A Procedure for Splitting Processes and its Application to Coordination

Sung-Shik T.Q. Jongmans

Centrum Wiskunde & Informatica
Amsterdam, the Netherlands

jongmans@cwi.nl

Dave Clarke

IBBT-DistriNet
Department of Computer Science
Katholieke Universiteit Leuven
Leuven, Belgium

dave.clarke@cs.kuleuven.be

José Proença

IBBT-DistriNet
Department of Computer Science
Katholieke Universiteit Leuven
Leuven, Belgium

jose.proenca@cs.kuleuven.be

We present a procedure for splitting processes in a process algebra with multi-actions (a subset of the specification language mCRL2). This splitting procedure cuts a process into two processes along a set of actions A: roughly, one of these processes contains no actions from A, while the other process contains only actions from A. We state and prove a theorem asserting that the parallel composition of these two processes equals the original process under appropriate synchronization.

We apply our splitting procedure to the process algebraic semantics of the coordination language Reo: using this procedure and its related theorem, we formally establish the soundness of splitting Reo connectors along the boundaries of their (a)synchronous regions in implementations of Reo. Such splitting can significantly improve the performance of connectors.

1 Introduction

Over the past decades, coordination languages have emerged for the specification and implementation of interaction protocols among entities running concurrently (components, services, threads, etc.). This class of languages includes Reo [1], a graphical language for compositional construction of *connectors*: communication mediums through which entities can interact with each other. Figure 1 shows some example connectors in their usual graphical syntax. Intuitively, connectors consist of one or more *channels*, through which data items flow, and two or more *nodes*, on which channel ends coincide. Through channel composition—the act of gluing channels together on nodes—engineers can construct complex connectors. Channels often used include the reliable synchronous channel, called sync, and the reliable asynchronous channel fifo$n$, which has a buffer of capacity n. Importantly, while nodes have a fixed semantics, Reo features an open-ended set of channels. This allows engineers to define their own channels with custom semantics.

To use connectors in real applications, one must derive executable code from graphical specifications of connectors (e.g., those in Figure 1). Roughly two implementation approaches exist. In the *distributed approach*, one implements the behavior of each of the k constituents of a connector and runs these k implementations concurrently as a distributed system; in the *centralized approach*, one computes the behavior of a connector as a whole, implements this behavior, and runs this implementation sequentially as a centralized system. Neither of these two approaches unconditionally predominates the other: among other factors of influence, the hardware architecture on which to deploy the application plays an important role. For example, in the case of a service-oriented application, the distributed approach seems natural, because the services involved run on different machines and the network between them

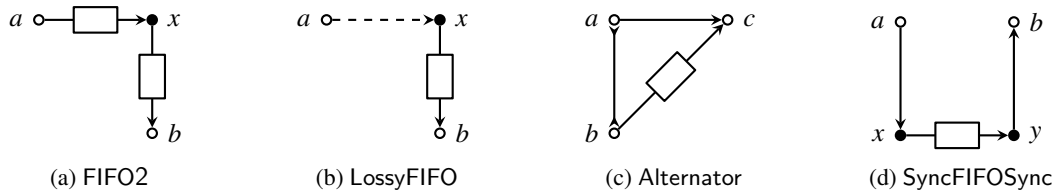| (a) FIFO2 | (b) LossyFIFO | (c) Alternator | (d) SyncFIFOSync |

Figure 1: Some example connectors.

may play a role in their coordination. However, if coordination involves threads running on the same machine in some multi-threading application, the centralized approach appears more appropriate, as it avoids communication among the constituents of a connector at runtime: in this scenario, one dedicates one thread to running the connector.

One optimization technique applicable to both of these approaches involves the identification of the *synchronous* and the *asynchronous regions* of a connector. A synchronous region contains exactly those nodes and channels of a connector that synchronize collectively to decide on their individual behavior; an asynchronous region connects synchronous regions in an asynchronous way. For instance, the connector consisting of a sync channel, a fifo1 channel, and another sync channel (see Figure 1d) has two synchronous regions, connected by an asynchronous region. Intuitively, two synchronous regions can run completely indepedently of each other;[1] an asynchronous regions connecting them takes care of transporting data from one synchronous region to the other. In the distributed approach, this means that nodes and channels need to share information only with those nodes and channels in the same synchronous region—not with every node or channel in the connector. In the centralized approach, this means that one does not need to compute the behavior of a connector as a whole, but rather on a per-region basis.

Recent work shows that the optimization based on identifying regions can significantly improve performance [6, 18, 19]. However, while intuitively valid, a formal argument establishing the soundness of this optimization does not exist yet. In this paper, we present such a proof, based on the process algebraic semantics of Reo [16, 13, 14, 15]. In this semantics, one associates every connector with a process term describing its behavior. More concretely, we identify the following contributions:

- We introduce a splitting procedure for a subset of the specification language mCRL2 [8, 9]—the basis of the existing process algebraic semantics of Reo—and prove its soundness.

- We formalize the notion of (a)synchronous regions in the process algebraic semantics of Reo.

- We apply this splitting procedure to the process algebraic semantics of Reo, thereby justifying the (a)synchronous regions optimization for Reo implementations. In particular, we discuss how we can implement and use the splitting procedure in the distributed approach, exploiting the local concurrency available on the computational nodes.

- We lay the foundations for the definition and analysis of new splitting operations for Reo.

This paper is organized as follows. In Section 2, we give an overview of the fragment of mCRL2 that we use. In Section 3, we summarize the process algebraic semantics of Reo. In Section 4, we introduce our splitting procedure, and in Section 5, we apply this procedure to connectors. We conclude this paper with future work in Section 6. See [11] for a version of this paper with an appendix including full proofs for all the intermediary lemmas.

---

[1]To see this, suppose that two synchronous regions cannot run completely independently of each other. In that case, there exist at least one constituent of the one region that synchronizes with at least one constituent of the other region. But then, these two constituents belong to the same synchronous region—a contradiction.

$$
\begin{array}{ll}
a & ::= \text{any element from } \mathbb{A}\text{ct} \\
a^{\tau} & ::= a \mid \tau \\
\alpha, \beta & ::= a^{\tau} \mid \alpha \sqcup \beta
\end{array}
$$

(a) Syntax of multi-actions.

$$
\begin{array}{ll}
\alpha^{\delta} & ::= \alpha \mid \delta \\
p & ::= \alpha^{\delta} \mid P \mid p + q \mid p \cdot q \\
& \mid p \| q \mid p \Vert q \mid p \mid q \\
& \mid \nabla_V(p) \mid \partial_B(p) \mid \rho_R(p) \mid \Gamma_C(p) \mid \mathscr{T}_I(p)
\end{array}
$$

(b) Syntax of processes.

Figure 2: Syntax.

2 A Process Algebra with Multi-Actions

The process algebra considered in this work comprises the data-free and untimed fragment of mCRL2, a specification language based on ACP [4] and the basis of the existing process algebraic semantics of Reo. Among other useful constructs, mCRL2 has one feature that makes it particularly well-suited as a semantic formalism for Reo, namely *multi-actions*: collections of actions that occur at the same time. We postpone an explanation of how to use multi-actions for describing the behavior of connectors until Section 3. In this section, we summarize (our subset of) mCRL2.

Figure 2a shows the syntax of multi-actions. Let $\mathbb{A}\text{ct}$ denote the set of actions, ranged over by the symbols a, b, c, etc. The distinguished symbol τ denotes the empty multi-action, i.e., the multi-action consisting of no observable actions. Let the symbols a^{τ}, b^{τ}, c^{τ}, etc., range over the elements in the set $\mathbb{A}\text{ct} \cup \{\tau\}$. The operator $\sqcup$ (commutative and associative) combines multi-actions to form larger multi-actions; let $\mathbb{M}\mathbb{A}\text{ct}$ denote the set of all multi-actions, ranged over by α, β, γ, etc. Processes, ranged over by p, q, r, etc., combine multi-actions using the operators shown in Figure 2b.

Basic operators The distinguished symbol—or nullary operator—δ denotes the deadlock process, i.e., the process performing no multi-actions. Let the symbols α^{δ}, β^{δ}, γ^{δ}, etc., range over the processes in the set $\mathbb{M}\mathbb{A}\text{ct} \cup \{\delta\}$. The operators $+$ and $\cdot$ combine processes alternatively and sequentially in the usual way.[2] Let $\mathbb{S}\text{eq}$ denote the set of sequential processes, which consist only of basic operators and multi-actions. Finally, let P, Q, R, etc., denote references that refer to process definitions of the form $P \mapsto p$, $Q \mapsto q$, $R \mapsto r$, etc. For technical convenience, we currently disallow mutual recursion: if $P \mapsto p$, then only P can occur as a reference in p.

Parallel operators The operator $\|$ interleaves and synchronizes processes. The operator $\Vert$ serves as an auxiliary operator in the axiomatization of $\|$: it makes the process on its left-hand side perform a multi-action, and afterwards, it combines the remaining process with the process on its right-hand side the same way $\|$ does. The operator $\mid$ synchronizes processes on the first multi-actions they perform, and it combines the remaining processes the same way $\|$ does.

Additional operators Four additional operators constrain the behavior of processes composed in parallel. The operator ∇ restricts a process p to the multi-actions in a set of nonempty multi-actions $V \subseteq \mathbb{M}\mathbb{A}\text{ct} \setminus \{\tau\}$ (modulo commutativity and associativity of $\sqcup$). The operator ∂ blocks those actions in a process p that occur also in a set of actions $B \subseteq \mathbb{A}\text{ct}$. The operator ρ renames the actions in a process p according to a set of renaming rules $R \subseteq \mathbb{A}\text{ct} \times \mathbb{A}\text{ct}$. Finally, the operator Γ applies the communications in a set $C \subseteq \mathbb{M}\mathbb{A}\text{ct} \times \mathbb{A}\text{ct}$ to a process p. We write communication rules as $\alpha \to a$ and require that τ does not occur in α.

---

[2]We skip the basic operators for conditional composition and summation, because they have no meaning in the data-free fragment of mCRL2 considered. Similarly, we skip those operators that have no meaning in the untimed fragment of mCRL2.
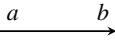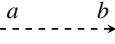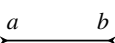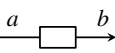
Graphical syntax	Textual syntax	Semantics
$\xrightarrow{\ a \qquad\qquad b\ }$	$\mathsf{sync}\langle a;b\rangle$	Atomically accepts an item on its source end a and dispenses it on its sink end b.
$a \dashrightarrow b$	$\mathsf{lossysync}\langle a;b\rangle$	Atomically accepts an item on its source end a and, non-deterministically, either dispenses it on its sink end b or loses it.
$a \quad\quad b$ (syncdrain)	$\mathsf{syncdrain}\langle a,b;\rangle$	Atomically accepts (and loses) items on both of its source ends a and b.
$a \;\square\; b$ (fifo1)	$\mathsf{fifo1}\langle a;b\rangle$	Atomically accepts an item on its source end and stores it in its buffer, and atomically dispenses the item d on its sink end and clears its buffer.

Figure 3: Syntax and semantics of common channels.

Abstraction operator The operator $\mathscr{T}$ hides those actions in a process p that occur also in a set of actions $I \subseteq \mathbb{A}\mathrm{ct}$. The act of hiding an action a, which means "replacing a by τ," differs from the act of blocking a, which means "replacing a by δ."

We adopt the following usual operator precedence (in decreasing order): $\sqcup, |, \cdot, \|, \|\!|, +$. We write as few parentheses as possible, omitting them also in the case of associative or commutative operators. For example, we write $a \cdot b \cdot c + d + e$ instead of $(a \cdot (b \cdot c)) + (d + e)$.

See Section A for an axiomatization of the operators discussed above.

3   Reo and its Process Algebraic Semantics

Before we continue with our splitting procedure in Section 4, we briefly discuss Reo and its process algebraic semantics [16, 13, 14, 15]; this helps in relating the abstract discussion in Section 4 to a concrete case. Recall from Section 1 that connectors consist of channels and nodes. Below, we outline how these channels and nodes behave and how to describe such behavior as procesess.

Channels. Every channel has exactly two ends, each of which has one of two types: *source ends* accept data, while *sink ends* dispense data. Besides this assumption on their number of ends, Reo makes no assumptions about channels. This means, for example, that Reo allows channels with two source ends. Figure 3 shows the graphical syntax of four common channels, a textual syntax, and an informal description of their behavior. In the process algebraic semantics of Reo, one associates every channel end with an action. For source ends, such an action represents the acceptance of data; for sink ends, it represents the dispersal of data. By combining these actions in multi-actions, one can describe channels that atomically accept and dispense data on their ends. For example, the following *recursive* process definitions describe the behavior of the channels in Figure 3.

$$Sync\langle a;b\rangle \qquad\qquad \mapsto a \sqcup b \cdot Sync\langle a;b\rangle \qquad\qquad SyncDrain\langle a,b;\rangle \mapsto a \sqcup b \cdot SyncDrain\langle a,b;\rangle$$
$$LossySync\langle a;b\rangle \mapsto (a \sqcup b + a) \cdot LossySync\langle a;b\rangle \qquad Fifo1\langle a;b\rangle \qquad \mapsto a \cdot b \cdot Fifo1\langle a;b\rangle$$

The definition $Sync\langle a;b\rangle$ models synchronous flow through channel ends a and b, represented by the multi-action $a \sqcup b$. The definition $LossySync\langle a;b\rangle$ models a (nondeterministic) choice between flow through ends a and b and flow through only a, represented by the multi-action $a \sqcup b + a$. The definition $Fifo1\langle a;b\rangle$ models flow through a followed by flow through b. The recursion found in each of the four process definitions above indicates that the channels modeled by them repeat their behavior indefinitely.

In this paper, we adopt the context-*in*sensitive process algebraic semantics of Reo, originally based on *constraint automata* [2]. In context-insensitive semantic formalisms, one cannot directly describe

channels and connectors whose behavior depends not only on their internal state but also on the presence or absence of I/O operations—their *context*. In contrast, one can describe such channels and connectors in semantic formalisms that do support context-sensitivity. For instance, a context-sensitive version of lossysync should lose a data item only in the absence of I/O operations on its sink end. A context-sensitive process algebraic semantics of Reo exists, originally based on *connector coloring* with three colors [5]. However, because this semantics depends on the data component of mCRL2, we do not consider it in this paper. We remark that we could *encode* a context-sensitive process algebraic semantics along the lines of [12], which makes our splitting procedure applicable also to context-sensitive channels and connectors. For simplicity, however, we do not pursue that in this paper. See [10] for an extensive overview of context-insensitive and context-sensitive semantic formalisms for Reo.

Nodes Entities communicating through a connector perform I/O operations—writes and takes—on its nodes. Reo features three kinds of nodes: *source nodes* on which only source ends coincide, *sink nodes* on which only sink ends coincide, and *mixed nodes* on which both kinds of channel end coincide. Nodes have the following semantics.

- A source node n has *replicator semantics*. Once an entity attempts to write a data item d on n, this node first suspends this operation. Subsequently, n notifies the channels whose source ends coincide on n that it offers d. Once each of these channels has notified n that it accepts d, n resolves the write: atomically, n dispenses d to each of its coincident source ends.

- A sink node n has *nondeterministic merger semantics*. Once an entity attempts to take a data item from n, this node first suspends this operation. Subsequently, n notifies the channels whose sink ends coincide on n that it accepts a data item. Once at least one of these channels has notified n that it offers a data item, n resolves the take: atomically, n fetches this data item 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.

- A mixed node n has *pumping station semantics*: a combination of the replicator semantics and merger semantics discussed above, where fetching and dispensing occurs atomically.

In the process algebraic semantics of Reo, one associates each of the m source ends of a node with an action $src_{1 \le i \le m}$ and each of its n sink ends with an action $snk_{1 \le i \le n}$. Then, one can describe nodes by combining the processes for a binary replicator (one sink end to two source ends), a binary merger (two sink ends to one source end), a one-to-one pumping station, and a process for boundary nodes:

$$
\begin{aligned}
Replicator\langle snk; src_1, src_2 \rangle &\mapsto snk \sqcup src_1 \sqcup src_2 \cdot Replicator\langle snk; src_1, src_2 \rangle \\
Merger\langle snk_1, snk_2; src \rangle &\mapsto (snk_1 \sqcup src + snk_2 \sqcup src) \cdot Merger\langle snk_1, snk_2; src \rangle \\
PumpingStation\langle snk; src \rangle &\mapsto snk \sqcup src \cdot PumpingStation\langle snk; src \rangle \\
Boundary\langle bnd \rangle &\mapsto bnd \cdot Boundary\langle bnd \rangle
\end{aligned}
$$

Connectors. To get the behavior of a connector as a process, one composes the processes of the constituents of that connector in parallel and synchronizes their actions appropriately. Below, we give the processes of the connectors in Figures 1a and 1c. See [16, 13, 14, 15] for more examples.

$$
\begin{aligned}
Fig1a = \partial_{\{a_1, \bar{a}_1, x_1, \bar{x}_1, x_2, \bar{x}_2, b_1, \bar{b}_1\}} \big(&\Gamma_{\{a_1 \sqcup \bar{a}_1 \to a, x_1 \sqcup \bar{x}_1 \to x, x_2 \sqcup \bar{x}_2 \to x, b_1 \sqcup \bar{b}_1 \to b\}} \big(\\
&Boundary\langle \bar{a}_1 \rangle \,\|\, Fifo1\langle a_1; x_1 \rangle \,\|\, PumpingStation\langle \bar{x}_1; \bar{x}_2 \rangle \,\|\, Fifo1\langle x_2; b_1 \rangle \,\|\, Boundary\langle \bar{b}_1 \rangle \big) \big)
\end{aligned}
$$

$$
\begin{aligned}
Fig1c = \partial_{\{*_{bnd}, \bar{*}_{bnd}, *_i, \bar{*}_i \,|\, * \in \{a,b,c\} \land i \in \{1,2\}\}} \big(&\Gamma_{\{*_{bnd} \sqcup \bar{*}_{bnd} \sqcup *_i \sqcup \bar{*}_i \to * \,|\, * \in \{a,b,c\} \land i \in \{1,2\}\}} \big(\\
&Boundary\langle a_{bnd} \rangle \,\|\, Replicator\langle \bar{a}_{bnd}; \bar{a}_1, \bar{a}_2 \rangle \,\|\, Boundary\langle b_{bnd} \rangle \,\|\, Replicator\langle \bar{b}_{bnd}; \bar{b}_1, \bar{b}_2 \rangle \,\| \\
&SyncDrain\langle a_2; b_2 \rangle \,\|\, Sync\langle a_1; c_1 \rangle \,\|\, Fifo1\langle b_1; c_2 \rangle \,\|\, Merger\langle \bar{c}_1, \bar{c}_2; \bar{c}_{bnd} \rangle \,\|\, Boundary\langle c_{bnd} \rangle \big) \big)
\end{aligned}
$$

4   Splitting Processes

Recall from Section 1 that we aim at establishing the validity of optimizing implementations of Reo through the identification of (a)synchronous regions. Essentially, we want to show that *splitting* connectors along the boundaries of their (a)synchronous regions (and running the resulting subconnectors concurrently) does not give rise to inadmissible behavior. In this section, we lay the foundation for this kind of splitting in terms of a splitting procedure for processes. Later, in Section 5, we apply this procedure to the process algebraic semantics of Reo, thereby justifying the splitting of connectors. Here, we start by explaining the intuition behind our splitting procedure; formal definitions appear in Section 4.1, followed by theorems and proofs in Section 4.2. We note that our notion of "splitting" differs from "decomposition" in the spirit of [17]: in our context, primality or uniqueness do not matter.

Let $\mathsf{Acts}(p)$ denote the set of actions occurring in a process p. We introduce the function $\mathtt{split}$, which splits a process p along a set of actions A into two processes: one of these processes contains no actions in $\mathsf{Acts}(p) \setminus A$, while the other process contains no actions in A. We call the former process the *A-isolation of p* and the latter process the *A-coisolation of p*. We aim at constructing p's isolation and its coisolation such that their parallel composition equals p under appropriate synchronization. Informally, to construct p's A-isolation, replace every action in p as follows:

- If $a \in A$, replace a with the multi-action $a \sqcup \xi(a)$, where $\xi(a)$ denotes a fresh action with respect to $\mathsf{Acts}(p)$. Intuitively, $\xi(a)$ represents the act of "disseminating that this process performs a."

- If $b \notin A$, replace b with the action $\overline{\xi}(b)$, where $\overline{\xi}(b)$ denotes a fresh action with respect to $\mathsf{Acts}(p)$. Intuitively, $\overline{\xi}(b)$ represents the act of "discovering that another process performs b."

Symmetrically, to construct the A-coisolation of a process p, replace in p every $b \in A$ with $\overline{\xi}(b)$ and every $b \notin A$ with $b \sqcup \xi(b)$. Note that because the foregoing affects only multi-actions, p's isolation and its coisolation have the same structure as p. In other words: the process p, its isolation, and its coisolation have the same transition system modulo transition labels.

To illustrate isolation and coisolation, consider the process $q = a \cdot b$ as a running example. This process has $q_1 = a \sqcup \xi(a) \cdot \overline{\xi}(b)$ as its $\{a\}$-isolation and $q_2 = \overline{\xi}(a) \cdot b \sqcup \xi(b)$ as its $\{a\}$-coisolation. However, the parallel composition of q_1 and q_2 is not equal to q yet: to ensure that a process equals the parallel composition of its isolation and its coisolation, these latter two processes should synchronize on $\xi(a)$ and $\overline{\xi}(a)$ for each a. To this end, we apply the communication operator Γ to such compositions. In our running example, this yields the process $\Gamma_C(q_1 \parallel q_2)$ with $C = \{\xi(a) \sqcup \overline{\xi}(a) \to \mathtt{tau}, \xi(b) \sqcup \overline{\xi}(b) \to \mathtt{tau}\}$. The special action $\mathtt{tau}$ serves as a placeholder action for τ, and we can hide it immediately using the abstraction operator $\mathscr{T}$;[3] henceforth, without loss of generality, we assume $\mathtt{tau} \notin \mathsf{Acts}(p)$ for each p. In our running example, this yields the process $\mathscr{T}_I(\Gamma_C(q_1 \parallel q_2))$ with $I = \{\mathtt{tau}\}$ and C as before. But also this process is not equal to q yet: only synchronization and abstraction do not suffice—we must also block those actions whose performance in isolation "makes no sense." For instance, we consider every unpaired occurrence of $\overline{\xi}(a)$ in a multi-action α nonsensical: intuitively, performing $\overline{\xi}(a)$ suggests that some process discovers that another process performs a, even though this does not happen (otherwise, also $\xi(a)$ would occur in α). By symmetry, we consider also every unpaired occurrence of $\xi(a)$ nonsensical. To block unpaired occurrences of $\xi(a)$ and $\overline{\xi}(a)$, we apply the blocking operator ∂. In our running example, this yields the process $\partial_B(\mathscr{T}_I(\Gamma_C(q_1 \parallel q_2)))$ with $B = \{\xi(a), \overline{\xi}(a), \xi(b), \overline{\xi}(b)\}$ and I and C as before. This process equals q.

---

[3]We use this construction, because mCRL2 does not permit communications to map directly to τ.

$$
\begin{aligned}
\mathrm{dom}(\Xi) &= dom(\xi_w) \cap dom(\overline{\xi}_w) \\
\mathrm{img}(\Xi) &= img(\xi_w) \cup img(\overline{\xi}_w) \\
\mathrm{comm}(\Xi) &= \{\xi_w(a) \sqcup \overline{\xi}_w(a) \to \mathtt{tau} \mid (a,w) \in \mathrm{dom}(\Xi)\}
\end{aligned}
$$

Q1	$?_\Xi(\tau) \simeq \tau$
Q2	$?_\Xi(\delta) \simeq \delta$
Q3	$?_\Xi(p+q) \simeq ?_\Xi(p) + ?_\Xi(q)$
Q4	$?_\Xi(p \cdot q) \simeq ?_\Xi(p) \cdot ?_\Xi(q)$

Figure 4: Auxiliary functions for substitution environments.   Figure 5: Axioms for ?.

$$
\begin{aligned}
&\mathsf{isol}_\Xi(a,A,w) = a \sqcup \xi_w(a) \ \text{ if } a \in A && \overline{\mathsf{isol}}_\Xi(a,A,w) = \overline{\xi}_w(a) && \text{if } a \in A \\
&\mathsf{isol}_\Xi(b,A,w) = \overline{\xi}_w(b) \qquad \text{ if } b \notin A && \overline{\mathsf{isol}}_\Xi(b,A,w) = b \sqcup \xi_w(b) \ \text{ if } b \notin A
\end{aligned}
$$

$$
\begin{aligned}
\widehat{\mathsf{isol}}_\Xi(\vartheta,A,w) &= \vartheta && \text{for } \vartheta \in \{\tau,\delta\} \\
\widehat{\mathsf{isol}}_\Xi(p \oplus q,A,w) &= \widehat{\mathsf{isol}}_\Xi(p,A,w) \oplus \widehat{\mathsf{isol}}_\Xi(q,A,w) && \text{for } \oplus \in \{\cdot,\sqcup\} \\
\widehat{\mathsf{isol}}_\Xi(p+q,A,w) &= \widehat{\mathsf{isol}}_\Xi(p,A,w1) + \widehat{\mathsf{isol}}_\Xi(q,A,w2)
\end{aligned}
$$

Figure 6: The functions isol and $\overline{\mathsf{isol}}$. Let $p \in \mathbb{S}\mathrm{eq}$ and $\widehat{\mathsf{isol}} \in \{\mathsf{isol}, \overline{\mathsf{isol}}\}$.

4.1 Formal Definitions

We proceed with formal definitions of the splitting procedure outlined above. We start with a formal account of the fresh auxiliary actions of the form $\xi(a)$ and $\overline{\xi}(a)$. As suggested by this notation, ξ and $\overline{\xi}$ denote functions that take an action a as their input and produce another action as their output. We collect such pairs of functions in *substitution environments* as follows. Let $\{1,2\}^*$ denote the set of finite strings over $\{1,2\}$, ranged over by w, v, u, etc.

Definition 1. *A substitution environment, typically denoted by Ξ, is a quintuple $(\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ consisting of a process definition $\dot{P} \mapsto \dot{p}$, a set $\mathbb{A} \subseteq \mathbb{A}\mathrm{ct}$, an action $\mathtt{tau} \in \mathbb{A}\mathrm{ct} \setminus \mathbb{A}$ and injective functions $\xi, \overline{\xi} : \mathbb{A} \times \{1,2\}^* \rightarrowtail \mathbb{A}\mathrm{ct} \setminus (\mathbb{A} \cup \mathtt{tau})$ such that $img(\xi) \cap img(\overline{\xi}) = \emptyset$.*

Henceforth, we write $\xi_w(a)$ and $\overline{\xi}_w(a)$ instead of $\xi(a,w)$ and $\overline{\xi}(a,w)$. Note that we dropped the w subscripts in our running example above: as we did not need this extra string of information, we omitted it for simplicity. In the general case, however, this information plays a key role, as explained shortly. The process definition in a substitution environment represents the main process to be split.

Figure 4 shows auxiliary functions for substitution environment. The functions "dom" and "img" map substitution environments to their domain and image. The function "comm" maps substitution environments to communications derivable from them.

To formalize the notions of A-isolation and A-coisolation, we introduce the functions isol and $\overline{\mathsf{isol}}$. Figure 6 shows their definitions. The functions isol and $\overline{\mathsf{isol}}$ take for arguments a sequential process, a set of actions $A \subseteq \mathbb{A}\mathrm{ct}$, a string $w \in \{1,2\}^*$, and a substitution environment (as a subscript for notational convenience). For most processes p, $\mathsf{isol}_\Xi(p,A,w)$ and $\overline{\mathsf{isol}}_\Xi(p,A,w)$ invoke themselves recursively on p's immediate subprocesses, the same set A, and the same string w. One exception exists: processes of the form $p+q$. For such processes, isol and $\overline{\mathsf{isol}}$ invoke themselves recursively on $w1$ and $w2$ instead of w. This ensures that in their parallel composition, the process $\mathsf{isol}_\Xi(p+q,A,w)$ can "track" which choice the process $\overline{\mathsf{isol}}_\Xi(p+q,A,w)$ makes and vice versa.

To clarify this, let us illustrate what would happen if $\mathsf{isol}_\Xi(p+q,A,w)$ and $\overline{\mathsf{isol}}_\Xi(p+q,A,w)$ invoke themselves recursively without changing w. In that case, w has no influence on the behavior of isol and $\overline{\mathsf{isol}}$, and we can omit it from our definitions. Now, suppose that we want to compose the $\{a\}$-isolation

and $\{a\}$-coisolation of the process $r = a \cdot b + a \cdot c$ in parallel. We have:

$$\mathsf{isol}_\Xi(r, \{a\}) = \qquad\qquad a \sqcup \xi(a) \cdot \quad \overline{\xi}(b) + a \sqcup \xi(a) \cdot \overline{\xi}(c)$$
$$\overline{\mathsf{isol}}_\Xi(r, \{a\}) = \overline{\xi}(a) \cdot b \sqcup \xi(b) + \quad \overline{\xi}(a) \cdot c \sqcup \xi(c)$$

Thus, the process $\mathsf{isol}_\Xi(r, \{a\})$ can erroneously synchronize its left-most multi-action $a \sqcup \xi(a)$ with the right-most multi-action $\overline{\xi}(a)$ of the process $\overline{\mathsf{isol}}_\Xi(r, \{a\})$. By changing w in the recursive invocations of $\mathsf{isol}_\Xi(p + q, A, w)$ and $\overline{\mathsf{isol}}_\Xi(p + q, A, w)$, this problem does not arise: it ensures that $a \sqcup \xi_{w1}(a)$ (on the left) can synchronize only with $\overline{\xi}_{w1}(a)$ (also on the left)—not with $\overline{\xi}_{w2}(a)$ (on the right). Note that this depends on the injectivity of ξ and $\overline{\xi}$ (see Definition 1).

The definition of the function split follows straightforwardly now that we have the functions isol and $\overline{\mathsf{isol}}$. We also introduce an auxiliary operator, denoted by ?, which encapsulates the communication, hiding, and blocking necessary to get equality of processes. Figure 5 shows axioms for this operator.[4]

Definition 2. $?_\Xi(p) = \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(\Gamma_{\mathrm{comm}(\Xi)}(p)))$

Definition 3. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \subseteq \mathbb{A}$,

$$\mathsf{split}_\Xi(p, A, w) = \begin{cases} ?_\Xi(\mathsf{isol}_\Xi(p, A, w) \,\|\, \overline{\mathsf{isol}}_\Xi(p, A, w)) & \text{if } p \in \mathbb{S}\mathrm{eq} \\ \mathsf{split}_\Xi(p_1, A, w) \oplus \mathsf{split}_\Xi(p_2, A, w) & \text{if } p \notin \mathbb{S}\mathrm{eq} \text{ and } p = p_1 \oplus p_2 \text{ and } \oplus \in \{\cdot, +, \|, \mathbb{L}, |\} \\ \dagger(\mathsf{split}_\Xi(p_1, A, w)) & \text{if } p = \dagger(p_1) \text{ and } \dagger \in \{\nabla_V, \partial_B, \rho_R, \Gamma_C, \mathscr{T}_I\} \\ \mathrm{SPLIT}_\Xi(\dot{P}, A, w) & \text{if } p = \dot{P} \end{cases}$$

where $\mathrm{SPLIT}_\Xi(\dot{P}, A, w)$ *denotes a reference to the process* $\mathsf{split}_\Xi(\dot{p}, A, w)$.

4.2 Theorems

Suppose an execution environment $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$. We prove that splitting $\dot{p}$ as described above yields a process equal to $\dot{p}$. We proceed in three steps. First, we prove our result for multi-actions. Then, we extend this result to sequential processes. Finally, we establish it for general processes. In each of these theorems we restrict our attention to *syntactically τ-free* specifications, because we work under strong bisimulation. Under equivalences weaker than strong bisimulation, we can relax this τ-freeness.

The axioms occasionally referred to in the remainder of this section appear in Figure 11, Section A.

4.2.1 A theorem for multi-actions

We start with a theorem for multi-actions, which states that splitting a syntactically τ-free multi-action equals that multi-action. Let τ-free(α) denote that τ does not occur in α (see [11] for a formal definition).

Theorem 1. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(\alpha) \cup A \subseteq \mathbb{A}$,

$$\tau\text{-free}(\alpha) \textbf{ implies } \mathsf{split}_\Xi(\alpha, A, w) \simeq \alpha$$

To prove this theorem, we need some auxiliary lemmas. We formulate these lemmas below; detailed proofs, as well as additional propositions on which these proofs rely, appear in [11]. The first lemma states that the parallel composition of the isolation and the coisolation of a process equals their synchronous composition (after applying communication, hiding, and blocking).

Lemma 1. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup A \subseteq \mathbb{A}$,

$$\big[\tau\text{-free}(p) \textbf{ and } p \in \mathbb{S}\mathrm{eq}\big] \textbf{ implies}$$
$$?_\Xi(\mathsf{isol}_\Xi(p, A, w) \,\|\, \overline{\mathsf{isol}}_\Xi(p, A, w)) \simeq ?_\Xi(\mathsf{isol}_\Xi(p, A, w) \,|\, \overline{\mathsf{isol}}_\Xi(p, A, w))$$

---

[4]The axiom Q1 follows from the axioms C1, H1, and B1 in Figure 11 in Section A; Q2 follows from C2, H5, and B5; Q3 follows from C3, H6, and B6; Q4 follows from C4, H7, and B7.

Proof (sketch). By the axioms M and A6, we must show that $?_\Xi(\mathsf{isol}_\Xi(p,A,w) \,\|\, ?_\Xi(\overline{\mathsf{isol}}_\Xi(p,A,w)$ and $?_\Xi(\overline{\mathsf{isol}}_\Xi(p,A,w) \,\|\, ?_\Xi(\mathsf{isol}_\Xi(p,A,w)$ equal δ. Both of these processes start with a multi-action α^δ. By the definition of isol and $\overline{\mathsf{isol}}$ (and, in particular, the injectivity and image-disjointness of ξ and $\overline{\xi}$), if $\alpha^\delta \neq \delta$, it must contain an action $\xi_w(a)$ without $\overline{\xi}_w(a)$ (or vice versa). But then, the blocking operator in ? (combined with SMA) will equate α^δ to δ. This suffices to show that $?_\Xi(\mathsf{isol}_\Xi(p,A,w) \,\|\, ?_\Xi(\overline{\mathsf{isol}}_\Xi(p,A,w)$ and $?_\Xi(\overline{\mathsf{isol}}_\Xi(p,A,w) \,\|\, ?_\Xi(\mathsf{isol}_\Xi(p,A,w)$ equal δ by A7 (because these processes start with α^δ). See [11] for a detailed proof. $\square$

Note that Lemma 1 involves sequential processes rather than only multi-actions. This enables us to use this lemma also in our proof of Theorem 2, below.

The following lemma consists of two parts. The first part states that one can rewrite every multi-action composed of the isolation and the coisolation of a multi-action α into a representation with the following characteristics: (i) for every communication $\beta_i \to \mathtt{tau}$ induced by the substitution environment involved, β_i occurs zero or more times; (ii) the *remainder* $\breve{\alpha}$ does not contain any fragment of any β_i and vice versa, denoted as $\breve{\alpha} \smile \beta_i$. (See [11] for a formal definition of the latter relation.) The second part of the following lemma states the additivity property $\Gamma_C(\alpha_1 \sqcup \alpha_2) \simeq \Gamma_C(\alpha_1) \sqcup \Gamma_C(\alpha_2)$ when α_1 and α_2 each have such a representation. Let $\bigsqcup_n \beta$ denote the sequence $\beta \sqcup \cdots \sqcup \beta$ of length n.

Lemma 2.

1. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(\alpha) \cup A \subseteq \mathbb{A}$ *and* $\mathsf{dom}(\mathsf{comm}(\Xi)) = \{\beta_1, \ldots, \beta_k\}$,

$$\mathsf{isol}_\Xi(\alpha,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha,A,w) \simeq \bigsqcup_{n_1} \beta_1 \sqcup \cdots \sqcup \bigsqcup_{n_k} \beta_k \sqcup \breve{\alpha} \text{ and } \breve{\alpha} \smile \beta_i$$

2. *For all* $C = \{\beta_1 \to b_1, \ldots, \beta_k \to b_k\}$,

$$\left[\begin{array}{c} \alpha_1 = \bigsqcup_{n_1} \beta_1 \sqcup \cdots \sqcup \bigsqcup_{n_k} \beta_k \ \sqcup \breve{\alpha}_1 \text{ and } \breve{\alpha}_1 \smile \beta_i \\ \text{and } \alpha_2 = \bigsqcup_{m_1} \beta_1 \sqcup \cdots \sqcup \bigsqcup_{m_k} \beta_k \sqcup \breve{\alpha}_2 \text{ and } \breve{\alpha}_2 \smile \beta_i \end{array}\right] \textbf{ implies } \left[\begin{array}{c} \Gamma_C(\alpha_1 \sqcup \alpha_2) \simeq \\ \Gamma_C(\alpha_1) \sqcup \Gamma_C(\alpha_2) \end{array}\right]$$

Proof (sketch).

1. If $\alpha = a$, there exists a $\beta_\ell = \xi_w(a) \sqcup \overline{\xi}_w(a)$ for some ℓ. By the definition of isol and $\overline{\mathsf{isol}}$, we have that $\mathsf{isol}_\Xi(\alpha,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha,A,w) = \beta_\ell \sqcup a$. Identifying $\breve{\alpha}$ with a, we must show that a does not occur in any β_i. This follows from the fact that ξ and $\overline{\xi}$ have disjoint domains and images by their definition. The general case follows by structural induction.

2. Because $\breve{\alpha}_1$ and $\breve{\alpha}_2$ do not contain any fragment of any β_i (i.e., $\breve{\alpha}_1 \smile \beta_i$ and $\breve{\alpha}_2 \smile \beta_i$), combining them in the same multi-action does not make the communication operator applicable: there exists no communication in $\alpha_1 \sqcup \alpha_2$ that did not exist already in α_1 or in α_2.

See [11] for a detailed proof. $\square$

The following corollary follows from the previous lemma: it asserts the additivity property $\Gamma_C(\alpha_1 \sqcup \alpha_2) \simeq \Gamma_C(\alpha_1) \sqcup \Gamma_C(\alpha_2)$ for $\alpha_1 = \mathsf{isol}_\Xi(\alpha,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha,A,w)$ and $\alpha_2 = \mathsf{isol}_\Xi(\beta,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\beta,A,w)$.

Corollary 1. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(\alpha) \cup A \subseteq \mathbb{A}$,

$$\left[\begin{array}{c} \Gamma_{\mathsf{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha,A,w) \sqcup \mathsf{isol}_\Xi(\beta,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\beta,A,w)) \simeq \\ \Gamma_{\mathsf{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha,A,w)) \sqcup \Gamma_{\mathsf{comm}(\Xi)}(\mathsf{isol}_\Xi(\beta,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\beta,A,w)) \end{array}\right]$$

Finally, Figure 7 shows a proof of Theorem 1.

Suppose $\tau\text{-free}(\alpha)$ (Prem). We proceed by structural induction on α.

Base: $\alpha = a^\tau$. If $a^\tau = \tau$, we get a contradition with Prem. If $a^\tau = a$:

$$?_\Xi(\mathsf{split}_\Xi(\alpha,A,w)) \overset{\alpha,a^\tau,\mathsf{split}}{=} ?_\Xi(\mathsf{isol}_\Xi(a,A,w) \parallel \overline{\mathsf{isol}}_\Xi(a,A,w))$$

$$\overset{\mathrm{Prem}\to\mathrm{Lem.\ 1}}{\simeq} ?_\Xi(\mathsf{isol}_\Xi(a,A,w) \mid \overline{\mathsf{isol}}_\Xi(a,A,w))$$

$$\overset{\mathrm{SMA,isol},\overline{\mathrm{isol}}}{\simeq} ?_\Xi(a \sqcup \xi_w(a) \sqcup \overline{\xi}_w(a))$$

$$\overset{?}{=} \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(\Gamma_{\mathrm{comm}(\Xi)}(a \sqcup \xi_w(a) \sqcup \overline{\xi}_w(a))))$$

$$\overset{\mathrm{C1,SMA}}{\simeq} \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(a \mid \mathtt{tau}))$$

$$\overset{\mathrm{H4,H3,H2}}{\simeq} \partial_{\mathrm{img}(\Xi)}(a \mid \tau) \overset{\mathrm{B4,SMA}}{\simeq} a \sqcup \tau \overset{\mathrm{MA3}}{\simeq} a \overset{a^\tau,\alpha}{\simeq} \alpha$$

Step: $\alpha = \alpha_1 \sqcup \alpha_2$. Suppose that this proposition holds for α_1 (IH1) and α_2 (IH2).

$$?_\Xi(\mathsf{split}_\Xi(\alpha,A,w))$$

$$\overset{\alpha,\mathsf{split}}{=} ?_\Xi(\mathsf{isol}_\Xi(\alpha_1 \sqcup \alpha_2,A,w) \parallel \overline{\mathsf{isol}}_\Xi(\alpha_1 \sqcup \alpha_2,A,w))$$

$$\overset{\mathrm{Prem}\to\mathrm{Lem.\ 1}}{\simeq} ?_\Xi(\mathsf{isol}_\Xi(\alpha_1 \sqcup \alpha_2,A,w) \mid \overline{\mathsf{isol}}_\Xi(\alpha_1 \sqcup \alpha_2,A,w))$$

$$\overset{\mathrm{SMA,isol},\overline{\mathrm{isol}}}{\simeq} ?_\Xi(\mathsf{isol}_\Xi(\alpha_1,A,w) \sqcup \mathsf{isol}_\Xi(\alpha_2,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))$$

$$\overset{?}{=} \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(\Gamma_{\mathrm{comm}(\Xi)}($$
$$\mathsf{isol}_\Xi(\alpha_1,A,w) \sqcup \mathsf{isol}_\Xi(\alpha_2,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))))$$

$$\overset{\mathrm{Cor.\ 1}}{\simeq} \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}($$
$$\Gamma_{\mathrm{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha_1,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w)) \sqcup$$
$$\Gamma_{\mathrm{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha_2,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))))$$

$$\overset{\mathrm{SMA,B4,H4}}{\simeq} \partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(\Gamma_{\mathrm{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha_1,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w)))) \mid$$
$$\partial_{\mathrm{img}(\Xi)}(\mathscr{T}_{\{\mathtt{tau}\}}(\Gamma_{\mathrm{comm}(\Xi)}(\mathsf{isol}_\Xi(\alpha_2,A,w) \sqcup \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))))$$

$$\overset{?,\mathrm{SMA}}{\simeq} ?_\Xi(\mathsf{isol}_\Xi(\alpha_1,A,w) \mid \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w)) \mid ?_\Xi(\mathsf{isol}_\Xi(\alpha_2,A,w) \mid \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))$$

$$\overset{\mathrm{Prem}\to\mathrm{Lem.\ 1}}{\simeq} ?_\Xi(\mathsf{isol}_\Xi(\alpha_1,A,w) \parallel \overline{\mathsf{isol}}_\Xi(\alpha_1,A,w)) \mid ?_\Xi(\mathsf{isol}_\Xi(\alpha_2,A,w) \parallel \overline{\mathsf{isol}}_\Xi(\alpha_2,A,w))$$

$$\overset{\mathsf{split}}{\simeq} ?_\Xi(\mathsf{split}_\Xi(\alpha_1,A,w)) \mid ?_\Xi(\mathsf{split}_\Xi(\alpha_2,A,w)) \overset{\mathrm{IH1,IH2}}{\simeq} \alpha_1 \mid \alpha_2 \overset{\mathrm{SMA}}{\simeq} \alpha_1 \sqcup \alpha_2 \overset{\alpha}{=} \alpha$$

Figure 7: Proof of Theorem 1

4.2.2   Theorems for processes

The following theorem generalizes Theorem 1 from multi-actions to processes in $\mathbb{S}\mathrm{eq}$: it states that splitting such a syntactically τ-free process equals that process.

Theorem 2. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup A \subseteq \mathbb{A}$,

$$\big[\tau\text{-free}(p) \text{ and } p \in \mathbb{S}\mathrm{eq}\big] \text{ implies } \mathsf{split}_\Xi(p,A,w) \simeq p$$

As for Theorem 1, we need some auxiliary lemmas to prove this theorem. We formulate these lemmas below; proofs, as well as additional propositions on which these proofs rely, appear in [11]. The first lemma states the additivity property $?_\Xi(r_1 \mid r_2) = ?_\Xi(r_1) \mid ?_\Xi(r_2)$ when r_1 and r_2 denote the isolation and the coisolation of the processes p and q. Importantly, while p and q may denote the same process, their isolation and coisolation must involve different strings over $\{1,2\}$ for the additivity to hold.

Lemma 3. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup \mathsf{Acts}(q) \cup A \subseteq \mathbb{A}$,

$$\big[\tau\text{-free}(p) \text{ and } \tau\text{-free}(q) \text{ and } p,q \in \mathbb{S}\mathrm{eq} \text{ and } w \neq v\big] \text{ implies}$$
$$?_\Xi(\mathsf{isol}_\Xi(p,A,w) \mid \overline{\mathsf{isol}}_\Xi(q,A,v)) \simeq ?_\Xi(\mathsf{isol}_\Xi(p,A,w)) \mid ?_\Xi(\overline{\mathsf{isol}}_\Xi(q,A,v))$$

Proof (sketch). The actions occurring in $\mathsf{isol}_\Xi(p,A,w)$ differ from those occurring in $\mathsf{isol}_\Xi(q,A,v)$ (except for the original actions in p and q) because ξ and $\overline{\xi}$ have disjoint images by their definition *and* because $w \neq v$. In that case, there exists no communication in $\mathsf{isol}_\Xi(p,A,w) \mid \overline{\mathsf{isol}}_\Xi(p,A,w)$ that did not exist already in $\mathsf{isol}_\Xi(p,A,w)$ or in $\overline{\mathsf{isol}}_\Xi(p,A,w)$, enabling one to distribute Γ in ? among them. We can do the same for $\mathscr{T}$ and ∂ in ? (by B4 and H4). See [11] for a detailed proof. □

The following lemma states that the process $?_\Xi(r)$ deadlocks when r denotes only the isolation or only the coisolation of a process p.

Lemma 4. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup A \subseteq \mathbb{A}$,

$$\big[\tau\text{-free}(p) \text{ and } p \in \mathbb{S}\mathsf{eq}\big] \text{ implies } \big[?_\Xi(\mathsf{isol}_\Xi(p,A,w)) \simeq \delta \text{ and } ?_\Xi(\overline{\mathsf{isol}}_\Xi(p,A,w)) \simeq \delta\big]$$

Proof (sketch). Similar to the proof sketch of Lemma 1. See [11] for a detailed proof. □

Suppose that we have two sequential processes, namely $r_1 = \mathsf{isol}_\Xi(p,A,w) \cdot \mathsf{isol}_\Xi(q,A,w)$ and $r_2 = \overline{\mathsf{isol}}_\Xi(p,A,w) \cdot \overline{\mathsf{isol}}_\Xi(q,A,w)$. Moreover, suppose that we take their parallel composition $r_1 \| r_2$. Our final lemma states that instead of taking this parallel composition, one can compose the parallel composition $r^\sharp = \mathsf{isol}_\Xi(p,A,w) \mid \overline{\mathsf{isol}}_\Xi(p,A,w)$ and the parallel composition $r^\flat = \mathsf{isol}_\Xi(q,A,w) \mid \overline{\mathsf{isol}}_\Xi(q,A,w)$ sequentially and get the same process. In short: $r^\sharp \cdot r^\flat$ equals $r_1 \| r_2$.

Lemma 5. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup A \subseteq \mathbb{A}$,

$$\big[\tau\text{-free}(p) \text{ and } p \in \mathbb{S}\mathsf{eq}\big] \text{ implies}$$
$$?_\Xi((\mathsf{isol}_\Xi(p,A,w) \cdot \mathsf{isol}_\Xi(q,A,w)) \| (\overline{\mathsf{isol}}_\Xi(p,A,w) \cdot \overline{\mathsf{isol}}_\Xi(q,A,w))) \simeq$$
$$?_\Xi(\mathsf{isol}_\Xi(p,A,w) \| \overline{\mathsf{isol}}_\Xi(p,A,w) \cdot \mathsf{isol}_\Xi(q,A,w) \| \overline{\mathsf{isol}}_\Xi(q,A,w))$$

Proof (sketch). The processes $\mathsf{isol}_\Xi(p,A,w)$ and $\overline{\mathsf{isol}}_\Xi(p,A,w)$ always stay "synchronized" when composed in parallel due to the ? operator. This implies that these processes "finish" at the same time. Consequently, $\mathsf{isol}_\Xi(q,A,w)$ and $\overline{\mathsf{isol}}_\Xi(q,A,w)$ start at the same time, which implies the desired result. See [11] for a detailed proof. □

Finally, Figure 8 shows a proof of Theorem 2. Our last theorem generalizes Theorem 2 from sequential processes to parallel processes; Figure 9 shows a proof.

Theorem 3. *For all* $\Xi = (\dot{P} \mapsto \dot{p}, \mathbb{A}, \mathtt{tau}, \xi, \overline{\xi})$ *such that* $\mathsf{Acts}(p) \cup A \subseteq \mathbb{A}$,

$$\tau\text{-free}(\dot{p}) \text{ implies } \mathsf{split}_\Xi(\dot{p}, A, w) \simeq \dot{p}$$

5 Application: Splitting Connectors

5.1 Formalization of (A)synchronous Regions

We provide a formal definition of the synchronous regions of a connector, based on the mCRL2 semantics of Reo. Let p denote a process describing the behavior of a Reo connector, and let $\longrightarrow$ denote its transition relation (labeled with multi-actions). Recall that every action in p represents a channel end or a node end. Let $a \in \mathsf{Acts}(p)$ denote one such an end. We define the *a*-synchronous region of p as the smallest set $X_a \subseteq \mathsf{Acts}(p)$ such that:

- $a \in X_a$.

- If $b \in X_a$ then $\mathsf{Acts}(\beta) \subseteq X_a$ for all β such that $q \xrightarrow{\beta} q'$ and $b \in \mathsf{Acts}(\beta)$.

Suppose $\big[\tau\text{-free}(p)$ **and** $p \in \mathbb{S}\text{eq}\big]$ (Prem). We have:

$$\text{split}_\Xi(p,A,w) \overset{\text{split}}{=} ?_\Xi(\text{isol}_\Xi(p,A,w) \,\|\, \overline{\text{isol}}_\Xi(p,A,w)) \overset{\text{Prem}\to\text{Lem. 1}}{\simeq} ?_\Xi(\text{isol}_\Xi(p,A,w) \,|\, \overline{\text{isol}}_\Xi(p,A,w))$$

Denote this property by Obs. We proceed by structural induction on p.

Base: $p = \alpha^\delta$. If $\alpha^\delta = \alpha$, the theorem follows by Theorem 1. If $\alpha^\delta = \delta$:

$$\text{split}_\Xi(p,A,w) \overset{\text{Obs},p}{=} ?_\Xi(\text{isol}_\Xi(\delta,A,w) \,|\, \overline{\text{isol}}_\Xi(\delta,A,w)) \overset{\text{isol},\overline{\text{isol}}}{=} ?_\Xi(\delta \,|\, \delta) \overset{\text{S4,Q2}}{\simeq} \delta \overset{\alpha^\delta,p}{=} p$$

Step: $p = p_1 \oplus p_2$ with $\oplus \in \{+,\cdot\}$. Suppose that this theorem holds for p_1 (IH1) and p_2 (IH2).

Case: $p = p_1 + p_2$.

$$\text{split}_\Xi(p,A,w)$$
$$\overset{\text{Obs},p}{\simeq} ?_\Xi(\text{isol}_\Xi(p_1 + p_2,A,w) \,|\, \overline{\text{isol}}_\Xi(p_1 + p_2,A,w))$$
$$\overset{\text{isol},\overline{\text{isol}}}{=} ?_\Xi((\text{isol}_\Xi(p_1,A,w1) + \text{isol}_\Xi(p_2,A,w2)) \,|\, (\overline{\text{isol}}_\Xi(p_1,A,w1) + \overline{\text{isol}}_\Xi(p_2,A,w2)))$$
$$\overset{\text{S7}}{\simeq} ?_\Xi(\text{isol}_\Xi(p_1,A,w1) \,|\, \overline{\text{isol}}_\Xi(p_1,A,w1) + \text{isol}_\Xi(p_1,A,w1) \,|\, \overline{\text{isol}}_\Xi(p_2,A,w2) +$$
$$\quad \text{isol}_\Xi(p_2,A,w2) \,|\, \overline{\text{isol}}_\Xi(p_1,A,w1) + \text{isol}_\Xi(p_2,A,w2) \,|\, \overline{\text{isol}}_\Xi(p_2,A,w2))$$
$$\overset{\text{Q3}}{\simeq} ?_\Xi(\text{isol}_\Xi(p_1,A,w1) \,|\, \overline{\text{isol}}_\Xi(p_1,A,w1)) + ?_\Xi(\text{isol}_\Xi(p_1,A,w1) \,|\, \overline{\text{isol}}_\Xi(p_2,A,w2)) +$$
$$\quad ?_\Xi(\text{isol}_\Xi(p_2,A,w2) \,|\, \overline{\text{isol}}_\Xi(p_1,A,w1)) + ?_\Xi(\text{isol}_\Xi(p_2,A,w2) \,|\, \overline{\text{isol}}_\Xi(p_2,A,w2))$$
$$\overset{\text{Obs}}{\simeq} ?_\Xi(\text{split}_\Xi(p_1,A,w1)) + ?_\Xi(\text{isol}_\Xi(p_1,A,w1) \,|\, \overline{\text{isol}}_\Xi(p_2,A,w2)) +$$
$$\quad ?_\Xi(\text{isol}_\Xi(p_2,A,w2) \,|\, \overline{\text{isol}}_\Xi(p_1,A,w1)) + \text{split}_\Xi(p_2,A,w2)$$
$$\overset{\text{Prem}\to\text{Lem. 3}}{\simeq} \text{split}_\Xi(p_1,A,w1) + ?_\Xi(\text{isol}_\Xi(p_1,A,w1)) \,|\, ?_\Xi(\overline{\text{isol}}_\Xi(p_2,A,w2)) +$$
$$\quad ?_\Xi(\text{isol}_\Xi(p_2,A,w2)) \,|\, ?_\Xi(\overline{\text{isol}}_\Xi(p_1,A,w1)) + \text{split}_\Xi(p_2,A,w2)$$
$$\overset{\text{Prem}\to\text{Lem. 4}}{\simeq} \text{split}_\Xi(p_1,A,w1) + \delta \,|\, \delta + \delta \,|\, \delta + \text{split}_\Xi(p_2,A,w2) \overset{\text{IH1,IH2,S4,A6}}{\simeq} p_1 + p_2 \overset{p}{\simeq} p$$

Case: $p = p_1 \cdot p_2$.

$$\text{split}_\Xi(p,A,w)$$
$$\overset{p,\text{split}}{=} ?_\Xi(\text{isol}_\Xi(p_1 \cdot p_2,A,w) \,\|\, \overline{\text{isol}}_\Xi(p_1 \cdot p_2,A,w))$$
$$\overset{\text{isol},\overline{\text{isol}}}{=} ?_\Xi((\text{isol}_\Xi(p_1,A,w) \cdot \text{isol}_\Xi(p_2,A,w)) \,\|\, (\overline{\text{isol}}_\Xi(p_1,A,w) \cdot \overline{\text{isol}}_\Xi(p_2,A,w)))$$
$$\overset{\text{Prem}\to\text{Lem. 5}}{\simeq} ?_\Xi(\text{isol}_\Xi(p_1,A,w) \,\|\, \overline{\text{isol}}_\Xi(p_1,A,w) \cdot \text{isol}_\Xi(p_2,A,w) \,\|\, \overline{\text{isol}}_\Xi(p_2,A,w))$$
$$\overset{\text{Q4}}{\simeq} ?_\Xi(\text{isol}_\Xi(p_1,A,w) \,\|\, \overline{\text{isol}}_\Xi(p_1,A,w)) \cdot ?_\Xi(\text{isol}_\Xi(p_2,A,w) \,\|\, \overline{\text{isol}}_\Xi(p_2,A,w))$$
$$\overset{\text{split}}{\simeq} \text{split}_\Xi(p_1,A,w) \cdot \text{split}_\Xi(p_2,A,w) \overset{\text{IH1,IH2}}{\simeq} p_1 \cdot p_2 \overset{p}{\simeq} p$$

Figure 8: Proof of Theorem 2.

- If $b \in X_a$ then $\text{Acts}(\beta') \subseteq X_a$ for all β, β' such that $q \overset{\beta}{\to} q'$ and $q \overset{\beta'}{\to} q''$ and $b \in \text{Acts}(\beta)$.

The second rule states that all the ends that occur in the same multi-action belong to the same synchronous region. The third rule states that all the ends that can have flow in some state q, but possibly in different transitions leaving q, belong to the same synchronous region. In that case, channel ends may exclude each other from flow, which requires them to synchronize and communicate about their behavior.

To exemplify the previous definition, consider the connector modeled by the process $p = a \sqcup b \cdot c + d$. Informally, either this connector has flow through a and b, followed by flow through c, or it has flow through d. We construct its a-synchronous region starting from the singleton set $X_a = \{a\}$ (first rule). Subsequently, due to the multi-action $a \sqcup b$, we add b to this set (second rule). The transition system of p contains a state with two outgoing transitions: one labeled by $a \sqcup b$, the other labeled by d. Hence, because $a \in X_a$, we add d to X_a (third rule). This concludes the construction: $X_a = X_b = X_d = \{a,b,d\}$.

First, we prove $\text{split}_\Xi(p,A,w) \simeq p[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$ for all p such that τ-free(p) and in which only $\dot{P}$ occurs as a reference (Prem), where $p[Q/R]$ denotes the syntactic substitution of the process reference Q for the process reference R in p, by structural induction on p.

Base: $p \in \mathbb{S}\text{eq}$ or $p = \dot{P}$. If $p \in \mathbb{S}\text{eq}$, this theorem follows by Theorem 2. If $p = \dot{P}$:

$$\text{split}_\Xi(p,A,w) \stackrel{p}{=} \text{split}_\Xi(\dot{P},A,w) \stackrel{\text{split}}{=} \text{SPLIT}_\Xi(\dot{P},A,w)$$
$$\stackrel{[Q/R]}{=} \dot{P}[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}] \stackrel{p}{=} p[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$$

Step: $p = p_1 \oplus p_2$ or $p = \dagger(p_1)$ for $\oplus \in \{\cdot,+,\|,\|,|\}$ and $\dagger \in \{\nabla_V, \partial_B, \rho_R, \Gamma_C, \mathscr{T}_I\}$. Suppose that this lemma holds for p_1 (IH1) and p_2 (IH2). We proceed by case distinction.

Case: $p = p_1 \oplus p_2$.

$$\text{split}_\Xi(p) \stackrel{p}{=} \text{split}_\Xi(p_1 \oplus p_2) \stackrel{\text{split}}{=} \text{split}_\Xi(p_1) \oplus \text{split}_\Xi(p_2)$$
$$\stackrel{\text{IH1,IH2}}{\simeq} p_1[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}] \oplus p_2[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$$
$$\stackrel{[Q/R]}{=} (p_1 \oplus p_2)[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}] \stackrel{p}{=} p[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$$

Case: $p = \dagger(p_1)$.

$$\text{split}_\Xi(p) \stackrel{p}{=} \text{split}_\Xi(\dagger(p_1)) \stackrel{\text{split}}{=} \dagger(\text{split}_\Xi(p_1))$$
$$\stackrel{\text{IH1}}{\simeq} \dagger(p_1[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}])$$
$$\stackrel{[Q/R]}{=} \dagger(p_1)[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}] \stackrel{p}{=} p[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$$

Recall $\dot{P} \mapsto \dot{p}$ (such that only $\dot{P}$ occurs as a process reference in $\dot{p}$—see Section 2) and $\text{SPLIT}_\Xi(\dot{P},A,w) \mapsto \text{split}_\Xi(\dot{p},A,w)$. To establish the equality of $\dot{p}$ and $\text{split}_\Xi(\dot{p},A,w)$, i.e., $\dot{p}[\text{SPLIT}_\Xi(\dot{P},A,w)/\dot{P}]$, we must show that there exists a process operator Φ of which $\dot{P}$ and $\text{SPLIT}_\Xi(\dot{P},A,w)$ are fixed points (see also Section 9.6 in [9]). Let $\Phi = \lambda Z \bullet \dot{p}[Z/\dot{P}]$. It follows that $\dot{P} = \Phi\dot{P}$ and that $\text{SPLIT}_\Xi(\dot{P},A,w) = \Phi\text{SPLIT}_\Xi(\dot{P},A,w)$.

Figure 9: Proof of Theorem 3

We define the set of the synchronous regions of the connector modeled by a process p as

$$\mathscr{X} = \bigcup_{a \in \text{Acts}(p)}\{X_a\}$$

and the set containing its asynchronous regions as

$$\mathscr{Y} = \{(a,b) \mid connected(a,b) \text{ and } a \in X \text{ and } b \in X' \text{ and } X \neq X' \text{ and } X,X' \in \mathscr{X}\},$$

where *connected*(a,b) denotes that the ends a and b belong to the same channel.

5.2 Splitting Connectors

We set out to establish the soundness of splitting connectors along the boundaries of their (a)synchronous regions. However, we can split any (syntactically τ-free) process along any set of actions by Theorem 3. This suggests that regardless of its (a)synchronous regions, one can split a connector in any possible way *and* preserve its original semantics. While true in theory, there is a catch for implementations of splitted connectors in practice: the parallel composition of the isolation and the coisolation of a connector process must synchronize, represented by the ? operator in Definition 3. Depending on the particular implementation approach, which in turn may depend on the underlying hardware architecture (see Section 1), performing ? at run-time may cost an unreasonable amount of resources, if possible at all. Next, we demonstrate that arbitrary splitting, therefore, makes no sense in practice despite its theoretical validity. Splitting based on (a)synchronous regions, in contrast, does.

We start with an example of splitting based on (a)synchronous regions. Suppose that we split fifo1$\langle a,b \rangle$ into two parts: one part contains only a, while the other part contains only b. Recall from Section 3 that the semantics of this channel is given by the process definition $Fifo1\langle a;b \rangle \mapsto a \cdot b \cdot Fifo1\langle a;b \rangle$. Splitting along $\{a\}$ (or equivalently, along $\{b\}$) yields:

$$
\begin{aligned}
\mathsf{SPLIT}_\Xi(Fifo1\langle a;b\rangle, \{a\}, \varepsilon) &\mapsto \mathsf{split}_\Xi(a \cdot b \cdot Fifo1\langle a;b\rangle, \{a\}, \varepsilon) \\
&= \mathsf{split}_\Xi(a \cdot b, \{a\}, \varepsilon) \cdot \mathsf{split}_\Xi(Fifo1\langle a;b\rangle, \{a\}, \varepsilon) \\
&= \mathord{?}_\Xi(a \sqcup \xi_\varepsilon(a) \cdot \quad \overline{\xi}_\varepsilon(b) \parallel \\
&\qquad\quad \overline{\xi}_\varepsilon(a) \cdot b \sqcup \xi_\varepsilon(b) \quad) \cdot \mathsf{SPLIT}_\Xi(Fifo1\langle a;b\rangle, \{a\}, \varepsilon))
\end{aligned}
$$

with $\Xi = (Fifo1\langle a;b\rangle \mapsto a \cdot b \cdot Fifo1\langle a;b\rangle, \{a,b\}, \mathtt{tau}, \xi, \overline{\xi})$. Here, ? represents the asynchronous region of fifo1$\langle a;b \rangle$. Suppose that we want to implement $p = a \sqcup \xi_\varepsilon(a) \cdot \overline{\xi}_\varepsilon(b)$ and $q = \overline{\xi}_\varepsilon(a) \cdot b \sqcup \xi_\varepsilon(b)$ such that, when run in parallel, they behave as $a \cdot b$. These implementations should perform the synchronization implied by ?. Recall from Section 4 that intuitively, $\xi_\varepsilon(a)$ represents the act of "disseminating the performance of a," while $\overline{\xi}_\varepsilon(a)$ represents the act of "discovering the performance of a." Thus, the implementation of p should: (1) accept data on a and disseminate this acceptance, and (2) discover the dispersal of data on b. Meanwhile, the implementation of q should: (1) discover the acceptance of data on a, and (2) dispense data on b and disseminate this dispersal. Thus, in each step, the implementations of p and q require only unidirectional communication about their behavior to synchronize: first, the implementation of p performs $\xi_\varepsilon(a)$ and the implementation of q takes notice of this (by performing $\overline{\xi}_\varepsilon(a)$); afterwards, p and q switch roles to perform $\overline{\xi}_\varepsilon(b)$ and $\xi_\varepsilon(b)$. This shows that synchronous regions can decide on their behavior independently of each other: the region $\{a\}$ does not need to know that the region $\{b\}$ dispenses data before it can accept data—it can decide to do so without communication.

In practice, this can yield performance improvements: although the isolation and the coisolation of a process p have the same transition system modulo transition labels, benefits can arise if one composes them in parallel with *another* split process q. In that case, there may exist a transition t of the (co)isolation of p that can proceed independently—without communication among the ends involved—of a transition t' of the (co)isolation of q. Without splitting, in contrast, communication among those ends must always take place to decide on whether to behave according to t, t', or both. For instance, if we put two split fifo1 instances in sequence (as in Figure 1a), the source end a of the first fifo1 can proceed independently of the sink end b of the second fifo1. This means that, if empty, the first fifo1 can accept a data item on a (and place it in its buffer) without communicating with b. Similarly, if full, the second fifo1 can dispense a data item on b (and remove it from its buffer) without communicating with a. In contrast, if we put two unsplit fifo1 instances in sequence, the source end a and the sink end b communicate with each other to decide on their joint behavior, even though the behavior of those ends does not depend on each other. By splitting, one avoids this unnecessary communication, reducing resource consumption at runtime.

To demonstrate that splitting arbitrarily makes no sense, suppose that we split sync$\langle a,b \rangle$ into two parts: one part contains only a, while the other part contains only b. Recall from Section 3 that the semantics of this channel is given by the process definition $Sync\langle a;b \rangle \mapsto a \sqcup b \cdot Sync\langle a;b \rangle$. Splitting along $\{a\}$ (or equivalently, along $\{b\}$) yields:

$$
\begin{aligned}
\mathsf{SPLIT}_\Xi(Sync\langle a;b\rangle, \{a\}, \varepsilon) &\mapsto \mathsf{split}_\Xi(a \sqcup b \cdot Sync\langle a;b\rangle, \{a\}, \varepsilon) \\
&= \mathsf{split}_\Xi(a \sqcup b, \{a\}, \varepsilon) \cdot \mathsf{split}_\Xi(Sync\langle a;b\rangle, \{a\}, \varepsilon) \\
&= \mathord{?}_\Xi(a \sqcup \xi_\varepsilon(a) \sqcup \quad \overline{\xi}_\varepsilon(b) \parallel \\
&\qquad\quad \overline{\xi}_\varepsilon(a) \sqcup b \sqcup \xi_\varepsilon(b) \quad) \cdot \mathsf{SPLIT}_\Xi(Sync\langle a;b\rangle, \{a\}, \varepsilon)
\end{aligned}
$$

with $\Xi = (Sync\langle a;b\rangle \mapsto a \sqcup b \cdot Sync\langle a;b\rangle, \{a,b\}, \mathtt{tau}, \xi, \overline{\xi})$. Now, similar to the previous example, suppose that we want to implement $p = a \sqcup \xi_\varepsilon(a) \sqcup \overline{\xi}_\varepsilon(b)$ and $q = \overline{\xi}_\varepsilon(a) \sqcup b \sqcup \xi_\varepsilon(b)$ such that, when run

in parallel, they behave as $a \sqcup b$. As before, these implementations should perform the synchronization implied by ?. Thus, the implementation of p should accept data on a, disseminate this acceptance, and discover the dispersal of data on b. Meanwhile, the implementation of q should discover the acceptance of data on a, dispense data on b, and disseminate this dispersal. All of these actions must occur at the same time. This means that, in contrast to our previous example, the implementations of p and q must engage in *bi*directional communication with each other about the acceptance of data on a and the dispersal of data on b. This suggests that the two ends of $\mathsf{sync}\langle a, b \rangle$ must synchronize with each other—they belong to the same synchronous region and cannot decide on their behavior independently—making it unreasonable to split them in the first place: the communication necessary to realize the synchronization necessary inflicts overhead, making it more attractive to run the original $\mathsf{sync}\langle a, b \rangle$ without splitting.

Depending on the hardware architecture, one can implement unidirectional communication efficiently; we sketch an implementation of the split $\mathsf{fifo1}\langle a, b \rangle$ on a shared memory machine with multi-threading. First, we instantiate two threads, A and B, for the processes $p = a \sqcup \xi_\varepsilon(a) \cdot \overline{\xi}_\varepsilon(b)$ and $q = \overline{\xi}_\varepsilon(a) \cdot b \sqcup \xi_\varepsilon(b)$. Every multi-action α translates to the atomic execution of a block of code representing the actions occurring in α. We implement the action $\xi_\varepsilon(a)$ as setting a shared Boolean flag and the action $\overline{\xi}_\varepsilon(a)$ as waiting for the value of this flag to change. Once the latter happens, thread B unsets the flag and knows that thread A has accepted data from a. Subsequently, it can dispense the data on b and set another shared flag for the actions $\xi_\varepsilon(b)$ and $\overline{\xi}_\varepsilon(b)$. In general, rather than simple Boolean flags, threads can share more complex data structures to keep track of which actions they have performed.

Now, suppose that $\mathsf{fifo1}\langle a, b \rangle$ constitutes some arbitrarily large connector with a distributed implementatation across multiple machines in a network. In the standard distributed approach (see Section 1), the implementation of $\mathsf{fifo1}\langle a, b \rangle$ has to share information with each of its neighbors in every step. We can reduce the amount of communication necessary for this sharing (and improve performance) by using the implementation of the split $\mathsf{fifo1}\langle a, b \rangle$ as described above (under the assumption that the machine on which we run this implementation features multi-threading and shared memory). The validity of doing this follows from Theorem 3: $\partial_B(\Gamma_C(\cdots \| \mathit{Fifo1}\langle a, b \rangle \| \cdots)) \simeq \partial_B(\Gamma_C(\cdots \| \mathsf{SPLIT}_\Xi(\mathit{Fifo1}\langle a, b \rangle) \| \cdots))$.

6 Future Work

We identify three main directions for future work.

- Implementing the splitting procedure to facilitate automatic splitting of processes, as well as a tool for the automatic detection of (a)synchronous regions of Reo connectors. Combined, they allow us to mechanically split connectors along their (a)synchronous regions. We can then integrate this in one of the code generation frameworks currently under development for Reo.

- Extending the splitting procedure to full mCRL2, including data and time. We see no fundamental difficulties along this path, although we expect the technical details and proofs to involve rather cumbersome derivations.

- Investigating other ways of splitting processes. The procedure we introduced in this paper splits processes in a synchronous manner, meaning that the action $\xi_w(a)$ occurs at the same time as the action a itself. We imagine at least two other ways of splitting processes. In one approach, $\xi_w(a)$ occurs after a but before the next action. Then, the process $q = a \cdot b$ has $a \cdot \xi_w(a) \cdot \overline{\xi}_w(b)$ as its $\{a\}$-isolation (instead of $a \sqcup \xi_w(a) \cdot \overline{\xi}_w(b)$. In another approach, $\xi_w(a)$ occurs after a but possibly concurrently with the next action. Then, q has $a \cdot (\xi_w(a) \| \overline{\xi}_w(b))$ as its isolation. We speculate that these splitting approaches are sound only under equivalences weaker than strong bisimulation.

This line of research seems related to existing work on delay-insensitive circuits (e.g., [20]) and desynchronization (e.g., [3, 7]), the derivation of an asynchronous system from a synchronous system: for the class of *desynchronizable systems*, the original synchronous system and the newly constructed asynchronous system are semantically equivalent. If we use the splitting procedure presented in our paper to obtain such an original synchronous system, we may use—perhaps with modifications—results from desynchronization for our splitting purpose.

References

[1] Farhad Arbab (2004): *Reo: a channel-based coordination model for component composition*. *MSCS* 14(3), pp. 329–366, doi:10.1017/S0960129504004153.

[2] Christel Baier, Marjan Sirjani, Farhad Arbab & Jan Rutten (2006): *Modeling component connectors in Reo by constraint automata*. *SCICO* 61(2), pp. 75–113, doi:10.1016/j.scico.2005.10.008.

[3] Harsh Beohar & Pieter Cuijpers (2010): *A theory of desynchronisable closed loop systems*. *EPTCS* 38(1), pp. 99–114, doi:10.4204/EPTCS.38.10.

[4] Jan Bergstra & Jan Willem Klop (1984): *Process Algebra for Synchronous Communication*. *Information and Control* 60(1–3), pp. 109–137, doi:10.1016/S0019-9958(84)80025-X.

[5] Dave Clarke, David Costa & Farhad Arbab (2007): *Connector colouring I: Synchronisation and context dependency*. *SCICO* 66(3), pp. 205–225, doi:10.1016/j.scico.2007.01.009.

[6] Dave Clarke & José Proença (2012): *Partial Connector Colouring*. In Marjan Sirjani, editor: *Coordination Models and Languages*, *LNCS* 7274, Springer, pp. 59–73, doi:10.1007/978-3-642-30829-1_5.

[7] Clemens Fischer and Wil Janssen (1996): *Synchronous development of asynchronous systems*. In Ugo Montanari & Vladimiro Sassone, editors: *CONCUR*, *LNCS* 1119, Springer, pp. 735–750, doi:10.1007/3-540-61604-7_87.

[8] Jan Friso Groote, Aad Mathijssen, Michel Reniers, Yaroslav Usenko & Muck van Weerdenburg (2007): *The Formal Specification Language mCRL2*. In: *Proceedings of MMOSS 2007*, pp. 1–34. Available at `http://drops.dagstuhl.de/opus/volltexte/2007/862/`.

[9] Jan Friso Groote & Mohammad Reza Mousavi (2010): *Modelling and Analysis of Communicating Systems*. Available at `http://www.win.tue.nl/~jfg/educ/2IW26/herfst2011/mcrl2-book.pdf`. To appear.

[10] Sung-Shik Jongmans & Farhad Arbab (2012): *Overview of Thirty Semantic Formalisms for Reo*. *SACS* 22(1), pp. 201–251, doi:10.7561/SACS.2012.1.201.

[11] Sung-Shik Jongmans, Dave Clarke & José Proença (2012): *A Procedure for Splitting Processes and its Application to Coordination (Technical Report)*. Technical Report SEN-1203, CWI.

[12] Sung-Shik Jongmans, Christian Krause & Farhad Arbab (2011): *Encoding Context-Sensitivity in Reo into Non-Context-Sensitive Semantic Models*. In Wolfgang de Meuter & Gruia-Catalin Roman, editors: *COORDINATION*, *LNCS* 6721, Springer, pp. 31–48, doi:10.1007/978-3-642-21464-6_3.

[13] Natallia Kokash, Christian Krause & Erik de Vink (2010): *Data-Aware Design and Verification of Service Compositions with Reo and mCRL2*. In: *Proceedings of SAC 2010*, pp. 2406–2413, doi:10.1145/1774088.1774590.

[14] Natallia Kokash, Christian Krause & Erik de Vink (2010): *Time and Data-Aware Analysis of Graphical Service Models in Reo*. In: *Proceedings of SEFM 2010*, pp. 125–134, doi:10.1109/SEFM.2010.26.

[15] Natallia Kokash, Christian Krause & Erik de Vink (2010): *Verification of Context-Dependent Channel-Based Service Models*. In Frank de Boer, Marcello Bonsangue, Stefan Hallerstede & Michael Leuschel, editors: *FMCO*, *LNCS* 6286, Springer, pp. 21–40, doi:10.1007/978-3-642-17071-3_2.

[16] Natallia Kokash, Christian Krause & Erik de Vink (2012): *Reo+mCRL2: A framework for model-checking dataflow in service compositions*. *FAC* 24(2), pp. 187–216, doi:10.1007/s00165-011-0191-6.

$$\mathscr{C}_C(\alpha) \;=\; \begin{cases} \mathscr{C}_{C_1}(\mathscr{C}_{C_2}(\alpha)) & \text{if } C = C_1 \cup C_2 \text{ and } C_1 \neq \emptyset \text{ and } C_2 \neq \emptyset \\ b \sqcup \mathscr{C}_C(\alpha \setminus \beta) & \text{if } C = \{\beta \rightarrow b\} \text{ and } \beta \sqsubseteq \alpha \\ \alpha & \text{otherwise} \end{cases}$$
for $C \subseteq \mathbb{MA}ct \times \mathbb{A}ct$ a set of communications

$$\ltimes(p) \;=\; \begin{cases} \{\alpha\} & \text{if } p = \alpha \\ \emptyset & \text{if } p \in \{\tau, \delta\} \\ \ltimes(q) \cup \ltimes(r) & \text{if } p = q \oplus r \text{ with } \oplus \in \{+, \cdot\} \end{cases}$$

$$\Downarrow(V) \;=\; \{\beta \sqsubseteq \alpha \,|\, \alpha \in V\} \setminus \{\tau\}$$
for $V \subseteq \mathbb{MA}ct$ a set of multi-actions

$$\mathsf{dom}(C) \;=\; \begin{cases} \mathsf{dom}(C_1) \cup \mathsf{dom}(C_2) & \text{if } C = C_1 \cup C_2 \text{ and } C_1 \neq \emptyset \text{ and } C_2 \neq \emptyset \\ \{\beta\} & \text{if } C = \{\beta \rightarrow b\} \end{cases}$$
for $C \subseteq \mathbb{MA}ct \times \mathbb{A}ct$ a set of communications

Figure 10: Auxiliary functions.

[17] Robin Milner & Faron Moller (1993): *Unique decomposition of processes*. *TCS* 107(2), pp. 357–363, doi:10.1016/0304-3975(93)90176-T.

[18] José Proença, Dave Clarke, Erik de Vink & Farhad Arbab (2012): *Dreams: a framework for distributed synchronous coordination*. In: *Proceedings of SAC 2012*, pp. 1510–1515, doi:10.1145/2245276.2232017.

[19] José Proença (2011): *Synchronous Coordination of Distributed Components*. Ph.D. thesis, Leiden University. Available at `http://hdl.handle.net/1887/17624`.

[20] Jan Tijmen Udding (1984): *Classification and Composition of Delay-Insensitive Circuits*. Ph.D. thesis, Eindhoven University of Technology. Available at `http://repository.tue.nl/25052`.

A   Axiomatization

Every process has an associated transition system describing its semantics (see [8] for the SOS rules). Let $\simeq$ denote equality of processes. Figure 11 shows a sound and complete axiomatization—for strong bisimulation—of the operators shown in Figure 2. Figure 11b axiomatizes two additional operators on multi-actions. Informally, the operator $\setminus$ subtracts the multi-action on its right-hand side from the multi-action on its left-hand side; the operator $\sqsubseteq$ checks if the multi-action on its right-hand side contains the multi-action on its left-hand side.

The axioms C1 and CL1 in Figure 11d refer to several auxiliary functions; Figure 10 shows their definitions. The function $\mathscr{C}_C$ applies the communications in a set C to a multi-action. The function $\ltimes$ maps a basic process p to its *alphabet*, i.e., the multi-actions that occur in p. The function $\Downarrow$ maps a set of multi-actions V to those nonempty multi-actions contained in at least one multi-action in V. Finally, the function dom maps a set of communications C to their domains.

MA1	$\alpha \sqcup \beta \simeq \beta \sqcup \alpha$
MA2	$(\alpha \sqcup \beta) \sqcup \gamma \simeq \alpha \sqcup (\beta \sqcup \gamma)$
MA3	$\alpha \sqcup \tau \simeq \alpha$
A1	$p + q \simeq q + p$
A2	$p + (q + r) \simeq (p + q) + r$
A3	$p + p \simeq p$
A4	$(p + q) \cdot r \simeq p \cdot r + q \cdot r$
A5	$(p \cdot q) \cdot r \simeq p \cdot (q \cdot r)$
A6	$p + \delta \simeq p$
A7	$\delta \cdot p \simeq \delta$

(a) Axioms for multi-actions and for the basic operators.

MD1	$\tau \setminus \alpha \simeq \tau$
MD2	$\alpha \setminus \tau \simeq \alpha$
MD3	$\alpha \setminus (\beta \sqcup \gamma) \simeq (\alpha \setminus \beta) \setminus \gamma$
MD4	$(a \sqcup \alpha) \setminus a \simeq \alpha$
MD5	$(a \sqcup \alpha) \setminus b \simeq a \sqcup (\alpha \setminus b)$ if $a \neq b$
MS1	$\tau \sqsubseteq \alpha \simeq true$
MS2	$\alpha \sqsubseteq \tau \simeq false$ if $\alpha \not\simeq \tau$
MS3	$a \sqcup \alpha \sqsubseteq a \sqcup \beta \simeq \alpha \sqsubseteq \beta$
MS4	$a \sqcup \alpha \sqsubseteq b \sqcup \beta \simeq$
	$\quad a \sqcup (\alpha \setminus b) \sqsubseteq \beta$ if $a \neq b$

(b) More axioms for multi-actions.

M	$p \parallel q \simeq p \Vert q + q \Vert p + p \mid q$
LM1	$\alpha^\delta \Vert p \simeq \alpha^\delta \cdot p$
LM2	$\delta \Vert p \simeq \delta$
LM3	$\alpha \cdot p \Vert q \simeq \alpha \cdot (p \parallel q)$
LM4	$(p + q) \Vert r \simeq p \Vert r + q \Vert r$
S1	$p \mid q \simeq q \mid p$
S2	$(p \mid q) \mid r \simeq p \mid (q \mid r)$
S3	$p \mid \tau \simeq p$
S4	$\alpha^\delta \mid \delta \simeq \delta$
S5	$(\alpha^\delta \cdot p) \mid \beta^\delta \simeq \alpha^\delta \mid \beta^\delta \cdot p$
S6	$(\alpha^\delta \cdot p) \mid (\beta^\delta \cdot q) \simeq \alpha^\delta \mid \beta^\delta \cdot (p \parallel q)$
S7	$(p + q) \mid r \simeq p \mid r + q \mid r$
SMA	$\alpha \mid \beta \simeq \alpha \sqcup \beta$

(c) Axioms for the parallel operators.

V1	$\nabla_V(\alpha) \simeq \alpha$ if $\alpha \in V \cup \{\tau\}$
V2	$\nabla_V(\alpha) \simeq \delta$ if $\alpha \notin V \cup \{\tau\}$
B1	$\partial_B(\tau) \simeq \tau$
B2	$\partial_B(a) \simeq a$ if $a \notin B$
B3	$\partial_B(a) \simeq \delta$ if $a \in B$
B4	$\partial_B(\alpha \mid \beta) \simeq \partial_B(\alpha) \mid \partial_B(\beta)$
R1	$\rho_R(\tau) \simeq \tau$
R2	$\rho_R(a) \simeq b$ if $a \to b \in R$ for some b
R3	$\rho_R(a) \simeq a$ if $a \to b \notin R$ for all b
R4	$\rho_R(\alpha \mid \beta) \simeq \rho_R(\alpha) \mid \rho_R(\beta)$
C1	$\Gamma_C(\alpha) \simeq \mathscr{C}_C(\alpha)$
CL1	$\Gamma_C(p) \simeq p$ if $\Downarrow(\ltimes(p)) \cap dom(C) = \emptyset$

(d) Axioms for the additional operators.

For all $\dagger \in \{\nabla_V, \partial_B, \rho_R, \Gamma_C\}$,

V3, B5, R5, C2	$\dagger(\delta) \simeq \delta$
V4, B6, R6, C3	$\dagger(\alpha + \beta) \simeq \dagger(\alpha) + \dagger(\beta)$
V5, B7, R7, C4	$\dagger(\alpha \cdot \beta) \simeq \dagger(\alpha) \cdot \dagger(\beta)$

(e) More axioms for the additional operators.

H1	$\mathscr{T}_I(\tau) \simeq \tau$
H2	$\mathscr{T}_I(a) \simeq \tau$ if $a \in I$
H3	$\mathscr{T}_I(a) \simeq a$ if $a \notin I$
H4	$\mathscr{T}_I(\alpha \mid \beta) \simeq \mathscr{T}_I(\alpha) \mid \mathscr{T}_I(\beta)$
H5	$\mathscr{T}_I(\delta) \simeq \delta$
H6	$\mathscr{T}_I(\alpha + \beta) \simeq \mathscr{T}_I(\alpha) + \mathscr{T}_I(\beta)$
H7	$\mathscr{T}_I(\alpha \cdot \beta) \simeq \mathscr{T}_I(\alpha) \cdot \mathscr{T}_I(\beta)$

(f) Axioms for the abstraction operator.

Figure 11: Axioms.