Partially-Distributed Coordination with Reo (Technical Report)

Sung-Shik T.Q. Jongmans
Formal Methods
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
Email: Sung.Jongmans@cwi.nl

Francesco Santini
Contraintes
INRIA Paris-Rocquencourt
Le Chesnay, France
Email: Francesco.Santini@inria.fr

Farhad Arbab
Formal Methods
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
Email: Farhad.Arbab@cwi.nl

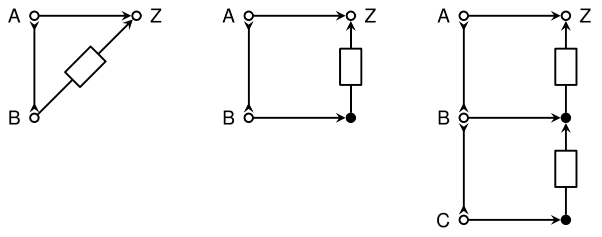
***Abstract*—Coordination languages, as Reo, have emerged for the specification and implementation of interaction protocols among concurrent entities. In this paper, we propose a framework for generating partially-distributed, partially-centralized implementations of Reo connectors to improve 1) build-time compilation and 2) run-time throughput and parallelism. Our framework relies on the definition of a new formal product operator on constraint automata (Reo's formal semantics), which enables the formally correct distribution of disjoint parts of a coordination scheme over different machines according to several possible motivations (e.g., performance, privacy, QoS constraints, resource availability, network topology). First, we describe the design and a proof-of-concept implementation of our framework. Then, in a case study, we show and explain how a generated connector implementation can be executed in the Cloud and supports Big Data coordination.**

***Keywords*-Reo coordination language; distributed computation; Web services; Cloud; Big Data;**

I. Introduction

A. Context

Coordination languages have emerged for the specification and implementation of interaction protocols among concurrent entities (components, services, threads, etc.). This class of languages includes Reo [1, 2], a graphical dataflow language for compositional construction of *connectors*: communication media through which entities can interact with each other. Figure 1 shows example connectors in their usual graphical syntax. Briefly, connectors consist of one or more *channels*, through which data items flow, and a number of (named) *nodes*, on which channel ends coincide. Through connector *composition* (the act of gluing connectors together on their shared nodes), developers can construct arbitrarily complex connectors. While nodes have fixed dataflow behavior, Reo features an open-ended set of channels: developers can define their own channels with custom semantics.



(a) Alternator (b) Alternator$_2$ (c) Alternator$_3$

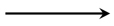
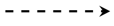
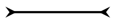
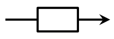
Figure 1: Example connectors: the Alternator family.

Compared to general-purpose languages, Reo has several software engineering advantages as a domain-specific language for programming interaction protocols [3]. For instance, the use of Reo forces developers to separate their computation code from their protocol code instead of intermixing computations with protocols (as usually done when both are implemented in the same language). This separation facilitates verbatim reuse, independent modification, and compositional construction of protocol implementations (i.e., Reo connectors) in a straightforward way. Moreover, Reo has a formal foundation, which enables formal analysis of connectors (e.g., model checking [4]). This makes statically verifying that a given protocol does not deadlock, for instance, relatively easy. Such protocol analyses are much harder on lower-level general-purpose languages, especially when computations and protocols are intermixed.

B. Problem

To use connectors in real applications, one must derive implementations from their graphical specification, as precompiled executable code or using a run-time interpretation engine. Roughly two implementation approaches currently exist. In the *distributed approach* [5, 6, 7], one implements the behavior of each of the k constituents of a connector (its nodes and channels) and runs these k implementations concurrently as a distributed system; in the *centralized approach* [3, 8], one computes the behavior of a connector as a whole, implements this behavior, and runs this implementation sequentially as a centralized system. The distributed approach has the advantage of fast compilation at build-time and high parallelism at run-time. However, this comes at the cost of reduced throughput at run-time (because of communication necessary for executing distributed algorithms). In contrast, the centralized approach has the advantage of high throughput at run-time but at the cost of slow compilation and reduced parallelism. Moreover, the amount of code generated in the centralized approach may

Table I: Syntax and semantics of common channels.

Name	*Graphical syntax*	*Semantics*
sync		Atomically takes a data item d from its source end and writes d to its sink end.
lossysync-nd		Atomically takes a data item d from its source end and nondeterministically either writes d to its sink end or loses d.
syncdrain		Atomically takes data items from both its source ends and loses them.
fifo		Takes a data item d from its source end, then writes d to its sink end.

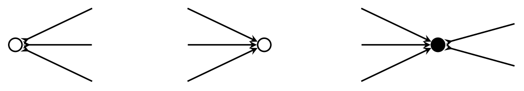

(a) Source (b) Sink (c) Mixed

Figure 2: Node types.

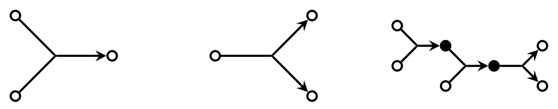

(a) Merger (b) Replicator (c) Node

Figure 3: Merger, Replicator, and Node for $n = 3$ and $m = 2$.

be exponential in k, in which case the output is prohibitively big and the time to produce it prohibitively long. Proença et al. observe that a partially-distributed *hybrid approach*, where parts of a connector are compiled according to the centralized approach and deployed in a distributed fashion, is generally ideal [6, 7]: a hybrid approach strikes a middle ground between throughput and parallelism at run-time while achieving reasonably fast compilation at build-time.

In this paper, we address the problem that no systematic, formally justified way of automatically constructing connector implementations according to the hybrid approach exists.

C. Contribution

We use Reo's notion of *(a)synchronous regions* and an extended recent result on connector composition to automatically construct formally sound hybrid connector implementations (using *constraint automata* as our semantic formalism [9]). We implement this construction on top of an existing Reo-to-Java centralized-code generator [3]. This enables a case study on *distributed service orchestration* by reusing an existing Reo-based orchestration framework [8]. This case study shows that the hybrid approach can improve the centralized approach not only in terms of run-time parallelism but also throughput.

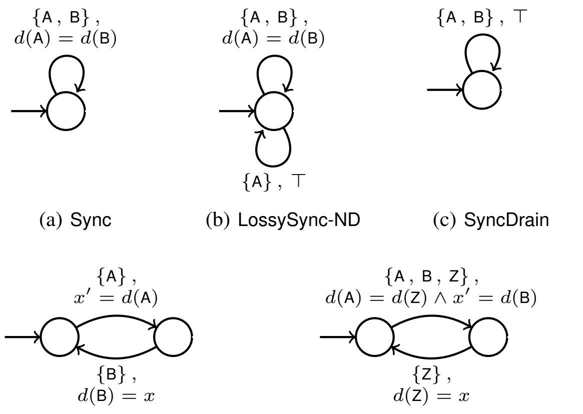
We organize this paper as follows. In Sec. II, we give a concise overview of Reo. In Sec. III, we explain the formal theory behind our hybrid connector implementations and how to automatically generate them. In Sec. IV, we discuss the salient aspects of our implementation. In Sec. V, we exemplify our code generator with a distributed service orchestration scenario. In Sec. VI, we discuss related work, after which Sec. VII concludes this paper.

II. Reo Coordination Language

Reo is a language for compositional construction of concurrency protocols, manifested as connectors [1, 2]. Connectors consist of channels and nodes, organized in a graph-like structure. Every channel consists of two ends and a constraint that relates the timing and the contents of the data-flows at those ends. A channel end has one of two types: *source ends* accept data (i.e., a source end of a channel connects to that channel's data source/producer), while *sink ends* dispense data (i.e., a sink end of a channel connects to that channel's data sink/consumer). Reo makes no other assumptions about channels. This means, for instance, that Reo allows channels with two source ends. Table I shows the syntax and an informal description of common channels.

Entities communicating through a connector perform I/O operations—writes and takes—on its nodes. Reo features three types of nodes: *source nodes* on which only source ends coincide (see Figure 2a), *sink nodes* on which only sink ends coincide (see Figure 2b), and *mixed nodes* on which both types of channel end coincide (see Figure 2c). Informally, nodes behave as follows.

- A source node n has *replicator semantics*. Once a communicating entity attempts to write a piece of data d on n, this node first suspends that operation. Subsequently, n notifies the channels whose source ends coincide on n that it offers d. Once each of those channels has notified n that it accepts d, n resolves the write: the write operation terminates successfully and atomically, n dispenses (a copy of) d to each of its coincident source ends. Source nodes forbid takes.
- A sink node n has *merger semantics*. Once a communicating entity attempts to take a piece of data from n, this node first suspends that operation. Subsequently, n notifies the channels whose sink ends coincide on n that it accepts a piece of data. Once at least one of these channels has notified n that it offers a piece of data d, n resolves the take: atomically, n fetches d from the appropriate channel end and dispenses it to the entity attempting to take. If multiple sink ends offer a data item, n chooses one of them nondeterministically. Sink nodes forbid writes.
- A mixed node has *pumping station semantics*: the atomic execution of the replicator semantics and merger semantics discussed above. Mixed nodes forbid I/O.


{A , B} , $d(\mathsf{A}) = d(\mathsf{B})$
{A , B} , $d(\mathsf{A}) = d(\mathsf{B})$
{A} , $\top$
{A , B} , $\top$
{A} , $x' = d(\mathsf{A})$
{B} , $d(\mathsf{B}) = x$
{A , B , Z} , $d(\mathsf{A}) = d(\mathsf{Z}) \wedge x' = d(\mathsf{B})$
{Z} , $d(\mathsf{Z}) = x$

(a) Sync (b) LossySync-ND (c) SyncDrain

(d) FIFO (e) Alternator

Figure 4: Constraint automata for the channels in Table I (between ports A and B) and for Alternator in Figure 1a.

$$\frac{q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \quad \textbf{and}\ \mathsf{Port}(\alpha) \cap P_\beta = \mathsf{Port}(\beta) \cap P_\alpha}{(q_\alpha\, ,\, q_\beta) \xrightarrow{P_\alpha \cup P_\beta, f_\alpha \wedge f_\beta} (q'_\alpha\, ,\, q'_\beta)} \quad (1)$$

$$\frac{q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \quad \textbf{and} \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\beta \ \textbf{or}\ P_\beta = \mathsf{Port}(\beta) \cap P_\alpha \\ \textbf{or}\ \mathsf{Port}(\alpha) \cap P_\beta = \emptyset = \mathsf{Port}(\beta) \cap P_\alpha \end{bmatrix}}{(q_\alpha\, ,\, q_\beta) \xrightarrow{P_\alpha \cup P_\beta, f_\alpha \wedge f_\beta} (q'_\alpha\, ,\, q'_\beta)} \quad (2)$$

$$\frac{q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \in Q_\beta \ \textbf{and}\ P_\alpha \cap \mathsf{Port}(\beta) = \emptyset}{(q_\alpha\, ,\, q_\beta) \xrightarrow{P_\alpha, f_\alpha} (q'_\alpha\, ,\, q_\beta)} \quad (3)$$

$$\frac{q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \ \textbf{and}\ q_\alpha \in Q_\alpha \ \textbf{and}\ P_\beta \cap \mathsf{Port}(\alpha) = \emptyset}{(q_\alpha\, ,\, q_\beta) \xrightarrow{P_\beta, f_\beta} (q_\alpha\, ,\, q'_\beta)} \quad (4)$$

Figure 5: Rules for combining transitions.

So far, we explicitly distinguished between three language constructs in Reo: channels, nodes, and connectors. In the rest of this paper, for simplicity (but without loss of generality), we represent every channel c as its *corresponding connector*, which consists of two nodes connected by c. To define the semantics of c, it suffices to define the semantics of its corresponding connector. Furthermore, we assume that at most one source end and at most one sink end coincides on every node [9]. We model such binary nodes with *ports*. Ports occur either on the boundary between a connector and its environment or inside a connector. If a connector consists only of boundary ports, we call it a *primitive*; otherwise, we call it a *composite*. To simulate the semantics of nodes with more than two coincident channel ends, we use two ternary primitives: Merger and Replicator. These primitives can compose into a Node composite, which connects $n \geq 0$ input ports to $m \geq 0$ output ports and behaves as a node on which n source ends and m sink ends coincide (where $n+m > 0$). Figure 3 shows an example.

Many formalisms exist for mathematically defining the semantics of connectors [10]. In this paper, we adopt the same formalism as the existing code generator that we use: constraint automata (CA) [9]. A constraint automaton consists of finite sets of states and transitions. States represent the internal configurations of a connector; transitions describe the atomic steps of the protocol specified by a connector. Formally, we represent a transition as a tuple of four elements: a source state, a *synchronization constraint*, a *data constraint*, and a target state. A synchronization constraint is a set that specifies on which ports a data item flows (i.e., which ports synchronize); a data constraint is a logical formula that specifies which particular data items flow on which of those ports.

Figure 4 shows example CA, where A and B refer to ports. Informally, the data constraint $d(\mathsf{A}) = d(\mathsf{B})$ means that the data item flowing on port A equals the data item flowing on port B; the data constraint $\top$ means that it does not matter which particular data items flow; the data constraint $x' = d(\mathsf{A})$ means that the value of x, a private (to the connector) memory cell, *after completing* the transition equals the data flowing on A *during* the transition (i.e., x is written to in the end); finally, the data constraint $d(\mathsf{B}) = x$ means that the data flowing on B equals the value of private memory cell x *before* starting the transition (i.e., x is read from in the beginning).

Let $\mathbb{S}$TATE, $\mathbb{P}$ORT, $\mathbb{M}$EM, and $\mathbb{D}$C, defined in Appx. B, denote universes of states, ports, memory cells, and data constraints.

Definition 1. The universe of CA, denoted by $\mathbb{C}$A, is the largest set of tuples $(Q\, ,\, \mathcal{P}\, ,\, \mathcal{M}\, ,\, \longrightarrow\, ,\, \imath)$ where:

- $Q \subseteq \mathbb{S}\text{TATE}$; (states)
- $\mathcal{P} \subseteq \mathbb{P}\text{ORT}$; (ports)
- $\mathcal{M} \subseteq \mathbb{M}\text{EM}$; (memory cells)
- $\longrightarrow \subseteq Q \times \wp(\mathcal{P}) \times \mathbb{D}\text{C} \times Q$; (transitions)
- and $\imath \in Q$. (initial state)

If α denotes a CA, let $\mathsf{State}(\alpha)$, $\mathsf{Port}(\alpha)$, $\mathsf{Mem}(\alpha)$, and $\mathsf{init}(\alpha)$, defined in Appx. B, denote its states, ports, memory cells, and initial state. We adopt bisimilarity on CA as behavioral equivalence [9]: if α and β are bisimilar, denoted by $\alpha \approx \beta$, α can "simulate" every transition of β in every state and vice versa. Henceforth, we consider CA up to bisimilarity.

Individual CA describe the behavior of individual connectors; the application of the existing *product operator* to such CA models connector composition [9]. The formal definition and a congruence lemma, proved in [9], follow below.

Definition 2. The product operator, denoted by $\_ \boxtimes \_$, is the operator on $\mathbb{C}\textsc{a} \times \mathbb{C}\textsc{a}$ defined by the following equation:

$$\alpha \boxtimes \beta = \begin{pmatrix} \mathsf{State}(\alpha) \times \mathsf{State}(\beta)\,,\ \mathsf{Port}(\alpha) \cup \mathsf{Port}(\beta)\,, \\ \mathsf{Mem}(\alpha) \cup \mathsf{Mem}(\beta)\,,\ \longrightarrow,\ (\mathsf{init}(\alpha)\,,\ \mathsf{init}(\beta)) \end{pmatrix}$$

where $\longrightarrow$ denotes the smallest relation induced by Rule (1), Rule (3), and Rule (4) in Figure 5.

Lemma 1. $\big[\alpha \approx \beta$ **and** $\gamma \approx \delta\big]$ **implies** $\alpha \boxtimes \gamma \approx \beta \boxtimes \delta$

Henceforth, let $\boxtimes\{\alpha_1\,,\ \ldots\,,\ \alpha_k\}$ denote $\alpha_1 \boxtimes \cdots \boxtimes \alpha_k$. This is well-defined, because $\boxtimes$ is associative and commutative.

III. Design: Theoretical Justification

A. Hybrid Connector Implementations

We start with presenting our design of hybrid connector implementations; in Sec. III-B, we discuss the design of a tool that automatically generates corresponding code.

To first better explain the different implementation approaches for connectors with CA as formal semantics, let $X = \{\alpha_1\,,\ \ldots\,,\ \alpha_k\}$ denote a set of "small" CA, each of which models one of the k primitive constituents of the connector Conn to implement. We associate with X an *interpretation*, denoted by $[\![X]\!]$, which models the composition of the constituents in X (i.e., the full behavior of Conn):

$$[\![X]\!] = \boxtimes X$$

Every implementation of Conn—be it distributed, centralized, or hybrid—must behave as $[\![X]\!]$.

With the distributed approach, one first writes code for every $\alpha \in X$ and then deploys those k CA-implementations in k parallel processing units (e.g., processes, threads, actors). To ensure that those CA-implementations behave as $[\![X]\!]$, at run-time, they must communicate with each other to check which of their transitions can fire: generally, the enabledness of a transition of one CA-implementation depends on the enabledness of transitions of other CA-implementations—an example follows shortly. Essentially, with the distributed approach, the k CA-implementations compute the $\boxtimes$-operators amongst them dynamically at run-time. With the centralized approach, in contrast, one first computes $\boxtimes X$ (i.e., the full behavior of Conn), then writes code for the resulting "big" CA, and finally deploys this single CA-implementation in a single processing unit. By its construction, this single CA-implementation behaves as $[\![X]\!]$. Finally, with the hybrid approach, one first constructs a *partition* $\mathcal{A} = \{A_1\,,\ \ldots\,,\ A_\ell\}$ of X,[1] then computes $\boxtimes A$ for every *part* $A \in \mathcal{A}$, then writes code for the resulting "medium" CA, and finally deploys those ℓ CA-implementations in ℓ concurrent processing units. We associate with $\mathcal{A}$ an interpretation, denoted by $[\![\mathcal{A}]\!]$, by straightforwardly lifting the interpretation of sets of CA:

$$[\![\mathcal{A}]\!] = [\![\{[\![A_1]\!]\,,\ \ldots\,,\ [\![A_\ell]\!]\}]\!]$$

[1]A partition of X is a disjoint collection $\mathcal{A}$ of nonempty subsets of X, called parts, whose union is X [11, Sec. 7].

One can easily show that the interpretation of X equals the interpretation of any of its partitions (i.e., $[\![X]\!] = [\![\mathcal{A}]\!]$). Essentially, with the hybrid approach, a code generator computes the $\boxtimes$-operators embedded in the "inner" interpretations statically at build-time (as in the centralized approach), while the ℓ CA-implementations compute the $\boxtimes$-operators embedded in the "outer" interpretation dynamically at run-time (as in the distributed approach).

To carry out the second, third, and fourth steps of the hybrid approach we can use existing techniques from the distributed/centralized approach. The current challenge, thus, lies in the first step: finding a *reasonable* partition of X in a potentially huge search space.[2] At one extreme, if we put every $\alpha \in X$ in its own part (i.e., $\mathcal{A} = \{\{\alpha_1\}\,,\ \ldots\,,\ \{\alpha_k\}\}$), we get the distributed approach, whose communication overhead reduces throughput at run-time. At the other extreme, if we put every $\alpha \in X$ in the same part (i.e., $\mathcal{A} = \{\{\alpha_1\,,\ \ldots\,,\ \alpha_k\}\}$), we get the centralized approach, which sequentializes all parallelism. The hybrid approach should avoid both these unwelcome phenomena. As a compromise, we therefore adopt the following guideline for constructing partitions:

> Put those CA whose implementations *would* require "expensive" communication at run-time in the same part; separate those CA that *will* require only "cheap" or no communication in different parts.

Without having explained the more technical meaning of "expensive" and "cheap" in this context, also the natural language interpretation of this guideline seems sensible. First, a partition $\mathcal{A}$ constructed according to this guideline yields improved throughput (cf. the distributed approach), because those CA responsible for most of the communication overhead—caused by expensive communication—are compiled into one CA-implementation at build-time (i.e., they populate the same part in $\mathcal{A}$). Second, $\mathcal{A}$ yields improved parallelism (cf. the centralized approach), because those CA requiring only cheap or no communication are deployed in parallel processing units at run-time (i.e., each of them populates its own part in $\mathcal{A}$).

To exemplify the meaning of "expensive" and "cheap," let Alice, Bob, and Carol be three CA-implementations . In particular, let Alice, Bob, and Carol implement three Sync primitives in sequence between ports A, B, C, and D. Suppose that Alice's input port A has a pending write operation coming from the environment and that she wants to fire her $\{\mathsf{A}\,,\ \mathsf{B}\}$-transition (see Figure 4a). Operationally, to fire this transition, Alice must atomically take the data item written on A from that port and write it to B (see Table I). However, to guarantee atomicity, she must first ascertain that Bob

[2]Theoretically, the total number of partitions of a k-cardinality set equals the k-th *Bell number*, denoted by B_k, which grows rapidly in k. For instance, the number of possible partitions for Alternator in Figure 1a, which consists of only 6 constituents, is $B_6 = 203$.

is in fact *ready* to take a data item from B.[3] So, before Alice takes the data item from A, she first asks Bob if he is ready to take from B. But Bob cannot immediately answer that question: he, in turn, must first ask Carol if she is ready to take a data item from C.[4] In this simple example, Carol can answer Bob's question without further *derivative* communication by locally checking if D has a pending take operation. Generally, however, the chain of derivative communication can be much longer (ignoring, for the moment, the possibility of loops, which can be resolved). This makes the *initial* communication between Alice and Bob potentially very expensive. In summary, if Bob initiates derivative communication to answer a question from Alice, we call communication between Alice and Bob "expensive" (and "cheap" otherwise).

To identify which CA require only cheap communication, we extend the recent *local product theory* from CA without data constraints to unrestricted CA [12]. The idea is to introduce a new product operator on CA, called *l-product*, which models connector composition with cheap communication. This contrasts the existing product in Definition 2, which models standard connector composition with potentially expensive communication: *l-product is generally not faithful to Reo's semantics*. The formal definition and a congruence lemma, proved in Appx. B, follow below. Afterward, we address the issue of determining when substituting the existing product with l-product is sound (i.e., faithful to Reo's semantics).

Definition 3. The l-product (where "l" stands for "local"), denoted by $\_ \boxdot \_$, is the operator on $\mathbb{C}\text{A} \times \mathbb{C}\text{A}$ defined by the following equation:

$$\alpha \boxdot \beta = \begin{pmatrix} \mathsf{State}(\alpha) \times \mathsf{State}(\beta) \,,\, \mathsf{Port}(\alpha) \cup \mathsf{Port}(\beta) \,, \\ \mathsf{Mem}(\alpha) \cup \mathsf{Mem}(\beta) \,,\, \longrightarrow \,,\, (\mathsf{init}(\alpha) \,,\, \mathsf{init}(\beta)) \end{pmatrix}$$

where $\longrightarrow$ denotes the smallest relation induced by Rule (2), Rule (3), and Rule (4) in Figure 5.

Lemma 2. $\big[\alpha \approx \beta$ **and** $\gamma \approx \delta\big]$ **implies** $\alpha \boxdot \gamma \approx \beta \boxdot \delta$

The difference with Definition 2 is that the transition relation is induced by Rule (2) in Figure 5 instead of Rule (1).[5]

To construct a reasonable—according to our guideline—partition $\mathcal{A} = \{A_1 \,, \ldots \,, A_\ell\}$, we must ensure that only cheap communication occurs among (implementations of

```
function PARTITION($\{\alpha_1 \,, \ldots \,, \alpha_k\}$)
  $(\mathcal{B}_0 \,, \mathcal{C}_0) := (\emptyset \,, \emptyset)$
  for all $1 \leq i \leq k$ do
    if $\xrightarrow{1} \alpha_i$ then
      $(\mathcal{B}_i \,, \mathcal{C}_i) := (\mathcal{B}_{i-1} \cup \{\{\alpha_i\}\} \,, \mathcal{C}_{i-1})$
    else
      $\overline{\mathcal{C}}_i := \{C \in \mathcal{C}_{i-1} \mid \gamma \in C \text{ and } \alpha_i \not\asymp \gamma\}$
      $(\mathcal{B}_i \,, \mathcal{C}_i) := (\mathcal{B}_{i-1} \,, (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\})$
  return $(\mathcal{B}_k \,, \mathcal{C}_k)$
```

Figure 6: Algorithm for computing reasonable partitions.

interpretations of) parts in $\mathcal{A}$ at run-time. Using the l-product just introduced, we can formally state this property as follows:

$$[\![A_1]\!] \boxtimes \cdots \boxtimes [\![A_\ell]\!] \approx [\![A_1]\!] \boxdot \cdots \boxdot [\![A_\ell]\!]$$

By extending [12, Theorem 2] from port automata to CA, we obtain two conditions on $[\![A_1]\!] \,, \ldots \,, [\![A_\ell]\!]$ that ensure the validity of that equation. This reduces the problem of finding a reasonable partition to finding CA that satisfy those conditions. Doing so is relatively straightforward (see Sec. III-B). To express the first condition, let $\xrightarrow{1} \alpha$ denote that each of α's transitions has a singleton synchronization constraint (e.g., the CA of FIFO in Figure 4d). To express the second condition, let $\alpha_1 \asymp \alpha_2$ denote that α_1 and α_2 have disjoint sets of ports. The following lemma, proved in Appx. C, defines those conditions.

Lemma 3.

$$\left[\begin{bmatrix} \xrightarrow{1} \beta_i \text{ for all} \\ 1 \leq i \leq m \end{bmatrix} \textbf{ and } \begin{bmatrix} [i \neq j \textbf{ implies } \gamma_i \asymp \gamma_j] \\ \textbf{for all } 1 \leq i \,, j \leq n \end{bmatrix}\right]$$

$$\textbf{implies} \quad \begin{aligned} &\beta_1 \boxtimes \cdots \boxtimes \beta_m \boxtimes \gamma_1 \boxtimes \cdots \boxtimes \gamma_n \\ \approx\, &\beta_1 \boxdot \cdots \boxdot \beta_m \boxdot \gamma_1 \boxdot \cdots \boxdot \gamma_n \end{aligned}$$

So, if we construct a partition $\mathcal{A}$ that contains m parts B_1 , ..., B_m with the property $\xrightarrow{1} [\![B_i]\!]$ and n parts $C_1 \,, \ldots \,,$ C_n with the property $[\![C_i]\!] \asymp [\![C_j]\!]$, Lemma 3 implies that only cheap communication (modeled by $\boxdot$) instead of expensive communication (modeled by $\boxtimes$) is necessary among their corresponding CA-implementations at run-time,[6] while, collectively, they behave as $[\![X]\!]$. The following derivation establishes this:

$$\begin{aligned} [\![X]\!] &= [\![\mathcal{A}]\!] \\ &= [\![\{[\![B_1]\!] \,, \ldots \,, [\![B_m]\!] \,, [\![C_1]\!] \,, \ldots \,, [\![C_n]\!]\}]\!] \\ &= [\![B_1]\!] \boxtimes \cdots \boxtimes [\![B_m]\!] \boxtimes [\![C_1]\!] \boxtimes \cdots \boxtimes [\![C_n]\!] \\ &\approx [\![B_1]\!] \boxdot \cdots \boxdot [\![B_m]\!] \boxdot [\![C_1]\!] \boxdot \cdots \boxdot [\![C_n]\!] \end{aligned}$$

B. Hybrid-Code Generator

The new task of a hybrid-code generator, in addition to the tasks of a centralized-code generator [3], is dividing

[3]If not, the data item that Alice writes to B has nowhere to go, which Reo's semantics forbids (i.e., ports cannot buffer data).

[4]If not, the data item that Bob atomically takes from B and writes to C has nowhere to go.

[5]The second line of the premise in Rule (1) states that, if two transitions agree on their shared ports, they can fire together. These transitions may involve any number of other, unshared ports, which causes expensive communication when computing $\boxtimes$ at run-time (e.g., Alice asks Bob about their shared port, but Bob wants to involve a port not shared with Alice for which he asks Carol). In contrast, the second line of the premise in Rule (2) restricts the involvement of unshared ports: either one of the transitions fires only shared ports, or both transitions fire only unshared ports.

[6]Mathematically, one can formally compensate for non-associativity of $\boxdot$ along the same lines as [12, Sec. 5], manifested as proper locking schemes at run-time, as mentioned in Sec. IV-A.

the CA of the k primitive constituents of the connector to implement over parts in a partition $\mathcal{A}$ such that the two conditions in Lemma 3 hold; it can subsequently use existing techniques to generate code for every part in $\mathcal{A}$. To carry out this new task, a hybrid-code generator can run the algorithm in Figure 6 with time complexity upper bound $\mathcal{O}(k^2)$. The following lemma, proved in Appx. D, establishes its functional correctness.

Lemma 4. $\text{PARTITION}(X) = (\mathcal{B}, \mathcal{C})$ **implies**

1) $\mathcal{B} \cup \mathcal{C}$ *is a partition of* X
2) $\xrightarrow{1} [\![B]\!]$ **for all** $B \in \mathcal{B}$
3) $\big[C \neq C'$ **implies** $[\![C]\!] \asymp [\![C']\!]\big]$ **for all** $C, C' \in \mathcal{C}$

The reasonable partitions $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$ computed by the algorithm in Figure 6 correspond to the *synchronous* and the *asynchronous regions* of a connector [6, 7]. First, every part $B \in \mathcal{B}$ seems to represent an asynchronous region (e.g., a FIFO primitive): the fact that $[\![B]\!]$ has only singleton synchronization constraints (i.e., $\xrightarrow{1} [\![B]\!]$) models that its ports can*not*—neither intentionally nor coincidentally—synchronize at run-time. Dually, every part $C \in \mathcal{C}$ seems to represent a synchronous region (e.g., the sequence of three Sync primitives modeled by Alice, Bob, and Carol in a previous example): by the duality of (a)synchronous regions, a CA for a synchronous region has at least one transition with a nonsingleton synchronization constraint. When such a transition fires, synchronization between at least two ports takes place.

IV. Implementation: Practical Realization

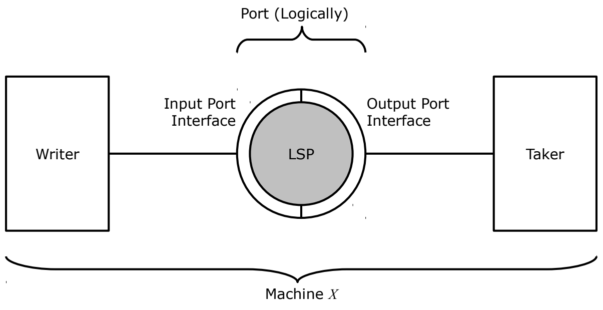
A. Hybrid Connectors

We extended the existing Reo-to-Java code generator [3]. This tool translates CA to implementations of Java's `Runnable` interface. Every such a CA-implementation can run in its own Java thread. The control-flow inside the main `run()` method follows a conceptually simple (event-driven) state machine pattern—details appear elsewhere [3]. We focus on the process of firing a transition $q \xrightarrow{P,f} q'$. The following enumeration of steps summarizes this process.

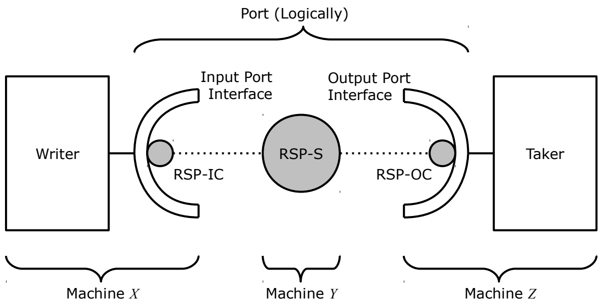
1) Check synchronization constraint P.
   a) Check the ports in P for pending I/O. Let $P' \subseteq P$ denote the set of ports without pending I/O.
   b) Ask neighbors with ports in P' which data constraints must hold for them to be ready for I/O on those ports. Let F denote that set of data constraints.
2) Solve extended data constraint $f \wedge \bigwedge F$.
3) Commit and conclude.
   a) Distribute data items among pending take operations according to the solution found for $f \wedge \bigwedge F$.
   b) Mark pending I/O completed, and update state to q'.
   c) Perform I/O on ports in P' according to the solution found for $f \wedge \bigwedge F$.
   d) Notify neighbors with ports in P' that they should fire their transitions involving those ports.
   e) Await the completion of those transition.

During steps (1) and (2), a CA-implementation *can* still abort the firing process, and it *will* do so if either the synchronization constraint does not hold—in which case F contains $\perp$—or the data constraint has no solution under the then-pending I/O operations. Once step (3) starts, however, the transition *must* run to completion. To ensure that the whole firing process runs atomically, that pending I/O operations do not timeout during the firing process, and that neighbors do not change state, a CA-implementation uses a two-phase locking scheme [13].

Ports are implemented as interfaces that provide access to concurrent data structures called *synchronization points* [3]. Essentially, synchronization points register pending write and take operations. There exist two kinds of port interfaces: input ports expose only `write(...)` methods for performing write operations, while output ports expose only `take(...)` methods for performing take operations. Application developers can use input and output ports for letting concurrent fragments of their computation code interact with each other via a CA-implementation (i.e., via a connector). Internal classes in the Reo run-time libraries, as well as generated CA-implementations, call also other methods on input and output ports (e.g., checking for pending I/O operations, communicating with neighbors via synchronization points).



(a) Local synchronization point (shared-memory port)



(b) Remote synchronization point (distributed-memory port)

Figure 7: Synchronization points

The run-time libraries of the original implementation of the Reo-to-Java code generator contain only one *shared-memory* implementation of input and output ports. Figure 7a shows an infographic, where "LSP" stands for "local synchronization point." This name reflects that the Java objects constituting such a synchronization point live on the same machine as the CA-implementations that access that synchronization point through input and output ports. Logically, a triple of an LSP, an input port, and an output port makes up a single port.

To enable developers to fully exploit the improved parallelism in the hybrid connector implementations generated by our tool extension (cf. the centralized approach), we implemented a new *distributed-memory* implementation of input and output ports. In particular, this allows developers to deploy connector implementations generated by our tool extension on different machines in a network. Figure 7b shows an infographic, where "RSP" stands for "remote synchronization point," "-S" for "server," "-IC" for "input client," and "-OC" for "output client." Deployment of a remote synchronization point starts with deploying an RSP-S on some machine in the network. Essentially, an RSP-S is a Web service, implemented using JAX-WS,[7] whose operations provide its clients access to a "classical" LSP inside of it. Once an RSP-S has been deployed, one can construct input and output ports to access it, including input and output clients. During execution, those port interfaces use such clients for delegating, to the deployed RSP-S, those method calls that they cannot process locally (e.g., checking for pending I/O operations). The CA-implementations that call methods on port interfaces do not know whether those calls require network communication: whether synchronization points accessed through port interfaces run locally or remotely is completely transparent.

B. Hybrid Code Generator

Our extension to the Reo-to-Java code generator is—as the original—implemented in Java as an Eclipse plug-in. This plug-in depends on the *Extensible Coordination Tools* (ECT): a collection of Eclipse plug-ins that constitute an IDE for Reo.[8] Our extension outputs ℓ Java classes—one for every part in the reasonable partition computed using the algorithm in Figure 6—each of which implements the `Runnable` interface. Instances of those classes can be deployed on different machines and connected to each other via distributed-memory ports (including the required remote synchronization point servers and clients). For testing purposes, the code generator also generates a default main program which deploys an instance of each of the ℓ classes and each of the required distributed-memory ports on the same machine.[9]

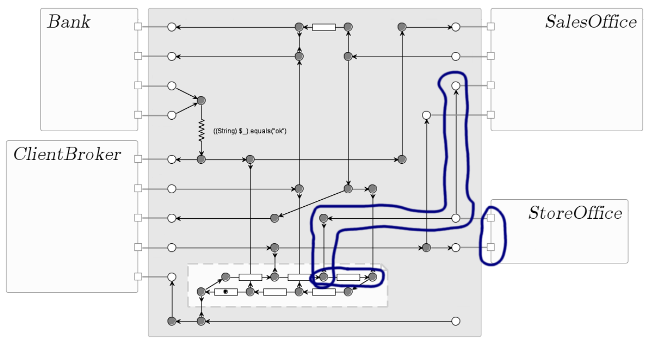


Figure 8: Connector for orchestrating the four WSs. Colored regions denote deployment of medium CA-implementations and the *StoreOffice* service on machine Blue.

For performance reasons, we simplified the implementation of the algorithm in Figure 6 and the actual code generation process (using ANTLR's StringTemplate engine [14]) by exploiting the observation that the only primitive currently supported by the ECT that satisfies the first condition in Lemma 3 is FIFO. This, for instance, reduced checking the condition of the **if**-statement in Figure 6 from iterating over all transitions of a CA to checking that CA's type.

V. Case Study

In the following case study, we elaborate on the same example as given in [8], which implements a classical online-purchase scenario. This interaction involves four Web services (WS) named *ClientBroker*, *StoreOffice*, *SalesOffice*, and *Bank*. The *ClientBroker* service takes care of interfacing a client to the other services, which deal with: the information about the store (i.e., the *StoreOffice* service), the procedure to prepare the invoice (i.e., the *SalesOffice* service), and the effective payment management (i.e., the *Bank* service). See [8] for a more detailed description. Figure 8 shows a Reo connector, named Orchestrator, for orchestrating the four WSs. A Reo expert designed this connector by hand. Alternatively, depending on the expertise available in an organization, one can specify the orchestration protocol as an automaton, as a BPEL program, or as a UML sequence/activity diagram, and use mechanical connector synthesis technology to obtain this (or a behaviorally equivalent) connector [15, 16].

In [8], Jongmans et al. generate a centralized implementation of Orchestrator and deploy that implementation on a single machine in a network. We improve on that by using our hybrid-code generator to obtain a hybrid implementation and deploy it across multiple machines.

First, our code generator establishes that Orchestrator consists of 43 channels and 42 nodes. It then matches each of

[7]http://jax-ws.java.net
[8]http://reo.project.cwi.nl

[9]Our implementation of distributed-memory ports works also for machines X, Y, and Z in Figure 7b such that $X = Y = Z$. However, in that case, using shared-memory ports is more sensible.

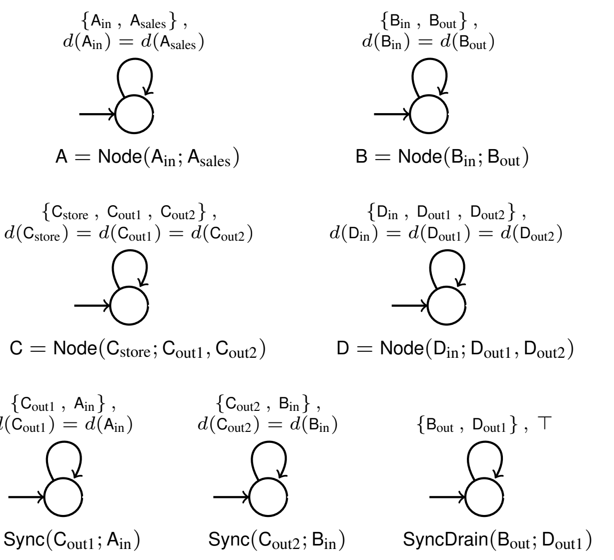
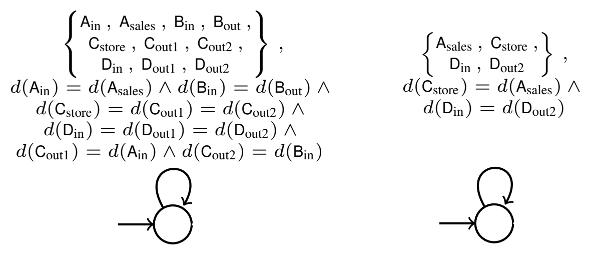

$\{A_{in}, A_{sales}\}$, $d(A_{in}) = d(A_{sales})$
$A = \mathrm{Node}(A_{in}; A_{sales})$
$\{B_{in}, B_{out}\}$, $d(B_{in}) = d(B_{out})$
$B = \mathrm{Node}(B_{in}; B_{out})$
$\{C_{store}, C_{out1}, C_{out2}\}$, $d(C_{store}) = d(C_{out1}) = d(C_{out2})$
$C = \mathrm{Node}(C_{store}; C_{out1}, C_{out2})$
$\{D_{in}, D_{out1}, D_{out2}\}$, $d(D_{in}) = d(D_{out1}) = d(D_{out2})$
$D = \mathrm{Node}(D_{in}; D_{out1}, D_{out2})$
$\{C_{out1}, A_{in}\}$, $d(C_{out1}) = d(A_{in})$
$\mathrm{Sync}(C_{out1}; A_{in})$
$\{C_{out2}, B_{in}\}$, $d(C_{out2}) = d(B_{in})$
$\mathrm{Sync}(C_{out2}; B_{in})$
$\{B_{out}, D_{out1}\}$, $\top$
$\mathrm{SyncDrain}(B_{out}; D_{out1})$

Figure 9: Constraint automata for the channels and nodes in the largest colored region in Figure 8. Nodes are named A, B, C, and D, from top to bottom and from left to right. In this CA semantics, every node is represented by a number of input and output ports, which are shared with channels.


$\left\{\begin{matrix} A_{in}, A_{sales}, B_{in}, B_{out}, \\ C_{store}, C_{out1}, C_{out2}, \\ D_{in}, D_{out1}, D_{out2} \end{matrix}\right\}$, $d(A_{in}) = d(A_{sales}) \wedge d(B_{in}) = d(B_{out}) \wedge d(C_{store}) = d(C_{out1}) = d(C_{out2}) \wedge d(D_{in}) = d(D_{out1}) = d(D_{out2}) \wedge d(C_{out1}) = d(A_{in}) \wedge d(C_{out2}) = d(B_{in})$
$\left\{\begin{matrix} A_{sales}, C_{store}, \\ D_{in}, D_{out2} \end{matrix}\right\}$, $d(C_{store}) = d(A_{sales}) \wedge d(D_{in}) = d(D_{out2})$

(a) Without abstraction

(b) With abstraction

Figure 10: ⊠-product of the CA in Figure 9, with and without abstracting away internal ports.

those constituents with a CA describing that constituent's behavior, which yields $k = 85$ small CA. For instance, Figure 9 shows seven CA describing the constituents in the largest colored region in Figure 8. Next, our code generator runs the algorithm in Figure 6 to obtain a reasonable partition. This partition consists of thirteen parts: seven singleton parts for asynchronous regions (i.e., FIFOs, which satisfy $\xrightarrow{1}$) and six parts for synchronous regions. For instance, the CA in Figure 9 form a complete part of the latter kind (none of those seven CA shares a port with any CA in any other part). The code generator then computes, for every part, the ⊠-product of the CA in that part. This yields $\ell = 13$ medium CA. For instance, Figure 10 shows the ⊠-product of the CA in Figure 9. Finally, the code generator compiles the thirteen medium CA to as many CA-implementations. For instance, at run-time, the generated implementation of the CA in Figure 10 has only one execution step (i.e., transition) that it repeats infinitely often: it atomically transports a piece of data from *StoreOffice* to *SalesOffice* and, simultaneously, it transports a piece of data from the output port of one FIFO to the input port of another one. The latter ensures that the generated CA-implementation can perform this execution step only if the whole connector is in a state that allows this (i.e., if the FIFOs are, respectively, full and empty). See [8] for details.

Next, we discuss the deployment of the generated code. Logically, we have five machines: called Red (for the actual client), Green (for *ClientBroker*), Blue (for *StoreOffice*), Cyan (for *SalesOffice*), and Magenta (for *Bank*). Although any distribution of the thirteen generated CA-implementations (and the ports between them, through which CA-implementations communicate with each other) over the five available machines would technically work, some of those distributions make more sense than others. Generally, the problem of (automatically) optimally distributing CA-implementations over machines is an interesting research challenge, which we regard as important future work (see also Sec. VII). For now, we adopted the following ad hoc approach: to minimize network traffic, we manually distributed CA-implementations in such a way that every piece of data goes over the network exactly once. In contrast, in the centralized implementation in [8], every piece of data goes over the network twice: first from the sending WS machine to the Orchestrator machine and then from the Orchestrator machine to the receiving WS machine. Thus, the hybrid approach can improve the centralized approach not only in terms of run-time parallelism but, as a consequence of less network traffic and no single-point contention, also in terms of throughput (especially with large data). Figure 8 shows the deployment of parts of the orchestration on machine Blue; see Appx. A for the full distribution.

For deploying ports, we first analyzed which ports are shared between CA-implementations running on the same machine. We deployed those ports as shared-memory ports (similar to [8]) and all other ports as distributed-memory ports. For every distributed-memory port p, we deployed its remote synchronization point server (RSP-S) on the same machine as the CA-implementation that uses p's input interface. This means that pieces of data involved in write operations on p stay on that machine as long as no other CA-implementation (running on a different machine) wants to take that data from p's RSP-S (via p's output interface). Consequently, pieces of data travel over the network only if absolutely necessary.

To actually run this case study on five machines, we use *Amazon Web Services* (AWS),[10] which is a collection of WSs that together make up a Cloud computing platform,

[10]http://aws.amazon.com

offered over the Internet by Amazon.com. In particular, we take advantage of *Elastic Compute Cloud* (EC2): a WS that provides resizable compute capacity in the Cloud. It is designed to make Web-scale computing easier for developers, by allowing them to rent virtual machines on which to run their own applications. We rented five of those virtual machines and deployed the generated code for this case study as explained above. This shows that the framework underlying Reo can be integrated with Cloud technologies (i.e., "Reo goes to the Cloud").

Actually, by using the Cloud in this way, we can position the example in [8] in the following new scenario. Suppose we run a (large) online business company that offers a purchase service through AWS: the motivation is that our company desires to scale its business in the Cloud, benefiting from a third-party infrastructure that can efficiently manage Big Data. For instance, combined with EC2, our company can use another Cloud service from the AWS platform called *Simple Storage Service* (S3): an online technology for managing large amounts of information at any time, capable of handling Big Data such as transaction logs of our clients' orders.[11] Because the hybrid approach allows us to sensibly distribute CA-implementations over machines (minimizing network traffic, as explained above), it is—contrasting the centralized approach—suitable for Big Data coordination.

VI. Related Work

In this section, we discuss related work on Reo and distributed orchestration/workflow.

A. Related Work on Reo

Closest to ours is the work on splitting connectors into (a)synchronous regions for better performance. Proença developed the first implementation based on these ideas, demonstrated its merit through benchmarks, and invented a new automaton model to reason about split connectors [5, 6]. Furthermore, Clarke and Proença explored connector splitting in the context of the connector coloring semantics [17]. They discovered that the standard version of that semantics has undesirable properties in the context of splitting: some split connectors that intuitively *should* be equivalent to the original connector are not equivalent under the standard version. To address this problem, Clarke and Proença propose a new variant—*partial connector coloring*—which allows one to better model locality and independencies between different parts of a connector. Recently, Jongmans et al. studied a formal justification of connector splitting in a process algebraic setting [18].

Also related to the work presented in this paper is the work of Kokash et al. on *action constraint automata* (ACA) [19]. Kokash et al. argue that ordinary constraint automata describe the behavior of Reo connectors too coarsely, which makes it impossible to express certain fine parallel behavior. While ACA better describe the behavior of existing connector implementations (under certain assumptions), the increased granularity of ACA comes at the price of substantially larger models. This makes them less suitable for code generation.

[11]Nowadays, these qualify as Big Data: *"Wal-Mart handles more than one million customer transactions every hour, feeding databases estimated at more than 2.5 petabytes"* (http://www.economist.com/node/15557443).

B. Related Work on Distributed Orchestration/Workflow

To put our previous case study in perspective, we conclude with related work on distributed orchestration/workflow.

In [20], Nanda et al. present a technique for partitioning a composite service written as a single BPEL [21] program into an equivalent set of decentralized processes. In their approach, Nanda et al. construct the dependency graph of a BPEL program with the aim of minimizing communication costs while maximizing throughput. In [22], Chafle et al. decentralize the orchestration of a *FindRoute* service by partitioning the BPEL code into four parts, which are executed by four distinct Java engines. Afterward, Chafle et al. compare the performance of the centralized and the decentralized implementation by using both service-time and message-size metrics. Chafle et al. also estimate the additional complexity in error recovery and fault handling in a decentralized orchestration. In [23], Mostarda et al. use a BPEL-based language for distributed orchestration in the context of pervasive computing. In their approach, Mostarda et al. automatically decompose a centralized workflow into implementations of finite state machines that, when synchronized using a consensus protocol, execute the original workflow. In [24], Fernández et al. introduce an execution model for distributed orchestration based on a metaphor from chemistry. In this approach, services communicate with each other through a distributed shared multiset ("chemical solution"), which contains both control-flow and data-flow information ("molecules"). A workflow engine then executes an orchestration protocol by applying formal rewrite rules ("chemical reactions"). Fernández et al. propose translating BPEL programs to chemical representations that the workflow engine they describe can process. The main difference between these existing BPEL-based approaches to distributed orchestration and our approach is the use of Reo instead of BPEL. At least from a software engineering perspective, Reo has several advantages compared to BPEL (declarative style, verbatim reuse/composition of connectors, smaller gap between verification and execution) [8]. Whether our Reo-based approach also outperforms those existing BPEL-based approaches remains a topic for future study.

In [25], Barker et al. present an architecture (including a proxy API) for optimizing data-flow in workflow execution. In their architecture, control-flow messages are still sent to a centralized orchestration engine, while data-flow messages (i.e., the actual, potentially large, pieces of data) go directly from one service to another. The latter reduces network traffic, and it coincides with the heuristic we adopted for distributing the CA-implementations over machines in our case study. However, our technique distributes also control-flow. This gives our approach, in addition to data-flow optimization through clever deployment, also other advantages such as control-flow parallelism, no single point-of-failure, and potentially easier dynamic reconfiguration. Similarly motivated as the work of Barker et al. (i.e., optimizing data-flow), in [26], Binder et al. propose a distributed orchestration methodology based on decomposing an orchestration protocol (or workflow) into a directed acyclic graph of service invocations, represented as *triggers*. Every trigger encodes the data dependencies of the invocation it represents (i.e., the parents and children of the corresponding node in the graph). Triggers act as proxies: they collect all input data before actually invoking the service, and they transmit all output data directly to the dependent triggers for subsequent service invocations. One may regard the medium CA-implementations that we generate with our code generator as triggers. In that case, this report improves the work of Binder et al. by presenting an automatic procedure for decomposing orchestration protocols into triggers for a more expressive protocol language (i.e., Reo): we support also loops and conditions, which Binder et al. forbid.

In [27], Tretola and Zimeo present a technique for improving concurrency in workflow execution. Their technique works by concurrently executing otherwise sequential, data-dependent service invocations with dummy data until those services require the actual data. If service invocations have a data-independent initialization phase, for instance, the workflow engine of Tretola and Zimeo executes those intialization phases in parallel instead of in sequence. For workflows and services implemented on top of the framework of Tretola and Zimeo, this process happens automatically. Although Tretola and Zimeo parallelize service invocations, they neither parallelize nor distribute the execution of the orchestration protocol. Also, their technique seems unable to handle third-party/black-box services. In contrast, our approach does distribute the execution of the orchestration protocol and works with black-box services (via proxies [8]), but we currently cannot exploit potential concurrency between service invocations the way Tretola and Zimeo do. Thus, the technique of Tretola and Zimeo seems to complement our work. (However, we do not understand yet which notion of equivalence their technique preserves and what consequences this has for properties that the orchestration protocol satisfies on the modeling level. This requires further study.)

In [28], Pedraza and Estublier present FOCAS. This framework consumes as input an annotated APEL [29] specification of an orchestration scenario and produces as output a number of suborchestrations and a deployment plan (for distributing suborchestrations over machines). The main difference between our approach and the approach of Pedraza and Estublier seems that Reo allows for expressing more complex data-flow behavior than APEL does. This directly influences the complexity of automatically computing suborchestrations. On the other hand, our code generator currently does not compute a deployment plan, because we do not support deployment annotations, which FOCAS does.

Finally, in [30], Muth et al. use the formal semantics of state/activity charts to develop an algorithm for transforming centralized state/activity charts into equivalent partitioned ones, suitable for distributed execution by a workflow engine. Muth et al. subsequently refine their basic approach to also reduce communication overhead and exploit parallelism between parts in partitions. We had similar motivations for doing the work presented in this paper, but the underlying formalisms differ. Comparing the strengths and weaknesses of state/activity charts and Reo in the context of workflows, orchestration protocols, and coordination in general seems an interesting topic for future work.

VII. Conclusion

We presented a hybrid approach for the implementation of Reo connectors by partitioning them into several (a)synchronous regions at build-time. Every such region can be executed on a different machine at run-time. We use the term “hybrid” because our approach is neither purely centralized (regions run in parallel at run-time), nor purely distributed (the elements inside a region are compiled to a sequential program at build-time and require no distributed algorithms or communication at run-time). In this way, we have the benefits of both pure approaches: the high run-time throughput of a centralized scheme combined with the high parallelism and fast compilation of a distributed scheme.

In the future, we plan to design an algorithm to automatically find the best partitioning of CA-implementations according to user and system-defined constraints. These constraints may need to be either “crisply” or “softly” satisfied (in relation to their indispensability), and concern different criteria as hardware requirements of software, QoS/QoE/performance desiderata, and issues correlated with security (e.g., preventing attacks based on *Business Process Discovery*), privacy (of both data and workflow), and fault handling.

References

[1] F. Arbab, “Reo: a channel-based coordination model for component composition,” *MSCS*, vol. 14, no. 3, pp. 329–366, 2004.

[2] ——, “Puff, The Magic Protocol,” in *Talcott Festschrift*, ser. LNCS. Springer, 2011, vol. 7000, pp. 169–206.

[3] S.-S. Jongmans and F. Arbab, “Modularizing and Specifying Protocols among Threads,” in *Proceedings of PLACES 2012*, ser. EPTCS. CoRR, 2013, vol. 109, pp. 34–45.

[4] N. Kokash, C. Krause, and E. de Vink, "Reo+mCRL2: A framework for model-checking dataflow in service compositions," *FAC*, vol. 24, no. 2, pp. 187–216, 2012.

[5] J. Proença, D. Clarke, E. de Vink, and F. Arbab, "Decoupled execution of synchronous coordination models via behavioural automata," in *Proceedings of FOCLASA 2011*, ser. EPTCS. CoRR, 2011, vol. 58, pp. 65–79.

[6] J. Proença, D. Clarke, E. de Vink, and F. Arbab, "Dreams: a framework for distributed synchronous coordination," in *Proceedings of SAC 2012*. ACM, 2012, pp. 1510–1515.

[7] J. Proença, "Synchronous Coordination of Distributed Components," Ph.D. dissertation, Leiden University, 2011.

[8] S.-S. Jongmans, F. Santini, M. Sargolzaei, F. Arbab, and H. Afsarmanesh, "Automatic Code Generation for the Orchestration of Web Services with Reo," in *Proceedings of ESOCC 2012*, ser. LNCS. Springer, 2012, vol. 7592, pp. 1–16.

[9] C. Baier, M. Sirjani, F. Arbab, and J. Rutten, "Modeling component connectors in Reo by constraint automata," *SCP*, vol. 61, no. 2, pp. 75–113, 2006.

[10] S.-S. Jongmans and F. Arbab, "Overview of Thirty Semantic Formalisms for Reo," *SACS*, vol. 22, no. 1, pp. 201–251, 2012.

[11] P. Halmos, "Relations," in *Naive Set Theory*. Springer, 1974, pp. 26–29.

[12] S.-S. Jongmans and F. Arbab, "Global Consensus through Local Synchronization," in *Preproceedings of FOCLASA 2013*, 2013, pp. 86–82.

[13] P. Bernstein, V. Hadzilacos, and N. Goodman, "Two Phase Locking," in *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987, pp. 47–111.

[14] T. Parr, "Generating Structured Text with Templates and Grammars," in *The Definitive ANTLR Reference: Building Domain-Specific Languages*. The Pragmatic Bookshelf, 2007, pp. 208–242.

[15] F. Arbab, C. Baier, F. de Boer, J. Rutten, and M. Sirjani, "Synthesis of Reo Circuits for Implementation of Component-Connector Automata Specifications," in *Proceedings of COORDINATION 2005*, ser. LNCS. Springer, 2005, vol. 3454, pp. 236–251.

[16] B. Changizi, "Model Based Analysis of Business Process Models," Ph.D. dissertation, Leiden University, 2014, (in preparation).

[17] D. Clarke and J. Proença, "Partial Connector Colouring," in *Coordination Models and Languages*, ser. LNCS. Springer, 2012, vol. 7274, pp. 59–73.

[18] S.-S. Jongmans, D. Clarke, and J. Proença, "A Procedure for Splitting Processes and its Application to Coordination," in *Proceedings of FOCLASA 2012*, ser. EPTCS. CoRR, 2012, vol. 91, pp. 79–96.

[19] N. Kokash, B. Changizi, and F. Arbab, "A Semantic Model for Service Composition with Coordination Time Delays," in *Proceedings of ICFEM*, ser. LNCS. Springer, 2010, vol. 6447, pp. 106–121.

[20] M. G. Nanda, S. Chandra, and V. Sarkar, "Decentralizing Execution of Composite Web Services," in *Proceedings of OOPSLA 2004*. ACM, 2004, pp. 170–187.

[21] D. Jordan and J. Evdemon, "Web Services Business Process Execution Language Version 2.0," OASIS, Standard ws-bpel-v2.0-OS, 2007.

[22] G. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized orchestration of composite web services," in *Proceedings of WWW Alt. 2004*. ACM, 2004, pp. 134–143.

[23] L. Mostarda, S. Marinovic, and N. Dulay, "Distributed Orchestration of Pervasive Services," in *Proceedings of AINA 2010*. IEEE, 2010, pp. 166–173.

[24] H. Fernández, T. Priol, and C. Tedeschi, "Decentralized Approach for Execution of Composite Web Services Using the Chemical Paradigm," in *Proceedings of ICWS 2010*. IEEE, 2010, pp. 139–146.

[25] A. Barker, J. Weissman, and J. van Hemert, "Orchestrating Data-Centric Workflows," in *Proceedings of CCGRID 2008*. IEEE, 2008, pp. 210–217.

[26] W. Binder, I. Constantinescu, and B. Faltings, "Decentralized Orchestration of Composite Web Services," in *Proceedings of ICWS 2006*. IEEE, 2006, pp. 869–876.

[27] G. Tretola and E. Zimeo, "Workflow fine-grained concurrency with automatic continuation," in *Proceedings of IPDPS 2006*. IEEE, 2006, pp. 253–260.

[28] G. Pedraza and J. Estublier, "Distributed Orchestration Versus Choreography: The FOCAS Approach," in *Proceedings of ICSP 2009*, ser. LNCS. Springer, 2009, no. 5543, pp. 75–86.

[29] S. Dami, J. Estublier, and M. Amiour, "APEL: A Graphical Yet Executable Formalism for Process Modeling," *ASE*, vol. 5, pp. 61–91, 1998.

[30] P. Muth, D. Wodtke, J. Weissenfels, A. K. Dittrich, and G. Weikum, "From Centralized Workflow Specification to Distributed Workflow Execution," *JIIS*, vol. 10, no. 2, pp. 159–184, 1998.

[31] S.-S. Jongmans and F. Arbab, "Global Consensus through Local Synchronization (Technical Report)," CWI, Tech. Rep. FM-1303, 2013.

APPENDIX A.
CASE STUDY: DISTRIBUTION OF CA-IMPLEMENTATIONS

Figures 11–15 show the deployment of the four Web services and thirteen CA-implementations (generated by our hybrid-code generator) on machines Red, Green, Blue, Cyan, and Magenta in the case study in Sec. V. Additionally, Figure 16 shows an overview of the full distribution.

APPENDIX B.
PROOF OF LEMMA 2

Before we prove Lemma 2, we first give formal definitions of notation that we introduced in Sec. II. We start with defining the universes of states, ports, memory cells, and data constraints. In all definitions, we make the tacit assumptions that no two different universes overlap (i.e., they are pairwise disjoint). After those definitions, we proceed with defining the accessor functions on constraint automata. Finally, we define bisimilarity in terms of the auxiliary notion of *similarity*.

Definition 4 (Universe of states)**.** The universe of states, denoted by $\mathbb{S}\text{TATE}$, is a set satisfying the following rule:

$$\frac{q_\alpha \in \mathbb{S}\text{TATE} \ \textbf{ and } \ q_\beta \in \mathbb{S}\text{TATE}}{(q_\alpha \, , \, q_\beta) \in \mathbb{S}\text{TATE}}$$

Definition 5 (Universe of ports)**.** The universe of ports, denoted by $\mathbb{P}\text{ORT}$, is a set.

Definition 6 (Universe of memory cells)**.** The universe of memory cells, denoted by $\mathbb{M}\text{EM}$, is a set.

Definition 7 (Universe of data constraints)**.** The universe of data constraints, denoted by $\mathbb{D}\text{C}$, is the set of data formulas generated by the following grammar:

$$\begin{array}{llr}
t & ::= d(p) \ \textbf{for} \ p \in \mathbb{P}\text{ORT} \ \mid \ d(x) \ \textbf{for} \ x \in \mathbb{M}\text{EM} \ \mid \ d(x') \ \textbf{for} \ x \in \mathbb{M}\text{EM} & \text{(data terms)} \\
a & ::= \top \ \mid \ t = t & \text{(data atoms)} \\
f & ::= a \ \mid \ \neg f \ \mid \ f \wedge f \ \mid \ f \vee f & \text{(data formulas)}
\end{array}$$

Definition 8 (Accessor functions on constraint automata)**.** The accessor functions on constraint automata, denoted by State, Port, Mem, and init, are functions from $\mathbb{C}\text{A}$ to $\wp(\mathbb{S}\text{TATE})$, $\wp(\mathbb{P}\text{ORT})$, $\wp(\mathbb{M}\text{EM})$, and $\mathbb{S}\text{TATE}$ defined as:

$$\begin{aligned}
\mathsf{State}((Q \, , \mathcal{P} \, , \mathcal{M} \, , \longrightarrow , \imath)) &= Q \\
\mathsf{Port}((Q \, , \mathcal{P} \, , \mathcal{M} \, , \longrightarrow , \imath)) &= \mathcal{P} \\
\mathsf{Mem}((Q \, , \mathcal{P} \, , \mathcal{M} \, , \longrightarrow , \imath)) &= \mathcal{M} \\
\mathsf{init}((Q \, , \mathcal{P} \, , \mathcal{M} \, , \longrightarrow , \imath)) &= \imath
\end{aligned}$$

Definition 9 (Similarity relation)**.** The similarity relation, denoted by $\preceq$, is the relation on $\mathbb{C}\text{A} \times \wp(\mathbb{S}\text{TATE}^2) \times \mathbb{C}\text{A}$ defined as:

$$\begin{array}{c}(Q_\alpha \, , \mathcal{P}_\alpha \, , \mathcal{M}_\alpha \, , \longrightarrow_\alpha , \imath_\alpha) \\ \preceq^R (Q_\beta \, , \mathcal{P}_\beta \, , \mathcal{M}_\beta \, , \longrightarrow_\beta , \imath_\beta)\end{array} \ \textbf{iff} \ \left[\begin{array}{c} R \subseteq Q_\alpha \times Q_\beta \ \textbf{ and } \ \mathcal{P}_\alpha = \mathcal{P}_\beta \ \textbf{ and } \ \mathcal{M}_\alpha = \mathcal{M}_\beta \ \textbf{ and } \ \imath_\alpha \, R \, \imath_\beta \ \textbf{ and} \\ \left[\begin{array}{c} \left[\left[\begin{array}{l} q_\alpha \xrightarrow{P, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } \ q_\alpha \, R \, q_\beta \end{array}\right] \ \textbf{implies} \ f_\alpha \leq \bigvee \left\{ f_\beta \ \middle| \ \begin{array}{c} [q_\beta \xrightarrow{P, f_\beta}_\beta q'_\beta \ \textbf{ and } \ q'_\alpha \, R \, q'_\beta] \\ \textbf{for some } \ q'_\beta \end{array} \right\} \right] \\ \textbf{for all } \ f_\alpha \, , P \, , q_\alpha \, , q'_\alpha \, , q_\beta \end{array}\right] \end{array}\right]$$

Definition 10 (Bisimilarity relation)**.** The bisimilarity relation, denoted by $\approx$, is the relation on $(\mathbb{C}\text{A} \times \wp(\mathbb{S}\text{TATE}^2) \times \mathbb{C}\text{A}) \cup (\mathbb{C}\text{A} \times \mathbb{C}\text{A})$ defined as:

$$\begin{array}{lll}
\alpha \approx^R \beta & \textbf{iff} & \left[\alpha \preceq^R \beta \ \textbf{ and } \ \beta \preceq^{R^{-1}} \alpha\right] \\
\alpha \approx \beta & \textbf{iff} & \left[\alpha \approx^R \beta \ \textbf{ for some } \ R\right]
\end{array}$$

Next, we prove Lemma 2 by reusing as much as possible the proofs for port automata in [12]. In this case, the main result that we want to reuse is [12, Lemma 2], which states exactly the same property as Lemma 2 for port automata. After carefully analyzing the proof of that result (including the definitions and other results it relies on), we discovered that the differences between port automata and constraint automata manifest only in [12, Lemma 12].[12] To prove Lemma 2, it therefore suffices to prove a similar result as [12, Lemma 12] for constraint automata instead of port automata. We do so in the following lemma.

[12]That lemma appears only in the full version [31] of [12].

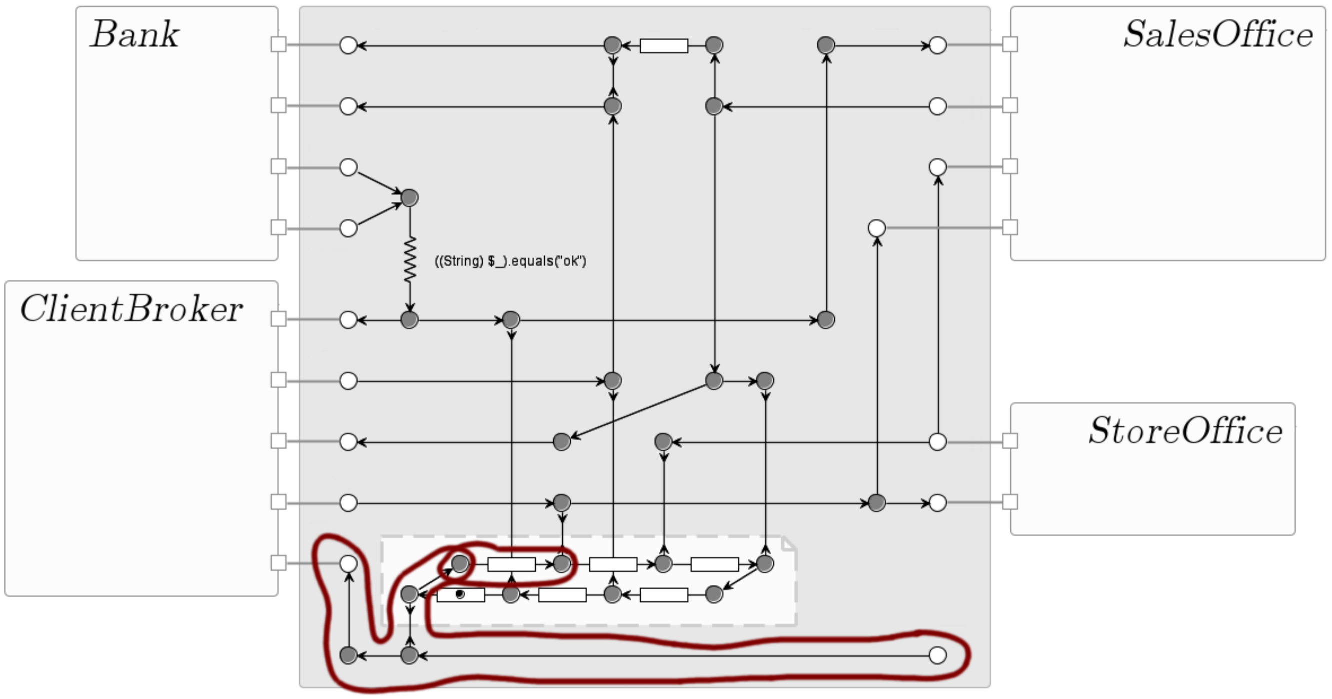


Figure 11: Deployment of two CA-implementations on machine Red.

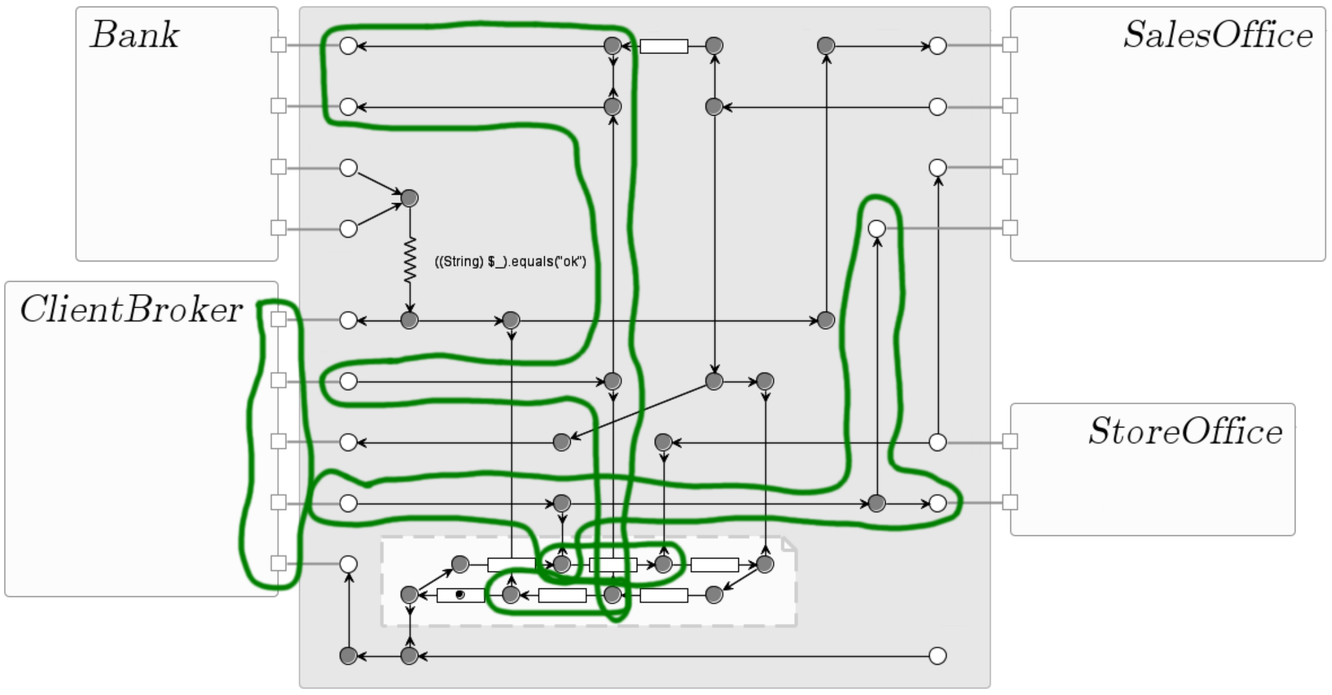


Figure 12: Deployment of four CA-implementations and the *ClientBroker* service on machine Green.

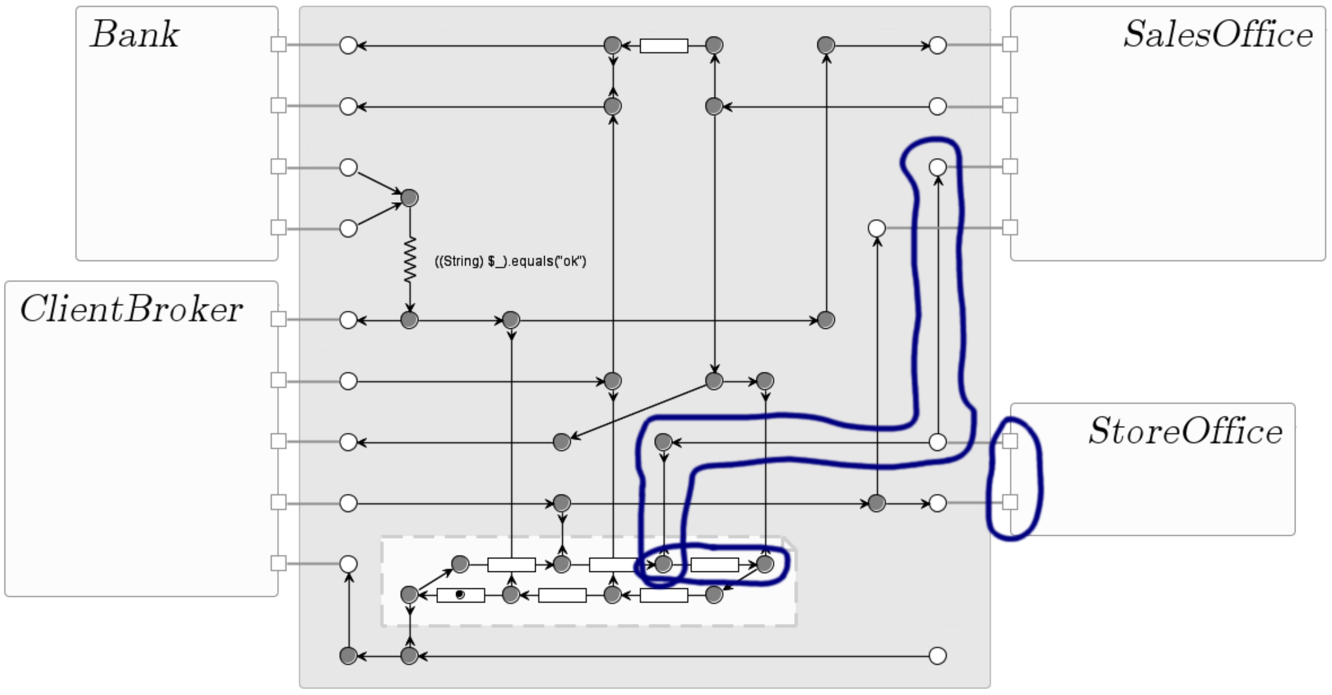


Figure 13: Deployment of two CA-implementations and the *StoreOffice* service on machine Blue.

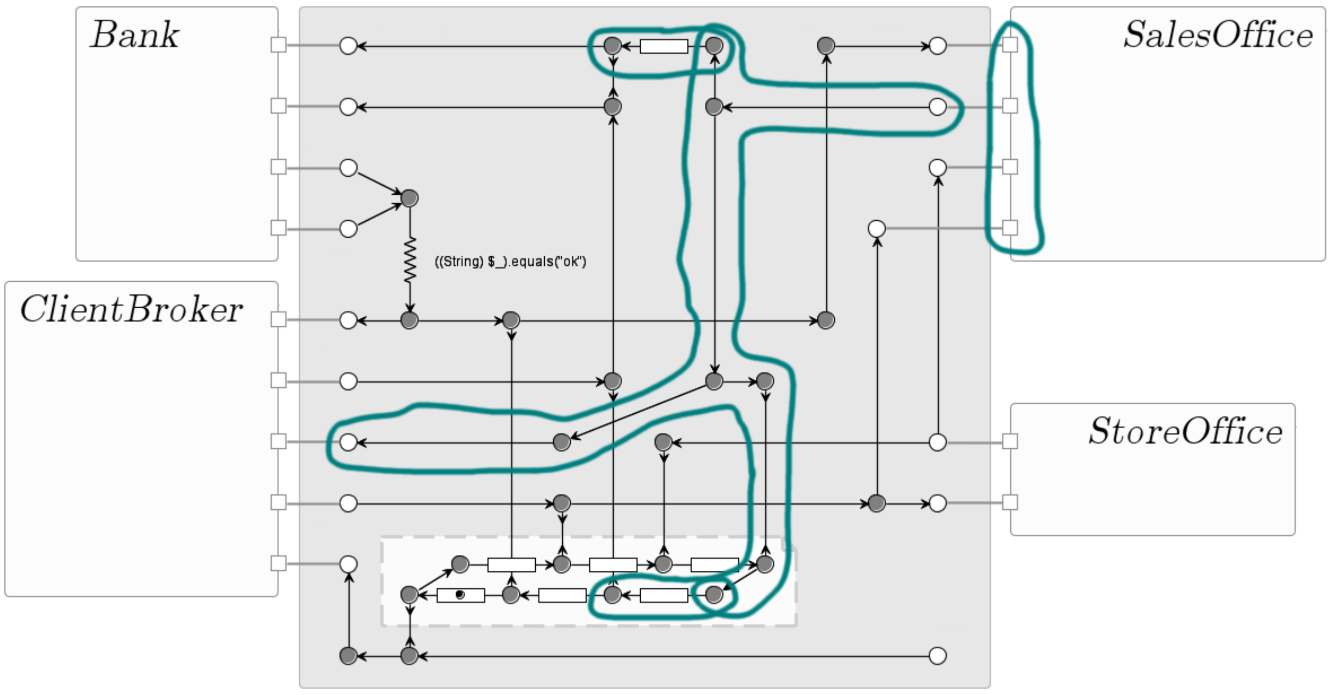


Figure 14: Deployment of three CA-implementations and the *SalesOffice* service on machine Cyan.

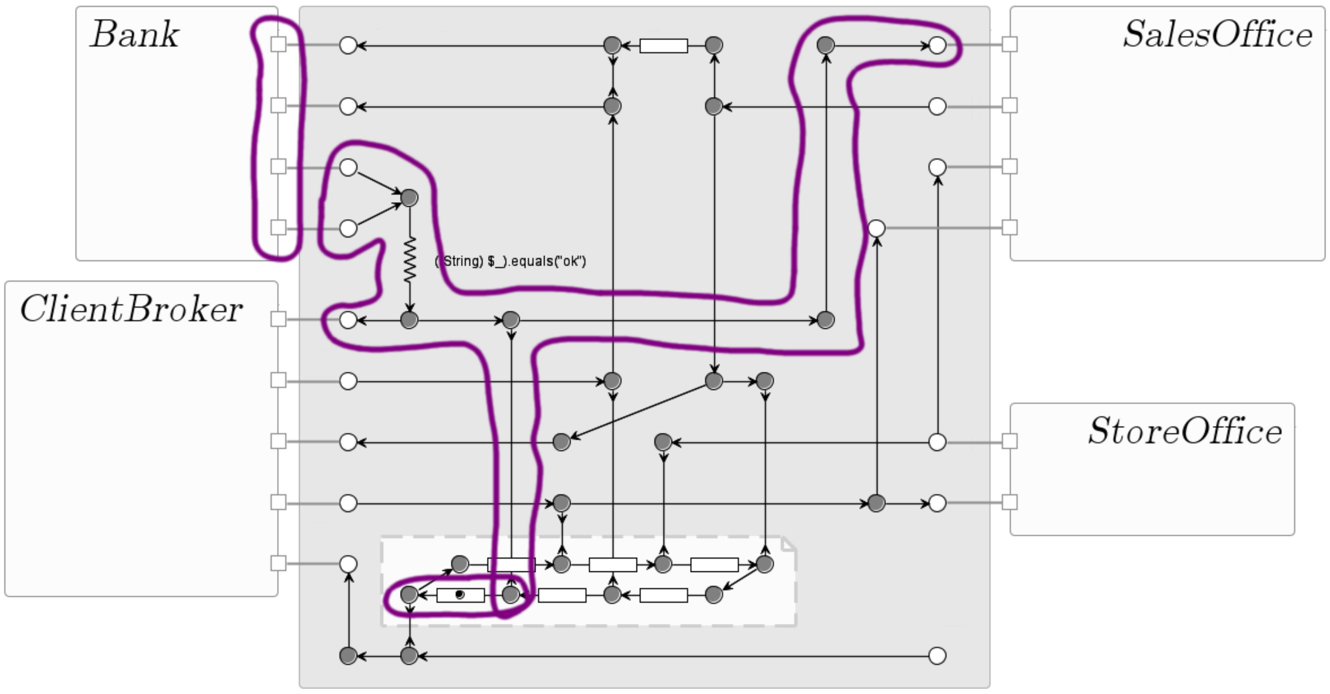


Figure 15: Deployment of two CA-implementations and the *Bank* service on machine Magenta.

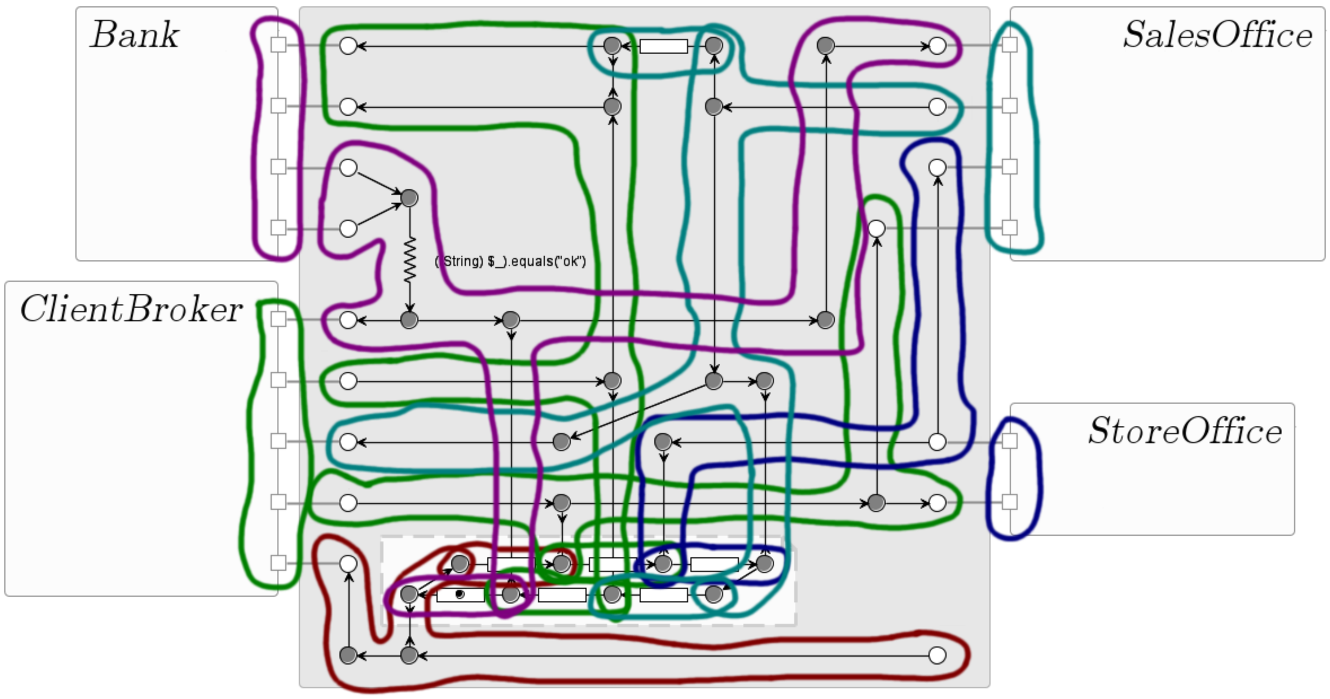


Figure 16: Distribution of all CA-implementations and all services over all machines.

Lemma 5.

$$\big[\alpha \preceq^{R_1} \beta \textbf{ and } \gamma \preceq^{R_2} \delta \textbf{ and} \begin{bmatrix} (q_\alpha\,,\,q_\gamma)\ R\ (q_\beta\,,\,q_\delta)\ \textbf{iff} \\ \big[q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\big] \end{bmatrix}\big] \textbf{ implies } \alpha \boxdot \gamma \preceq^{R} \beta \boxdot \delta$$

Proof: Assume:

$\textcircled{\texttt{A1}}$ $\alpha \preceq^{R_1} \beta$

$\textcircled{\texttt{A2}}$ $\gamma \preceq^{R_2} \delta$

$\textcircled{\texttt{A3}}$ $(q_\alpha\,,\,q_\gamma)\ R\ (q_\beta\,,\,q_\delta)$ **iff** $\big[q_\alpha\ R_1\ q_\beta$ **and** $q_\gamma\ R_2\ q_\delta\big]$

$\textcircled{\texttt{A4}}$ $\alpha = (Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)$

$\textcircled{\texttt{A5}}$ $\beta = (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$

$\textcircled{\texttt{A6}}$ $\gamma = (Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma)$

$\textcircled{\texttt{A7}}$ $\delta = (Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta)$

$\textcircled{\texttt{A8}}$ $\longrightarrow_\dagger$ denotes the smallest relation induced by Rule (2), Rule (3), and Rule (4) in Figure 5 under α and γ.

$\textcircled{\texttt{A9}}$ $\longrightarrow_\ddagger$ denotes the smallest relation induced by Rule (2), Rule (3), and Rule (4) in Figure 5 under β and δ.

Observe:

$\textcircled{\texttt{Z1}}$ Recall $\gamma \preceq^{R_2} \delta$ from $\textcircled{\texttt{A2}}$. Then, by applying $\textcircled{\texttt{A6}}$, conclude $(Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} \delta$. Then, by applying $\textcircled{\texttt{A7}}$, conclude $(Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} (Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta)$. Then, by applying Definition 9 of $\preceq$, conclude $\mathcal{P}_\gamma = \mathcal{P}_\delta$. Then, by applying Definition 8 of Port, conclude $\mathsf{Port}((Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma)) = \mathsf{Port}((Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta))$. Then, by applying $\textcircled{\texttt{A6}}$, conclude $\mathsf{Port}(\gamma) = \mathsf{Port}((Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta))$. Then, by applying $\textcircled{\texttt{A7}}$, conclude $\mathsf{Port}(\gamma) = \mathsf{Port}(\delta)$.

$\textcircled{\texttt{Z2}}$ Recall $\gamma \preceq^{R_2} \delta$ from $\textcircled{\texttt{A2}}$. Then, by introducing $\textcircled{\texttt{A1}}$, conclude $\big[\alpha \preceq^{R_1} \beta$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A4}}$, conclude $\big[(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} \beta$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A5}}$, conclude $\big[(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A6}}$, conclude $(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$ **and** $(Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} \delta$. Then, by applying $\textcircled{\texttt{A7}}$, conclude:

$$\begin{gathered}(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)\\ \textbf{and } (Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} (Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta)\end{gathered}$$

Then, by applying Definition 9 of $\preceq$, conclude $\big[\mathcal{P}_\alpha = \mathcal{P}_\beta$ **and** $\mathcal{M}_\alpha = \mathcal{M}_\beta$ **and** $\imath_\alpha\ R\ \imath_\beta\big]$ **and** $\big[\mathcal{P}_\gamma = \mathcal{P}_\delta$ **and** $\mathcal{M}_\gamma = \mathcal{M}_\delta$ **and** $\imath_\gamma\ R\ \imath_\delta\big]$. Then, by basic rewriting, conclude $\big[\big[\mathcal{P}_\alpha = \mathcal{P}_\beta$ **and** $\mathcal{P}_\gamma = \mathcal{P}_\delta\big]$ **and** $\big[\mathcal{M}_\alpha = \mathcal{M}_\beta$ **and** $\mathcal{M}_\gamma = \mathcal{M}_\delta\big]$ **and** $\big[\imath_\alpha\ R\ \imath_\beta$ **and** $\imath_\gamma\ R\ \imath_\delta\big]\big]$. Then, by applying $\textcircled{\texttt{A3}}$, conclude $\big[\big[\mathcal{P}_\alpha = \mathcal{P}_\beta$ **and** $\mathcal{P}_\gamma = \mathcal{P}_\delta\big]$ **and** $\big[\mathcal{M}_\alpha = \mathcal{M}_\beta$ **and** $\mathcal{M}_\gamma = \mathcal{M}_\delta\big]$ **and** $(\imath_\alpha\,,\,\imath_\gamma)\ R\ (\imath_\beta\,,\,\imath_\delta)\big]$. Then, by rewriting under ZFC, conclude, $\big[\mathcal{P}_\alpha \cup \mathcal{P}_\gamma = \mathcal{P}_\beta \cup \mathcal{P}_\delta$ **and** $\mathcal{M}_\alpha \cup \mathcal{M}_\gamma = \mathcal{M}_\beta \cup \mathcal{M}_\delta$ **and** $(\imath_\alpha\,,\,\imath_\gamma)\ R\ (\imath_\beta\,,\,\imath_\delta)\big]$.

$\textcircled{\texttt{Z3}}$ Recall $\gamma \preceq^{R_2} \delta$ from $\textcircled{\texttt{A2}}$. Then, by introducing $\textcircled{\texttt{A1}}$, conclude $\big[\alpha \preceq^{R_1} \beta$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A4}}$, conclude $\big[(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} \beta$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A5}}$, conclude $\big[(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$ **and** $\gamma \preceq^{R_2} \delta\big]$. Then, by applying $\textcircled{\texttt{A6}}$, conclude $(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$ **and** $(Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} \delta$. Then, by applying $\textcircled{\texttt{A7}}$, conclude:

$$\begin{gathered}(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)\\ \textbf{and } (Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} (Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta)\end{gathered}$$

Then, by applying Definition 9 of $\preceq$, conclude $\big[R_1 \subseteq Q_\alpha \times Q_\beta$ **and** $R_2 \subseteq Q_\gamma \times Q_\delta\big]$. Then, by rewriting under ZFC, conclude:

$$\begin{gathered}\big[\big[q_\alpha\ R_1\ q_\beta \textbf{ implies } \big[q_\alpha \in Q_\alpha \textbf{ and } q_\beta \in Q_\beta\big]\big] \textbf{ for all } q_\alpha\,,\,q_\beta\big]\\ \textbf{and } \big[\big[q_\gamma\ R_2\ q_\delta \textbf{ implies } \big[q_\gamma \in Q_\gamma \textbf{ and } q_\delta \in Q_\delta\big]\big] \textbf{ for all } q_\gamma\,,\,q_\delta\big]\end{gathered}$$

Then, by basic rewriting, conclude:

$$\big[\big[q_\alpha\ R_1\ q_\beta \textbf{ and } q_\gamma\ R_2\ q_\delta\big] \textbf{ implies } \begin{bmatrix} q_\alpha \in Q_\alpha \textbf{ and } q_\beta \in Q_\beta \\ \textbf{and } q_\gamma \in Q_\gamma \textbf{ and } q_\delta \in Q_\delta\end{bmatrix}\big] \textbf{ for all } q_\alpha\,,\,q_\beta\,,\,q_\gamma\,,\,q_\delta$$

Then, by rewriting under ZFC, conclude:

$$\big[\big[q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\big]\ \textbf{ implies }\ \left[\begin{array}{c}(q_\alpha\,,\,q_\gamma)\in Q_\alpha\times Q_\gamma\\ \textbf{and }\ (q_\beta\,,\,q_\delta)\in Q_\beta\times Q_\delta\end{array}\right]\big]\ \textbf{ for all }\ q_\alpha\,,\,q_\beta\,,\,q_\gamma\,,\,q_\delta$$

Then, by applying $\textcircled{A3}$, conclude:

$$\big[(q_\alpha\,,\,q_\gamma)\ R\ (q_\beta\,,\,q_\delta)\ \textbf{ implies }\ \left[\begin{array}{c}(q_\alpha\,,\,q_\gamma)\in Q_\alpha\times Q_\gamma\\ \textbf{and }\ (q_\beta\,,\,q_\delta)\in Q_\beta\times Q_\delta\end{array}\right]\big]\ \textbf{ for all }\ q_\alpha\,,\,q_\beta\,,\,q_\gamma\,,\,q_\delta$$

Then, by rewriting under ZFC, conclude $R\subseteq(Q_\alpha\times Q_\gamma)\times(Q_\beta\times Q_\delta)$.

Reasoning to a generalization, suppose:

$$\big[(q_\alpha\,,\,q_\gamma)\xrightarrow{P,f}_\dagger(q'_\alpha\,,\,q'_\gamma)\ \textbf{ and }\ (q_\alpha\,,\,q_\gamma)\ R\ (q_\beta\,,\,q_\delta)\big]\ \textbf{ for some }\ P\,,\,f\,,\,q_\alpha\,,\,q'_\alpha\,,\,q_\beta\,,\,q_\gamma\,,\,q'_\gamma\,,\,q_\delta$$

Then, by applying $\textcircled{A3}$, conclude $\big[(q_\alpha\,,\,q_\gamma)\xrightarrow{P,f}_\dagger(q'_\alpha\,,\,q'_\gamma)\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\big]$. Then, by applying $\textcircled{A8}$, conclude $\big[\big[\big[\text{Rule (2) applies}\big]\ \textbf{ or }\ \big[\text{Rule (3) applies}\big]\ \textbf{ or }\ \big[\text{Rule (4) applies}\big]\big]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\big]$. Then, by basic rewriting, conclude:

$$\left[\begin{array}{c}\big[\text{Rule (2) applies}\big]\ \textbf{ and}\\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}\right]\ \textbf{ or }\ \left[\begin{array}{c}\big[\text{Rule (3) applies}\big]\ \textbf{ and}\\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}\right]\ \textbf{ or }\ \left[\begin{array}{c}\big[\text{Rule (4) applies}\big]\ \textbf{ and}\\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}\right]$$

Proceed by case distinction.

- **Case:** $\big[\big[\text{Rule (2) applies}\big]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\big]$.
  Then, by applying the definition of Rule (2) in Figure 5, conclude:

$$\left[\begin{array}{l}P=P_\alpha\cup P_\gamma\ \textbf{ and }\ f=f_\alpha\wedge f_\gamma\ \textbf{ and }\ q_\alpha\xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha\ \textbf{ and }\ q_\gamma\xrightarrow{P_\gamma,f_\gamma}_\gamma q'_\gamma\ \textbf{ and}\\ \left[\begin{array}{c}P_\alpha=\mathsf{Port}(\alpha)\cap P_\gamma\ \textbf{ or }\ P_\gamma=\mathsf{Port}(\gamma)\cap P_\alpha\\ \textbf{or }\ \mathsf{Port}(\alpha)\cap P_\gamma=\emptyset=\mathsf{Port}(\gamma)\cap P_\alpha\end{array}\right]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}\right]\ \textbf{ for some }\ f_\alpha\,,\,f_\gamma\,,\,P_\alpha\,,\,P_\gamma$$

  Then, by introducing $\textcircled{A2}$, conclude:

$$\begin{array}{c}\gamma\preceq^{R_2}\delta\ \textbf{ and }\ P=P_\alpha\cup P_\gamma\ \textbf{ and }\ f=f_\alpha\wedge f_\gamma\ \textbf{ and }\ q_\alpha\xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha\ \textbf{ and }\ q_\gamma\xrightarrow{P_\gamma,f_\gamma}_\gamma q'_\gamma\\ \textbf{and }\ \left[\begin{array}{c}P_\alpha=\mathsf{Port}(\alpha)\cap P_\gamma\ \textbf{ or }\ P_\gamma=\mathsf{Port}(\gamma)\cap P_\alpha\\ \textbf{or }\ \mathsf{Port}(\alpha)\cap P_\gamma=\emptyset=\mathsf{Port}(\gamma)\cap P_\alpha\end{array}\right]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}$$

  Then, by introducing $\textcircled{A1}$, conclude:

$$\begin{array}{c}\alpha\preceq^{R_1}\beta\ \textbf{ and }\ \gamma\preceq^{R_2}\delta\ \textbf{ and }\ P=P_\alpha\cup P_\gamma\ \textbf{ and }\ f=f_\alpha\wedge f_\gamma\ \textbf{ and }\ q_\alpha\xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha\ \textbf{ and }\ q_\gamma\xrightarrow{P_\gamma,f_\gamma}_\gamma q'_\gamma\\ \textbf{and }\ \left[\begin{array}{c}P_\alpha=\mathsf{Port}(\alpha)\cap P_\gamma\ \textbf{ or }\ P_\gamma=\mathsf{Port}(\gamma)\cap P_\alpha\\ \textbf{or }\ \mathsf{Port}(\alpha)\cap P_\gamma=\emptyset=\mathsf{Port}(\gamma)\cap P_\alpha\end{array}\right]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}$$

  Then, by applying $\textcircled{A4}$, conclude:

$$\begin{array}{c}(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)\preceq^{R_1}\beta\ \textbf{ and }\ \gamma\preceq^{R_2}\delta\ \textbf{ and}\\ P=P_\alpha\cup P_\gamma\ \textbf{ and }\ f=f_\alpha\wedge f_\gamma\ \textbf{ and }\ q_\alpha\xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha\ \textbf{ and }\ q_\gamma\xrightarrow{P_\gamma,f_\gamma}_\gamma q'_\gamma\\ \textbf{and }\ \left[\begin{array}{c}P_\alpha=\mathsf{Port}(\alpha)\cap P_\gamma\ \textbf{ or }\ P_\gamma=\mathsf{Port}(\gamma)\cap P_\alpha\\ \textbf{or }\ \mathsf{Port}(\alpha)\cap P_\gamma=\emptyset=\mathsf{Port}(\gamma)\cap P_\alpha\end{array}\right]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}$$

  Then, by applying $\textcircled{A5}$, conclude:

$$\begin{array}{c}(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)\preceq^{R_1}(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)\ \textbf{ and }\ \gamma\preceq^{R_2}\delta\ \textbf{ and}\\ P=P_\alpha\cup P_\gamma\ \textbf{ and }\ f=f_\alpha\wedge f_\gamma\ \textbf{ and }\ q_\alpha\xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha\ \textbf{ and }\ q_\gamma\xrightarrow{P_\gamma,f_\gamma}_\gamma q'_\gamma\\ \textbf{and }\ \left[\begin{array}{c}P_\alpha=\mathsf{Port}(\alpha)\cap P_\gamma\ \textbf{ or }\ P_\gamma=\mathsf{Port}(\gamma)\cap P_\alpha\\ \textbf{or }\ \mathsf{Port}(\alpha)\cap P_\gamma=\emptyset=\mathsf{Port}(\gamma)\cap P_\alpha\end{array}\right]\ \textbf{ and }\ q_\alpha\ R_1\ q_\beta\ \textbf{ and }\ q_\gamma\ R_2\ q_\delta\end{array}$$

Then, by applying $\textcircled{\texttt{A6}}$, conclude:

$$\begin{gathered}
(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \textbf{ and } (Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} \delta \textbf{ and}\\
P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\gamma \xrightarrow{P_\gamma , f_\gamma}_\gamma q'_\gamma\\
\textbf{and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix} \textbf{ and } q_\alpha \; R_1 \; q_\beta \textbf{ and } q_\gamma \; R_2 \; q_\delta
\end{gathered}$$

Then, by applying $\textcircled{\texttt{A7}}$, conclude:

$$\begin{gathered}
(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \preceq^{R_1} (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \textbf{ and}\\
(Q_\gamma\,,\,\mathcal{P}_\gamma\,,\,\mathcal{M}_\gamma\,,\,\longrightarrow_\gamma\,,\,\imath_\gamma) \preceq^{R_2} (Q_\delta\,,\,\mathcal{P}_\delta\,,\,\mathcal{M}_\delta\,,\,\longrightarrow_\delta\,,\,\imath_\delta) \textbf{ and}\\
P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\gamma \xrightarrow{P_\gamma , f_\gamma}_\gamma q'_\gamma\\
\textbf{and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix} \textbf{ and } q_\alpha \; R_1 \; q_\beta \textbf{ and } q_\gamma \; R_2 \; q_\delta
\end{gathered}$$

Then, by applying Definition 9 of $\preceq$, conclude:

$$\begin{gathered}
\begin{bmatrix} \Big[\begin{bmatrix} q_\alpha \xrightarrow{P , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } q_\alpha \; R_1 \; q_\beta \end{bmatrix} \textbf{ implies } f_\alpha \leq \bigvee \left\{ f_\beta \;\middle|\; \begin{matrix} [q_\beta \xrightarrow{P , f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha \; R_1 \; q'_\beta] \\ \textbf{for some } q'_\beta \end{matrix} \right\} \Big] \\ \textbf{for all } f_\alpha\,,\,P\,,\,q_\alpha\,,\,q'_\alpha\,,\,q_\beta \end{bmatrix} \textbf{ and}\\
\begin{bmatrix} \Big[\begin{bmatrix} q_\gamma \xrightarrow{P , f_\gamma}_\gamma q'_\gamma \\ \textbf{and } q_\gamma \; R_2 \; q_\delta \end{bmatrix} \textbf{ implies } f_\gamma \leq \bigvee \left\{ f_\delta \;\middle|\; \begin{matrix} [q_\delta \xrightarrow{P , f_\delta}_\delta q'_\delta \textbf{ and } q'_\gamma \; R_2 \; q'_\delta] \\ \textbf{for some } q'_\delta \end{matrix} \right\} \Big] \\ \textbf{for all } f_\gamma\,,\,P\,,\,q_\gamma\,,\,q'_\gamma\,,\,q_\delta \end{bmatrix} \textbf{ and}\\
P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\gamma \xrightarrow{P_\gamma , f_\gamma}_\gamma q'_\gamma\\
\textbf{and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix} \textbf{ and } q_\alpha \; R_1 \; q_\beta \textbf{ and } q_\gamma \; R_2 \; q_\delta
\end{gathered}$$

Then, by basic rewriting, conclude:

$$\begin{gathered}
f_\alpha \leq \bigvee \left\{ f_\beta \;\middle|\; \begin{matrix} [q_\beta \xrightarrow{P_\alpha , f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha \; R_1 \; q'_\beta] \\ \textbf{for some } q'_\beta \end{matrix} \right\} \textbf{ and } f_\gamma \leq \bigvee \left\{ f_\delta \;\middle|\; \begin{matrix} [q_\delta \xrightarrow{P_\gamma , f_\delta}_\delta q'_\delta \textbf{ and } q'_\gamma \; R_2 \; q'_\delta] \\ \textbf{for some } q'_\delta \end{matrix} \right\}\\
\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix}
\end{gathered}$$

Then, by rewriting under ZFC, conclude:

$$\begin{gathered}
f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \;\middle|\; \begin{matrix} [q_\beta \xrightarrow{P_\alpha , f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha \; R_1 \; q'_\beta] \\ \textbf{for some } q'_\beta \end{matrix} \right\} \wedge \bigvee \left\{ f_\delta \;\middle|\; \begin{matrix} [q_\delta \xrightarrow{P_\gamma , f_\delta}_\delta q'_\delta \textbf{ and } q'_\gamma \; R_2 \; q'_\delta] \\ \textbf{for some } q'_\delta \end{matrix} \right\}\\
\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix}
\end{gathered}$$

Then, by rewriting under ZFC, conclude:

$$\begin{gathered}
f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \wedge f_\delta \;\middle|\; \begin{matrix} [q_\beta \xrightarrow{P_\alpha , f_\beta}_\beta q'_\beta \textbf{ and } q_\delta \xrightarrow{P_\gamma , f_\delta}_\delta q'_\delta \textbf{ and } q'_\alpha \; R_1 \; q'_\beta \textbf{ and } q'_\gamma \; R_2 \; q'_\delta] \\ \textbf{for some } q'_\beta\,,\,q'_\delta \end{matrix} \right\}\\
\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix}
\end{gathered}$$

Then, by applying $\textcircled{\scriptsize A3}$, conclude:

$$f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \wedge f_\delta \;\middle|\; \begin{array}{c} \big[q_\beta \xrightarrow{P_\alpha , f_\beta}_\beta q'_\beta \textbf{ and } q_\delta \xrightarrow{P_\gamma , f_\delta}_\delta q'_\delta \textbf{ and } (q'_\alpha \,,\, q'_\gamma)\ R\ (q'_\beta \,,\, q'_\delta)\big] \\ \textbf{for some } q'_\beta \,,\, q'_\delta \end{array} \right\}$$

$$\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma \textbf{ and } \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix}$$

Then, by rewriting under ZFC, conclude:

$$f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \wedge f_\delta \;\middle|\; \begin{array}{c} \left[\begin{array}{c} \begin{bmatrix} P_\alpha = \mathsf{Port}(\alpha) \cap P_\gamma \textbf{ or } P_\gamma = \mathsf{Port}(\gamma) \cap P_\alpha \\ \textbf{or } \mathsf{Port}(\alpha) \cap P_\gamma = \emptyset = \mathsf{Port}(\gamma) \cap P_\alpha \end{bmatrix} \textbf{ and} \\ q_\beta \xrightarrow{P_\alpha , f_\beta}_\beta q'_\beta \textbf{ and } q_\delta \xrightarrow{P_\gamma , f_\delta}_\delta q'_\delta \textbf{ and } (q'_\alpha \,,\, q'_\gamma)\ R\ (q'_\beta \,,\, q'_\delta) \end{array}\right] \\ \textbf{for some } q'_\beta \,,\, q'_\delta \end{array} \right\}$$

$$\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma$$

Then, by applying the definition of Rule (2) in Figure 5, conclude:

$$f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \wedge f_\delta \;\middle|\; \begin{array}{c} \big[\big[\text{Rule (2) applies}\big] \textbf{ and } (q'_\alpha \,,\, q'_\gamma)\ R\ (q'_\beta \,,\, q'_\delta)\big] \\ \textbf{for some } q'_\beta \,,\, q'_\delta \end{array} \right\}$$

$$\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma$$

Then, by applying $\textcircled{\scriptsize A9}$, conclude:

$$f_\alpha \wedge f_\gamma \leq \bigvee \left\{ f_\beta \wedge f_\delta \;\middle|\; \begin{array}{c} \big[(q_\beta \,,\, q_\delta) \xrightarrow{P_\alpha \cup P_\gamma , f_\beta \wedge f_\delta}_\ddagger (q'_\beta \,,\, q'_\delta) \textbf{ and } (q'_\alpha \,,\, q'_\gamma)\ R\ (q'_\beta \,,\, q'_\delta)\big] \\ \textbf{for some } q'_\beta \,,\, q'_\delta \end{array} \right\}$$

$$\textbf{and } P = P_\alpha \cup P_\gamma \textbf{ and } f = f_\alpha \wedge f_\gamma$$

Then, by basic rewriting, conclude:

$$f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c} \big[(q_\beta \,,\, q_\delta) \xrightarrow{P , f'}_\ddagger q' \textbf{ and } (q'_\alpha \,,\, q'_\gamma)\ R\ q'\big] \\ \textbf{for some } q' \end{array} \right\}$$

- **Case:** $\big[\big[\text{Rule (3) applies}\big] \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q_\gamma\ R_2\ q_\delta\big]$.

Then, by applying the definition of Rule (3) in Figure 5, conclude:

$$\begin{bmatrix} P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\gamma = q'_\gamma \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q_\gamma\ R_2\ q_\delta \end{bmatrix} \textbf{ for some } f_\alpha \,,\, P_\alpha$$

Then, by basic rewriting, conclude:

$$\begin{array}{c} P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta \end{array}$$

Then, by introducing $\textcircled{\scriptsize A2}$, conclude:

$$\begin{array}{c} \gamma \preceq^{R_2} \delta \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta \end{array}$$

Then, by introducing $\textcircled{\scriptsize A1}$, conclude:

$$\begin{array}{c} \alpha \preceq^{R_1} \beta \textbf{ and } \gamma \preceq^{R_2} \delta \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta \end{array}$$

Then, by applying $\textcircled{\scriptsize A4}$, conclude:

$$\begin{array}{c} (Q_\alpha \,,\, \mathcal{P}_\alpha \,,\, \mathcal{M}_\alpha \,,\, \longrightarrow_\alpha \,,\, \imath_\alpha) \preceq^{R_1} \beta \textbf{ and } \gamma \preceq^{R_2} \delta \textbf{ and} \\ P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta \end{array}$$

Then, by applying $\textcircled{\texttt{A5}}$, conclude:

$$\begin{gathered}(Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha) \preceq^{R_1} (Q_\beta\,, \mathcal{P}_\beta\,, \mathcal{M}_\beta\,, \longrightarrow_\beta\,, \imath_\beta) \textbf{ and } \gamma \preceq^{R_2} \delta \textbf{ and}\\ P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by applying $\textcircled{\texttt{A6}}$, conclude:

$$\begin{gathered}(Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha) \preceq^{R_1} (Q_\beta\,, \mathcal{P}_\beta\,, \mathcal{M}_\beta\,, \longrightarrow_\beta\,, \imath_\beta) \textbf{ and } (Q_\gamma\,, \mathcal{P}_\gamma\,, \mathcal{M}_\gamma\,, \longrightarrow_\gamma\,, \imath_\gamma) \preceq^{R_2} \delta \textbf{ and}\\ P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by applying $\textcircled{\texttt{A7}}$, conclude:

$$\begin{gathered}(Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha) \preceq^{R_1} (Q_\beta\,, \mathcal{P}_\beta\,, \mathcal{M}_\beta\,, \longrightarrow_\beta\,, \imath_\beta) \textbf{ and}\\ (Q_\gamma\,, \mathcal{P}_\gamma\,, \mathcal{M}_\gamma\,, \longrightarrow_\gamma\,, \imath_\gamma) \preceq^{R_2} (Q_\delta\,, \mathcal{P}_\delta\,, \mathcal{M}_\delta\,, \longrightarrow_\delta\,, \imath_\delta) \textbf{ and}\\ P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by applying Definition 9 of $\preceq$, conclude:

$$\begin{gathered}\left[\begin{array}{c}\Big[\left[\begin{matrix} q_\alpha \xrightarrow{P, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } q_\alpha\ R_1\ q_\beta\end{matrix}\right] \textbf{ implies } f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\big[q_\beta \xrightarrow{P, f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha\ R_1\ q'_\beta\big] \\ \textbf{for some } q'_\beta\end{matrix}\right\}\Big] \\ \textbf{for all } f_\alpha\,, P\,, q_\alpha\,, q'_\alpha\,, q_\beta\end{array}\right] \textbf{ and } R_2 \subseteq Q_\gamma \times Q_\delta \\ \textbf{and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \\ \textbf{and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q_\alpha\ R_1\ q_\beta \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by basic rewriting, conclude:

$$\begin{gathered}f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\big[q_\beta \xrightarrow{P_\alpha, f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha\ R_1\ q'_\beta\big] \\ \textbf{for some } q'_\beta\end{matrix}\right\} \textbf{ and } R_2 \subseteq Q_\gamma \times Q_\delta \\ \textbf{and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by rewriting under ZFC, conclude:

$$\begin{gathered}f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\big[q_\beta \xrightarrow{P_\alpha, f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha\ R_1\ q'_\beta\big] \\ \textbf{for some } q'_\beta\end{matrix}\right\} \textbf{ and } q_\delta \in Q_\delta \\ \textbf{and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } P_\alpha \cap \mathsf{Port}(\gamma) = \emptyset \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by applying $\textcircled{\texttt{Z1}}$, conclude:

$$\begin{gathered}f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\big[q_\beta \xrightarrow{P_\alpha, f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha\ R_1\ q'_\beta\big] \\ \textbf{for some } q'_\beta\end{matrix}\right\} \textbf{ and } q_\delta \in Q_\delta \\ \textbf{and } P = P_\alpha \textbf{ and } f = f_\alpha \textbf{ and } P_\alpha \cap \mathsf{Port}(\delta) = \emptyset \textbf{ and } q'_\gamma\ R_2\ q_\delta\end{gathered}$$

Then, by rewriting under ZFC, conclude:

$$f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\left[\begin{matrix}q_\delta \in Q_\delta \textbf{ and } P_\alpha \cap \mathsf{Port}(\delta) = \emptyset \textbf{ and } q'_\gamma\ R_2\ q_\delta \\ \textbf{and } q_\beta \xrightarrow{P_\alpha, f_\beta}_\beta q'_\beta \textbf{ and } q'_\alpha\ R_1\ q'_\beta\end{matrix}\right] \\ \textbf{for some } q'_\beta\end{matrix}\right\} \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha$$

Then, by applying $\textcircled{\texttt{A3}}$, conclude:

$$f_\alpha \leq \bigvee\left\{ f_\beta \;\middle|\; \begin{matrix}\left[\begin{matrix}q_\delta \in Q_\delta \textbf{ and } P_\alpha \cap \mathsf{Port}(\delta) = \emptyset \textbf{ and} \\ q_\beta \xrightarrow{P_\alpha, f_\beta}_\beta q'_\beta \textbf{ and } (q'_\alpha\,, q'_\gamma)\ R\ (q'_\beta\,, q_\delta)\end{matrix}\right] \\ \textbf{for some } q'_\beta\end{matrix}\right\} \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha$$

Then, by applying the definition of Rule (3) in Figure 5, conclude:

$$f_\alpha \leq \bigvee \left\{ f_\beta \;\middle|\; \begin{array}{c}\big[[\text{Rule (3) applies}] \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; (q'_\beta\,,\, q_\delta)\big]\\ \textbf{for some } q'_\beta\end{array}\right\} \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha$$

Then, by applying $\textcircled{\texttt{A9}}$, conclude:

$$f_\alpha \leq \bigvee \left\{ f_\beta \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P_\alpha, f_\beta}_\ddagger (q'_\beta\,,\, q_\delta) \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; (q'_\beta\,,\, q_\delta)\big]\\ \textbf{for some } q'_\beta\end{array}\right\} \textbf{ and } P = P_\alpha \textbf{ and } f = f_\alpha$$

Then, by basic rewriting, conclude:

$$f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P, f'}_\ddagger q' \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; q'\big]\\ \textbf{for some } q'\end{array}\right\}$$

- **Case:** $\big[[\text{Rule (4) applies}] \textbf{ and } q_\alpha\; R_1\; q_\beta \textbf{ and } q_\gamma\; R_2\; q_\delta\big]$. Symmetrically.

Hence, after considering all cases, conclude:

$$f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P, f'}_\ddagger q' \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; q'\big]\\ \textbf{for some } q'\end{array}\right\}$$

Then, by generalizing the premise, conclude:

$$\begin{array}{c}\left[\begin{bmatrix}(q_\alpha\,,\, q_\gamma) \xrightarrow{P}_\dagger (q'_\alpha\,,\, q'_\gamma)\\ \textbf{and } (q_\alpha\,,\, q_\gamma)\; R\; (q_\beta\,,\, q_\delta)\end{bmatrix} \textbf{ implies } f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P, f'}_\ddagger q' \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; q'\big]\\ \textbf{for some } q'\end{array}\right\}\right]\\ \textbf{for all } q_\alpha\,,\, q_\beta\,,\, q_\gamma\,,\, q_\delta\,,\, q'_\alpha\,,\, q'_\gamma\,,\, P\end{array}$$

Then, by introducing $\textcircled{\texttt{Z2}}$, conclude:

$$\begin{array}{c}\mathcal{P}_\alpha \cup \mathcal{P}_\gamma = \mathcal{P}_\beta \cup \mathcal{P}_\delta \textbf{ and } \mathcal{M}_\alpha \cup \mathcal{M}_\gamma = \mathcal{M}_\beta \cup \mathcal{M}_\delta \textbf{ and } (\imath_\alpha\,,\, \imath_\gamma)\; R\; (\imath_\beta\,,\, \imath_\delta) \textbf{ and }\\ \begin{bmatrix}\left[\begin{bmatrix}(q_\alpha\,,\, q_\gamma) \xrightarrow{P}_\dagger (q'_\alpha\,,\, q'_\gamma)\\ \textbf{and } (q_\alpha\,,\, q_\gamma)\; R\; (q_\beta\,,\, q_\delta)\end{bmatrix} \textbf{ implies } f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P, f'}_\ddagger q' \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; q'\big]\\ \textbf{for some } q'\end{array}\right\}\right]\\ \textbf{for all } q_\alpha\,,\, q_\beta\,,\, q_\gamma\,,\, q_\delta\,,\, q'_\alpha\,,\, q'_\gamma\,,\, P\end{bmatrix}\end{array}$$

Then, by introducing $\textcircled{\texttt{Z3}}$, conclude:

$$\begin{array}{c}R \subseteq (Q_\alpha \times Q_\gamma) \times (Q_\beta \times Q_\delta) \textbf{ and } \mathcal{P}_\alpha \cup \mathcal{P}_\gamma = \mathcal{P}_\beta \cup \mathcal{P}_\delta \textbf{ and } \mathcal{M}_\alpha \cup \mathcal{M}_\gamma = \mathcal{M}_\beta \cup \mathcal{M}_\delta \textbf{ and } (\imath_\alpha\,,\, \imath_\gamma)\; R\; (\imath_\beta\,,\, \imath_\delta) \textbf{ and }\\ \begin{bmatrix}\left[\begin{bmatrix}(q_\alpha\,,\, q_\gamma) \xrightarrow{P}_\dagger (q'_\alpha\,,\, q'_\gamma)\\ \textbf{and } (q_\alpha\,,\, q_\gamma)\; R\; (q_\beta\,,\, q_\delta)\end{bmatrix} \textbf{ implies } f \leq \bigvee \left\{ f' \;\middle|\; \begin{array}{c}\big[(q_\beta\,,\, q_\delta) \xrightarrow{P, f'}_\ddagger q' \textbf{ and } (q'_\alpha\,,\, q'_\gamma)\; R\; q'\big]\\ \textbf{for some } q'\end{array}\right\}\right]\\ \textbf{for all } q_\alpha\,,\, q_\beta\,,\, q_\gamma\,,\, q_\delta\,,\, q'_\alpha\,,\, q'_\gamma\,,\, P\end{bmatrix}\end{array}$$

Then, by applying Definition 9 of $\preceq$, conclude $(Q_\alpha \times Q_\gamma\,,\, \mathcal{P}_\alpha \cup \mathcal{P}_\gamma\,,\, \mathcal{M}_\alpha \cup \mathcal{M}_\gamma\,,\, \longrightarrow_\dagger\,,\, (\imath_\alpha\,,\, \imath_\gamma)) \preceq^R (Q_\beta \times Q_\delta\,,\, \mathcal{P}_\beta \cup \mathcal{P}_\delta\,,\, \mathcal{M}_\beta \cup \mathcal{M}_\delta\,,\, \longrightarrow_\ddagger\,,\, (\imath_\beta\,,\, \imath_\delta))$. Then, by introducing $\textcircled{\texttt{A9}}$, conclude $\big[\textcircled{\texttt{A9}} \textbf{ and } (Q_\alpha \times Q_\gamma\,,\, \mathcal{P}_\alpha \cup \mathcal{P}_\gamma\,,\, \mathcal{M}_\alpha \cup \mathcal{M}_\gamma\,,\, \longrightarrow_\dagger\,,\, (\imath_\alpha\,,\, \imath_\gamma)) \preceq^R (Q_\beta \times Q_\delta\,,\, \mathcal{P}_\beta \cup \mathcal{P}_\delta\,,\, \mathcal{M}_\beta \cup \mathcal{M}_\delta\,,\, \longrightarrow_\ddagger\,,\, (\imath_\beta\,,\, \imath_\delta))\big]$. Then, by introducing $\textcircled{\texttt{A8}}$, conclude $\big[\textcircled{\texttt{A8}} \textbf{ and } \textcircled{\texttt{A9}}$ **and** $(Q_\alpha \times Q_\gamma\,,\, \mathcal{P}_\alpha \cup \mathcal{P}_\gamma\,,\, \mathcal{M}_\alpha \cup \mathcal{M}_\gamma\,,\, \longrightarrow_\dagger\,,\, (\imath_\alpha\,,\, \imath_\gamma)) \preceq^R (Q_\beta \times Q_\delta\,,\, \mathcal{P}_\beta \cup \mathcal{P}_\delta\,,\, \mathcal{M}_\beta \cup \mathcal{M}_\delta\,,\, \longrightarrow_\ddagger\,,\, (\imath_\beta\,,\, \imath_\delta))\big]$. Then, by applying Definition 3 of $\boxdot$, conclude $(Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \boxdot (Q_\gamma\,,\, \mathcal{P}_\gamma\,,\, \mathcal{M}_\gamma\,,\, \longrightarrow_\gamma\,,\, \imath_\gamma) \preceq^R (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \boxdot (Q_\delta\,,\, \mathcal{P}_\delta\,,\, \mathcal{M}_\delta\,,\, \longrightarrow_\delta\,,\, \imath_\delta)$. Then, by applyling $\textcircled{\texttt{A4}}$, conclude $\alpha \boxdot (Q_\gamma\,,\, \mathcal{P}_\gamma\,,\, \mathcal{M}_\gamma\,,\, \longrightarrow_\gamma\,,\, \imath_\gamma) \preceq^R (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \boxdot (Q_\delta\,,\, \mathcal{P}_\delta\,,\, \mathcal{M}_\delta\,,\, \longrightarrow_\delta\,,\, \imath_\delta)$. Then, by applying $\textcircled{\texttt{A6}}$, conclude $\alpha \boxdot \gamma \preceq^R (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \boxdot (Q_\delta\,,\, \mathcal{P}_\delta\,,\, \mathcal{M}_\delta\,,\, \longrightarrow_\delta\,,\, \imath_\delta)$. Then, by applying $\textcircled{\texttt{A5}}$, conclude $\alpha \boxdot \gamma \preceq^R \beta \boxdot (Q_\delta\,,\, \mathcal{P}_\delta\,,\, \mathcal{M}_\delta\,,\, \longrightarrow_\delta\,,\, \imath_\delta)$. Then, by applying $\textcircled{\texttt{A7}}$, conclude $\alpha \boxdot \gamma \preceq^R \beta \boxdot \delta$. ■

APPENDIX C. PROOF OF LEMMA 3

Similar to how we proved Lemma 2 in Section B, we prove Lemma 3 by reusing as much as possible the proofs for port automata in [12]. In this case, the main result that we want to reuse is [12, Theorem 2]. However, because that theorem states not exactly the same property for port automata as Lemma 3 for constraint automata, we need an auxiliary lemma to bridge the gap. To that end, we first define the unary relation denoted by $\xrightarrow{1}$ (informally introduced in Sec. III) and a new binary relation on constraint automata called *slavery* (originally introduced in [12] on port automata). Afterward, we state and prove the auxiliary lemma.

Definition 11 (No-synchronization relation)**.** The no-synchronization relation, denoted by $\xrightarrow{1}$, is the relation on $\mathbb{C}\text{A}$ defined as:

$$\xrightarrow{1} (Q\,,\,\mathcal{P}\,,\,\mathcal{M}\,,\,\longrightarrow\,,\,\imath)\ \ \textbf{iff}\ \ \begin{bmatrix}\big[q \xrightarrow{P,f} q'\ \ \textbf{implies}\ \ |P|=1\big]\\ \textbf{for all}\ \ f\,,\,P\,,\,q\,,\,q'\end{bmatrix}$$

Definition 12 (Slave relation)**.** The slave relation, denoted by $\mapsto$, is the relation on $\mathbb{C}\text{A} \times \mathbb{C}\text{A}$ defined as:

$$(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \mapsto \alpha\ \ \textbf{iff}\ \ \begin{bmatrix}\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ P_\beta \subseteq \mathsf{Port}(\alpha)\Big]\\ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\end{bmatrix}$$

Lemma 6. $\xrightarrow{1}\beta$ **implies** $\beta \mapsto \alpha$

Proof: Assume:

(A1) $\xrightarrow{1}\beta$

(A2) $\beta = (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$

Recall $\xrightarrow{1}\beta$ from (A1). Then, by applying Definition 11 of $\xrightarrow{1}$, conclude $\big[\big[q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta$ **implies** $|P_\beta| = 1\big]$ **for all** $f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\big]$. Then, by basic rewriting, conclude $\big[\big[\big[q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta$ **and** $P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]$ **implies** $\big[|P_\beta| = 1$ **and** $P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]\big]$ **for all** $f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\big]$. Then, by rewriting under ZFC, conclude:

$$\Big[\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ \big[\big[P_\beta = \{p\}\ \ \textbf{for some}\ \ p\big]\ \ \textbf{and}\ \ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]\Big]\ \ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\Big]$$

Then, by basic rewriting, conclude:

$$\Big[\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ \big[\big[P_\beta = \{p\}\ \ \textbf{and}\ \ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]\ \ \textbf{for some}\ \ p\big]\Big]\ \ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\Big]$$

Then, by basic rewriting, conclude:

$$\Big[\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ \big[\big[P_\beta = \{p\}\ \ \textbf{and}\ \ \{p\} \cap \mathsf{Port}(\alpha) \neq \emptyset\big]\ \ \textbf{for some}\ \ p\big]\Big]\ \ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\Big]$$

Then, by rewriting under ZFC, conclude:

$$\Big[\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ \big[\big[P_\beta = \{p\}\ \ \textbf{and}\ \ p \in \mathsf{Port}(\alpha)\big]\ \ \textbf{for some}\ \ p\big]\Big]\ \ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\Big]$$

Then, by rewriting under ZFC, conclude:

$$\Big[\Big[\begin{bmatrix}q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta\ \ \textbf{and}\\ P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\end{bmatrix}\ \ \textbf{implies}\ \ \big[\big[P_\beta = \{p\}\ \ \textbf{and}\ \ \{p\} \subseteq \mathsf{Port}(\alpha)\big]\ \ \textbf{for some}\ \ p\big]\Big]\ \ \textbf{for all}\ \ f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\Big]$$

Then, by rewriting under ZFC, conclude $\big[\big[\big[q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta$ **and** $P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]$ **implies** $\big[P_\beta \subseteq \mathsf{Port}(\alpha)$ **for some** $p\big]\big]$ **for all** $f_\beta\,,\,P_\beta\,,\,q_\beta\,,\,q'_\beta\big]$. Then, by basic rewriting, conclude $\big[\big[\big[q_\beta \xrightarrow{P_\beta,f_\beta} q'_\beta$ **and** $P_\beta \cap \mathsf{Port}(\alpha) \neq \emptyset\big]$ **implies**

$$\frac{q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta , f_\beta}_\beta q'_\beta \textbf{ and } (\mathsf{Port}(\alpha) , P_\alpha) \lozenge (\mathsf{Port}(\beta) , P_\beta)}{(q_\alpha , q_\beta) \xrightarrow{P_\alpha \cup P_\beta , f_\alpha \wedge f_\beta} (q'_\alpha , q'_\beta)} \quad (1)$$

$$\frac{q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta , f_\beta}_\beta q'_\beta \textbf{ and } (\mathsf{Port}(\alpha) , P_\alpha) \blacklozenge (\mathsf{Port}(\beta) , P_\beta)}{(q_\alpha , q_\beta) \xrightarrow{P_\alpha \cup P_\beta , f_\alpha \wedge f_\beta} (q'_\alpha , q'_\beta)} \quad (2)$$

$$\frac{q_\alpha \xrightarrow{P_\alpha , f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \in Q_\beta \textbf{ and } P_\alpha \cap \mathsf{Port}(\beta) = \emptyset}{(q_\alpha , q_\beta) \xrightarrow{P_\alpha , f_\alpha} (q'_\alpha , q_\beta)} \quad (3)$$

$$\frac{q_\beta \xrightarrow{P_\beta , f_\beta}_\beta q'_\beta \textbf{ and } q_\alpha \in Q_\alpha \textbf{ and } P_\beta \cap \mathsf{Port}(\alpha) = \emptyset}{(q_\alpha , q_\beta) \xrightarrow{P_\beta , f_\beta} (q_\alpha , q'_\beta)} \quad (4)$$

Figure 17: Reformulated rules for combining transitions (cf. Figure 5).

$P_\beta \subseteq \mathsf{Port}(\alpha)\big]$ **for all** $f_\beta , P_\beta , q_\beta , q'_\beta\big]$. Then, by applying Definition 12 of $\mapsto$, conclude $(Q_\beta , \mathcal{P}_\beta , \mathcal{M}_\beta , \longrightarrow_\beta , \imath_\beta) \mapsto \alpha$. Then, by applying (A2), conclude $\beta \mapsto \alpha$. ■

Using Lemma 6, we can show that the two conditions in the premise of Lemma 3 satisfy the conditions in the premise of [12, Definition 12]. Then, combined with [12, Proposition 2], we can use [12, Theorem 2] to establish the conclusion of Lemma 3. In principle, this concludes our proof. What remains, however, is showing that we in fact can extend the proof of [12, Theorem 2] from port automata to constraint automata. After carefully analyzing the structure of that proof (including the definitions and other results it relies on), we discovered that the differences between port automata and constraint automata manifest in [12, Lemmas 4, 5, 6, 8, 9]. Consequently, for our proof of Lemma 3 to be valid, we must prove similar results as [12, Lemmas 4, 5, 6, 8, 9] for constraint automata instead of port automata. We do so in the remainder of this section.

First, to simplify notation in the upcoming proofs, we adopt the same symbols $\lozenge$ and $\blacklozenge$ as in [12] to abbreviate the second line of the premise in Rule (1) and Rule (2) in Figure 5. Formally, those symbols denote the following relations.

Definition 13 (Weak agreement relation)**.** The weak agreement relation, denoted by $\lozenge$, is the relation on $\wp(\mathbb{P}\text{ORT})^2 \times \wp(\mathbb{P}\text{ORT})^2$ defined as:

$$(\mathcal{P}_\alpha , P_\alpha) \lozenge (\mathcal{P}_\beta , P_\beta) \textbf{ iff } \big[P_\alpha \subseteq \mathcal{P}_\alpha \textbf{ and } P_\beta \subseteq \mathcal{P}_\beta \textbf{ and } \mathcal{P}_\alpha \cap P_\beta = \mathcal{P}_\beta \cap P_\alpha\big]$$

Definition 14 (Strong agreement relation)**.** The strong agreement relation, denoted by $\blacklozenge$, is the relation on $\wp(\mathbb{P}\text{ORT})^2 \times \wp(\mathbb{P}\text{ORT})^2$ defined as:

$$(\mathcal{P}_\alpha , P_\alpha) \blacklozenge (\mathcal{P}_\beta , P_\beta) \textbf{ iff } \left[P_\alpha \subseteq \mathcal{P}_\alpha \textbf{ and } P_\beta \subseteq \mathcal{P}_\beta \textbf{ and } \begin{bmatrix} P_\alpha = \mathcal{P}_\alpha \cap P_\beta \textbf{ or } P_\beta = \mathcal{P}_\beta \cap P_\alpha \\ \textbf{or } \mathcal{P}_\alpha \cap P_\beta = \emptyset = \mathcal{P}_\beta \cap P_\alpha \end{bmatrix}\right]$$

Using those relations, we can reformulate Rule (1) and Rule (2) more concisely as in Figure 17. In addition to $\lozenge$ and $\blacklozenge$, we also adopt the notion of *subautomata* from [12].

Definition 15 (Subautomaton relation)**.** The subautomaton relation, denoted by $\sqsubseteq$, is the relation on $\mathbb{C}\text{A} \times \mathbb{C}\text{A}$ defined as:

$$(Q , \mathcal{P} , \mathcal{M} , \longrightarrow_\alpha , \imath) \sqsubseteq (Q , \mathcal{P} , \mathcal{M} , \longrightarrow_\beta , \imath) \textbf{ iff } \longrightarrow_\alpha \subseteq \longrightarrow_\beta$$

Now, the following lemma extends [12, Lemma 4] from port automata to constraint automata.

Lemma 7. $\alpha \boxdot \beta \sqsubseteq \alpha \boxtimes \beta$

Proof: Assume:

(A1) $\alpha = (Q_\alpha , \mathcal{P}_\alpha , \mathcal{M}_\alpha , \longrightarrow_\alpha , \imath_\alpha)$

(A2) $\beta = (Q_\beta , \mathcal{P}_\beta , \mathcal{M}_\beta , \longrightarrow_\beta , \imath_\beta)$

(A3) $\longrightarrow_\boxtimes$ denotes the smallest relation induced by the rules Rule (1), Rule (3), and Rule (4) in Figure 17 under α and β.

(A4) $\longrightarrow_{\boxdot}$ denotes the smallest relation induced by the rules Rule (2), Rule (3), and Rule (4) in Figure 17 under α and β.

Reasoning to a generalization, suppose:

$$(q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\,q'_\beta) \text{ **for some** } f\,,\,P\,,\,q_\alpha\,,\,q_\beta\,,\,q'_\alpha\,,\,q'_\beta$$

Then, by applying (A4), $\big[\big[$Rule (2) applies$\big]$ **or** $\big[$Rule (3) applies$\big]$ **or** $\big[$Rule (4) applies$\big]\big]$. Proceed by case distinction.

- **Case:** $\big[$Rule (2) applies$\big]$.
  Then, by applying the definition of Rule (2) in Figure 17, conclude:

$$\left[\begin{array}{c} P = P_\alpha \cup P_\beta \ \textbf{and}\ f = f_\alpha \wedge f_\beta \ \textbf{and}\ q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \\ \textbf{and}\ (\mathsf{Port}(\alpha)\,,\,P_\alpha) \blacklozenge (\mathsf{Port}(\beta)\,,\,P_\beta) \end{array}\right] \ \textbf{for some}\ f_\alpha\,,\,f_\beta\,,\,P_\alpha\,,\,P_\beta$$

  Then, by applying [12, Lemma 3], conclude $\big[P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha$ **and** $q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta$ **and** $(\mathsf{Port}(\alpha)\,,\,P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\,P_\beta)\big]$. Then, by applying the definition of Rule (1) in Figure 17, conclude $\big[$ $P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $\big[$Rule (1) applies$\big]\big]$. Then, by applying (A3), conclude $\big[P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $(q_\alpha\,,\,q_\beta) \xrightarrow{P_\alpha\cup P_\beta,f_\alpha\wedge f_\beta}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)\big]$. Then, by basic rewriting, conclude $(q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)$.
- **Case:** $\big[$Rule (3) applies$\big]$.
  Then, by applying (A3), conclude $(q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)$.
- **Case:** $\big[$Rule (4) applies$\big]$.
  Then, by applying (A3), conclude $(q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)$.

Hence, after considering all cases, conclude $(q_\alpha\,,\,q_\beta) \xrightarrow{P}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)$. Then, by generalizing the premise, conclude:

$$\big[(q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\,q'_\beta) \ \textbf{implies}\ (q_\alpha\,,\,q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\,q'_\beta)\big] \ \textbf{for all}\ f\,,\,P\,,\,q_\alpha\,,\,q_\beta\,,\,q'_\alpha\,,\,q'_\beta$$

Then, by rewriting under ZFC, conclude $\longrightarrow_{\boxdot} \subseteq \longrightarrow_{\boxtimes}$. Then, by introducing (A4), conclude $\big[$(A4) **and** $\longrightarrow_{\boxdot} \subseteq \longrightarrow_{\boxtimes}\big]$. Then, by introducing (A3), conclude $\big[$(A3) **and** (A4) **and** $\longrightarrow_{\boxdot} \subseteq \longrightarrow_{\boxtimes}\big]$. Then, by applying Definition 15 of $\sqsubseteq$, conclude:

$$\text{(A3) } \textbf{and} \text{ (A4) } \textbf{and}\ (Q_\alpha{\times}Q_\beta\,,\,\mathcal{P}_\alpha{\cup}\mathcal{P}_\beta\,,\,\mathcal{M}_\alpha{\cup}\mathcal{M}_\beta\,,\,\longrightarrow_{\boxdot}\,,\,(\imath_\alpha\,,\,\imath_\beta)) \sqsubseteq (Q_\alpha{\times}Q_\beta\,,\,\mathcal{P}_\alpha{\cup}\mathcal{P}_\beta\,,\,\mathcal{M}_\alpha{\cup}\mathcal{M}_\beta\,,\,\longrightarrow_{\boxtimes}\,,\,(\imath_\alpha\,,\,\imath_\beta))$$

Then, by applying Definition 3 of $\boxdot$, conclude $\big[$(A3) **and** $(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \boxdot (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \sqsubseteq (Q_\alpha \times Q_\beta\,,\,\mathcal{P}_\alpha \cup \mathcal{P}_\beta\,,\,\mathcal{M}_\alpha \cup \mathcal{M}_\beta\,,\,\longrightarrow_{\boxtimes}\,,\,(\imath_\alpha\,,\,\imath_\beta))\big]$. Then, by applying Definition 2 of $\boxtimes$, conclude $(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)\boxdot(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \sqsubseteq (Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)\boxtimes(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$. Then, by applying (A1), conclude $\alpha\boxdot(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \sqsubseteq \alpha\boxtimes(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$. Then, by applying (A2), conclude $\alpha\boxdot\beta \sqsubseteq \alpha\boxtimes\beta$.

■

To generalize the next lemma from port automata to constraint automata, we need to introduce yet another notion from [12].

Definition 16 (Conditional strong agreement relation)**.** The conditional strong agreement relation, denoted by $\lozenge\!\!\!\!\blacklozenge$, is the relation on $\mathbb{C}\text{A} \times \mathbb{C}\text{A}$ defined as:

$$(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha) \lozenge\!\!\!\!\blacklozenge (Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta) \ \textbf{iff} \ \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \ \textbf{and} \\ (\mathsf{Port}(\alpha)\,,\,P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\,P_\beta) \end{array}\right] \\ \textbf{implies}\ (\mathsf{Port}(\alpha)\,,\,P_\alpha) \blacklozenge (\mathsf{Port}(\beta)\,,\,P_\beta) \end{array}\right] \\ \textbf{for all}\ f_\alpha\,,\,f_\beta\,,\,P_\alpha\,,\,P_\beta\,,\,q_\alpha\,,\,q_\beta\,,\,q'_\alpha\,,\,q'_\beta \end{array}\right]$$

Now, the following lemma extends [12, Lemma 5] from port automata to constraint automata.

Lemma 8. $\alpha \lozenge\!\!\!\!\blacklozenge \beta$ **implies** $\alpha \boxtimes \beta \sqsubseteq \alpha \boxdot \beta$

Proof: Assume:

(A1) $\alpha \lozenge\!\!\!\!\blacklozenge \beta$

(A2) $\alpha = (Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)$

$\textcircled{\texttt{A3}}$ $\beta = (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$

$\textcircled{\texttt{A4}}$ $\longrightarrow_{\boxtimes}$ denotes the smallest relation induced by the rules Rule (1), Rule (3), and Rule (4) in Figure 17 under α and β.

$\textcircled{\texttt{A5}}$ $\longrightarrow_{\boxdot}$ denotes the smallest relation induced by the rules Rule (2), Rule (3), and Rule (4) in Figure 17 under α and β.

Reasoning to a generalization, suppose:

$$(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\, q'_\beta) \ \textbf{for some}\ f\,,\, P\,,\, q_\alpha\,,\, q_\beta\,,\, q'_\alpha\,,\, q'_\beta$$

Then, by applying $\textcircled{\texttt{A4}}$, $\big[\big[\text{Rule (1) applies}\big]$ **or** $\big[\text{Rule (3) applies}\big]$ **or** $\big[\text{Rule (4) applies}\big]\big]$. Proceed by case distinction.

- **Case:** $\big[\text{Rule (1) applies}\big]$.
  Then, by applying the definition of Rule (1) in Figure 17, conclude:

$$\left[\begin{array}{c} P = P_\alpha \cup P_\beta \ \textbf{and}\ f = f_\alpha \wedge f_\beta \ \textbf{and}\ q_\alpha \xrightarrow{P_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta}_\beta q'_\beta \\ \textbf{and}\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right] \ \textbf{for some}\ f_\alpha\,,\, f_\beta\,,\, P_\alpha\,,\, P_\beta$$

  Then, by introducing $\textcircled{\texttt{A1}}$, conclude:

$$\alpha \lozenge\!\!\blacklozenge \beta \ \textbf{and}\ \left[\begin{array}{c} P = P_\alpha \cup P_\beta \ \textbf{and}\ f = f_\alpha \wedge f_\beta \ \textbf{and}\ q_\alpha \xrightarrow{P_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta}_\beta q'_\beta \\ \textbf{and}\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right]$$

  Then, by applying Definition 16 of $\lozenge\!\!\blacklozenge$, conclude:

$$\left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \ \textbf{and} \\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right] \\ \textbf{implies}\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \blacklozenge (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right] \\ \textbf{for all}\ q_\alpha\,,\, q_\beta\,,\, q'_\alpha\,,\, q'_\beta\,,\, P_\alpha\,,\, P_\beta \end{array}\right] \ \textbf{and}$$

$$\big[P = P_\alpha \cup P_\beta \ \textbf{and}\ f = f_\alpha \wedge f_\beta \ \textbf{and}\ q_\alpha \xrightarrow{P_\alpha}_\alpha q'_\alpha \ \textbf{and}\ q_\beta \xrightarrow{P_\beta}_\beta q'_\beta \ \textbf{and}\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \lozenge (\mathsf{Port}(\beta)\,,\, P_\beta)\big]$$

  Then, by rewriting under ZFC, conclude $\big[P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $q_\alpha \xrightarrow{P_\alpha}_\alpha q'_\alpha$ **and** $q_\beta \xrightarrow{P_\beta}_\beta q'_\beta$ **and** $(\mathsf{Port}(\alpha)\,,\, P_\alpha) \blacklozenge (\mathsf{Port}(\beta)\,,\, P_\beta)\big]$. Then, by applying the definition of Rule (2) in Figure 17, conclude $\big[P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $\big[\text{Rule (2) applies}\big]\big]$. Then, by applying $\textcircled{\texttt{A5}}$, conclude $\big[P = P_\alpha \cup P_\beta$ **and** $f = f_\alpha \wedge f_\beta$ **and** $(q_\alpha\,,\, q_\beta) \xrightarrow{P_\alpha \cup P_\beta, f_\alpha \wedge f_\beta}_{\boxdot} (q'_\alpha\,,\, q'_\beta)\big]$. Then, by basic rewriting, conclude $(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\, q'_\beta)$.

- **Case:** $\big[\text{Rule (3) applies}\big]$.
  Then, by applying $\textcircled{\texttt{A5}}$, conclude $(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\, q'_\beta)$.

- **Case:** $\big[\text{Rule (4) applies}\big]$.
  Then, by applying $\textcircled{\texttt{A5}}$, conclude $(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\, q'_\beta)$.

Hence, after considering all cases, conclude $(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\, q'_\beta)$. Then, by generalizing the premise, conclude:

$$\big[(q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxtimes} (q'_\alpha\,,\, q'_\beta) \ \textbf{implies}\ (q_\alpha\,,\, q_\beta) \xrightarrow{P,f}_{\boxdot} (q'_\alpha\,,\, q'_\beta)\big] \ \textbf{for all}\ f\,,\, P\,,\, q_\alpha\,,\, q_\beta\,,\, q'_\alpha\,,\, q'_\beta$$

Then, by rewriting under ZFC, conclude $\longrightarrow_{\boxtimes} \subseteq \longrightarrow_{\boxdot}$. Then, by introducing $\textcircled{\texttt{A5}}$, conclude $\big[\textcircled{\texttt{A5}}$ **and** $\longrightarrow_{\boxtimes} \subseteq \longrightarrow_{\boxdot}\big]$. Then, by introducing $\textcircled{\texttt{A4}}$, conclude $\big[\textcircled{\texttt{A4}}$ **and** $\textcircled{\texttt{A5}}$ **and** $\longrightarrow_{\boxtimes} \subseteq \longrightarrow_{\boxdot}\big]$. Then, by applying Definition 15 of $\sqsubseteq$, conclude:

$$\textcircled{\texttt{A4}} \ \textbf{and}\ \textcircled{\texttt{A5}} \ \textbf{and}\ (Q_\alpha{\times}Q_\beta\,,\, \mathcal{P}_\alpha{\cup}\mathcal{P}_\beta\,,\, \mathcal{M}_\alpha{\cup}\mathcal{M}_\beta\,,\, \longrightarrow_{\boxtimes}\,,\, (\imath_\alpha\,,\, \imath_\beta)) \sqsubseteq (Q_\alpha{\times}Q_\beta\,,\, \mathcal{P}_\alpha{\cup}\mathcal{P}_\beta\,,\, \mathcal{M}_\alpha{\cup}\mathcal{M}_\beta\,,\, \longrightarrow_{\boxdot}\,,\, (\imath_\alpha\,,\, \imath_\beta))$$

Then, by applying Definition 2 of $\boxtimes$, conclude $\big[\textcircled{\texttt{A5}}$ **and** $(Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \boxtimes (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \sqsubseteq (Q_\alpha \times Q_\beta\,,\, \mathcal{P}_\alpha \cup \mathcal{P}_\beta\,,\, \mathcal{M}_\alpha \cup \mathcal{M}_\beta\,,\, \longrightarrow_{\boxdot}\,,\, (\imath_\alpha\,,\, \imath_\beta))\big]$. Then, by applying Definition 3 of $\boxdot$, conclude $(Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \boxtimes (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \sqsubseteq (Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \boxdot (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$. Then, by applying $\textcircled{\texttt{A2}}$

, conclude $\alpha \boxtimes (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta) \sqsubseteq \alpha \boxdot (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$. Then, by applying $\textcircled{\scriptsize A3}$, conclude $\alpha \boxtimes \beta \sqsubseteq \alpha \boxdot \beta$. ■

To generalize the next lemma from port automata to constraint automata, we first define the binary relation denoted by $\asymp$ (informally introduced in Section III).

Definition 17 (Independence relation)**.** The independence relation, denoted by $\asymp$, is the relation on $\mathbb{C}\text{A} \times \mathbb{C}\text{A}$ defined as:

$$\alpha \asymp \beta \;\; \textbf{iff} \;\; \mathsf{Port}(\alpha) \cap \mathsf{Port}(\beta) = \emptyset$$

Now, the following lemma extends [12, Lemma 6] from port automata to constraint automata.

Lemma 9. $\alpha \asymp \beta$ **implies** $\alpha \mathrel{\langle\!\blacklozenge} \beta$

Proof: Assume:

$\textcircled{\scriptsize A1}$ $\alpha \asymp \beta$.

$\textcircled{\scriptsize A2}$ $\alpha = (Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha)$

$\textcircled{\scriptsize A3}$ $\beta = (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$

Reasoning to a generalization, suppose:

$$(\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta) \;\; \textbf{for some} \;\; P_\alpha\,,\, P_\beta$$

Then, by introducing $\textcircled{\scriptsize A1}$, conclude $\big[\alpha \asymp \beta \;\textbf{and}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)\big]$. Then, by applying Definition 17 of $\asymp$, conclude $\big[\mathsf{Port}(\alpha) \cap \mathsf{Port}(\beta) = \emptyset \;\textbf{and}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)\big]$. Then, by applying Definition 13 of $\lozenge$, conclude $\big[\mathsf{Port}(\alpha) \cap \mathsf{Port}(\beta) = \emptyset \;\textbf{and}\; P_\alpha \subseteq \mathsf{Port}(\alpha) \;\textbf{and}\; P_\beta \subseteq \mathsf{Port}(\beta) \;\textbf{and}\; \mathsf{Port}(\alpha) \cap P_\beta = \mathsf{Port}(\beta) \cap P_\alpha\big]$. Then, by rewriting under ZFC, conclude $\big[\mathsf{Port}(\alpha) \cap P_\beta = \emptyset \;\textbf{and}\; P_\alpha \subseteq \mathsf{Port}(\alpha) \;\textbf{and}\; P_\beta \subseteq \mathsf{Port}(\beta) \;\textbf{and}\; \mathsf{Port}(\alpha) \cap P_\beta = \mathsf{Port}(\beta) \cap P_\alpha\big]$. Then, by rewriting under ZFC, conclude $\big[P_\alpha \subseteq \mathsf{Port}(\alpha) \;\textbf{and}\; P_\beta \subseteq \mathsf{Port}(\beta) \;\textbf{and}\; \mathsf{Port}(\alpha) \cap P_\beta = \emptyset = \mathsf{Port}(\beta) \cap P_\alpha\big]$. Then, by applying Definition 14 of $\blacklozenge$, conclude $(\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\blacklozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)$. Then, by generalizing the premise, conclude:

$$\big[(\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta) \;\textbf{implies}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\blacklozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)\big] \;\; \textbf{for all} \;\; P_\alpha\,,\, P_\beta$$

Then, by basic rewriting, conclude:

$$\left[\begin{array}{l} \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \;\textbf{and}\; q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \;\textbf{and} \\ (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right] \\ \textbf{implies}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\blacklozenge} (\mathsf{Port}(\beta)\,,\, P_\beta) \end{array}\right] \;\; \textbf{for all} \;\; q_\alpha\,,\, q_\beta\,,\, q'_\alpha\,,\, q'_\beta\,,\, P_\alpha\,,\, P_\beta$$

Then, by applying Definition 16 of $\langle\!\blacklozenge$, conclude $(Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \mathrel{\langle\!\blacklozenge} (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$. Then, by applying $\textcircled{\scriptsize A2}$, conclude $\alpha \mathrel{\langle\!\blacklozenge} (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$. Then, by applying $\textcircled{\scriptsize A3}$, conclude $\alpha \mathrel{\langle\!\blacklozenge} \beta$. ■

The following lemma extends [12, Lemma 8] from port automata to constraint automata.

Lemma 10. $\beta \mapsto \alpha$ **implies** $\beta \mathrel{\langle\!\blacklozenge} \alpha$

Proof: Assume:

$\textcircled{\scriptsize A1}$ $\alpha \mapsto \beta$

$\textcircled{\scriptsize A2}$ $\alpha = (Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha)$

$\textcircled{\scriptsize A3}$ $\beta = (Q_\beta\,,\, \mathcal{P}_\beta\,,\, \mathcal{M}_\beta\,,\, \longrightarrow_\beta\,,\, \imath_\beta)$

Reasoning to a generalization, suppose:

$$\Big[q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \;\textbf{and}\; q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \;\textbf{and}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)\Big] \;\; \textbf{for some} \;\; f_\alpha\,,\, f_\beta\,,\, P_\alpha\,,\, P_\beta\,,\, q_\alpha\,,\, q_\beta\,,\, q'_\alpha\,,\, q'_\beta$$

Then, by introducing $\textcircled{\scriptsize A1}$, conclude $\Big[\alpha \mapsto \beta \;\textbf{and}\; \Big[q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \;\textbf{and}\; q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \;\textbf{and}\; (\mathsf{Port}(\alpha)\,,\, P_\alpha) \mathrel{\lozenge} (\mathsf{Port}(\beta)\,,\, P_\beta)\Big]\Big]$. Then, by applying $\textcircled{\scriptsize A2}$, conclude $\Big[(Q_\alpha\,,\, \mathcal{P}_\alpha\,,\, \mathcal{M}_\alpha\,,\, \longrightarrow_\alpha\,,\, \imath_\alpha) \mapsto \beta \;\textbf{and}\; \Big[q_\alpha \xrightarrow{P_\alpha, f_\alpha}_\alpha q'_\alpha \;\textbf{and}\; q_\beta \xrightarrow{P_\beta, f_\beta}_\beta q'_\beta \;\textbf{and}$

$(\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)]\big]$. Then, applying Definition 12 of $\mapsto$, conclude:

$$\Big[\left[\begin{array}{l}\left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha, f_\alpha} q'_\alpha \textbf{ and}\\ P_\alpha\cap\mathsf{Port}(\beta)\neq\emptyset\end{array}\right]\\ \textbf{implies } P_\alpha\subseteq\mathsf{Port}(\beta)\end{array}\right] \textbf{ for all } f_\alpha\,,\,P_\alpha\,,\,q_\alpha\,,\,q'_\alpha\Big] \textbf{ and } \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]$$

Then, by basic rewriting, conclude:

$$\left[\begin{array}{l}\left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha, f_\alpha} q'_\alpha \textbf{ and}\\ P_\alpha\cap\mathsf{Port}(\beta)\neq\emptyset\end{array}\right]\\ \textbf{implies } P_\alpha\subseteq\mathsf{Port}(\beta)\end{array}\right] \textbf{ and } \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]$$

Then, by basic rewriting, conclude:

$$\left[\begin{array}{c} (\textbf{not } q_\alpha \xrightarrow{P_\alpha, f_\alpha} q'_\alpha)\\ \textbf{or } P_\alpha\cap\mathsf{Port}(\beta)=\emptyset\\ \textbf{or } P_\alpha\subseteq\mathsf{Port}(\beta)\end{array}\right] \textbf{ and } \left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]$$

Then, by basic rewriting, conclude:

$$\begin{array}{r} \left[\begin{array}{c} (\textbf{not } q_\alpha \xrightarrow{P_\alpha, f_\alpha} q'_\alpha) \textbf{ and } q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta\\ \textbf{and } (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]\\ \textbf{or } \left[\begin{array}{c} P_\alpha\cap\mathsf{Port}(\beta)=\emptyset \textbf{ and } q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta\\ \textbf{and } (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]\\ \textbf{or } \left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\beta) \textbf{ and } q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta\\ \textbf{and } (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]\end{array}$$

Then, by basic rewriting, conclude:

$$\textbf{false or } \left[\begin{array}{c} P_\alpha\cap\mathsf{Port}(\beta)=\emptyset \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right] \textbf{ or } \left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\beta) \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]$$

Then, by basic rewriting, conclude:

$$\left[\begin{array}{c} P_\alpha\cap\mathsf{Port}(\beta)=\emptyset \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right] \textbf{ or } \left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\beta) \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]$$

Then, by applying Definition 13 of $\Diamond$, conclude:

$$\left[\begin{array}{c} P_\alpha\cap\mathsf{Port}(\beta)=\emptyset \textbf{ and}\\ P_\alpha\subseteq\mathsf{Port}(\alpha) \textbf{ and } P_\beta\subseteq\mathsf{Port}(\beta)\\ \textbf{and } \mathsf{Port}(\alpha)\cap P_\beta=\mathsf{Port}(\beta)\cap P_\alpha\end{array}\right] \textbf{ or } \left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\beta) \textbf{ and}\\ P_\alpha\subseteq\mathsf{Port}(\alpha) \textbf{ and } P_\beta\subseteq\mathsf{Port}(\beta)\\ \textbf{and } \mathsf{Port}(\alpha)\cap P_\beta=\mathsf{Port}(\beta)\cap P_\alpha\end{array}\right]$$

Then, by rewriting under ZFC, conclude:

$$\left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\alpha) \textbf{ and } P_\beta\subseteq\mathsf{Port}(\beta) \textbf{ and}\\ \mathsf{Port}(\alpha)\cap P_\beta=\emptyset=\mathsf{Port}(\beta)\cap P_\alpha\end{array}\right] \textbf{ or } \left[\begin{array}{c} P_\alpha\subseteq\mathsf{Port}(\alpha) \textbf{ and } P_\beta\subseteq\mathsf{Port}(\beta)\\ \textbf{and } \mathsf{Port}(\alpha)\cap P_\beta=\mathsf{Port}(\beta)\cap P_\alpha\end{array}\right]$$

Then, by applying Definition 14 of $\blacklozenge$, conclude $\big[(\mathsf{Port}(\alpha)\,,\,P_\alpha)\blacklozenge(\mathsf{Port}(\beta)\,,\,P_\beta)$ **or** $(\mathsf{Port}(\alpha)\,,\,P_\alpha)\blacklozenge(\mathsf{Port}(\beta)\,,\,P_\beta)\big]$. Then, by basic rewriting, conclude $(\mathsf{Port}(\alpha)\,,\,P_\alpha)\blacklozenge(\mathsf{Port}(\beta)\,,\,P_\beta)$. Then, by generalizing the premise, conclude:

$$\left[\begin{array}{l}\left[\begin{array}{c} q_\alpha \xrightarrow{P_\alpha,f_\alpha}_\alpha q'_\alpha \textbf{ and } q_\beta \xrightarrow{P_\beta,f_\beta}_\beta q'_\beta \textbf{ and}\\ (\mathsf{Port}(\alpha)\,,\,P_\alpha)\Diamond(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right]\\ \textbf{implies } (\mathsf{Port}(\alpha)\,,\,P_\alpha)\blacklozenge(\mathsf{Port}(\beta)\,,\,P_\beta)\end{array}\right] \textbf{ for all } f_\alpha\,,\,f_\beta\,,\,P_\alpha\,,\,P_\beta\,,\,q_\alpha\,,\,q_\beta\,,\,q'_\alpha\,,\,q'_\beta$$

Then, by applying Definition 16 of $\langle\!\blacklozenge\!\rangle$, conclude $(Q_\alpha\,,\,\mathcal{P}_\alpha\,,\,\mathcal{M}_\alpha\,,\,\longrightarrow_\alpha\,,\,\imath_\alpha)\langle\!\blacklozenge\!\rangle(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$. Then, by applying $\textcircled{\texttt{A2}}$, conclude $\alpha\,\langle\!\blacklozenge\!\rangle\,(Q_\beta\,,\,\mathcal{P}_\beta\,,\,\mathcal{M}_\beta\,,\,\longrightarrow_\beta\,,\,\imath_\beta)$. Then, by applying $\textcircled{\texttt{A3}}$, conclude $\alpha\,\langle\!\blacklozenge\!\rangle\,\beta$. ■

Finally, the following lemma extends [12, Lemma 9] from port automata to constraint automata. Note that, after this lemma, we extended each of [12, Lemma 4, 5, 6, 8, 9] from port automata to constraint automata, which was exactly our remaining proof obligation for establishing Lemma 3.

Lemma 11. $\beta \mapsto \alpha$ **implies** $\beta \mapsto \alpha \boxdot \gamma$

Proof: Assume:

(A1) $\alpha \mapsto \beta$

(A2) $\alpha = (Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha)$

Recall $\alpha \mapsto \beta$ from (A1). Then, by applying (A2), conclude $(Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha) \mapsto \beta$. Then, applying Definition 12 of $\mapsto$, conclude $\big[\big[\big[q_\alpha \xrightarrow{P_\alpha , f_\alpha} q'_\alpha$ **and** $P_\alpha \cap \mathsf{Port}(\beta) \neq \emptyset\big]$ **implies** $P_\alpha \subseteq \mathsf{Port}(\beta)\big]$ **for all** $f_\alpha\,, P_\alpha\,, q_\alpha\,, q'_\alpha\big]$. Then, by introducing [12, Proposition 6],[13] conclude:

$$\mathsf{Port}(\beta \boxdot \gamma) = \mathsf{Port}(\beta) \cup \mathsf{Port}(\gamma) \ \ \textbf{and} \ \ \left[\begin{array}{c}\big[\big[q_\alpha \xrightarrow{P_\alpha , f_\alpha} q'_\alpha \ \textbf{and} \ P_\alpha \cap \mathsf{Port}(\beta) \neq \emptyset\big] \ \textbf{implies} \ P_\alpha \subseteq \mathsf{Port}(\beta)\big] \\ \textbf{for all} \ f_\alpha\,, P_\alpha\,, q_\alpha\,, q'_\alpha\end{array}\right]$$

Then, by rewriting under ZFC, conclude:

$$\mathsf{Port}(\beta) \subseteq \mathsf{Port}(\beta \boxdot \gamma) \ \ \textbf{and} \ \ \left[\begin{array}{c}\big[\big[q_\alpha \xrightarrow{P_\alpha , f_\alpha} q'_\alpha \ \textbf{and} \ P_\alpha \cap \mathsf{Port}(\beta) \neq \emptyset\big] \ \textbf{implies} \ P_\alpha \subseteq \mathsf{Port}(\beta)\big] \\ \textbf{for all} \ f_\alpha\,, P_\alpha\,, q_\alpha\,, q'_\alpha\end{array}\right]$$

Then, by rewriting under ZFC, conclude $\big[\big[\big[q_\alpha \xrightarrow{P_\alpha , f_\alpha} q'_\alpha$ **and** $P_\alpha \cap \mathsf{Port}(\beta \boxdot \gamma) \neq \emptyset\big]$ **implies** $P_\alpha \subseteq \mathsf{Port}(\beta)\big]$ **for all** $f_\alpha\,, P_\alpha\,, q_\alpha\,, q'_\alpha\big]$. Then, by applying Definition 12 of $\mapsto$, conclude $(Q_\alpha\,, \mathcal{P}_\alpha\,, \mathcal{M}_\alpha\,, \longrightarrow_\alpha\,, \imath_\alpha) \mapsto \beta \boxdot \gamma$. Then, by applying (A2), conclude $\alpha \mapsto \beta \boxdot \gamma$. ■

APPENDIX D. PROOF OF LEMMA 4

Assume:

(A1) $\textsc{Partition}(X) = (\mathcal{B}\,, \mathcal{C})$

(A2) $X = \{\alpha_1\,, \ldots\,, \alpha_k\}$

1) To show that $(\mathcal{B}\,, \mathcal{C})$ is a partition of X, we must show that $(\mathcal{B}_k\,, \mathcal{C}_k)$ is a partition of X. To show that, we must show $\big[\bigcup_{B \in \mathcal{B}_k} B \cup \bigcup_{B \in \mathcal{C}_k} C = X$ **and** $\big[\big[\big[A \neq A'$ **and** $A\,, A' \in \mathcal{B} \cup \mathcal{C}\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,, A'\big]\big]$. We do so by induction on $0 \leq i \leq k$.
   - **Base:** $i = 0$.
     Observe:

     (Z1) Conclude $\emptyset = \emptyset$. Then, by rewriting under ZFC, conclude $\emptyset \cap \emptyset = \emptyset$. Then, by applying the definition of $\textsc{Partition}$, conclude $\mathcal{B}_0 \cap \mathcal{C}_0 = \emptyset$. Then, by applying $\boxed{\textbf{Base}}$, conclude $\mathcal{B}_i \cap \mathcal{C}_i = \emptyset$.

     (Z2) Conclude $\emptyset = \{\alpha_1\,, \ldots\,, \alpha_0\}$. Then, by rewriting under ZFC, conclude $\bigcup_{B \in \emptyset} B \cup \bigcup_{C \in \emptyset} C = \{\alpha_1\,, \ldots\,, \alpha_0\}$. Then, by applying the definition of $\textsc{Partition}$, conclude $\bigcup_{B \in \mathcal{B}_0} B \cup \bigcup_{C \in \mathcal{C}_0} C = \{\alpha_1\,, \ldots\,, \alpha_0\}$. Then, by applying $\boxed{\textbf{Base}}$, conclude $\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$.

     (Z3) Conclude $\emptyset \cap \{\alpha_1\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \emptyset} B \cup \bigcup_{C \in \emptyset} C) \cap \{\alpha_1\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by applying the definition of $\textsc{Partition}$, conclude $(\bigcup_{B \in \mathcal{B}_0} B \cup \bigcup_{C \in \mathcal{C}_0} C) \cap \{\alpha_1\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by applying $\boxed{\textbf{Base}}$, conclude $(\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$.

     (Z4) Conclude $\big[\big[\big[A \neq A'$ **and** $A\,, A' \in \emptyset\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,, A'\big]$. Then, by rewriting under ZFC, conclude $\big[\big[\big[A \neq A'$ **and** $A\,, A' \in \emptyset\big] \cup \emptyset$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,, A'\big]$. Then, by applying the definition of $\textsc{Partition}$, conclude $\big[\big[\big[A \neq A'$ **and** $A\,, A' \in \mathcal{B}_0 \cup \mathcal{C}_0\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,, A'\big]$. Then, by applying $\boxed{\textbf{Base}}$, conclude $\big[\big[\big[A \neq A'$ **and** $A\,, A' \in \mathcal{B}_i \cup \mathcal{C}_i\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,, A'\big]$.

     Conclude the base case by **and**-ing (Z1), (Z2), (Z3), (Z4).

[13]That proposition appears only in the full version [31] of [12].

- **IH:**

$$\begin{gathered}\mathcal{B}_{i-1} \cap \mathcal{C}_{i-1} = \emptyset \ \textbf{and} \ \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1 , \ldots , \alpha_{i-1}\} \\ \textbf{and} \ (\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} = \emptyset \ \textbf{and} \\ \big[\big[[A \neq A' \ \textbf{and} \ A , A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big]\end{gathered}$$

- **Step:** $0 < i \leq k$.
Conclude $\big[\xrightarrow{1} \alpha_i$ **or** $[$**not** $\xrightarrow{1} \alpha_i]\big]$. Proceed by case distinction.

  - **Case:** $\xrightarrow{1} \alpha_i$.
  Observe:

    $\textcircled{Y1}$ Recall $\xrightarrow{1} \alpha_i$ by $\boxed{\textbf{Case}}$. Then, by applying the definition of PARTITION, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **and** $\mathcal{C}_i = \mathcal{C}_{i-1}\big]$.

    $\textcircled{Y2}$ Reasoning to **false**, suppose $\{\{\alpha_i\}\} \in \mathcal{C}_{i-1}$. Then, by rewriting under ZFC, conclude $\alpha_i \in \bigcup_{C \in \mathcal{C}_{i-1}} C$. Then, by rewriting under ZFC, conclude $\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$. Then, by rewriting under ZFC, conclude $\big[\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$ **and** $\alpha_i \in \{\alpha_i , \ldots , \alpha_k\}\big]$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} \neq \emptyset$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} = \emptyset$ **and** $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} \neq \emptyset\big]$. Then, by basic rewriting, conclude **false**. Then, by negating the premise, conclude $\{\{\alpha_i\}\} \notin \mathcal{C}_{i-1}$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{B}_{i-1} \cap \mathcal{C}_{i-1} = \emptyset$ **and** $\{\{\alpha_i\}\} \notin \mathcal{C}_{i-1}\big]$. Then, by rewriting under ZFC, conclude $(\mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}) \cap \mathcal{C}_{i-1} = \emptyset$. Then, by applying $\textcircled{Y1}$, conclude $\mathcal{B}_i \cap \mathcal{C}_i = \emptyset$.

    $\textcircled{Y3}$ Recall $\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1 , \ldots , \alpha_{i-1}\}$ from $\boxed{\textbf{IH}}$. Then, by rewriting under ZFC, conclude $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_i\} \cup \{\alpha_1 , \ldots , \alpha_{i-1}\}$. Then, by rewriting under ZFC, conclude $\bigcup_{B \in \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1 , \ldots , \alpha_i\}$. Then, by applying $\textcircled{Y1}$, conclude $\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C = \{\alpha_1 , \ldots , \alpha_i\}$.

    $\textcircled{Y4}$ Recall $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} = \emptyset$ from $\boxed{\textbf{IH}}$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap (\{\alpha_i\} \cup \{\alpha_{i+1} , \ldots , \alpha_k\}) = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C \cup \{\alpha_i\}) \cap \{\alpha_{i+1} , \ldots , \alpha_k\} = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_{i+1} , \ldots , \alpha_k\} = \emptyset$. Then, by applying $\textcircled{Y1}$, conclude $(\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C) \cap \{\alpha_{i+1} , \ldots , \alpha_k\} = \emptyset$.

    $\textcircled{Y5}$ Recall $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i , \ldots , \alpha_k\} = \emptyset$ from $\boxed{\textbf{IH}}$. Then, by rewriting under ZFC, conclude $\alpha_i \notin \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$. Then, by rewriting under ZFC, conclude $\big[[A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **implies** $\alpha_i \notin A']$ **for all** $A'\big]$. Then, by rewriting under ZFC, conclude $\big[[A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **implies** $\{\alpha_i\} \cap A' = \emptyset]$ **for all** $A'\big]$. Then, by rewriting under ZFC, conclude $\big[\big[[A \in \{\{\alpha_i\}\}$ **and** $A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A , A'\big]$. Then, by rewriting under ZFC, conclude:

    $$\begin{gathered}\big[\big[[A \in \{\{\alpha_i\}\} \ \textbf{and} \ A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big] \\ \textbf{and} \ \big[\big[[A \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1} \ \textbf{and} \ A' \in \{\{\alpha_i\}\}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big]\end{gathered}$$

    Then, by introducing $\boxed{\textbf{IH}}$, conclude:

    $$\begin{gathered}\big[\big[[A \neq A' \ \textbf{and} \ A , A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big] \\ \textbf{and} \ \big[\big[[A \in \{\{\alpha_i\}\} \ \textbf{and} \ A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big] \\ \textbf{and} \ \big[\big[[A \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1} \ \textbf{and} \ A' \in \{\{\alpha_i\}\}] \ \textbf{implies} \ A \cap A' = \emptyset\big] \ \textbf{for all} \ A , A'\big]\end{gathered}$$

    Then, by rewriting under ZFC, conclude $\big[\big[[A \neq A'$ **and** $A , A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1} \cup \{\{\alpha_i\}\}]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A , A'\big]$. Then, by applying $\textcircled{Y1}$, conclude $\big[\big[[A \neq A'$ **and** $A , A' \in \mathcal{B}_i \cup \mathcal{C}_i]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A , A'\big]$.

  Conclude the inductive step by **and**-ing $\textcircled{Y2}$, $\textcircled{Y3}$, $\textcircled{Y4}$, $\textcircled{Y5}$.

  - **Case:** $[$**not** $\xrightarrow{1} \alpha_i]$.
  Observe:

$\textcircled{\texttt{X1}}$ Recall $\big[\textbf{not} \xrightarrow{1} \alpha_i\big]$ by $\boxed{\textbf{Case}}$. Then, by applying the definition of PARTITION, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1}$ **and** $\mathcal{C}_i = (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}$ **and** $\overline{\mathcal{C}}_i = \{C \in \mathcal{C}_{i-1} \mid \gamma \in C$ **and** $\alpha_i \not\succ \gamma\}\big]$.

$\textcircled{\texttt{X2}}$ Recall $\overline{\mathcal{C}}_i = \{C \in \mathcal{C}_{i-1} \mid \gamma \in C$ **and** $\alpha_i \not\succ \gamma\}$ from $\textcircled{\texttt{X1}}$. Then, by rewriting under ZFC, conclude $\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$.

$\textcircled{\texttt{X3}}$ Reasoning to **false**, suppose $\{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\} \in \mathcal{B}_{i-1}$. Then, by rewriting under ZFC, conclude $\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B$. Then, by rewriting under ZFC, conclude $\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$. Then, by rewriting under ZFC, conclude $\big[\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$ **and** $\alpha_i \in \{\alpha_i\,, \ldots\,, \alpha_k\}\big]$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,, \ldots\,, \alpha_k\} \neq \emptyset$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,, \ldots\,, \alpha_k\} = \emptyset$ **and** $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,, \ldots\,, \alpha_k\} \neq \emptyset\big]$. Then, by basic rewriting, conclude **false**. Then, by negating the premise, conclude $\{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\} \notin \mathcal{B}_{i-1}$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{B}_{i-1} \cap \mathcal{C}_{i-1} = \emptyset$ **and** $\{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\} \notin \mathcal{B}_{i-1}\big]$. Then, by rewriting under ZFC, conclude $(\mathcal{B}_{i-1} \cap (\mathcal{C}_{i-1} \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}) = \emptyset$. Then, by applying $\textcircled{\texttt{X1}}$, conclude $\mathcal{B}_i \cap \mathcal{C}_i = \emptyset$.

$\textcircled{\texttt{X4}}$ Recall $\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1\,, \ldots\,, \alpha_{i-1}\}$ from $\boxed{\textbf{IH}}$. Then, by rewriting under ZFC, conclude $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_i\} \cup \{\alpha_1\,, \ldots\,, \alpha_{i-1}\}$. Then, by rewriting under ZFC, conclude $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$. Then, by introducing $\textcircled{\texttt{X2}}$, conclude $\big[\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$ **and** $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C = \{\alpha_1\,, \ldots\,, \alpha_i\}\big]$. Then, by rewriting under ZFC, conclude $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \overline{\mathcal{C}}_i} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$. Then, by rewriting under ZFC, conclude $\{\alpha_i\} \cup \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i} C \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$. Then, by rewriting under ZFC, conclude $\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$. Then, by applying $\textcircled{\texttt{X1}}$, conclude $\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C = \{\alpha_1\,, \ldots\,, \alpha_i\}$.

$\textcircled{\texttt{X5}}$ Recall $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,, \ldots\,, \alpha_k\} = \emptyset$ from $\boxed{\textbf{IH}}$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap (\{\alpha_i\} \cup \{\alpha_{i+1}\,, \ldots\,, \alpha_k\}) = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C \cup \{\alpha_i\}) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by introducing $\textcircled{\texttt{X2}}$, conclude $\big[\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$ **and** $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C \cup \{\alpha_i\}) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset\big]$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \overline{\mathcal{C}}_i} C \cup \{\alpha_i\}) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i} C \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C \cup \{\alpha_i\}) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}} C) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$. Then, by applying $\textcircled{\texttt{Y1}}$, conclude $(\bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_i} C) \cap \{\alpha_{i+1}\,, \ldots\,, \alpha_k\} = \emptyset$.

$\textcircled{\texttt{X6}}$ Reasoning to **false**, suppose:

$$\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \;\textbf{ and }\; A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \;\textbf{ and }\; A \cap A' \neq \emptyset\big] \;\textbf{ for some }\; A\,, A'$$

Then, by rewriting under ZFC, conclude:

$$\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \;\textbf{ and }\; A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \;\textbf{ and }\; \alpha \in A \;\textbf{ and }\; \alpha \in A'\big] \;\textbf{ for some }\; \alpha$$

Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i} C$ **and** $\alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\big]$. Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C)$ **and** $\alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\big]$. Then, by basic rewriting, conclude:

$$\big[\alpha \in \textstyle\bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C) \;\textbf{ and }\; \alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\big] \;\textbf{ and }\; \begin{bmatrix} \alpha = \alpha_i \\ \textbf{or }\; \alpha \neq \alpha_i \end{bmatrix}$$

Then, by basic rewriting, conclude:

$$\begin{array}{c}\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C) \;\textbf{ and }\; \alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C \;\textbf{ and }\; \alpha = \alpha_i\big] \\ \textbf{or }\; \big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C) \;\textbf{ and }\; \alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C \;\textbf{ and }\; \alpha \neq \alpha_i\big]\end{array}$$

Proceed by case distinction.

- **Case:** $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C)$ **and** $\alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C$ **and** $\alpha = \alpha_i\big]$.
  Then, by basic rewriting, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C)$ **and** $\alpha = \alpha_i\big]$. Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$ **and** $\alpha = \alpha_i\big]$. Then, by basic rewriting,

conclude $\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C$. Then, by rewriting under ZFC, conclude $\big[\alpha_i \in \{\alpha_i\,,\,\ldots\,,\,\alpha_k\}$ **and** $\alpha_i \in \bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C\big]$. Then, by rewriting under ZFC, conclude $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,,\,\ldots\,,\,\alpha_k\} \neq \emptyset$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,,\,\ldots\,,\,\alpha_k\} = \emptyset$ **and** $(\bigcup_{B \in \mathcal{B}_{i-1}} B \cup \bigcup_{C \in \mathcal{C}_{i-1}} C) \cap \{\alpha_i\,,\,\ldots\,,\,\alpha_k\} \neq \emptyset\big]$. Then, by basic rewriting, conclude **false**.

· **Case:** $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C)$ **and** $\alpha \in \{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C$ **and** $\alpha \neq \alpha_i\big]$.
Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B \cup (\bigcup_{C \in \mathcal{C}_{i-1}} C \setminus \bigcup_{C \in \overline{\mathcal{C}}_i} C)$ **and** $\alpha \in \bigcup_{C \in \overline{\mathcal{C}}_i} C\big]$. Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B$ **and** $\alpha \in \bigcup_{C \in \overline{\mathcal{C}}_i} C\big]$. Then, by introducing $\textcircled{\scriptsize X2}$, conclude $\big[\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$ **and** $\alpha \in \bigcup_{B \in \mathcal{B}_i} B$ **and** $\alpha \in \bigcup_{C \in \overline{\mathcal{C}}_i} C\big]$. Then, by rewriting under ZFC, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_i} B$ **and** $\alpha \in \bigcup_{C \in \mathcal{C}_{i-1}} C\big]$. Then, by applying $\textcircled{\scriptsize X1}$, conclude $\big[\alpha \in \bigcup_{B \in \mathcal{B}_{i-1}} B$ **and** $\alpha \in \bigcup_{C \in \mathcal{C}_{i-1}} C\big]$. Then, by rewriting under ZFC, conclude:

$$\big[B \in \mathcal{B}_{i-1} \textbf{ and } C \in \mathcal{C}_{i-1} \textbf{ and } \alpha \in B \textbf{ and } \alpha \in C'\big] \textbf{ for some } B\,,\,C$$

Then, by rewriting under ZFC, conclude $\big[B \in \mathcal{B}_{i-1}$ **and** $C \in \mathcal{C}_{i-1}$ **and** $B \cap C \neq \emptyset\big]$. Then, by rewriting under ZFC, conclude $\big[B \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **and** $C \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **and** $B \cap C \neq \emptyset\big]$. Then, by rewriting under ZFC, conclude $\big[B\,,\,C \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **and** $B \cap C \neq \emptyset\big]$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{B}_{i-1} \cap \mathcal{C}_{i-1} = \emptyset$ **and** $B\,,\,C \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **and** $B \cap C \neq \emptyset\big]$. Then, by rewriting under ZFC, conclude $\big[B \neq C$ **and** $B\,,\,C \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}$ **and** $B \cap C \neq \emptyset\big]$. Then, by introducing $\boxed{\textbf{IH}}$, conclude:

$$\begin{gathered}\big[\big[\big[A \neq A' \textbf{ and } A\,,\,A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[B \neq C \textbf{ and } B\,,\,C \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1} \textbf{ and } B \cap C \neq \emptyset\big]\end{gathered}$$

Then, by basic rewriting, conclude **false**.

Hence, after considering all cases, conclude **false**. Then, by negating the premise, conclude:

$$\big[\big[\textbf{not } A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)\big] \textbf{ or } \big[\textbf{not } A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \textbf{ or } \big[\textbf{not } A \cap A' \neq \emptyset\big]\big] \textbf{ for all } A\,,\,A'$$

Then, by basic rewriting, conclude $\big[\big[\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)$ **and** $A' \in \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,,\,A'\big]$. Then, by rewriting under ZFC, conclude:

$$\begin{gathered}\big[\big[\big[A \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \textbf{ and } A' \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[\big[\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \textbf{ and } A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big]\end{gathered}$$

Then, by introducing $\boxed{\textbf{IH}}$, conclude:

$$\begin{gathered}\big[\big[\big[A \neq A' \textbf{ and } A\,,\,A' \in \mathcal{B}_{i-1} \cup \mathcal{C}_{i-1}\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[\big[\big[A \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \textbf{ and } A' \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[\big[\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \textbf{ and } A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big]\end{gathered}$$

Then, by rewriting under ZFC, conclude:

$$\begin{gathered}\big[\big[\big[A \neq A' \textbf{ and } A\,,\,A' \in \mathcal{B}_{i-1} \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[\big[\big[A \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \textbf{ and } A' \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i)\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big] \\ \textbf{and } \big[\big[\big[A \in \mathcal{B}_i \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \textbf{ and } A' \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \textbf{ implies } A \cap A' = \emptyset\big] \textbf{ for all } A\,,\,A'\big]\end{gathered}$$

Then, by rewriting under ZFC, conclude $\big[\big[\big[A \neq A'$ **and** $A\,,\,A' \in \mathcal{B}_{i-1} \cup (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,,\,A'\big]$. Then, by applying $\textcircled{\scriptsize X1}$, conclude $\big[\big[\big[A \neq A'$ **and** $A\,,\,A' \in \mathcal{B}_i \cup \mathcal{C}_i\big]$ **implies** $A \cap A' = \emptyset\big]$ **for all** $A\,,\,A'\big]$.

Conclude the inductive step by **and**-ing $\textcircled{\scriptsize X3}$, $\textcircled{\scriptsize X4}$, $\textcircled{\scriptsize X5}$, $\textcircled{\scriptsize X6}$.

$\square$

2) To show $\big[\xrightarrow{1} [\![B]\!]$ **for all** $B \in \mathcal{B}\big]$, we must show $\big[\big[B \in \mathcal{B}_k$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$. We do so by induction on $0 \leq i \leq k$.

- **Base:** $i = 0$.
Conclude $\big[\big[$**false implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$. Then, by rewriting under ZFC, conclude $\big[\big[B \in \emptyset$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$. Then, by applying the definition of PARTITION, conclude $\big[\big[B \in \mathcal{B}_0$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$. Then, by applying $\boxed{\textbf{Base}}$, conclude $\big[\big[B \in \mathcal{B}_i$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$.

- **IH:** $\big[B \in \mathcal{B}_{i-1}$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** B
- **Step:** $0 < i \leq k$.
  Conclude $\big[\xrightarrow{1} \alpha_i$ **or** $\big[$**not** $\xrightarrow{1} \alpha_i\big]\big]$. Proceed by case distinction.
  - **Case:** $\xrightarrow{1} \alpha_i$.
    Then, by applying the definition of PARTITION, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **and** $\xrightarrow{1} \alpha_i\big]$. Then, by the definition of $[\![\_]\!]$ in Section III, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **and** $\xrightarrow{1} [\![\{\alpha_i\}]\!]\big]$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **and** $\big[\big[\big[B \in \mathcal{B}_{i-1}$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$ **and** $\xrightarrow{1} [\![\{\alpha_i\}]\!]\big]\big]$. Then, by rewriting under ZFC, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **and** $\big[\big[B \in \mathcal{B}_{i-1} \cup \{\{\alpha_i\}\}$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]\big]$. Then, by basic rewriting, conclude $\big[\big[B \in \mathcal{B}_i$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$.
  - **Case:** $\big[$**not** $\xrightarrow{1} \alpha_i\big]$.
    Then, by applying the definition of PARTITION, conclude $\mathcal{B}_i = \mathcal{B}_{i-1}$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{B}_i = \mathcal{B}_{i-1}$ **and** $\big[\big[B \in \mathcal{B}_{i-1}$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]\big]$. Then, by basic rewriting, conclude $\big[\big[B \in \mathcal{B}_i$ **implies** $\xrightarrow{1} [\![B]\!]\big]$ **for all** $B\big]$.

□

3) We first state and prove an auxiliary lemma.

Lemma 12. $[\![\{\alpha_1 , \dots , \alpha_k\}]\!] \not\asymp \beta$ **iff** $\big[\big[\alpha \not\asymp \beta$ **and** $\alpha \in \{\alpha_1 , \dots , \alpha_k\}\big]$ **for some** $\alpha\big]$

Proof:

$\Rightarrow$ Suppose $[\![\{\alpha_1 , \dots , \alpha_k\}]\!] \not\asymp \beta$. Then, by applying the definition of $[\![\_]\!]$ in Section III, conclude $\alpha_1 \boxtimes \cdots \boxtimes \alpha_k \not\asymp \beta$. Then, by applying Definition 17 of $\asymp$, conclude $\mathsf{Port}(\alpha_1 \boxtimes \cdots \boxtimes \alpha_k) \cap \mathsf{Port}(\beta) \neq \emptyset$. Then, by applying Definition 2 of $\boxtimes$, conclude $(\mathsf{Port}(\alpha_1) \cup \cdots \cup \mathsf{Port}(\alpha_k)) \cap \mathsf{Port}(\beta) \neq \emptyset$. Then, by rewriting under ZFC, conclude $\big[\big[\mathsf{Port}(\alpha) \cap \mathsf{Port}(\beta) \neq \emptyset$ **and** $\alpha \in \{\alpha_1 , \dots , \alpha_k\}\big]$ **for some** $\alpha\big]$. Then, by applying Definition 17 of $\asymp$, conclude $\big[\alpha \not\asymp \beta$ **and** $\alpha \in \{\alpha_1 , \dots , \alpha_k\}$ **for some** $\alpha\big]$.

$\Leftarrow$ Suppose $\big[\big[\alpha \not\asymp \beta$ **and** $\alpha \in \{\alpha_1 , \dots , \alpha_k\}\big]$ **for some** $\alpha\big]$. Then, by applying Definition 17 of $\asymp$, conclude $\big[\big[\mathsf{Port}(\alpha) \cap \mathsf{Port}(\beta) \neq \emptyset$ **and** $\alpha \in \{\alpha_1 , \dots , \alpha_k\}\big]$ **for some** $\alpha\big]$. Then, by rewriting under ZFC, conclude $(\mathsf{Port}(\alpha_1) \cup \cdots \cup \mathsf{Port}(\alpha_k)) \cap \mathsf{Port}(\beta) \neq \emptyset$. Then, by applying Definition 2 of $\boxtimes$, conclude $\mathsf{Port}(\alpha_1 \boxtimes \cdots \boxtimes \alpha_k) \cap \mathsf{Port}(\beta) \neq \emptyset$. Then, by applying Definition 17 of $\asymp$, conclude $\alpha_1 \boxtimes \cdots \boxtimes \alpha_k \not\asymp \beta$. Then, by applying the definition of $[\![\_]\!]$ in Section III, conclude $[\![\{\alpha_1 , \dots , \alpha_k\}]\!] \not\asymp \beta$.

■

To show $\big[\big[C_1 \neq C_2$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2 \in \mathcal{C}\big]$, we must show $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_k\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$. We do so by induction on $0 \leq i \leq k$.

- **Base:** $i = 0$.
  Conclude $\big[\big[$**false implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$. Then, by basic rewriting, conclude $\big[\big[\big[C_1 \neq C_2$ **and false**$\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$. Then, by rewriting under ZFC, conclude $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \emptyset\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$. Then, by applying the definition of PARTITION, conclude $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_0\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$. Then, by applying $\boxed{\textbf{Base}}$, conclude $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_i\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$.
- **IH:** $\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_{i-1}\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** C_1 , C_2
- **Step:** $0 < i \leq k$.
  Conclude $\big[\xrightarrow{1} \alpha_i$ **or** $\big[$**not** $\xrightarrow{1} \alpha_i\big]\big]$. Proceed by case distinction.
  - **Case:** $\xrightarrow{1} \alpha_i$.
    Then, by applying the definition of PARTITION, conclude $\mathcal{C}_i = \mathcal{C}_{i-1}$. Then, by introducing $\boxed{\textbf{IH}}$, conclude $\big[\mathcal{C}_i = \mathcal{C}_{i-1}$ **and** $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_{i-1}\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]\big]$. Then, by basic rewriting, conclude $\big[\big[\big[C_1 \neq C_2$ **and** $C_1 , C_2 \in \mathcal{C}_i\big]$ **implies** $[\![C_1]\!] \asymp [\![C_2]\!]\big]$ **for all** $C_1 , C_2\big]$.
  - **Case:** $\big[$**not** $\xrightarrow{1} \alpha_i\big]$.
    Observe:

    (W1) Recall $\big[$**not** $\xrightarrow{1} \alpha_i\big]$ by $\boxed{\textbf{Case}}$. Then, by applying the definition of PARTITION, conclude $\big[\mathcal{C}_i = (\mathcal{C}_{i-1} \setminus$

$\overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C\in\overline{\mathcal{C}}_i} C\}$ **and** $\overline{\mathcal{C}}_i = \{C \in \mathcal{C}_{i-1} \mid \gamma \in C$ **and** $\alpha_i \not\asymp \gamma\}\big]$.

$\textcircled{\texttt{W2}}$ Recall $\overline{\mathcal{C}}_i = \{C \in \mathcal{C}_{i-1} \mid \gamma \in C$ **and** $\alpha_i \not\asymp \gamma\}$ from $\textcircled{\texttt{W1}}$. Then, by rewriting under ZFC, conclude $\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$.

Reasoning to **false**, suppose:

$$\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{and}\ \llbracket C_1\rrbracket \not\asymp \llbracket\{\alpha_i\} \cup \textstyle\bigcup_{C\in\overline{\mathcal{C}}_i} C\rrbracket\big]\ \textbf{for some}\ C_1$$

Then, by applying Lemma 12, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\big[\big[\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in \{\alpha_i\} \cup \bigcup_{C\in\overline{\mathcal{C}}_i} C\big]$ **for some** $\alpha\big]\big]$. Then, by basic rewriting, conclude:

$$\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{and}\ \llbracket C_1\rrbracket \not\asymp \alpha \ \textbf{and}\ \alpha \in \{\alpha_i\} \cup \textstyle\bigcup_{C\in\overline{\mathcal{C}}_i} C\big]\ \textbf{for some}\ \alpha$$

Then, by rewriting under ZFC, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\big[\alpha \in \{\alpha_i\}$ **or** $\alpha \in \bigcup_{C\in\overline{\mathcal{C}}_i} C\big]\big]$. Then, by basic rewriting, conclude $\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in \{\alpha_i\}\big]$ **or** $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in \bigcup_{C\in\overline{\mathcal{C}}_i} C\big]\big]$. Proceed by case distinction.

- **Case:** $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in \{\alpha_i\}\big]$.
  Then, by rewriting under ZFC, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha = \alpha_i\big]$. Then, by basic rewriting, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha_i\big]$. Then, by applying Lemma 12, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\big[\big[\gamma \not\asymp \alpha_i$ **and** $\gamma \in C_1\big]$ **for some** $\gamma\big]\big]$. Then, by basic rewriting, conclude:

  $$\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{and}\ \gamma \not\asymp \alpha_i \ \textbf{and}\ \gamma \in C_1\big]\ \textbf{for some}\ \gamma$$

  Then, by rewriting under ZFC, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $C_1 \in \{C \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \mid \gamma \in C$ **and** $\gamma \not\asymp \alpha_i\}\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $C_1 \in \{C \in \mathcal{C}_{i-1} \mid \gamma \in C$ **and** $\gamma \not\asymp \alpha_i\}\big]$. Then, by applying $\textcircled{\texttt{W1}}$, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $C_1 \in \overline{\mathcal{C}}_i\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \notin \overline{\mathcal{C}}_i$ **and** $C_1 \in \overline{\mathcal{C}}_i\big]$. Then, by basic rewriting, conclude **false**.

- **Case:** $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in \bigcup_{C\in\overline{\mathcal{C}}_i} C\big]$.
  Then, by rewriting under ZFC, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\big[\big[\alpha \in C_2$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$ **for some** $C_2\big]\big]$. Then, by basic rewriting, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\big[\big[\llbracket C_1\rrbracket \not\asymp \alpha$ **and** $\alpha \in C_2\big]$ **for some** $C_2\big]$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by applying Lemma 12, conclude $\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \notin \overline{\mathcal{C}}_i$ **and** $C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \neq C_2$ **and** $C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \neq C_2$ **and** $C_1 \in \mathcal{C}_{i-1}$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by introducing $\textcircled{\texttt{W2}}$, conclude $\big[\overline{\mathcal{C}}_i \subseteq \mathcal{C}_{i-1}$ **and** $C_1 \neq C_2$ **and** $C_1 \in \mathcal{C}_{i-1}$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \overline{\mathcal{C}}_i\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \neq C_2$ **and** $C_1 \in \mathcal{C}_{i-1}$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket$ **and** $C_2 \in \mathcal{C}_{i-1}\big]$. Then, by rewriting under ZFC, conclude $\big[C_1 \neq C_2$ **and** $C_1, C_2 \in \mathcal{C}_{i-1}$ **and** $\llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket\big]$. Then, by introducing $\boxed{\textbf{IH}}$, conclude:

  $$\begin{gathered}\big[\big[\big[C_1 \neq C_2 \ \textbf{and}\ C_1, C_2 \in \mathcal{C}_{i-1}\big]\ \textbf{implies}\ \llbracket C_1\rrbracket \asymp \llbracket C_2\rrbracket\big]\ \textbf{for all}\ C_1, C_2\big] \\ \textbf{and}\ \big[C_1 \neq C_2 \ \textbf{and}\ C_1, C_2 \in \mathcal{C}_{i-1} \ \textbf{and}\ \llbracket C_1\rrbracket \not\asymp \llbracket C_2\rrbracket\big]\end{gathered}$$

  Then, by basic rewriting, conclude **false**.

Hence, after considering all cases, conclude **false**. Then, by negating the premise, conclude:

$$\big[\big[\textbf{not}\ C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i\big]\ \textbf{or}\ \big[\textbf{not}\ \llbracket C_1\rrbracket \not\asymp \llbracket\{\alpha_i\} \cup \textstyle\bigcup_{C\in\overline{\mathcal{C}}_i} C\rrbracket\big]\big]\ \textbf{for all}\ C_1$$

Then, by basic rewriting, conclude $\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **implies** $\llbracket C_1\rrbracket \asymp \llbracket\{\alpha_i\} \cup \bigcup_{C\in\overline{\mathcal{C}}_i} C\rrbracket\big]$ **for all** $C_1\big]$. Then, by rewriting under ZFC, conclude $\big[\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i$ **and** $C_2 \in \{\{\alpha_i\} \cup \bigcup_{C\in\overline{\mathcal{C}}_i} C\}\big]$ **implies** $\llbracket C_1\rrbracket \asymp \llbracket C_2\rrbracket\big]$ **for all** $C_1, C_2\big]$. Then, by rewriting under ZFC, conclude:

$$\begin{gathered}\big[\big[\big[C_1 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C\in\overline{\mathcal{C}}_i} C\} \ \textbf{and}\ C_2 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i\big]\ \textbf{implies}\ \llbracket C_1\rrbracket \asymp \llbracket C_2\rrbracket\big]\ \textbf{for all}\ C_1, C_2\big] \\ \textbf{and}\ \big[\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{and}\ C_2 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C\in\overline{\mathcal{C}}_i} C\}\big]\ \textbf{implies}\ \llbracket C_1\rrbracket \asymp \llbracket C_2\rrbracket\big]\ \textbf{for all}\ C_1, C_2\big]\end{gathered}$$

Then, by introducing $\boxed{\textbf{IH}}$, conclude:

$$
\begin{aligned}
&\Big[\big[\big[C_1 \neq C_2 \ \textbf{ and } \ C_1\,,\,C_2 \in \mathcal{C}_{i-1}\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big] \\
\textbf{and} \ &\Big[\big[\big[C_1 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \ \textbf{ and } \ C_2 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big] \\
\textbf{and} \ &\Big[\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{ and } \ C_2 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big]
\end{aligned}
$$

Then, by rewriting under ZFC, conclude:

$$
\begin{aligned}
&\Big[\big[\big[C_1 \neq C_2 \ \textbf{ and } \ C_1\,,\,C_2 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big] \\
\textbf{and} \ &\Big[\big[\big[C_1 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\} \ \textbf{ and } \ C_2 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big] \\
\textbf{and} \ &\Big[\big[\big[C_1 \in \mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i \ \textbf{ and } \ C_2 \in \{\{\alpha_i\} \cup \textstyle\bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big]
\end{aligned}
$$

Then, by rewriting under ZFC, conclude $\Big[\big[\big[C_1 \neq C_2 \ \textbf{ and } \ C_1\,,\,C_2 \in (\mathcal{C}_{i-1} \setminus \overline{\mathcal{C}}_i) \cup \{\{\alpha_i\} \cup \bigcup_{C \in \overline{\mathcal{C}}_i} C\}\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big]$. Then, by applying ⓦ1, conclude $\Big[\big[\big[C_1 \neq C_2 \ \textbf{ and } \ C_1\,,\,C_2 \in \mathcal{C}_i\big] \ \textbf{ implies } \ [\![C_1]\!] \asymp [\![C_2]\!]\big] \ \textbf{ for all } \ C_1\,,\,C_2\Big]$

□