$13], a modern process specification language based on ACP.

The focus of this chapter is on formal relations between BIP and Reo. As such, detailed comparison of BIP or Reo with process algebras or other models that support multi-party synchronization is beyond our scope. However, support for multiparty synchronization in some other models, and the consensus in BIP and Reo to support this notion through first-order constructs confirms the practical

significance of this concept.

On the other hand, BIP and Reo treat the separation between computation and coordination differently. The BIP framework concretely *defines* what separates computation (BIP behavior) from coordination (BIP interaction), while Reo merely *separates* computation (Reo components) and coordination (Reo connector) structurally. Indeed, Reo does not force a fixed universal definition for computation and coordination in all applications. Without giving a fixed definition of separation criterion, Reo's structural separation of computation from coordination (i.e., component versus connector) simply means that, while this separation is always important, the distinction between the two is in the eye of the beholder: in different applications, different, or even the same people, may find it convenient to draw the line that separates computation and coordination at different places to suit their needs. For example, the stateful behavior of a FIFO with capacity of 1 strictly places what this entity does in the behavior layer of BIP, as a (computation) component. In Reo, such stateful components can, of course, be regarded and used as computation as well. However, when deemed appropriate, one can use the same component (i.e., a FIFO_1 channel) in the construction of a Reo connector as well, e.g., to express the stateful, turn-taking interaction between two components, as in Figure 2.4.

The property-preserving translations presented in this chapter enable us to lift the composition operator for data-sensitive Reo circuits to BIP architectures. Besides lifting theoretical results, it seems natural to investigate whether it is possible to transfer also other techniques, such as those used in compilation and model checking. For example, Reo's compositional approach to code generation [Jon16] may yield a very different distributed implementation of a BIP system. Comparing the performance of such a postulated implementation of BIP, can reveal valuable insights for compilation.

The results in this chapter show that both BIP and Reo can be interpreted as a variant of labeled transitions sytems (LTS). In the sequel of this thesis, we work for the large part with the semantics of coordination languages. In fact, one of our main contributions consists in developing coordination language semantics that improve expressiveness (Chapters 3 and 7) and tooling (Chapters 5 and 6). The only exception is Chapter 4, where we develop a syntax to build actual systems with these new semantics. Here, we adopt Reo's compositionally principle as a basis of our syntax.