Centrum voor Wiskunde en Informatica

REPORTRAPPORT

SEN

Software Engineering

Software ENgineering

Coordination models Orc and Reo compared

J.M.P. Proença, D.G. Clarke

REPORT SEN-R0802 MAY 2008

Centrum voor Wiskunde en Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

Coordination models Orc and Reo compared

ABSTRACT
Orc and Reo are two complementary approaches to the problem of coordinating components or services. On one hand, Orc is highly asynchronous, dynamic, and based on ephemeral connections to services. On the other hand, Reo is based on the interplay between synchronization and mutual exclusion, is more static, and establishes more continuous connections between components or services. The question of how Orc and Reo relate to each other naturally arises. In this paper, we present a detailed comparison of the two models. We demonstrate that embedding non-recursive Orc expressions into Reo connectors is straightforward, whereas recursive Orc expressions require an extension to the Reo model. For the other direction, we argue that embedding Reo into Orc would require, based on expressiveness results of Palamidessi, signifficantly more effort. We conclude with some general observations and comparisons between the two approaches.

1998 ACM Computing Classification System: D.3.1;D.3.2;F.1.1;F.1.2;F.3.2
Keywords and Phrases: Reo; Orc; Coordination languages; Software connectors

Coordination Models $\mathcal{O}$rc and $\mathcal{R}$eo Compared

José Proença[*,†] Dave Clarke[†]

April 29, 2008

Abstract

$\mathcal{O}$rc and $\mathcal{R}$eo are two complementary approaches to the problem of coordinating components or services. On one hand, $\mathcal{O}$rc is highly asynchronous, dynamic, and based on ephemeral connections to services. On the other hand, $\mathcal{R}$eo is based on the interplay between synchronization and mutual exclusion, is more static, and establishes more continuous connections between components or services. The question of how $\mathcal{O}$rc and $\mathcal{R}$eo relate to each other naturally arises. In this paper, we present a detailed comparison of the two models. We demonstrate that embedding non-recursive $\mathcal{O}$rc expressions into $\mathcal{R}$eo connectors is straightforward, whereas recursive $\mathcal{O}$rc expressions require an extension to the $\mathcal{R}$eo model. For the other direction, we argue that embedding $\mathcal{R}$eo into $\mathcal{O}$rc would require, based on expressiveness results of Palamidessi, significantly more effort. We conclude with some general observations and comparisons between the two approaches.

1 Introduction

Although the field of coordination languages and models has been around for some time, the recent interest in Service-oriented Computing (SoC) and Web-service choreography and orchestration[1] has precipitated greater interest in the field, resulting in both new models and new application domains for existing models. Service-oriented Computing is based on the idea that software is composed of services which reside on third-party machines [SH05]. Web services are a common realization of this idea [Cer02]. Since the conception of SoC, research has focussed on developing languages to compose or coordinate services into either composite services or complete applications.

Coordination languages and models are based on the philosophy that an application or system should be divided into the parts that perform computation and the parts that coordinate the results and resources required to perform the computations. The original coordination language, Linda [Gel85], played only a passive rôle in coordination, by providing a blackboard (tuple space) which data can be written to and read from. Since then many coordination models have been proposed [PA98, AHM96], and the trend is towards developing models that play a more active rôle in the coordination process. Two recent and interesting active coordination models, $\mathcal{O}$rc [CM07] and $\mathcal{R}$eo [Arb04], sit diametrically opposite of each other in their approaches to coordinating services or components. This paper sets out to explore these models in detail.

$\mathcal{O}$rc is a simple orchestration language designed by Misra and Cook [CM07], based on three connectives and the simple notion of a site call to model computations—the external actions to be orchestrated. Central to $\mathcal{O}$rc's design is the idea that accessing (web) sites is an asynchronous activity which can fail, and so the connectives are designed to be asynchronous and not susceptible to failure.

$\mathcal{R}$eo is a channel-based coordination language designed by Arbab [Arb04] that is based on a simple notion of channel composition. It differs from existing models in that composition of connectors from channels propagates synchronization and exclusion constraints through connectors. In combination with stateful channels, an expressive coordination language emerges.

[1]Supported by FCT grant 22485 – 2005, Portugal.
[2]Email: Jose.Proenca@cwi.nl D.G.Clarke@cwi.nl
[1]We take the words choreography and orchestration to fall under the more general notion of coordination.

This paper presents a comparison of $\mathcal{O}$rc and $\mathcal{R}$eo. By choosing two coordination languages at different ends of the spectrum for our comparison, we hope to gain insight into the design choices and the advantages and disadvantages of various approaches. In the long run, we should hope for a synthesis of their key ideas, to get the best of both worlds. We present a number of examples, compare features and the underlying philosophies and design choices, and formally embed $\mathcal{O}$rc into $\mathcal{R}$eo. We also discuss the difficulties of the embedding in the other direction, referring to known results about encoding the asynchronous π-calculus into the synchronous π-calculus. The main conclusions that we make from this work is that neither full $\mathcal{O}$rc can be encoded into $\mathcal{R}$eo, nor can $\mathcal{R}$eo be encoded into $\mathcal{O}$rc. We start by explaining informally why infinite $\mathcal{O}$rc expressions, defined using recursion, cannot be encoded into $\mathcal{R}$eo. We then define $\mathcal{O}$rc$^-$ as the subset of $\mathcal{O}$rc without recursion, and provide an encoding from $\mathcal{O}$rc$^-$ to $\mathcal{R}$eo, that we prove to be correct. This encoding is the main contribution of this paper. To show that the synchronous behaviour of $\mathcal{R}$eo cannot be encoded into $\mathcal{O}$rc, we use previous results about the π-calculus due to Palamidessi [Pal97], by encoding $\mathcal{O}$rc into the asynchronous π-calculus, and referring to the symmetric leader election problem. Here we do not formally prove the correctness of this last encoding.

Section 2 describes our encoding of $\mathcal{O}$rc into $\mathcal{R}$eo. Section 3 presents our argument that the other direction is not possible, in general. Section 4 compares the two models on a variety of points. Section 5 discusses some related work, and Section 6 concludes and discusses future work. But first, we introduce $\mathcal{O}$rc and $\mathcal{R}$eo, and give small examples.

1.1 $\mathcal{O}$rc

In this section we present $\mathcal{O}$rc's syntax and reduction semantics, and give simple examples of $\mathcal{O}$rc expressions. Work by Misra and others describes $\mathcal{O}$rc's semantics in more detail [CM07, KCM06].

$\mathcal{O}$rc expressions have the following syntax, where E is an expression name, M is a site name, x is a variable, v is a constant value, and $\overline{p}$ is a tuple of p's:

$$\begin{array}{rcl} f, g \in \mathit{Expr} & ::= & \mathbf{0} \;\mid\; M(\overline{p}) \;\mid\; E(\overline{p}) \;\mid\; f > x > g \;\mid\; f \,|\, g \;\mid\; f \textbf{ where } x :\in g \\ p \in \mathit{Actual} & ::= & x \;\mid\; v \\ \mathit{Definition} & ::= & E(\overline{x}) \stackrel{\text{def}}{=} f \end{array}$$

An $\mathcal{O}$rc program consists of an $\mathcal{O}$rc expression together with a set of definitions. Basic services, such as data manipulation, are assumed to be provided by primitive *sites*. An $\mathcal{O}$rc expression can be a primitive site call, a reference to another $\mathcal{O}$rc expression, or a composition of $\mathcal{O}$rc expressions. The computational model underlying $\mathcal{O}$rc consists on a number of expressions running in parallel, which ultimately call sites. Each of these sites may publish a result, that can be discarded or used in other expressions.

A site call is written $M(\overline{p})$, where $\overline{p}$ is a tuple of arguments, which can be constants or variables. During execution all variables have to be instantiated, that is, evaluation is strict, and the site returns at most one value. Example primitive sites include $\mathbf{0}$, a special primitive included in $\mathcal{O}$rc's syntax grammar which never responds, and $let(v)$, which responds value v. We use E to range over possibly recursive definitions of $\mathcal{O}$rc expressions.

Three combinators exist for composing expressions f and g: symmetric composition, written $f \mid g$; sequential composition, written $f > x > g$; and asymmetric composition, written f **where** $x :\in g$. The combinator $f \mid g$ calls f and g simultaneously and executes them in parallel. The values that it can publish are exactly all the values that f and g can publish. The sequential composition $f > x > g$ starts by calling f, and for each published value by f, a new thread of g is executed. The variable x is bound to each value published by f in the corresponding thread of g. The values published by $f > x > g$ consist of the values published by all threads of g. The last operator f **where** $x :\in g$ calls f and g in parallel, replacing x in f by the first published value of g. All the subsequent values published by g are discarded. This operator publishes only the values published by f.[2]

In the rest of this section we show some examples, borrowed from Kitchin *et al.* [KCM06], to provide a better understanding about the semantics of $\mathcal{O}$rc. In the end we formally present $\mathcal{O}$rc's semantics.

[2] These operations appear to be closely related to Friedman and Wise's *frons* construct [FW80].

Sequential vs asymmetric parallel composition Consider the following $\mathcal{O}\mathsf{rc}$ expressions:

$$\begin{aligned} \mathit{EmailNews}(d) &\stackrel{\text{def}}{=} (\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)) > x > \mathit{email}(me, x) \\ \mathit{EmailNewsOnce}(d) &\stackrel{\text{def}}{=} \mathit{email}(me, x) \textbf{ where } x :\in (\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)) \end{aligned}$$

Here, uk and me are constant values, x and d are variables, and $\mathit{CNN}(uk, d)$, $\mathit{BBC}(uk)$ and $\mathit{email}(me, x)$ are site calls that retrieve the news for the UK on the day d from CNN, retrieve the news for the UK from BBC for today, and send an email to me with value x. Thus $\mathit{EmailNews}(d)$ and $\mathit{EmailNewsOnce}(d)$ invoke the news service from CNN and BBC and send the resulting content by e-mail to me. The difference between these two expressions is that *EmailNews* sends the news from both CNN and BBC (when the services reply), while *EmailNewsOnce* mails only the value of the first reply, ignoring the second reply.

Time-out Let $\mathit{Rtimer}(t)$ be a site that, when called, waits t time units before publishing a signal. Using this site, we can express a call to a site M that can only wait t time units for its result using the following $\mathcal{O}\mathsf{rc}$ expression.

$$\mathit{let}(z) \textbf{ where } z :\in \big(M > x > \mathit{let}(x, \mathit{true}) \mid \mathit{Rtimer}(t) > x > \mathit{let}(x, \mathit{false})\big)$$

In this example *true* and *false* are constants that indicate whether M succeeded in publishing a value or not. When M is faster to publish a value than $\mathit{Rtimer}(t)$, then the full expression publishes the tuple (x, true), where x is the value published by M. Otherwise, it publishes (y, false), where y is the signal published by the timer site. In the semantics of $\mathcal{O}\mathsf{rc}$ that we present in the end of this Section we consider that, when both sides of the parallel composition are equally fast, then one is chosen non-deterministically.

Barrier Synchronization Consider the $\mathcal{O}\mathsf{rc}$ expressions $M > x > f$ and $N > y > g$. We can execute them in parallel, imposing that f and g are called at the same time, after both sites M and N have completed.

$$\big((\mathit{let}(u, v) \textbf{ where } u :\in M) \textbf{ where } v :\in N\big) > (x, y) > (f \mid g)$$

The two asymmetric parallel combinators join the results of the calls of M and N, and the result is forward to f and g via a single sequential composition combinator.

Recursion Infinite behaviour can be described using recursive definitions, as the following example shows.

$$\begin{aligned} \mathit{Metronome} &\stackrel{\text{def}}{=} \mathit{Signal} \mid (\mathit{Rtimer}(1) > x > \mathit{Metronome}) \\ \mathit{EmailNewsFrequently}(d) &\stackrel{\text{def}}{=} \mathit{Metronome} > x > \mathit{EmailNewsOnce}(d) \end{aligned}$$

Metronome is an $\mathcal{O}\mathsf{rc}$ expression that sends a signal published by *Signal* every time unit. The site $\mathit{Rtimer}(t)$, as in the time-out example, waits t time units before publishing a signal. Therefore, *EmailNewsFrequently* calls *EmailNewsOnce* every time unit, which in turn sends me an email from either *CNN* or *BBC*.

$\mathcal{O}\mathsf{rc}$'s semantics Instead of the standard, asynchronous semantics for $\mathcal{O}\mathsf{rc}$, we present a synchronous semantics which allows multiple events to occur at the same time. This approach enables a simpler formal comparison with $\mathcal{R}\mathsf{eo}$, without really changing the essence of $\mathcal{O}\mathsf{rc}$. The reduction rules for $\mathcal{O}\mathsf{rc}$ expressions have the form $f \xrightarrow{a} g$ and are presented below. Here a is a *set* of observations of base events, defined as follows:

$$\mathit{BaseEvent} ::= \tau \mid M_k(\overline{v}) \mid k?v \mid !v$$

We use the silent observation τ mainly to represent the binding of a variable to a value. $M_k(\overline{v})$ represents the call to site M, indexed by a fresh k and where $\overline{v}$ is a tuple of values used as argument. $k?v$ represents the return of value v by the site call indexed with k. $!v$ represents that a value v was published. Finally, we use a and b to range over sets of observations, following the convention that $\xrightarrow{\emptyset}$ denotes $\xrightarrow{\tau}$. The reduction rules are presented in Fig. 1.

$$\frac{k \text{ fresh}}{M(\overline{v}) \xrightarrow{M_k(\overline{v})} ?k} \ (\textsc{SiteCall}) \qquad \frac{}{?k \xrightarrow{k?v} let(v)} \ (\textsc{SiteRet}) \qquad \frac{}{let(v) \xrightarrow{!v} \mathbf{0}} \ (\textsc{Let})$$

$$\frac{f \xrightarrow{a} f'}{f \mid g \xrightarrow{a} f' \mid g} \ (\textsc{Sym1}) \qquad \frac{g \xrightarrow{a} g'}{f \mid g \xrightarrow{a} f \mid g'} \ (\textsc{Sym2}) \qquad \frac{f \xrightarrow{a} f' \quad g \xrightarrow{b} g'}{f \mid g \xrightarrow{a,b} f' \mid g'} \ (\textsc{Sym3})$$

$$\frac{g \xrightarrow{a} g'}{g \textbf{ where } x :\in f \xrightarrow{a} g' \textbf{ where } x :\in f} \ (\textsc{Asym1N}) \qquad \frac{f \xrightarrow{b} f' \quad !w \notin b}{g \textbf{ where } x :\in f \xrightarrow{b} g \textbf{ where } x :\in f'} \ (\textsc{Asym2})$$

$$\frac{g \xrightarrow{a} g' \quad f \xrightarrow{b} f' \quad !w \notin b}{g \textbf{ where } x :\in f \xrightarrow{a,b} g' \textbf{ where } x :\in f'} \ (\textsc{Asym3N}) \qquad \frac{f \xrightarrow{!v,b} f'}{g \textbf{ where } x :\in f \xrightarrow{\tau} [v/x].g} \ (\textsc{Asym1V})$$

$$\frac{g \xrightarrow{a} g' \quad f \xrightarrow{!v,b} f'}{g \textbf{ where } x :\in f \xrightarrow{a} [v/x].g'} \ (\textsc{Asym2V}) \qquad \frac{f \xrightarrow{!v_1,\ldots,!v_n,a} f' \quad !w \notin a \quad n \geq 0}{f >x> g \xrightarrow{a} (f' >x> g) \mid [v_1/x].g \mid .. \mid [v_n/x].g} \ (\textsc{Seq})$$

Figure 1: Operational semantics of $\mathcal{O}$rc

To describe the behaviour of a site call we introduce an extension to represent an intermediate state, $?k$, following [KCM06]. This denotes an instance of a site call that has not yet returned, and is used in rules (SiteCall) and (SiteRet). k is a fresh value used to uniquely identify the specific call. We also use the primitive site $let(v)$ as an intermediate state, to capture a site that has just returned value v, and will publish that value. For the case of the asymmetric composition g **where** $x :\in f$, five different rules were defined to distinguish the cases when only g is reduced, when only f is reduced (publishing or not a value), and the combination of both.

The reductions rules in Fig. 1 yield the following reduction of the expression *EmailNewsOnce* presented above. Here we assume that a and b are fresh, v is the value published by the *BBC* site, and v' is the value published by the *email* site.

$$\begin{array}{ll}
& email(me, x) \textbf{ where } x :\in (CNN(uk, d) \mid BBC(uk)) \\
\xrightarrow{BBC_a(uk)} & email(me, x) \textbf{ where } x :\in (CNN(uk, d) \mid ?a) \\
\xrightarrow{a?v} & email(me, x) \textbf{ where } x :\in (CNN(uk, d) \mid let(v)) \\
\xrightarrow{\tau} & email(me, v) \\
\xrightarrow{email_b(me,v)} & ?b \\
\xrightarrow{b?v'} & let(v') \\
\xrightarrow{!v'} & 0
\end{array}$$

1.2 $\mathcal{R}$eo

$\mathcal{R}$eo is a powerful coordination model based on channel composition. Channels impose synchronisation and other constraints on their ends. Behaviour arises from the propagation of these constraints through connectors formed by plugging channels together to form nodes, which themselves impose mutual exclusive data merging and synchronous data replication constraints. A key characteristic of $\mathcal{R}$eo is that synchrony and mutual exclusion constraints are propagated through composition. We present the semantics of $\mathcal{R}$eo connectors in an adaptation of the constraint automata model [BSAR06].

Firstly, we assume that connectors are defined over a denumerable set of node names, $\mathcal{N}ode$. Each connector C will have a set of input nodes $\boldsymbol{I} \subseteq \mathcal{N}ode$, and a (disjoint) set of output nodes, $\boldsymbol{O} \subseteq \mathcal{N}ode$. The input and output nodes of a connector define its *arity*, denoted $C : \boldsymbol{I} \rightarrow \boldsymbol{O}$. [3] We define $\mathcal{N}ames(C)$ to be $\boldsymbol{I} \cup \boldsymbol{O}$, which we call the boundary nodes of C.[4]

The semantics of a connector C is given as a reduction relation of the form $C \xrightarrow{N} C'$, where N is a (partial) map from the set of boundary nodes to the values that flow through those nodes. For example, we write $I(v), O(v)$ to denote the map from the boundary nodes I and O to the value v, and we write $\mathsf{nodes}(N)$ to denote the domain of N. We say that C

[3]For the purpose of this paper, we assume that primitive connectors are not plugged into themselves, *i.e.*, for a primitive with arity $\boldsymbol{I} \rightarrow \boldsymbol{O}$, we have that if $\boldsymbol{I} \rightarrow \boldsymbol{O}$ then $\boldsymbol{I} \cap \boldsymbol{O} = \emptyset$.

[4] We introduce a slightly different definition from the literature, where a boundary node is a node that is only an input or output node.

evolves to C' and fires nodes $\mathsf{nodes}(N)$. C' is the connector resulting from the particular step. Typically, C and C' will have the same primitives, just in different states. Table 1 presents some $\mathcal{R}$eo primitives, their arity, and axioms describing their behaviour. Each axiom gives a valid reduction of the corresponding primitive.

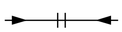
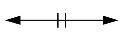
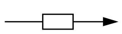
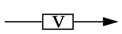
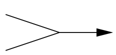
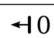
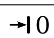
Visualisation	Representation	Arity	Axioms
	$\mathit{Sync}_{A,B}$	$A \to B$	$\mathit{Sync}_{A,B} \xrightarrow{A(v),B(v)} \mathit{Sync}_{A,B}$
	$\mathit{SDrain}_{A,B}$	$\{A,B\} \to \emptyset$	$\mathit{SDrain}_{A,B} \xrightarrow{A(v),B(w)} \mathit{SDrain}_{A,B}$
	$\mathit{SSpout}_{A,B}$	$\emptyset \to \{A,B\}$	$\mathit{SSpout}_{A,B} \xrightarrow{A(v),B(w)} \mathit{SSpout}_{A,B}$
	$\mathit{Lossy}_{A,B}$	$A \to B$	$\mathit{Lossy}_{A,B} \xrightarrow{A(v),B(v)} \mathit{Lossy}_{A,B}$ $\mathit{Lossy}_{A,B} \xrightarrow{A(v)} \mathit{Lossy}_{A,B}$
	$\mathit{ADrain}_{A,B}$	$\{A,B\} \to \emptyset$	$\mathit{ADrain}_{A,B} \xrightarrow{A(v)} \mathit{ADrain}_{A,B}$ $\mathit{ADrain}_{A,B} \xrightarrow{B(v)} \mathit{ADrain}_{A,B}$
	$\mathit{ASpout}_{A,B}$	$\emptyset \to \{A,B\}$	$\mathit{ASpout}_{A,B} \xrightarrow{A(v)} \mathit{ASpout}_{A,B}$ $\mathit{ASpout}_{A,B} \xrightarrow{B(v)} \mathit{ASpout}_{A,B}$
	$\mathit{FIFO1}_{A,B}$	$A \to B$	$\mathit{FIFO1}_{A,B} \xrightarrow{A(v)} \mathit{FIFO1}_{A,B}(v)$
	$\mathit{FIFO1}_{A,B}(v)$	$A \to B$	$\mathit{FIFO1}_{A,B}(v) \xrightarrow{B(v)} \mathit{FIFO1}_{A,B}$
	$\mathit{Merger}_{A,B,C}$	$\{A,B\} \to C$	$\mathit{Merger}_{A,B,C} \xrightarrow{A(v),C(v)} \mathit{Merger}_{A,B,C}$ $\mathit{Merger}_{A,B,C} \xrightarrow{B(v),C(v)} \mathit{Merger}_{A,B,C}$
	$0\text{-}\mathit{out}_A$	$\emptyset \to A$	–
	$0\text{-}\mathit{in}_A$	$A \to \emptyset$	–
	$\mathit{1Drain}_A$	$A \to \emptyset$	$\mathit{1Drain}_A \xrightarrow{A(v)} \mathit{1Drain}_A$

Table 1: Arity and behaviour of some $\mathcal{R}$eo primitives

The composition of connectors C and C' is denoted by $C * C'$. Well-formedness of the composition and the calculation of its arity is given by the following rule:

$$\frac{C : \boldsymbol{I} \to \boldsymbol{O} \qquad C' : \boldsymbol{I'} \to \boldsymbol{O'} \qquad \boldsymbol{I''} \stackrel{\text{def}}{=} \boldsymbol{I} \cup \boldsymbol{I'} \quad \boldsymbol{O''} \stackrel{\text{def}}{=} \boldsymbol{O} \cup \boldsymbol{O'} \quad \boldsymbol{O} \cap \boldsymbol{O'} = \emptyset}{C * C' : (\boldsymbol{I''} \setminus \boldsymbol{O''}) \to \boldsymbol{O''}}$$

This rule expresses that output and input nodes are plugged $1:n$, *i.e.*, each output node can be plugged into multiple input nodes. This results from the fact that the well-formedness conditions in the rule only impose that $\boldsymbol{O} \cup \boldsymbol{O'} = \emptyset$, and not that $\boldsymbol{I} \cup \boldsymbol{I'} = \emptyset$, and also from the fact that we only remove the repeated input nodes in the resulting arity. Regarding the behaviour, output nodes act as n-replicators, where data must flow to every connected input channel end. If $n = 0$, we assume that the data is consumed. The formal description we present differs slightly from the original description of $\mathcal{R}$eo, where nodes could result from an $n : m$ plugging, without fundamentally changing anything, in order to simplify our formal results.

Notation 1.1. *Given a map N and a set P. With a slight abuse of notation, define* $N \cap P \stackrel{def}{=} \{(n,d) \in N \mid n \in P\}$ *and* $N \setminus P \stackrel{def}{=} \{(n,d) \in N \mid n \notin P\}$.

The following two rules give the semantics for the composition of connectors $C_1 : \boldsymbol{I_1} \to \boldsymbol{O_1}$ and $C_2 : \boldsymbol{I_2} \to \boldsymbol{O_2}$. Note that a node set can only fire if it fires in both C_1 and C_2, with the same data value flowing in both cases.

$$\frac{C_1 \xrightarrow{N_1} C_1' \quad C_2 \xrightarrow{N_2} C_2' \quad N_1 \cap \mathcal{N}ames(C_2) = N_2 \cap \mathcal{N}ames(C_1)}{C_1 * C_2 \xrightarrow{N_1 \cup N_2} C_1' * C_2'}$$

$$\frac{C_1 \xrightarrow{N_1} C_1' \quad N_1 \cap \mathcal{N}ames(C_2) = \emptyset}{C_1 * C_2 \xrightarrow{N_1} C_1' * C_2}$$

Note that we do not address causality issues here, because the connectors we will build deliberately avoid causal loops. These can be trivially dealt with. We also introduce a restriction operator that hides some output nodes of a connector. Given a connector $C : \boldsymbol{I} \to \boldsymbol{O}$ and a set of nodes $\Omega \subseteq \boldsymbol{O}$, define $C\!\restriction_\Omega = C : \boldsymbol{I} \to \Omega$.

***Ordering* example** Consider the services *Politics*, *Sport* and *Email*, that return news about politics or news about sport, or send an email of a given message, respectively. In Fig. 2 we present a connector that coordinates these three services. Initially, the connector receives data from the *Politics* and the *Sport* services, and forwards data from *Politics* to the *Email* in a single step. After that, the data previously sent by the *Sport* service is sent to the *Email*. This way we guarantee that the two services alternate, and that we can only have politics news if sports news is also available—presumably as a sanity-preserving measure.

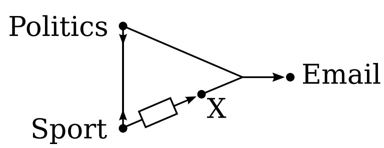


Figure 2: Example of a $\mathcal{R}$eo connector

Formally, we consider two different states of the connector:

$$\begin{array}{rcl} Ord & = & (FIFO1_{Sport,X} * SDrain_{Politics,Sport} * Merger_{Politics,X,Email})\restriction_{Email} \\ Ord(x) & = & (FIFO1_{Sport,X}(x) * SDrain_{Politics,Sport} * Merger_{Politics,X,Email})\restriction_{Email} \end{array}$$

The former corresponds to the connector depicted in Fig 2, while the latter corresponds to the same connector when the *FIFO1* channel is full with data x. Using the rules above, we can calculate the connector's arity, $Ord : \{Politics, Sport\} \to Email$, and behaviour:

$$\begin{array}{l} Ord \xrightarrow{Politics(v),Sport(w),Email(v)} Ord(w) \\ Ord(w) \xrightarrow{X(w),Email(w)} Ord. \end{array}$$

These transitions represent the only possible behaviour of the connector, given the axioms of each primitive and the reduction rules. The first transition goes to a state where the buffer is full, and indicates that data is flowing on the nodes *Politics*, *Sport*, and *Email*. The second transition goes back to the original state, and indicates that data is flowing on nodes X and *Email*.

***Synchronizing merge* example** We now present an example to illustrate how to define a more complex coordination pattern, without going into too much details. In Fig. 3 we represent a *synchronizing merge* connector, whose main idea is to control the execution of components or connectors A and B according to a specific pattern. This pattern is one of the workflow patterns defined by Van der Aalst [AHKB03].

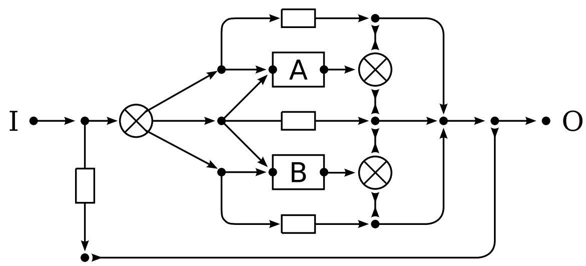


Figure 3: Synchronizing merge connector

In the connector presented in Fig. 3 we introduce some special notation. The nodes with a cross ($\otimes$) (to be introduced in Table 2) denote exclusive routers, which output the received data to precisely one of its outputs. We use nodes with more than one input to represent mergers connected to the node. The components denoted by A and B have an input node on the left side and an output node on the right side, and represent two external components or connectors that receive a signal to start executing, and return another signal after executing.

The composition of the behaviour of the primitives in Fig. 3, after hiding the details of every output node except **O**, yields the following behaviour. Initially, only the input node **I** can be fired, causing data to flow into the *FIFO1* channel on the left and through the exclusive router on the left. Data will flow also either to A, to B, or to both, depending on who is ready to receive data, and to one of the three *FIFO1* channels in the middle. The connector evolves to a new state, where the only possible step is to output data through node **O** after the components that were initiated return a signal, emptying the existing *FIFO1* channels. Therefore, if only A was executed, then B cannot execute until A finishes, and if both A and B were executed, then they must finish before any of them can be executed again.

2 A Static Encoding of $\mathcal{O}\mathsf{rc}$ in $\mathcal{R}\mathsf{eo}$

We present two translations of $\mathcal{O}\mathsf{rc}$ into $\mathcal{R}\mathsf{eo}$. The first translation, *the merged-output encoding*, attempts to directly model $\mathcal{O}\mathsf{rc}$ expressions, in particular, by merging the multiple results of a sequential composition. The second encoding, *the multiple-output encoding*, takes an alternative approach, duplicating the circuitry for each output of the $\mathcal{O}\mathsf{rc}$ expression. Note that we can only encode non-recursive $\mathcal{O}\mathsf{rc}$ expressions into a finite $\mathcal{R}\mathsf{eo}$ connector. For the remainder of the paper, we restrict ourselves to non-recursive $\mathcal{O}\mathsf{rc}$ expressions, denoted $\mathcal{O}\mathsf{rc}^-$. Basically, we assume that every invocation of a definition has been expanded. We also assume the existence of a $\mathcal{R}\mathsf{eo}$ component, with one input and one output node, for each primitive site. Initially, the component is ready to receive some data over the input node; after an unspecified amount of time, it may return a result over the output node.

Before presenting the encodings, we will introduce some useful $\mathcal{R}\mathsf{eo}$ connectors. We then give some formal properties concerning the second encoding on Section 2.4, and use weak bisimulation to prove its soundness with respect to $\mathcal{O}\mathsf{rc}$'s semantics.

2.1 Warming up

We now introduce the $\mathcal{R}\mathsf{eo}$ connectors used in the translations. Each connector is defined by presenting its arity and axioms, although they could equally have been defined as the composition of primitives.[5] The connectors are defined in Table 2.

Table 2 is divided into two parts. In the upper part we present a $\mathcal{R}\mathsf{eo}$ node and three connectors that play a rôle similar to nodes in that they connect a single output node to multiple input nodes A node ($\bullet$) with arity $I \to \{O_1, \ldots, O_n\}$ receives data in I and replicates the same data synchronously to O_1, ..., O_n. This can be derived from the rules introduced in Section 1.2. The behaviour of the remaining three connector in the upper part is as follows. Firstly, an *Exclusive Router* ($\otimes$) receives data in the input node and sends data

[5]The tupling connector T_n is an exception, as none of our primitives are capable of data manipulation.

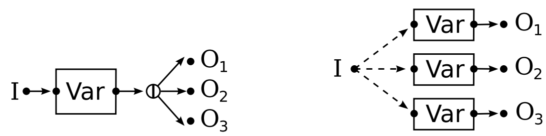
Visualisation	Arity	Axioms
$\bullet$	$I \to \{O_1, \ldots, O_n\}$	$\bullet \xrightarrow{I(x), O_1(x), \ldots, O_n(x)} \bullet$
$\otimes$	$I \to \{O_1, \ldots, O_n\}$	$\dfrac{O \in \{O_1, \ldots, O_n\}}{\otimes \xrightarrow{I(x), O(x)} \otimes}$
$\obar$	$I \to \{O_1, \ldots, O_n\}$	$\dfrac{\emptyset \subsetneq \{P_1, \ldots, P_m\} \subseteq \{O_1, \ldots, O_n\}}{\obar \xrightarrow{I(x), P_1(x), \ldots, P_m(x)} \obar}$
$\textcircled{\scriptsize 0}$	$I \to \{O_1, \ldots, O_n\}$	$\textcircled{\scriptsize 0} \xrightarrow{I(x), O_1(x_1), \ldots, O_n(x_n)} \textcircled{\scriptsize 1}$
$\textcircled{\scriptsize 1}$	$I \to \{O_1, \ldots, O_n\}$	–
T_n	$\{I_1, \ldots, I_n\} \to O$	$\mathsf{T}_n \xrightarrow{I_1(x_1), \ldots, I_n(x_n), O(x_1, \ldots, x_n)} \mathsf{T}_n$
C_p	$\emptyset \to O$	$\mathsf{C}_p \xrightarrow{O(p)} \mathsf{C}_p$
Var	$I \to O$	$\mathsf{Var} \xrightarrow{I(x)} \mathsf{Var}(x)$
$\mathsf{Var}(x)$	$I \to O$	$\mathsf{Var}(x) \xrightarrow{O(x)} \mathsf{Var}(x)$ $\mathsf{Var}(x) \xrightarrow{I(y)} \mathsf{Var}(y)$
P_n	$\{I_1, \ldots, I_n\} \to O$	$\dfrac{\emptyset \subsetneq \{P_1, \ldots, P_m\} \subseteq \{I_1, \ldots, I_n\} \quad k \in \{1, \ldots, m\}}{\mathsf{P}_n \xrightarrow{P_1(x_1), \ldots, P_m(x_m), O(x_k)} \mathsf{P}_n}$

Table 2: Definition of some $\mathcal{R}$eo connectors.

synchronously to exactly one of its output nodes. If more than one output node can receive the data, a non-deterministic choice is made. Secondly, an *Inclusive Router* ($\obar$) is a variation of the Exclusive Router that can send data to multiple output nodes instead of performing a non-deterministic choice. Third is a connector that acts like a node for one step, and then prevents flow for eternity, by becoming the connector in the fourth row.

Now consider the lower part of Table 2. Connector T_n tuples n values. It is a synchronous connector, *i.e.*, inputs and outputs succeed at the same time. Connector C_p always return a constant value p. Connectors Var and $\mathsf{Var}(x)$ represent a possibly-undefined variable. It is a buffer that replaces its content when new data arrives to the connector, and can output its content as many times as required. The last connector, P_n, coordinates n inputs into a single output. Data flows only if data can flow synchronously at one or more input nodes and the output node.

Before continuing with the encoding, an issue regarding the use of variables in $\mathcal{R}$eo needs to be resolved. A variable can be read by multiple connectors, either all at the same time or just by some at each step. To coordinate access to a variable, we propose two different approaches in Fig. 4: (a) replicate the output of the variable when necessary, or (b) replicate the input and create a variable connector for each possible access. The second approach has the advantage that the access to a variable does not require any synchronisation between the connectors that may also access the variable. Although more storage locations are required, it reduces the cost of coordination. This is the approach we use.



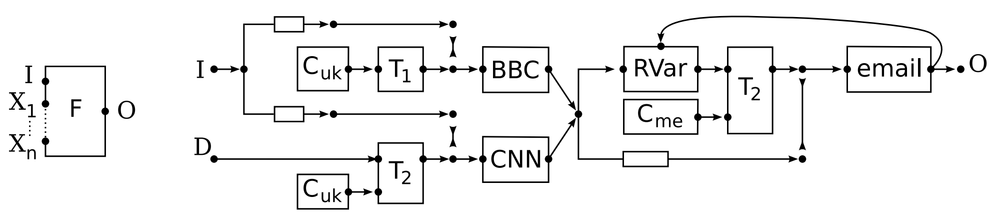
(a) (b)

Figure 4: (a) Replication *after* the storage of a variable. (b) Replication *of* the storage of a variable.

2.2 Merged-Output Encoding

This section presents an encoding of an $\mathcal{O}\mathsf{rc}^-$ expression into a connector which merges the multiple outputs of a parallel composition via a single output node. This is the most natural approach, but it is, as we shall see, problematic. We therefore only give an informal presentation, reserving a completely formal description for our second encoding.

An expression $h \in \mathcal{O}\mathsf{rc}^-$ is encoded as a connector with arity $\{I, X_1, \ldots, X_n\} \rightarrow O$, depicted in Fig. 5(a), where the X_i corresponds to the free variables of h. For example, the encoding of the expression $(\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)) >x> \mathit{email}(me, x)$ is presented in Fig. 5(b), recalling that d is a variable, whereas uk and me are constants. The connector starts by receiving data on input node I and buffering it. Site BBC can then be called, while site CNN needs to wait until data is available on node D. The results from the site calls are stored in the RVar component one at a time, which subsequently provides the value to site *email*, once for each value returned by BBC and CNN.



(a) (b)

Figure 5: Encodings into $\mathcal{R}\mathsf{eo}$ connectors with a single output: (a) a general $\mathcal{O}\mathsf{rc}$ expression; (b) a specific $\mathcal{O}\mathsf{rc}$ expression $(\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)) >x> \mathit{email}(me, x)$. RVar is a resettable variable. It acts like a variable (*e.g.*, Var from Section 2.1), but it cannot be updated until the reset (top) node is fired, removing the value of the variable.

The example encoding reveals the main problem of this approach. The outputs of $\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)$ are forwarded to a single instance of *email*, serializing the execution of *email*. As a consequence, it is possible that CNN finishes before BBC, but that site *email* hangs on the result of CNN, preventing *email* from even getting the result from BBC. The semantics of $\mathcal{O}\mathsf{rc}$ [KCM06], however, dictate that $(\mathit{CNN}(uk, d) \mid \mathit{BBC}(uk)) >x> \mathit{email}(me, x)$ is strongly bisimilar to $(\mathit{CNN}(uk, d) >x> \mathit{email}(me, x)) \mid (\mathit{BBC}(uk) >x> \mathit{email}(me, x))$, which means that *email* is not serialized and could respond to either results from CNN or BBC irrespective of their ordering or failure. This, however, is not true for the connectors resulting from the encoding. In the next section, we overcome this problem by duplicating parts of the connector.

Another solution for this termination problem is possible by introducing some observational behaviour corresponding to when a service does not publish any value, as done by Bruni *et al.* in their encoding of $\mathcal{O}\mathsf{rc}$ into Petri Nets [BMT06]. This could be achieved, for example, by adding timeouts to each primitive site call. An extension for $\mathcal{R}\mathsf{eo}$ that includes connectors capable of dealing explicitly with time was proposed by Arbab *et al.* [ABBR07]. The authors introduce the *Timed Constraint Automata*, which can be used to formally model the timeouts in $\mathcal{R}\mathsf{eo}$, allowing a precise definition of a component that fails to return any value. In our case we could attach a timeout connector to the input node of each site call, such as an *expiring FIFO1* channel, which loses the contents of its buffer after a certain time.

Using these ideas we expect that we could also encode recursive $\mathcal{O}\mathsf{rc}$ expressions, but we chose not to use this approach because we consider it to be less faithful to $\mathcal{O}\mathsf{rc}$'s semantics, where the failure to return a value cannot be observed.

2.3 Multiple-Output Encoding

A more faithful encoding of $\mathcal{O}\mathsf{rc}^-$ expressions is presented in this section. The encoding of an expression such as $f > x > g$ duplicates g for each output of f. The encoding is possible because we can obtain an upper bound on the number of outputs of an $\mathcal{O}\mathsf{rc}^-$ expression—this is not possible with full $\mathcal{O}\mathsf{rc}$ due to recursion. The following lemma captures this property.

Lemma 2.1. *Define function (#) on $\mathcal{O}\mathsf{rc}^-$ expressions (and internal representations: $?k$ and $let(v)$), and on sets of output actions as follows:*

$$\begin{array}{lcl}
\#(f \mid g) & = & \#(f) + \#(g) \\
\#(f > x > g) & = & \#(f) \times \#(g) \\
\#(g \textbf{ \textit{where} } x :\in f) & = & \#(g) \\
\#(M(v_1, \ldots, v_n)) & = & 1 \\
\#\mathbf{0} & = & 0
\end{array}
\qquad
\begin{array}{lcl}
\#(?k) & = & 1 \\
\#(let(v)) & = & 1 \\
 & & \\
 & & \\
\#(!v_1, \ldots, !v_n, a) & = & n \\
\multicolumn{3}{r}{\textit{where } !w \notin a}
\end{array}$$

This function gives an upper bound on the number of outputs produced by an $\mathcal{O}\mathsf{rc}^-$ expression, i.e., *for any $\mathcal{O}\mathsf{rc}^-$ expression h, $h \xrightarrow{a} h'$ implies $\#h \geq \#a + \#h'$.*

Proof. Note that substitution does not effect the number of outputs, *i.e.*, $\#([\sigma].h) = \#h$, because $\#(M(x))$ and $\#(M(v))$ are always 1 independently of the value of v, where x is a variable and v is a constant value.

The proof follows by induction on the structure of h. The base cases, $\mathbf{0}$, $let(v)$, $?k$, and $M(v)$, are trivial, since there is only one possible action for each, and only $let(v)$ produces one output. We also omit the case when $h = g \mid h$, since it is simpler than the other combinators, and the reasoning is analogous.

- $h = f > x > g$: In this case our induction hypothesis is: if $f \xrightarrow{a} f'$, then $\#f \geq \#a + \#f'$. Assume $f \xrightarrow{a} f'$, where $a = !v_1, .., !v_n, a'$ and $!w \notin a'$ (hence, $\#a = n$). The only possible reduction is given by rule Seq: $h \xrightarrow{a'} h'$, where $h' = f' > x > g \mid [v_1/x].g \mid \ldots \mid [v_n/x].g$. Since a' has no outputs, we know that $\#a' = 0$. Then we can conclude that:

$$\begin{array}{rcl}
\#a' + \#h' & = & \#h' \\
 & = & \#(f' > x > g \mid [v_1/x].g \mid \ldots \mid [v_n/x].g) \\
\{\text{Def. } (\#)\} & = & \#(f' > x > g) + \#([v_1,/x].g) + \cdots + \#([v_n/x].g) \\
 & = & \#(f' > x > g) + \#a \times \#g \\
\{\text{Def. } (\#)\} & = & (\#f' \times \#g) + \#a \times \#g \\
 & = & (\#a + \#f') \times \#g \\
\{\text{IH on } f\} & \leq & \#f \times \#g \\
 & = & \#h
\end{array}$$

- $h = g$ **where** $x :\in f$: Again, our induction hypothesis is: if $g \xrightarrow{a} g'$ and $f \xrightarrow{b} f'$, then $\#g \geq \#a + \#g'$ and $\#f \geq \#b + \#f'$. We consider 2 cases. The first case assumes that $g \xrightarrow{a} g'$ and $f \xrightarrow{b} f'$, where $!w \notin b$ (and therefore $\#b = 0$). In this case we can apply rule Asym3N: $h \xrightarrow{a,b} g'$ **where** $x :\in f'$, and conclude that $\#(a, b) + \#(g'$ **where** $x :\in f') = \#a + \#b + \#g' = \#a + \#g' \leq \#g = \#h$. For the second case we assume $g \xrightarrow{a} g'$ and $f \xrightarrow{!v,b} f'$, where b can be any set of actions. In this case we can apply rule Asym2V: $h \xrightarrow{a} [v/x].g'$, and conclude that $\#a + \#([v/x].g') = \#a + \#g' \leq \#g = \#h$. We arrive at a similar conclusion in the remaining possibilities for the reduction of h, after applying rule Asym1N, Asym2 or Asym1V.

□

Corollary 2.2. *Let $f \in \mathcal{O}\mathsf{rc}^-$, and $f \xrightarrow{a_1} f' \xrightarrow{a_2} \cdots \xrightarrow{a_n} f^{(n)}$ be a possible trace. Then $\#f \geq \#a_1 + \cdots + \#a_n + \#f^{(n)}$.*

We now define a function $[\![\cdot]\!]$ that converts an expression $f \in \mathcal{O}\mathsf{rc}^-$ into a $\mathcal{R}\mathsf{eo}$ connector. The arity of the resulting connector will be $\{I\} \cup \boldsymbol{V} \to \boldsymbol{O}$, where $I \notin \boldsymbol{V}$, I denotes the main input node, $\boldsymbol{V}$ denotes a set of nodes corresponding to the free variables of f, and $\boldsymbol{O}$ is the set of output nodes. Node I is used to initiate the connector, though nodes in $\boldsymbol{V}$ can be fired beforehand, which corresponds to the setting of these variables. The function $[\![\cdot]\!]$, presented in Fig. 6, is defined inductively on the shape of $\mathcal{O}\mathsf{rc}$ expressions, in such a way that the number of output nodes is given by the function ($\#$) defined above. In the definition of the encoding we use special primitives, also depicted in Fig. 6, where we show an F and G shaped hole to provide some intuition about where $[\![f]\!]$ and $[\![g]\!]$ are connected to. In the following description of these primitives, we use F, G, and M to denote the connectors encoding f, g and M, respectively.

Symmetric Parallel Composition:

$$\|_{\theta,\alpha,\beta} : I \to \{I_f, I_g\}$$

Initially $\theta = \alpha = \beta = 0$. The intuition behind the connector $\|_{\theta,\alpha,\beta}$, illustrated in Fig. 6(a), is that it is initialized by flow in node I, after which sends an initialization signal on nodes I_f and I_g. The data is buffered in buffers that can fire nodes I_f and I_g as soon as they are ready to be fired. As $[\![f \mid g]\!] = \|_{0,0,0} * \mathsf{F} * \mathsf{G}$, firing I_f and I_g will trigger the connectors F and G. The behaviour of $\|_{\theta,\alpha,\beta}$ is depicted in the diagram below.

$$\begin{tikzcd}
\|_{0,0,0} \arrow[r, "I(v)"] & \|_{1,v,v} \arrow[r, "I_f(v)"] \arrow[d, "I_g(v)"] \arrow[dr, "{I_f(v),I_g(v)}"] & \|_{1,0,v} \arrow[d, "I_g(v)"] \\
 & \|_{1,v,0} \arrow[r, "I_f(v)"] & \|_{1,0,0}
\end{tikzcd}$$

Sequential Composition:

$$\rangle\!\rangle x \rangle\!\rangle_{\theta,\langle\alpha_1,\ldots,\alpha_n\rangle} : \{I, O_{f1}, \ldots, O_{fn}\} \to \{I_f, I_{g1}, \ldots, I_{gn}, X_1, \ldots, X_n\}$$

The connector is illustrated in Fig. 6(b). The main idea is to execute F when data flows through the input node I, and to buffer each of its outputs in a different *FIFO1* channel. Each of these *FIFO1* channels is connected to a different instance of the encoded G, which can be executed in parallel after the corresponding *FIFO1* channel is filled.

To make the behaviour easier to describe, we factor $\rangle\!\rangle x \rangle\!\rangle_{\theta,\langle\alpha_1,\ldots,\alpha_n\rangle}$ into $n+1$ different connectors corresponding to unconnected parts of the main connector:

$$\rangle\!\rangle x \rangle\!\rangle_{\theta,\langle\alpha_1,\ldots,\alpha_n\rangle} = \rangle\!\rangle x \rangle\!\rangle^{\mathsf{F}}_{\theta} * \rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}1}_{\alpha_1} * \cdots * \rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}n}_{\alpha_n},$$

where $\rangle\!\rangle x \rangle\!\rangle^{\mathsf{F}}_{\theta} : I \to I_f$, $\rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}j}_{\alpha_j} : O_{fj} \to \{I_{gj}, X_j\}$, and $1 \le j \le n$. Initially $\theta = \alpha_1 = \ldots = \alpha_n = 0$. The possible behaviour of each of the subparts is the following:

$$\begin{array}{l}
\rangle\!\rangle x \rangle\!\rangle^{\mathsf{F}}_{0} \xrightarrow{I(v),I_f(v)} \rangle\!\rangle x \rangle\!\rangle^{\mathsf{F}}_{1} \\
\rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}j}_{0} \xrightarrow{O_{fj}(v),X_j(v)} \rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}j}_{v} \xrightarrow{I_{gj}(v)} \rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}j}_{0},
\end{array}$$

where $1 \le j \le n$. This means that $\rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,\ldots,0\rangle}$, when triggered by node I, synchronously triggers the input node of F. For each output of F (in node O_{fj}), the connector $\rangle\!\rangle x \rangle\!\rangle^{\mathsf{G}j}_{0}$ also synchronously fires node X_j (making the contents of variable x available in G), and evolves to a configuration where the input node of G_j can be fired whenever possible.

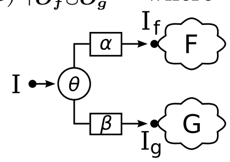
Asymmetric Parallel Composition:

$$\mathbb{W}^{x}_{\theta,\alpha,\beta,\delta} : \{I, O_{f1}, \ldots, O_{fn}\} \to \{I_f, I_g, X\}$$

The connector is illustrated in Fig. 6(c). The intuition is that F and G are executed in parallel. The output nodes of F are merged in such a way that only the first output value will flow through node X, which will be connected to G where the value of x is used. The output nodes of the connector $\mathbb{W}^{x}_{\theta,\alpha,\beta,\delta}$ are precisely the output nodes of G.

$[\![f \mid g]\!] = (\mathsf{F} * \|_{0,0,0} * \mathsf{G}) \restriction_{\boldsymbol{O_f} \cup \boldsymbol{O_g}}$ where

$\|_{\theta,\alpha,\beta} : I \to \{I_f, I_g\}$:=

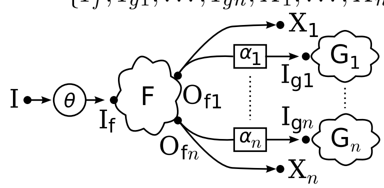


$\mathsf{F} := [\![f]\!] : \{I_f\} \cup \boldsymbol{V_f} \to \boldsymbol{O_f}$
$\mathsf{G} := [\![g]\!] : \{I_g\} \cup \boldsymbol{V_g} \to \boldsymbol{O_g}$

(a)

$[\![f >x> g]\!] = (\mathsf{F} * \rangle x \rangle_{0,\langle 0,\dots,0\rangle} * \mathsf{G}_1 * \cdots * \mathsf{G}_n) \restriction_{\bigcup_{i=1}^{n} \boldsymbol{O_{gi}}}$ where

$\rangle x \rangle_{\theta,\langle \alpha_1,\dots,\alpha_n\rangle} : \{I, O_{f1}, \dots, O_{fn}\} \to \{I_f, I_{g1}, \dots, I_{gn}, X_1, \dots, X_n\}$:=

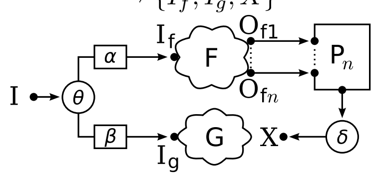


$\mathsf{F} := [\![f]\!] : \{I_f\} \cup \boldsymbol{V_f} \to \{O_{f1}, \dots, O_{fn}\}$
for $j \in \{1, \dots, n\}$:
$$\left\{ \begin{array}{l} \mathsf{G}_j := [\![[x_j/x].g]\!] : \\ \quad \{I_{gj}\} \cup \boldsymbol{V_g} \to \boldsymbol{O_{gj}} \\ x_j \text{ is a fresh} \\ \quad \text{variable name} \end{array} \right.$$

(b)

$[\![g \textbf{ where } x :\in f]\!] = (\mathbb{W}^x_{0,0,0,0} * \mathsf{F} * \mathsf{G}) \restriction_{\boldsymbol{O_g}}$ where

$\mathbb{W}^x_{\theta,\alpha,\beta,\delta} : \{I, O_{f1}, \dots, O_{fn}\} \to \{I_f, I_g, X\}$:=

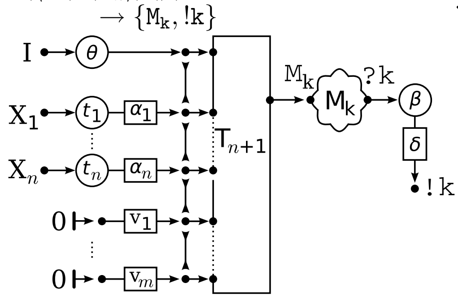


$\mathsf{F} := [\![f]\!] : I_f \cup \boldsymbol{V_f} \to \{O_{f1}, \dots, O_{fn}\}$
$\mathsf{G} := [\![g]\!] : I_g \cup \boldsymbol{V_g} \to \boldsymbol{O_g}$

(c)

$[\![M(x_1, \dots, x_n, v_1, \dots, v_m)]\!] = (\mathbb{M}_{0,\langle 0,\dots,0\rangle,V,0,0} * \mathsf{M_k}) \restriction_{!\mathtt{k}}$ where

$\mathbb{M}_{\theta,\langle \alpha_1,\dots,\alpha_n\rangle,V,\beta,\delta} : \{I, X_1, \dots, X_n, ?\mathtt{k}\} \to \{\mathtt{M_k}, !\mathtt{k}\}$:=



$V = \langle v_1, \dots, v_m\rangle$
$x_1, \dots, x_n$ are variables
$v_1, \dots, v_m$ are values
for $j \in \{1, \dots, n\}$:
$$t_j := \left\{ \begin{array}{l} 0 \text{ if } \theta = \alpha_j = 0 \\ 1 \text{ otherwise} \end{array} \right.$$
$\mathsf{M_k} : \mathtt{M_k} \to ?\mathtt{k}$:=
$\mathcal{R}$eo component of site M
k is fresh

(d)

Figure 6: Definition of the encoding function $[\![\cdot]\!]$ from $\mathcal{O}$rc into $\mathcal{R}$eo, where α, β, θ, and δ stand for the value of buffers (FIFO's or One Time nodes), whose value can be 0 (no value), 1 (some value), or a constant value. Nodes in the environment are associated with the variable with the same name in lower case. For example, node X_1 in $\mathcal{R}$eo corresponds to variable x_1 in $\mathcal{O}$rc.

To make the behaviour easier to describe, we factor $\mathbb{W}^x_{\theta,\alpha,\beta,\delta}$ into two different connectors corresponding to unconnected parts of the main connector:

$$\mathbb{W}^x_{\theta,\alpha,\beta,\delta} = \mathbb{W}^{\leftarrow}_{\theta,\alpha,\beta} * \mathbb{W}^{\rightarrow}_{\delta},$$

where $\mathbb{W}^{\leftarrow}_{\theta,\alpha,\beta} : I \to \{I_f, I_g\}$ and $\mathbb{W}^{\rightarrow}_{\delta} : \{O_{f1}, \ldots, O_{fn}\} \to \{X\}$. Initially $\theta = \alpha = \beta = \delta = 0$. The connector $\mathbb{W}^{\leftarrow}_{\theta,\alpha,\beta}$ is exactly the same as connector $\|_{\theta,\alpha,\beta}$, and therefore $\mathbb{W}^{\leftarrow}_{0,0,0}$ behaves as $\|_{0,0,0}$. The possible behaviour of $\mathbb{W}^{\rightarrow}_{0}$ is the following:

$$\mathbb{W}^{\rightarrow}_{0} \xrightarrow{\boldsymbol{O}, X(v_k)} \mathbb{W}^{\rightarrow}_{1},$$

where $\boldsymbol{O} \subseteq \{O_{f1}(v_1), \ldots, O_{fn}(v_n)\}$, and $v_k \in \{v_1, \ldots, v_n\}$ such that $O_{fk}(v_k) \in \boldsymbol{O}$. The choice of which node in $\{O_{f1}(v_1), \ldots, O_{fn}(v_n)\}$ will write into node X is made by connector P_n (see Table 2). This means that $\mathbb{W}^x_{0,0,0,0} * \mathsf{F} * \mathsf{G}$ behaves similarly to $\|_{0,0,0} * \mathsf{F} * \mathsf{G}$, except that the output nodes of F trigger the connector $\mathbb{W}^{\rightarrow}_{\delta}$. The output nodes of $\mathbb{W}^x_{\theta,\alpha,\beta,\delta}$ are restricted to the output nodes of F. This connector allows data to flow to node X, which is part of the environment of G and is made available to this instance. Note that the difference between $\|_{\theta,\alpha,\beta}$ and $\mathbb{W}^x_{\theta,\alpha,\beta,\delta}$ is captured, in part, by the combinator $\mathbb{W}^{\rightarrow}_{\delta}$.

Site call:

$$\mathbb{M}_{\theta,\Sigma,V,\beta,\delta} : \{I, X_1, \ldots, X_n, ?\mathtt{k}\} \to \{\mathtt{M_k}, !\mathtt{k}\}$$

The connector is illustrated in Fig. 6(d). The main idea is to tuple all the arguments required by site M before the site is executed. As in previous cases, we factor this connector into two different connectors corresponding to unconnected parts of $\mathbb{M}$ to make the behaviour easier to describe:

$$\mathbb{M}_{\theta,\Sigma,V,\beta,\delta} = \mathbb{M}^{\leftarrow}_{\theta,\Sigma,V} * \mathbb{M}^{\rightarrow}_{\beta,\delta}.$$

Initially $\theta = \beta = \delta = 0$ and $\Sigma = \langle 0, \ldots, 0\rangle$. The behaviour of each of the subparts is described below:

$$\begin{array}{l} \mathbb{M}^{\leftarrow}_{0,\Sigma_0,V} \xrightarrow{N_1} \mathbb{M}^{\leftarrow}_{0,\Sigma_1,V} \xrightarrow{N_2} \cdots \xrightarrow{N_j} \mathbb{M}^{\leftarrow}_{0,\Sigma_j,V} \xrightarrow{I(v),\mathtt{M_k}(v)} \mathbb{M}^{\leftarrow}_{1,\langle 0,\ldots,0\rangle,\langle 0,\ldots,0\rangle} \\ \mathbb{M}^{\rightarrow}_{0,0} \xrightarrow{\mathtt{M_k}(v)} \mathbb{M}^{\rightarrow}_{1,v} \xrightarrow{!\mathtt{k}(v)} \mathbb{M}^{\rightarrow}_{1,0} \end{array}$$

where $v = \langle v_1, \ldots, v_n\rangle$ is a tuple of data values, and for each $i \in \{0, \ldots, j\}$, $N_i \subseteq \{X_1(v_1), \ldots, X_n(v_n)\}$, $\bigcup_i N_i = \{X_1(v_1), \ldots, X_n(v_n)\}$, $\Sigma_j = \langle \alpha'_1, \ldots, \alpha'_n\rangle$ such that, for $i \in \{1, \ldots, n\}$, $\alpha'_i \neq 0$, and for each $X_m(v_m) \in N_i$, $\Sigma_i = [v_m/\alpha_m].\Sigma_{i-1}$. This means that initially the empty *FIFO1* channels in $\mathbb{M}^{\leftarrow}_{\theta,\Sigma,V}$ need to become full by the firing of the corresponding nodes. Only then node I can be fired, together with node $\mathtt{M_k}$ which triggers component $\mathsf{M_k}$. When this component returns data on node `?k`, the value is stored in a *FIFO1* channel, and in the next step the value is output by node `!k`.

Example revisited Recall the $\mathcal{O}$rc expression $(CNN(uk, d) \mid BBC(uk)) >x> email(me, x)$ presented in Section 1.1. We presented its merged-output encoding in Section 2.2. Fig. 7 presents the connector $[\![(CNN(uk, d) \mid BBC(uk)) >x> email(me, x)]\!]$ resulting from the multiple-output encoding. Data flowing through the input node I corresponds to the start of execution of the $\mathcal{O}$rc expression, and data flowing through the input node D corresponds to the binding of variable d.

Note that the resulting connector is not the most simple one, in the sense that there are consecutive *FIFO1* channels that could be merged into a single one, and there are some redundant One Time Nodes. If we wanted to actually run the encoding of an $\mathcal{O}$rc expression we could remove the One Time Nodes, relying on the assumption that site calls only return once, and the encoded connector is executed only once. Without these assumptions, the One Time Nodes are needed to derive a bisimulation between an $\mathcal{O}\mathsf{rc}^-$ expression and its encoding in $\mathcal{R}$eo.

2.4 Soundness

In this section we provide several important results about the translation presented in Fig. 6 that are required to understand and prove the main result, namely that every $h \in \mathcal{O}\mathsf{rc}^-$ is weakly bisimilar to its encoding in $\mathcal{R}$eo.

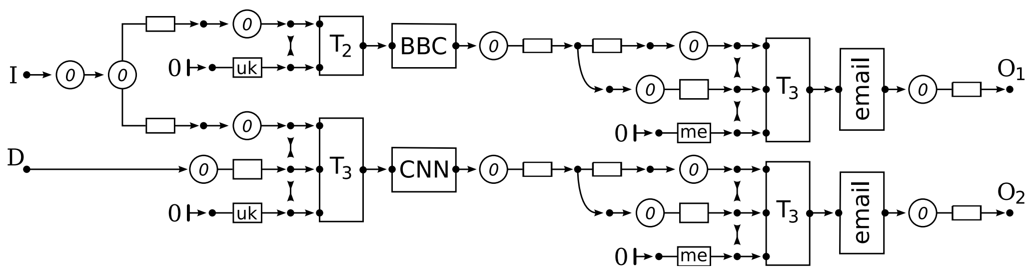


Figure 7: Multiple-output encoding of $(CNN(uk, d) \mid BBC(uk)) >x> email(me, x)$

Define the function $\widehat{\cdot}$ to map labels in $\mathcal{R}$eo's operational semantics to base events of $\mathcal{O}$rc as follows:

$$\begin{array}{rclrclrcl} \widehat{\mathtt{M_k}(\overline{v})} &=& M_k(\overline{v}) & \widehat{?\mathtt{k}(v)} &=& k?v & \widehat{!\mathtt{k}(v)} &=& !v \\ \widehat{\emptyset} &=& \emptyset & \widehat{a,b} &=& \widehat{a} \cup \widehat{b} & \widehat{a} &=& \tau \ \ \textit{otherwise}, \end{array}$$

where $\mathtt{M_k}$, $?\mathtt{k}$ and $!\mathtt{k}$ correspond to nodes in the $\mathcal{R}$eo connector obtained from the translation of a site call M.

Lemma 2.3. *Let $h \in \mathcal{O}\mathsf{rc}^-$. Each node in $\mathcal{N}ames([\![h]\!])$ can be fired at most once.*

Proof. Recall that the One Time node, depicted as θ, only allows data to flow once. We can verify that, in every rule of the translation, the input nodes are connected to a One Time node labelled by θ or t_n, for some number n, which means that the input nodes can only be fired once. The proof for the output nodes follows by induction on the structure of h.

- Base case – $h = M(v_1, .., v_n)$:
  We know that the input node can be triggered at most once because of the existence of a One Time node labelled by θ. The same way we have One Time nodes labelled by t_n, for some number n, and β, which guarantees that the nodes corresponding to input variables and the output node, respectively, can be fired at most once.
- $h = f \mid g$, $h = f >x> g$, or $h = g$ **where** $x :\in f$:
  By definition of $[\![h]\!]$, we can observe that the output nodes always correspond to the union of the output nodes of recursive calls to f or g. Since we know by induction hypothesis that the output nodes of each recursive call can be triggered at most once, than we can conclude that it is also the case for $[\![h]\!]$.

□

Lemma 2.4 relates the order in which input and output nodes are fired.

Lemma 2.4. *Let $h \in \mathcal{O}\mathsf{rc}^-$ and $H = [\![h]\!] : \{I\} \cup \boldsymbol{V} \to \boldsymbol{O}$. For any trace $\langle a_0, a_1, a_2, \ldots \rangle$ of sets of fired boundary nodes of H, we claim that:*

$$I \in a_n \quad \Rightarrow \quad \boldsymbol{O} \cap a_n = \emptyset, \ \textit{and for } 0 \leq j < n, \ a_j \subseteq \boldsymbol{V} \ \textit{and } \boldsymbol{O} \cap a_j = \emptyset.$$

This lemma can also be proved by structural induction on h. It is enough to verify that, for each case of the encoding function, the firing of the main input must precede the firing of the output node. By Lemma 2.4, the input and output nodes can be fired only once, so every action a occurring before the input node is fired is such that $\widehat{a} = \emptyset$ or $\widehat{a} = \tau$, because a can only refer to input or output nodes.

Since the main input node of the encoding of an $\mathcal{O}\mathsf{rc}^-$ expression can only be fired once, we introduce some notation to distinguish the states of the connector before and after the input node is fired. This simplifies the comparison of the evolution of $\mathcal{O}\mathsf{rc}^-$ expressions with different configurations of the encoded connector.

Definition 2.5. *Let $f \in \mathcal{O}\mathsf{rc}^-$ and $F = [\![f]\!] : \{I_f\} \cup \boldsymbol{V_f} \to \boldsymbol{O_f}$. We define two partitions of reachable configurations of F:*

$$F^{-I} = \left\{ F' \mid F \xrightarrow{a_1} \cdots \xrightarrow{a_n} F' \wedge I_f \notin \mathsf{nodes}(a_1 \cup \ldots \cup a_n) \wedge n \geq 0 \right\}$$
$$F^{+I} = \left\{ F' \mid F \xrightarrow{a_1} \cdots \xrightarrow{a_n} F' \wedge I_f \in \mathsf{nodes}(a_1 \cup \ldots \cup a_n) \wedge n \geq 1 \right\}$$

The first set consists of the configurations of F after zero or more steps up to when the input node is fired, and the second set consists of the possible configurations after the input node has fired. We say that a connector F' is reachable from F if $F' \in F^{-I} \cup F^{+I}$. Combining Definition 2.5 with Lemmas 2.3 and 2.4, we arrive at the following corollary.

Corollary 2.6. *Let $f \in \mathcal{O}\mathsf{rc}^-$ and $F = [\![f]\!] : \{I_f\} \cup \boldsymbol{V_f} \to \boldsymbol{O_f}$. Then:*

- *Assume $H \in F^{-I}$. If $H \xrightarrow{a} H'$, then $\mathsf{nodes}(a) \cap \boldsymbol{O_f} = \emptyset$. Furthermore, for all H'' reachable from F, if $H'' \xrightarrow{a} H$, then $H'' \in F^{-I}$, $I_f \notin \mathsf{nodes}(a)$, and either $\widehat{a} = \emptyset$ or $\widehat{a} = \tau$*
- *If $H \in F^{+I}$ and $H \xrightarrow{a} H'$, then $I \notin \mathsf{nodes}(a)$ and $H' \in F^{+I}$.*

The main result of this section is the existence of a weak bisimulation between an $\mathcal{O}\mathsf{rc}^-$ expression and its translation into $\mathcal{R}\mathsf{eo}$. We define the notion of weak transition and weak bisimulation inspired by Milner's definition of weak bisimilarity [Mil99].

Definition 2.7. *Let Q and Q' be $\mathcal{O}\mathsf{rc}$ expressions (or $\mathcal{R}\mathsf{eo}$ connectors), and a be a set of actions. We write $Q \overset{a}{\Rightarrow} Q'$ to denote $Q(\xrightarrow{\tau})^* \xrightarrow{a'} (\xrightarrow{\tau})^* Q'$, whenever $a \backslash \{\tau\} = a' \backslash \{\tau\}$,* i.e., *$Q$ evolves to Q' after performing a transition a' and any number of τ transitions before or after a'. When $a = \{\tau\}$, then $\overset{a}{\Rightarrow} \overset{def}{=} (\xrightarrow{\tau})^*$.*

Definition 2.8. *We say $\approx\, \subseteq \mathcal{O}\mathsf{rc}^- \times \mathcal{R}\mathsf{eo}$ is a weak bisimulation if for every pair $(f, C) \in \approx$, written $f \approx C$, where f is an $\mathcal{O}\mathsf{rc}$ expression, C is a connector configuration, and $a \subseteq BaseEvents$, we have:*

(i) if $f \xrightarrow{a} f'$, then $\exists b, C'$ such that $\widehat{b} = a$, $C \overset{b}{\Rightarrow} C'$ and $f' \approx C'$; and

(ii) if $C \xrightarrow{a} C'$, then there is an expression f' such that $f \overset{\widehat{a}}{\Rightarrow} f'$ and $f' \approx C'$.

We say f is weakly bisimilar to C, written $f \sim C$, if there is a weak bisimulation $\approx$ such that $f \approx C$.

Lemma 2.9 captures that substituting a variable in an $\mathcal{O}\mathsf{rc}$ expression is the same as triggering the input node associated with the corresponding variable.

Lemma 2.9. *Let $h \in \mathcal{O}\mathsf{rc}^-$ and $h_v \overset{def}{=} [v/x].h$, where x is a free variable in h, and v is a data value. Substitution does not change the behaviour of the translation,* i.e.*,*

$$\text{If } h \sim [\![h]\!] \text{ and } [\![h]\!] \xrightarrow{X(v)} H_v \text{ then } h_v \sim H_v,$$

where H_v is obtained by sending value v in node X.

Proof Outline. We start by verifying that the only relevant case is when $h = M(\overline{p})$, and $x \in \overline{p}$, because that is the only place where x can be used. We can prove that, in this case, the possible behaviour of $[\![h_v]\!]$ is the same as H_v, concluding that $h \sim [\![h]\!]$ implies $h_v \sim H_v$. ☐

Theorem 2.10 is the main result of this section, which relates $\mathcal{O}\mathsf{rc}$ expressions with their $\mathcal{R}\mathsf{eo}$ encodings. The proof uses the lemmas introduced above, in particular, Corollary 2.6 deals with inductive applications of the construction, and Lemma 2.9 handles the substitution of variables.

Theorem 2.10. *Let $h \in \mathcal{O}\mathsf{rc}^-$. We claim that $h \sim [\![h]\!] : I \cup \boldsymbol{V} \to \boldsymbol{O}$, where $\boldsymbol{V}$ contains only nodes associated to free variables of h.*

Proof Outline. This theorem follows by induction on the structure of h. We define the relation $\approx$ inductively for each constructor of $\mathcal{O}\mathsf{rc}$ as follows. We omit the prove that $\approx$ is a weak bisimulation, which can be done by analysing every possible element of $\approx$.

- $h = M(x_1, .., x_n, v_1, .., v_m)$
  Where $x_1, .., x_n$ are variables and $v_1, .., v_m$ are values. We assume that the last variables are always the first to be instantiated.

$$\begin{array}{rcl} \approx & = & \{(M(x_1, .., x_{i-1}, v'_i, .., v'_n, v_1, .., v_m), \mathbb{M}_{0,\langle 0,..,0,v'_j,..,v'_n\rangle,\langle v_1,..,v_m\rangle,0,0} * \mathsf{M_k}) \\ & & \quad \mid 1 \leq i \leq n\} \\ & \cup & \{(?k, \mathbb{M}_{1,\langle 0,..,0\rangle,\langle 0,..,0\rangle,0,0} * \mathsf{M_k}), \\ & & \ (let(v'), \mathbb{M}_{1,\langle 0,..,0\rangle\langle 0,..,0\rangle,1,v'} * \mathsf{M_k}), \\ & & \ (\mathbf{0}, \mathbb{M}_{1,\langle 0,..,0\rangle,\langle 0,..,0\rangle,1,0} * \mathsf{M_k})\} \end{array}$$

- $h = f \mid g$
  Let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G} = [\![g]\!]$. By the induction hypothesis there are two bisimulations, $\approx_f$ and $\approx_g$, such that $f \approx_f \mathsf{F}$ and $g \approx_g \mathsf{G}$.

$$
\begin{array}{rcl}
\approx & = & \{(f' \mid g', F' * \|_{0,0,0} * G')\ ,\ (f' \mid g', F' * \|_{1,v,v} * G') \\
 & & \qquad\quad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{-I}\} \\
 & \cup & \{(f' \mid g', F' * \|_{1,0,v} * G') \\
 & & \qquad\quad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I}\} \\
 & \cup & \{(f' \mid g', F' * \|_{1,v,0} * G') \\
 & & \qquad\quad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{+I}\} \\
 & \cup & \{(f' \mid g', F' * \|_{1,0,0} * G') \\
 & & \qquad\quad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{+I}\}
\end{array}
$$

- $h = f >x> g$
  Let $n = \#f$, and $1 \leq j \leq n$. Also let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G}_j = [\![g]\!]$. By the induction hypothesis there are $n+1$ bisimulations, $\approx_f$ and $\approx_{gj}$, such that $f \approx_f \mathsf{F}$ and $g \approx_{gj} \mathsf{G_j}$.

$$
\begin{array}{rcl}
\approx & = & \{(f' >x> g', F' * \rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'n) \\
 & & \qquad \mid f' \approx_f F' \wedge g' \approx_{gj} G'_j \wedge F' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}\} \\
 & \cup & \{(f' >x> g' \mid [v_1/x].g' \mid \ \ldots \ \mid [v_r/x].g', F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * \\
 & & \qquad G'_1 * \ldots * G'_n) \\
 & & \qquad \mid f' \approx_f F' \quad \wedge \quad F' \in \mathsf{F}^{+I} \quad \wedge \quad v_1,..,v_n \text{ are values} \\
 & & \qquad \wedge \quad \forall_{m \in \{1,..,r\}}.([v_m/x].g' \approx_{gs} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \\
 & & \qquad \wedge \quad \forall_{m \in \{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^-I \wedge \alpha_m = 0)\}
\end{array}
$$

- $h = g$ **where** $x :\in f$
  Let $n = \#f$, and $1 \leq j \leq n$. Also let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G} = [\![g]\!]$. By the induction hypothesis there are two bisimulations, $\approx_f$ and $\approx_g$, such that $f \approx_f \mathsf{F}$ and $g \approx_g \mathsf{G}$.

$$
\begin{array}{rcl}
\approx & = & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{0,0,0,0} * G') \\
 & & ,(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,v,v,0} * G') \\
 & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{-I}\} \\
 & \cup & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,v,0,0} * G') \\
 & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{+I}\} \\
 & \cup & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,0,v,\delta} * G') \\
 & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I} \wedge \delta \in \{0,1\}\} \\
 & \cup & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,0,0,\delta} * G') \\
 & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{+I} \wedge \delta \in \{0,1\}\}
\end{array}
$$

□

3 Encoding $\mathcal{R}$eo into $\mathcal{O}$rc

The encoding of $\mathcal{O}\mathsf{rc}^-$ into $\mathcal{R}$eo is local, in the sense that each $\mathcal{O}$rc combinator and each site call in an $\mathcal{O}$rc expression can be independently translated, and their composition yields the encoding of the main expression. On the other hand, we anticipate that the encoding of $\mathcal{R}$eo into $\mathcal{O}$rc would be global, since each $\mathcal{R}$eo connector needs to be considered as whole. Note that such an encoding will not be compositional. For example, $\mathcal{C}(Sync_{A,B} * Merger_{B,C,D})$ would not correspond to $\mathcal{C}(Sync_{A,B}) \mid \mathcal{C}(Merger_{B,C,D})$, since in the second case it is possible for data to flow from A to B, whereas in $\mathcal{R}$eo this could not occur if there was also data flowing from C to D. The encoding would become roughly the implementation of one of the known algorithms to combine the synchronous constraints imposed by $\mathcal{R}$eo primitives, such as Connector Colouring [CCA07].

The expressiveness of $\mathcal{O}$rc is closely related to the set of base primitive sites considered. An example use of more complex primitive site calls can be found in the work by Cook *et al.* [CPM06], where the authors encode into $\mathcal{O}$rc the set of workflow patterns proposed by Van der Aalst [AHKB03]. A similar approach could be attempted to encode $\mathcal{R}$eo into $\mathcal{O}$rc, using complex primitive site calls that can synchronize with each other, but still the encoding will not be compositional. We also analysed a synchronous semantics of $\mathcal{O}$rc, presented by

Cook *et al.* [CM07], where all events other than external response are processed as soon as possible. This allows, for example, to impose an order on how two primitive sites are called, which was not possible with the asynchronous semantics. However, it is still not possible to describe atomic blocks that can either succeed or rollback if one of the actions is not possible. A stronger model, for example, a transactional model, is required to capture the synchrony imposed by $\mathcal{R}$eo semantics.

Formal comparisons between synchronous and asynchronous communication have been explored in the context of the π-calculus. The asynchronous π-calculus, or π_a-calculus for short, is a subset of the π-calculus with no mixed choice operator, and whose syntax mandates that a process finishes after outputing a message in a channel. To have (polyadic) synchronisation in the π-calculus means that it is possible to constrain a fixed-sized tuple of more than one channel so that each element can be executed only if all the other elements of this tuple can also be executed. This notion of synchrony is closely related to synchrony in $\mathcal{R}$eo, since $\mathcal{R}$eo allows for the definition of constraints on the firing of more than one port in the same step. Unlike $\mathcal{R}$eo, the π-calculus does not propagate synchrony through composition.

In the remaining of this section we explore expressiveness results in the context of the π-calculus, in particular, the work by Palamidessi where she proves that the (synchronous) π-calculus cannot be encoded in the π_a-calculus [Pal97].

Sketch of a non-encodability result via π-calculus

We sketch a proof of the non-encodability of the $\mathcal{R}$eo into full $\mathcal{O}$rc, according to a reasonable notion of encodability, reusing the results of Palamidessi wherein she compares the expressiveness of the (synchronous) π-calculus and the π_a-calculus [Pal97]. Our argument relies on the assumption that different sites in $\mathcal{O}$rc can only communicate with each other through $\mathcal{O}$rc's combinators. Without this assumption it would be possible to use arbitrarily complex sites to produce the desired coordination.

Palamidessi proved that there is no uniform encoding from the π-calculus to the π_a-calculus that preserves a reasonable semantics. She defines an encoding to be uniform if it preserves distribution and permutations, *i.e.*, if the parallel operator on the π-calculus is encoded into the parallel operator on the π_a-calculus, and if for each renaming of variables before the encoding there is some permutation on the encoded process such that certain conditions hold. A reasonable semantics is characterised by distinguishing two processes P and Q whenever P can produce actions on certain intended channels that cannot be produced by Q. In her proof Palamidessi uses the argument that the leader election on a symmetric system cannot be solved using the π_a-calculus because the symmetry cannot be broken, while it is possible in the π-calculus, mainly because of the existence of the guarded choice construct.

It is not immediately clear how these results apply in our setting, as it is difficult to know the meaning of preserving the parallel operator on an encoding of $\mathcal{R}$eo into $\mathcal{O}$rc. We sketched our proof as follows. We present an encoding of $\mathcal{O}$rc into the π_a-calculus, allowing us to conclude that the symmetric leader election problem cannot be solved in $\mathcal{O}$rc. Note that we do not prove the correctness of the encoding. We then conclude by giving a brief explanation on how the leader election problem can be trivially solved in $\mathcal{R}$eo.

Encoding $\mathcal{O}$rc into the π_a-calculus

In this section we present briefly the syntax of the π_a-calculus, we define an encoding function $(\!|\cdot|\!)$ from $\mathcal{O}$rc to the π_a-calculus, and we translate our running example into the π_a-calculus.

The syntax of a π_a-calculus process is defined as follows, where x is a channel, and y is the message (or a tuple of messages) sent over a channel, which can be again a channel.

$$\textit{Processes} \qquad P \ ::= \ \overline{x}\langle y\rangle \ \mid \ x(y).P \ \mid \ (\nu x)P \ \mid \ P|P \ \mid \ !P$$

Informally $\overline{x}\langle y\rangle$ represents the output of message y through channel x, $x(y).P$ represents the reception of message through channel x, which becomes bounded to name y, and evolves to process P, $(\nu x)P$ represents the creation of a new channel name x, which can occur in P, $P|P$ stands for the parallel execution of two processes, that can communicate over common channels, and $!P$ represents the replication of process P, *i.e.*, an unbounded parallel execution of copies of the same process P. Note that replication in the π_a-calculus has been proven to be equivalent to a set of recursive processes. We omit the formal semantics of the π_a-calculus, which can be easily found in the literature [Pal97, CM03].

The general idea is, given an $\mathcal{O}\mathsf{rc}$ expression f and a channel name s, produce its translation P in the π_a-calculus such that P can be executed by sending a message *out* through channel s. We denote it by $(\![f, s]\!) = P$. The message *out* is the channel used by the resulting expression to output the possible results of the corresponding $\mathcal{O}\mathsf{rc}$ expression. The names of variables in $\mathcal{O}\mathsf{rc}$ are used as the names of the channels in the π_a-calculus, where the corresponding value is passed. We present the encoding in Fig.8, where we use the notation $(\nu x\ y\ z)P$ to denote $(\nu x)(\nu y)(\nu z)P$.

$$
\begin{array}{rcl}
(\![f \mid g, start]\!) & = & start(out).(\nu startf\ \ startg) \\
& & \big((\![f, startf]\!) \mid \overline{startf}\langle out\rangle \\
& & \mid (\![g, startg]\!) \mid \overline{startg}\langle out\rangle\big) \\[2ex]
(\![g \textbf{ where } x :\in f, start]\!) & = & start(out).(\nu startf\ \ startg\ \ outf\ \ x) \\
& & \big((\![f, startf]\!) \mid \overline{startf}\langle outf\rangle \\
& & \mid (\![g, startg]\!) \mid \overline{startg}\langle out\rangle \\
& & \mid outf(x').!\overline{x}\langle x'\rangle\big) \\[2ex]
(\![f > x > g, start]\!) & = & start(out).(\nu startf\ \ startg\ \ outf) \\
& & \big((\![f, startf]\!) \mid \overline{startf}\langle outf\rangle \\
& & \mid !(outf(x').(\nu\ startg\ x) \\
& & \quad ((\![g, startg]\!) \mid \overline{startg}\langle out\rangle \mid !\overline{x}\langle x'\rangle))\big) \\[2ex]
(\![M(p), start]\!) & = & start(out).p(p').\overline{\mathsf{M}}\langle p', out\rangle \\[2ex]
(\![E(p), start]\!) & = & start(out).\overline{\mathsf{E}}\langle p, out\rangle
\end{array}
$$

Figure 8: Translation of $\mathcal{O}\mathsf{rc}$ into π_a-calculus

The general definition of the encoding of an $\mathcal{O}\mathsf{rc}$ expression f into the π_a-calculus, after introducing a set $Defs = \{D_1, \ldots, D_n\}$ of definitions of auxiliary $\mathcal{O}\mathsf{rc}$ expressions, is as follows:

$$(\![f, Defs, start]\!) = (\![f, start]\!) \mid (\![D_1]\!)_{def} \mid \cdots \mid (\![D_1]\!)_{def} \mid Sites$$

where $(\![E(p) \stackrel{\text{def}}{=} f]\!)_{def} = !\big(\mathsf{E}(p, out).(\nu startf)((\![f, startf]\!) \mid \overline{startf}\langle out\rangle)\big)$. Furthermore, we assume *Sites* to consist on several processes in parallel, one for each site M used in f and in *Defs*, such that it can always receive a message through channel M to start the computation corresponding to the site call to M. Note that the parameters of M will be all the arguments of site M, and also the channel that should be used to return the result of the site call.

Fig. 9 presents the encoding $(\![\cdot]\!)$ applied to our running example, where *start* is the name of the channel that needs to be used to start the execution of the translated process.

Separation result

The encoding of $\mathcal{O}\mathsf{rc}$ into the π_a-calculus, if proved to be correct, shows that $\mathcal{O}\mathsf{rc}$ is not expressive enough to break the symmetry when solving the leader election problem. Carbone and Maffeis [CM03] extended Palamidessi's result to show that the expressive power of the π-calculus with polyadic synchronisation that can synchronize at most n channels is less than of the one that can synchronize at most $n+1$ channels. These results emphasise the idea that $\mathcal{R}\mathsf{eo}$ cannot be encoded in $\mathcal{O}\mathsf{rc}$, because $\mathcal{O}\mathsf{rc}$ is asynchronous whereas $\mathcal{R}\mathsf{eo}$ can synchronize an arbitrary number of ports.

The synchrony and exclusion inherent to $\mathcal{R}\mathsf{eo}$, unlike in $\mathcal{O}\mathsf{rc}$ and the π_a-calculus, allows the symmetry of a system to be easily broken. Combined with fact that the symmetric leader election problem cannot be solved in $\mathcal{O}\mathsf{rc}$, this is enough to show the non-encodability of $\mathcal{R}\mathsf{eo}$ into $\mathcal{O}\mathsf{rc}$. Although we do not prove formally that the leader election problem can be solved in the context of $\mathcal{R}\mathsf{eo}$, the connector in Fig. 10 provides the necessary intuition

$$
\begin{array}{l}
(\!\![(CNN(uk,d) \mid BBC(uk)) > x > email(me,x), start]\!\!) = \\
\quad start(out).(\nu startf\ startg\ outf\ x) \\
\quad\quad \big(startf(out).(\nu startf\ startg) \\
\quad\quad\quad \big(startf(out).uk(uk').d(d').\overline{CNN}\langle uk', d', out\rangle \\
\quad\quad\quad \mid \overline{startf}\langle out\rangle \\
\quad\quad\quad \mid startg(out).uk(uk').\overline{BBC}\langle uk', out\rangle \\
\quad\quad\quad \mid \overline{startg}\langle out\rangle\big) \\
\quad\quad \mid \overline{startf}\langle outf\rangle \\
\quad\quad \mid !\big(outf(x').(\nu startg\ x) \\
\quad\quad\quad (startg(out).me(me').x(x').\overline{email}\langle me', x', out\rangle \\
\quad\quad\quad \mid \overline{startg}\langle out\rangle \mid !\overline{x}\langle x'\rangle)\big)\big)
\end{array}
$$

Figure 9: Example of the encoding of $\mathcal{O}$rc into π_a-calculus

to understand how it could be solved. This example shows a connector built from three symmetric sub-connectors, each having an input port on the left and an output port on the right, connected with each other. The resulting connector guarantees that, after one step, all the output nodes will have received the same message from exactly one input port I_n, chosen non-deterministically. This results from (1) the synchronous replication of each of the messages received, which guarantees that data can only flow in an input node if the same data can also flow in all the output nodes; and (2) from the merge of the messages, that guarantees that the data flowing in each of the outputs can only come from one of the inputs, excluding the possibility of dataflow on the remaining inputs.

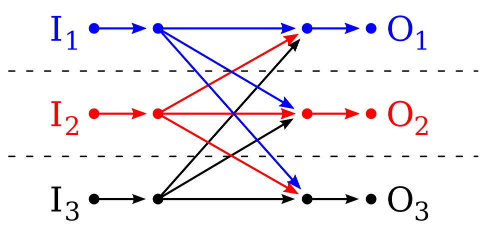


Figure 10: Leader election in $\mathcal{R}$eo

4 Discussion

We now compare $\mathcal{O}$rc and $\mathcal{R}$eo on some issues of philosophy and design.

Focus of Control In $\mathcal{O}$rc, control lies with the orchestrator: an $\mathcal{O}$rc expression initiates contact with external sites. On the other hand, $\mathcal{R}$eo assumes that control is initiated externally to a connector by a component. The request to write data to or read data from a node is subsequently handled by the connector. This is how $\mathcal{R}$eo coordinates, by controlling when such requests can succeed, though from the perspective of web services, control is inverted.

Component/Service Instantiation In $\mathcal{R}$eo, components are attached externally to a connector, whereas $\mathcal{O}$rc can dynamically initiate contact with services. $\mathcal{O}$rc is thus more dynamic, although it is tightly bound to the actual sites being called. These limitations seem easy to lift.

One-off Interaction vs. Streams $\mathcal{O}$rc expressions unfold over their life-time, so each piece of syntax is reduced once and each site call is performed once. On the other hand, $\mathcal{R}$eo establishes rigid connections between parties, as it makes the assumption that parties will continuously communicate.

Dynamics As an $\mathcal{O}$rc expression reduces, its 'configuration' changes dynamically. For instance, $f >x> g$ creates a new instance of g for each value produced by f. This was encoded in $\mathcal{R}$eo by calculating a bound on the number of values produced by f and duplicating the circuitry for g. As $\mathcal{R}$eo's connectivity is more or less fixed, and $\mathcal{O}$rc expressions 'fire' only once, our encoding introduces a lot of connectors that are used only once. In very recent work by Koehler *et al.* [KLA07, KCPA08] the authors present how to model dynamic reconfiguration of $\mathcal{R}$eo connectors using a high level approach based on graph transformation techniques. The authors go further with this idea, and propose a framework where dataflow triggers the reconfiguration process. This framework can be the basis for self adapting and dynamically reconfigurable connectors.

Asynchrony vs. Synchrony $\mathcal{O}$rc offers highly asynchronous connectives that gracefully deal with failing sites. $\mathcal{R}$eo is highly synchronous and susceptible to failure. Recall that failure can also be handled with timed connectors, as mentioned in Section 2.2, although this solution is less transparent, as failure must explicitly be handled. In principle, synchrony (or in any case, atomicity) can form the basis of high-level abstractions.

5 Related Work

Bruni *et al.* [BMT06] present a static encoding of $\mathcal{O}$rc into Petri nets. However, their encoding is not faithful to the $\mathcal{O}$rc model, as it assumes that each primitive site returns either a valid value or some value to state that it will not return a value. $\mathcal{O}$rc, on the other hand, gracefully deals with sites which do not return values. Our encoding into $\mathcal{R}$eo more accurately handles the absence of dataflow. Our encoding also considers the data values passed around, in contrast to Bruni *et al.*'s encoding, which passes only Petri net tokens. Bruni *et al.* also present an encoding of full $\mathcal{O}$rc into the Join calculus—an expressive calculus for concurrent processes developed at INRIA. The Join calculus provides a simple support for distributed programming, intentionally avoiding some communication constructs that are difficult to implement in a distributed setting. This calculus supports some synchrony, by introducing patterns that correspond to multiple events which must all be present for the pattern to be recognized. However, the Join-calculus is not highly synchronous like $\mathcal{R}$eo, as it does not propagate synchrony through composition. The precise relationship between the Join calculus and $\mathcal{R}$eo is left for future work.

Many other coordination languages exist, and these are compared in some earlier surveys [PA98, AHM96]. We can fairly safely say that few (coordination) languages offer the degree of synchrony that $\mathcal{R}$eo offers. Obvious exceptions are synchronous languages such as Esterel [Ber00]. These languages are useful for programming reactive systems, though they lack non-determinism, and in general seem not to be directly useful for coordinating distributed systems. To remedy this situation, the GALS (globally asynchronous, locally synchronous) model [Cha84, DMK$^+$06] has been adopted, whereby local computation is synchronous and communication between different machines is asynchronous.

As with $\mathcal{O}$rc, the GALS model adopts the arguably correct view that distributed systems must be programmed asynchronously. $\mathcal{R}$eo is also able to express such distinctions, and more, through the many choices of synchrony or asynchrony—the result depends upon how a connector is deployed to a distributed system. $\mathcal{R}$eo claims that instead of synchrony, it is really implementing atomicity, and hence a basic form of transaction [Arb04]. This has not yet been convincingly demonstrated.

A method for comparing expressiveness was proposed by de Boer and Palamidessi [BP94], where they introduce a notion of language embedding refined with some "reasonable" conditions. Brogi and Jaquet used this method to compare coordination models with Linda-like operations and a shared dataspace [BJ03]. Our attempt to prove that $\mathcal{R}$eo could not be encoded is based on a result of Palamidessi where she compares the expressiveness of the π-calculus and the π_a-calculus, which follows a similar approach to [BP94], but not for the same class of languages. However, it is not clear how this result could be used to prove the encodability of $\mathcal{O}$rc$^-$ into $\mathcal{R}$eo.

The idea of reusing the expressiveness results with the π-calculus was already successfully used to compare expressiveness in other contexts. A good example is the work from Philips and Vigliotti [PV04], where they compare the expressiveness of ambient calculi against different dialects of the π-calculus, providing also a good overall perspective on existing expressiveness results with respect to the π-calculus.

6 Conclusion and Future Work

We have compared $\mathcal{O}$rc and $\mathcal{R}$eo, by encoding the non-recursive fragment of $\mathcal{O}$rc into $\mathcal{R}$eo, by discussing the failure of the encoding in the other direction, and by comparing a number of design decisions. $\mathcal{O}$rc is highly asynchronous and deals well with failure. $\mathcal{R}$eo supports a high degree of synchrony, and potentially high-level abstractions. An obvious omission is a comparison of the efficiency of the two models. Unfortunately, both implementations are too preliminary for this to have any real meaning. The extension of our encoding to full $\mathcal{O}$rc requires either recursively-defined or dynamically reconfigurable $\mathcal{R}$eo connectors. These extensions to $\mathcal{R}$eo are interesting on their own, and are the subject of future work.

Note that, despite the expressiveness power provided by $\mathcal{R}$eo, we can still have feasible implementations in asynchronous networks. This is mainly because problems such as the leader election can be solved in real networks by assuming that the system is not completely symmetric, *i.e.*, we can assume unique identifiers exist for every entity in a network which can be used to break the symmetry.

References

[ABBR07] Farhad Arbab, Christel Baier, Frank de Boer, and Jan Rutten. Models and temporal logical specifications for timed component connectors. *Software and Systems Modeling (SoSyM)*, 6(1):59–82, March 2007.

[AHKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[AHM96] Jean-Marc Andreoli, Chris Hankin, and Daniel Le Metayer, editors. *Coordination Programming: Mechanisms, Models and Semantics*. Imperial College Press, 1996.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[Ber00] Gérard Berry. *Proof, language, and interaction: essays in honour of Robin Milner*, chapter The foundations of Esterel, pages 425–454. MIT Press, Cambridge, MA, USA, 2000.

[BJ03] Antonio Brogi and Jean-Marie Jacquet. On the expressiveness of coordination via shared dataspaces. *Sci. Comput. Program.*, 46(1-2):71–98, 2003.

[BMT06] Roberto Bruni, Hernán C. Melgratti, and Emilio Tuosto. Translating Orc features into Petri nets and the Join calculus. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2006.

[BP94] Frank S. de Boer and Catuscia Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.

[BSAR06] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan Rutten. Modeling component connectors in Reo by constraint automata. *Science of Computer Programming*, 61(2):75–113, 2006.

[CCA07] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency. *Sci. Comput. Program.*, 66(3):205–225, 2007.

[Cer02] Ethan Cerami. *Web Services Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[Cha84] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, October 1984.

[CM03] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nord. J. Comput.*, 10(2):70–98, 2003.

[CM07] William R. Cook and Jayadev Misra. Computation orchestration, a basis for wide-area computing. *Software and Systems Modeling*, 6(1):83–110, 2007.

[CPM06] William R. Cook, Sourabh Patwardhan, and Jayadev Misra. Workflow patterns in Orc. In Paolo Ciancarini and Herbert Wiklicky, editors, *COORDINATION*, volume 4038 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2006.

[DMK+06] Frederic Doucet, Massimiliano Menarini, Ingolf H. Krüger, Rajesh K. Gupta, and Jean-Pierre Talpin. A verification approach for gals integration of synchronous components. *Electr. Notes Theor. Comput. Sci.*, 146(2):105–131, 2006.

[FW80] Daniel P. Friedman and David S. Wise. An indeterminate constructor for applicative programming. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada*, pages 245–250, January 1980.

[Gel85] David Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[KCM06] David Kitchin, William R. Cook, and Jayadev Misra. A language for task orchestration and its semantic properties. In Christel Baier and Holger Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 2006.

[KCPA08] Christian Koehler, David Costa, José Proença, and Farhad Arbab. Reconfiguration of Reo connectors triggered by dataflow. In *The 8th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT)*, 2008. To appear.

[KLA07] Christian Koehler, Alexander Lazovik, and Farhad Arbab. Connector rewriting with high-level replacement systems. In *The 6th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 2007)*, 2007.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, June 1999.

[PA98] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *M. Zelkowitz (Ed.), The Engineering of Large Systems*, volume 46 of *Advances in Computers*, pages 329–400. Academic Press, 1998.

[Pal97] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *POPL*, pages 256–265, 1997.

[PV04] Iain Phillips and Maria Grazia Vigliotti. Electoral systems in ambient calculi. In Igor Walukiewicz, editor, *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 408–422. Springer, 2004.

[SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents.* John Wiley & Sons, 2005.

A Proofs

Proof. (Lemma 2.4) The proof follows by induction on the structure of h:

- $h = f \mid g$
  We know that $[\![h]\!] = (\mathsf{F} * \|_{0,0,0} * \mathsf{G}) \upharpoonright_{\vec{O_f} \cup \vec{O_g}}$, where $\mathsf{F} := [\![f]\!] : \{I_f\} \cup \vec{V_f} \to \vec{O_f}$ and $\mathsf{G} := [\![g]\!] : \{I_g\} \cup \vec{V_g} \to \vec{O_g}$. Therefore $[\![h]\!] : \{I\} \cup \vec{V_f} \cup \vec{V_g} \to \vec{O_f} \cup \vec{O_g}$ The only possible reduction step of $\|_{0,0,0}$ is by $I(v)$ to $\|_{1,v,v}$, where v is a data value, and only after this reduction step the input nodes of F and G can be fired. Since we know by induction hypothesis that the property is valid for F and G, then we conclude that node I is always triggered before the outputs. By induction hypothesis we also know that nodes in $\vec{V_f}$ and $\vec{V_g}$ can still be fired by F or G before node I is fired. We conclude that only nodes in $\vec{V_f} \cup \vec{V_g}$ can be fired by $[\![h]\!]$ before node I is fired.

- $h = f >x> g$
  We know that $[\![h]\!] = (\mathsf{F} * \rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle} * \mathsf{G_1} * \ldots * \mathsf{G_n}) \upharpoonright_{\vec{O_{g1}} \cup \ldots \cup \vec{O_{gn}}}$, where $\rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle}$: $\{I, O_{f_1}, .., O_{f_n}\} \to \{I_f, I_g, X\}$, $\mathsf{F} := [\![f]\!] : \{O_{f1}, .., O_{fn}\}$ and $\mathsf{G}_j := [\![g]\!] : \{I_{gj}\} \cup \vec{V_g} \to \vec{O_{gj}}$, for any j between 1 and n. Therefore $[\![h]\!] : \{I\} \cup \vec{V_f} \cup \vec{V_g} \to \vec{O_{g1}} \cup \ldots \cup \vec{O_{g_n}}$. By induction hypothesis we know that the output nodes of F and G cannot occur until their input nodes are fired, and consequently the nodes $X_1, .., X_n$ cannot be fired either. The only possible boundary nodes of $[\![h]\!]$ that can be fired are I and nodes in $\vec{V_f} \cup \vec{V_g}$. Note that firing of nodes in $\vec{V_f} \cup \vec{V_g}$ can occur before their input node is fired. Node I_f is only fired when node I is fired (by definition of $\rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle}$), and only after the output nodes of F are fired can the input nodes of G_j be fired.

- $h = g$ **where** $x :\in f$
  This case is very similar to when $h = f \mid g$. The same arguments presented for that case are also valid here: by induction hypothesis we can also claim that F and G can only fire their output nodes after their input nodes are fired, which only occurs after node I is fired. The difference with respect to $f \mid g$ is that we only need to consider the output nodes of G, and G may have an input node labelled by X, which will be dependant on the flow of data on one of the output nodes of F. The only possible observation of $[\![h]\!]$ until node I is fired correspond to dataflow in $\vec{V_f}$ or in $\vec{V_g} \backslash X$.

- $h = M(x_1, .., x_n, v_1, .., v_m)$
  We know that $[\![h]\!] = (\mathbb{M}_{0,\langle 0,..,0\rangle,V,0,0} * \mathsf{M_k}) \upharpoonright_{!\mathtt{k}}$, where k is fresh, $\mathsf{M_k}$ is a $\mathcal{R}\mathsf{eo}$ component corresponding to site M, and $V = \langle v_1, .., v_m\rangle$. We can derive that $[\![h]\!]$: $\{I, X_1, .., X_n, ?\mathtt{k}\} \to \{!\mathtt{k}\}$. The component $\mathsf{M_k}$ can only be executed when node $\mathtt{M_k}$ is fired, which can only occur in the same synchronous step as the firing of node I and of the output ends of the FIFO1 channels associated to the arguments of M (recall that the component T_{n+1} is synchronous). Initially the only possible behaviour of $[\![h]\!]$ is to fire input nodes other than I, *i.e.*, nodes in $\{X_1, .., X_n\}$, until every FIFO1 channel associated to each variable is full. When this occurs, the only possible behaviour is:

  $$\mathbb{M}_{0,\langle \alpha_1,..,\alpha_n\rangle,\langle v_1,..,v_m\rangle,0,0} \xrightarrow{I(y),\mathtt{M_k}(\langle \alpha_1,..,\alpha_n,v_1,..,v_m\rangle)} \mathbb{M}_{1,\langle 0,..,0\rangle,\langle 0,..,0\rangle,0,0},$$

  triggering the execution of site M, and only after the site returns a value in node $?\mathtt{k}$ the connector evolves, flowing data in its only output node $!\mathtt{k}$.

□

Proof. (Theorem 2.10)
In Section 2.4 we express a weak bisimulation between a non-recursive $\mathcal{O}\mathsf{rc}$ expression f and its encoding into $\mathcal{R}\mathsf{eo}$ $[\![f]\!]$. Here we present an exaustive proof that $f \sim [\![f]\!]$, by presenting a valid bisimulation.

For this proof we use Corolary 2.6, which guarantees that only nodes in the environment can succeed until the input node is fired, and that each input node is fired at most once. The proof follows by induction on the structure of h.

- $h = M(v_1, .., v_m)$, where $v_1, .., v_m$ are values.
  Let $v = \langle v_1, .., v_m\rangle$. The only possible reduction of h is:

  $$M(v) \xrightarrow{M_k(v)} ?k \xrightarrow{k?v'} let(v') \xrightarrow{!v'} \mathbf{0},$$

where v' is the value returned by site M. The only possible reduction of $[\![h]\!]$ is:

$$\mathbb{M}_{0,\langle\rangle,v,0,0} * \mathsf{M_k} \xrightarrow{I(x),\mathtt{M_k}(v)} \mathbb{M}_{1,\langle\rangle,\langle 0,..,0\rangle,0,0} * \mathsf{M_k} \xrightarrow{?\mathtt{k}(?v')} \mathbb{M}_{1,\langle\rangle\langle 0,..,0\rangle,1,v'} * \mathsf{M_k}$$
$$\xrightarrow{!\mathtt{k}(v')} \mathbb{M}_{1,\langle\rangle,\langle 0,..,0\rangle,1,0} * \mathsf{M_k}.$$

Since $\{\widehat{I(x),\mathtt{M_k}(v)}\} = \{M_k(v), \tau\}$, $\{\widehat{?\mathtt{k}(?v')}\} = \{k?v'\}$, and $\{\widehat{!\mathtt{k}(v')}\} = \{!v'\}$, we can define a weak bisimulation $R_M = \{(M(v), \mathbb{M}_{0,\langle\rangle,v,0,0}*\mathsf{M_k}), (?k, \mathbb{M}_{1,\langle\rangle,\langle 0,..,0\rangle,0,0}*\mathsf{M_k}), (let(v'), \mathbb{M}_{1,\langle\rangle\langle 0,..,0\rangle,1,v'} * \mathsf{M_k}), (\mathbf{0}, \mathbb{M}_{1,\langle\rangle,\langle 0,..,0\rangle,1,0} * \mathsf{M_k})\}$, which allow us to conclude that $h \sim [\![h]\!]$. Note that the value k referred in both systems is the same value. This means that the translation of a primitive site does not choose any fresh k but the exact same value as the reduction semantics of $\mathcal{O}$rc. Since we need one different value for each instance of the site M in both reduction semantics, the value will still be fresh.

- $h = M(x_1, .., x_n, v_1, .., v_m)$, where $x_1, .., x_n$ are variables.
To make the explanation easier, we will assume that the arguments of M are sorted: the first n arguments are variables, and the following m arguments are values. Furthermore, we assume that the last variables are always the first to be instantiated. Since the evaluation of M is strict, then the only possible behaviour is to instantiate variables, replacing them by data values. We consider the application of a substitution to be an internal action:

$$\begin{array}{rcl} M(x_1, .., x_n, v_1, .., v_m) & \xrightarrow{\tau} & [v'_j/x_j, .., v'_n/x_n].M(x_1, .., x_n, v_1, .., v_m) \\ & & = M(x_1, .., x_{j-1}, v'_j, .., v'_n, v_1, .., v_m) \end{array}$$

We can label this action by τ because, in $\mathcal{O}$rc' semantics, when a substitution occurs it is either labelled by τ, or ignored if some other action also occurs.
In this case $[\![h]\!] = \mathbb{M}_{0,\langle 0,..,0\rangle,\langle v_1,..,v_m\rangle,0,0} * \mathsf{M_k} : \{I, X_1, .., X_n\} \to \boldsymbol{O}$. Equivalently to the reduction of the $\mathcal{O}$rc expression, we have:

$$\mathbb{M}_{0,\langle 0,..,0\rangle,\langle v_1,..,v_m\rangle,0,0} * \mathsf{M_k} \xrightarrow{X_j(v'_j),..,X_n(v'_n)} \mathbb{M}_{0,\langle 0,..,0,v'_j,..,v'_n\rangle,\langle v_1,..,v_m\rangle,0,0} * \mathsf{M_k}$$

Note that $\{\widehat{X_j(v'_j),..,X_n(v'_n)}\} = \{\tau\}$. We can then define a relation

$$\begin{array}{rl} R'_M = & \{(M(x_1, .., x_{i-1}, v'_i, .., v'_n, v_1, .., v_m), \mathbb{M}_{0,\langle 0,..,0,v'_j,..,v'_n\rangle,\langle v_1,..,v_m\rangle,0,0} * \mathsf{M_k}) \\ & \quad \mid 1 \le i \le n\} \\ & \cup \{(?k, \mathbb{M}_{1,\langle 0,..,0\rangle,\langle 0,..,0\rangle,0,0} * \mathsf{M_k}), \\ & \quad (let(v'), \mathbb{M}_{1,\langle 0,..,0\rangle\langle 0,..,0\rangle,1,v'} * \mathsf{M_k}), \\ & \quad (\mathbf{0}, \mathbb{M}_{1,\langle 0,..,0\rangle,\langle 0,..,0\rangle,1,0} * \mathsf{M_k})\} \end{array}$$

The second part of R'_M is a bisimulation for the same reasons we presented to show that the relation R_M is a bisimulation, defined in the first case of the proof. The main difference with R_M is the R'_M has also all the possible combinations for when there are variables that are not instantiated. In this case $(h, [\![h]\!]) \in R'_M$, and the fact that the only possible behaviour of an expression with the same format as h and its translation is the behaviour described before yields that R'_M is in fact a bisimulation.

- $h = f \mid g$
Let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G} = [\![g]\!]$. We know by the induction hypothesis that there exist two bisimulations, $\approx_f$ and $\approx_g$, such that $f \approx_f \mathsf{F}$ and $g \approx_g \mathsf{G}$. Let v be a data value. Based on these bisimulations, we define $\approx$ such that $h \approx [\![h]\!]$:

$$\begin{array}{rcl} \approx & = & \{(f' \mid g', F' * \|_{0,0,0} * G')\ ,\ (f' \mid g', F' * \|_{1,v,v} * G') \\ & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{-I}\} \\ & \cup & \{(f' \mid g', F' * \|_{1,0,v} * G') \\ & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I}\} \\ & \cup & \{(f' \mid g', F' * \|_{1,v,0} * G') \\ & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{+I}\} \\ & \cup & \{(f' \mid g', F' * \|_{1,0,0} * G') \\ & & \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{+I}\} \end{array}$$

We now prove that $\approx$ is a weak bisimulation as defined in Definition 2.8, by proving that the two implications, numbered by (i) and (ii), are verified for every element in $\approx$.

– $f' \mid g' \approx F' * \|_{0,0,0} * G'$ where $f' \approx_f F'$, $g' \approx_g G'$, $F' \in \mathsf{F}^{-I}$, and $G' \in \mathsf{G}^{-I}$.
  (i) The possible reduction steps for $f' \mid g'$ are:

$$f' \mid g' \xrightarrow{a} f'' \mid g' \qquad f' \mid g' \xrightarrow{b} f' \mid g'' \qquad f' \mid g' \xrightarrow{a,b} f'' \mid g''$$

  Since $f' \approx_f F'$ and $g' \approx_g G'$, we can conclude for each of these cases:
  (i.1) Exists F'' and b such that $\widehat{a'} = a$, $F' \overset{a'}{\Rightarrow} F''$ and $f'' \approx_f F''$. With respect to F'' we can still consider two cases:
    (i.1.1) $F'' \in \mathsf{F}^{-I}$: The input node is not fired so $F' * \|_{0,0,0} * G' \overset{a'}{\Rightarrow} F'' * \|_{0,0,0} * G'$ and $f'' \mid g' \approx F'' * \|_{0,0,0} * G'$ (because $f'' \approx_f F'' \wedge g' \approx_g G' \wedge F'' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{-I}$).
    (i.1.2) $F'' \in \mathsf{F}^{+I}$: The input node is fired so $F' * \|_{0,0,0} * G' \overset{\tau}{\Rightarrow} F' * \|_{1,v,v} * G' \overset{a'}{\Rightarrow} F'' * \|_{1,0,v} * G'$ and $f'' \mid g' \approx F'' * \|_{1,0,v} * G'$ (because $f'' \approx_f F'' \wedge g' \approx_g G' \wedge F'' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I}$).
  (i.2) Analogous to (i.1).
  (i.3) Exists F'', G'', a', and b' such that $\widehat{a'} = a$, $\widehat{b'} = b$, $F' \overset{a}{\Rightarrow} F''$, $G' \overset{b}{\Rightarrow} G''$, $f'' \approx_f F''$, and $g'' \approx_g G''$. With respect to F'' and G'' we have four cases:
    (i.3.1) $F'' \in \mathsf{F}^{+I}$ and $G'' \in \mathsf{G}^{-I}$. The input node of F' is fired and the input node of G' is not fired. This means that $F' * \|_{0,0,0} * G' \xrightarrow{\tau} F' * \|_{1,v,v} * G' \xrightarrow{a',b'} \|_{1,0,v} * F'' * G''$, and $f'' \mid g'' \approx F'' * \|_{1,0,v} * G''$, for some data value v (because $f'' \approx_f F'' \wedge g'' \approx_g G'' \wedge F'' \in \mathsf{F}^{+I} \wedge G'' \in \mathsf{G}^{-I}$).
    (i.3.2) $F'' \in \mathsf{F}^{-I}$ and $G'' \in \mathsf{G}^{+I}$. Analogous to (i.3.1).
    (i.3.3) $F'' \in \mathsf{F}^{-I}$ and $G'' \in \mathsf{G}^{-I}$. Analogous to (i.3.1).
    (i.3.4) $F'' \in \mathsf{F}^{+I}$ and $G'' \in \mathsf{G}^{+I}$. Analogous to (i.3.1).
  (ii) The possible reduction steps for $F' * \|_{0,0,0} * G'$ are:
  (ii.1) $\|_{0,0,0} * F' * G' \xrightarrow{\tau} F' * \|_{1,v,v} * G'$.
  We know that $f' \mid g' \overset{\tau}{\Rightarrow} f' \mid g'$, $f' \approx_f F'$, $g' \approx_g G'$, $F' \in \mathsf{F}^{-I}$, and $F' \in \mathsf{F}^{-I}$, therefore $f' \mid g' \approx F' * \|_{1,v,v} * G'$.
  (ii.2) $F' * \|_{0,0,0} * G' \xrightarrow{\tau} F'' * \|_{0,0,0} * G'$.
  Since $F' \in \mathsf{F}^{-I}$ and the input node of F' is not fired (because of the combinator $\|_{0,0,0}$), then, by Corollary 2.6, $F'' \in \mathsf{F}^{-I}$. Since $F' \xrightarrow{\tau} F''$, then $f' \overset{\tau}{\Rightarrow} f''$ and $f'' \approx_f F''$. Therefore $f' \mid g' \overset{\tau}{\Rightarrow} f'' \mid g'$ and $f'' \mid g' \approx F'' * \|_{0,0,0} * G'$.
  (ii.3) $F' * \|_{0,0,0} * G' \xrightarrow{\tau} F' * \|_{0,0,0} * G''$.
  Analogous to (ii.2)
  (ii.4) Combination of the previous cases, for which the proves are analogous: (ii.2) and (ii.3), (ii.1) and (ii.2), (ii.1) and (ii.3), and (ii.1), (ii.2) and (ii.3).

– $f' \mid g' \approx F' * \|_{1,v,v} * G'$ where $f' \approx_f F'$, $g' \approx_g G'$, $F' \in \mathsf{F}^{-I}$, and $G' \in \mathsf{G}^{-I}$.
  This case is very similar to the previous one.

– $f' \mid g' \approx F' * \|_{1,0,v} * G'$ where $f' \approx_f F'$, $g' \approx_g G'$, $F' \in \mathsf{F}^{+I}$, and $G' \in \mathsf{G}^{-I}$.
  (i) The possible reduction steps for $f' \mid g'$ are:

$$f' \mid g' \xrightarrow{a} f'' \mid g' \qquad f' \mid g' \xrightarrow{b} f' \mid g'' \qquad f' \mid g' \xrightarrow{a,b} f'' \mid g''$$

  Since $f' \approx_f F'$ and $g' \approx_g G'$, we can conclude for each of these cases:
  (i.1) Exists F'' and a' such that $\widehat{a'} = a$, $F' \overset{a'}{\Rightarrow} F''$ and $f'' \approx_f F''$. Since $F' \in \mathsf{F}^{+I}$, then by Corollary 2.6 $F'' \in \mathsf{F}^{+I}$. Therefore $F' * \|_{1,0,v} * G' \overset{a'}{\Rightarrow} F'' * \|_{1,0,v} * G'$ and $f'' \mid g' \approx F'' * \|_{1,0,v} * G'$ (because $f'' \approx_f F'' \wedge g' \approx_g G' \wedge F'' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I}$).
  (i.2) Exists G'' and b' such that $\widehat{b'} = b$, $G' \overset{b}{\Rightarrow} G''$ and $g'' \approx_g G''$. With respect to G'' we can still consider two cases:
    (i.2.1) $G'' \in \mathsf{G}^{-I}$: The input node is not fired so $F' * \|_{1,0,v} * G' \overset{b'}{\Rightarrow} F' * \|_{1,0,v} * G''$ and $f' \mid g'' \approx F' * \|_{1,0,v} * G''$ (because $f' \approx_f F' \wedge g'' \approx_g G'' \wedge F' \in \mathsf{F}^{+I} \wedge G'' \in \mathsf{G}^{-I}$).

(i.2.2) $G'' \in \mathsf{G}^{+I}$: The input node is fired so $F' * \|_{1,0,v} * G' \overset{b'}{\Rightarrow} \|_{1,0,0} * F' * G''$ and $f' \mid g'' \approx F' * \|_{1,0,0} * G''$ (because $f' \approx_f F' \wedge g'' \approx_g G'' \wedge F' \in \mathsf{F}^{+I} \wedge G'' \in \mathsf{G}^{+I}$).

(i.3) Exists F'', G'', a', and b' such that $\widehat{a'} = a$, $\widehat{b'} = b$, $F' \overset{a'}{\Rightarrow} F''$, $G' \overset{b'}{\Rightarrow} G''$, $f'' \approx_f F''$, and $g'' \approx_g G''$. We know $F'' \in \mathsf{F}^{+I}$, as explained in (i.1). With respect to G'' we have two cases:

(i.3.1) $G'' \in \mathsf{G}^{-I}$: The input node is not fired so $F' * \|_{1,0,v} * G' \overset{a',b'}{\Rightarrow} F'' * \|_{1,0,v} * G''$ and $f'' \mid g'' \approx F'' * \|_{1,0,v} * G''$ (because $f'' \approx_f F'' \wedge g'' \approx_g G'' \wedge F'' \in \mathsf{F}^{+I} \wedge G'' \in \mathsf{G}^{-I}$).

(i.3.2) $G'' \in \mathsf{G}^{+I}$: The input node is fired so $F'' * \|_{1,0,v} * G' \overset{a',b'}{\Rightarrow} F'' * \|_{1,0,0} * G''$ and $f'' \mid g'' \approx F'' * \|_{1,0,0} * G''$ (because $f'' \approx_f F'' \wedge g'' \approx_g G'' \wedge F'' \in \mathsf{F}^{+I} \wedge G'' \in \mathsf{G}^{+I}$).

(ii) The possible reduction steps for $F' * \|_{1,0,v} * G'$ are:

(ii.1) $\|_{1,0,v} * F' * G' \xrightarrow{a} F'' * \|_{1,0,v} * G'$.
By Corollary 2.6, $F'' \in \mathsf{F}^{+I}$. Since $F' \xrightarrow{a} F''$, then $f' \overset{\widehat{a}}{\Rightarrow} f''$ and $f'' \approx_f F''$. Therefore $f' \mid g' \overset{\widehat{a}}{\Rightarrow} f'' \mid g'$ and $f'' \mid g' \approx F' * \|_{1,0,v} * G'$.

(ii.2) $F' * \|_{1,0,v} * G' \xrightarrow{b} F' * \|_{1,0,v} * G''$.
Since $G' \in \mathsf{G}^{-I}$ and the input node of G' is not fired (because of the combinator $\|_{1,0,v}$), then by Corollary 2.6 $G'' \in \mathsf{G}^{-I}$. Since $G' \xrightarrow{b} G''$, then $g' \overset{\widehat{b}}{\Rightarrow} g''$ and $g'' \approx_g G''$. Therefore $f' \mid g' \overset{\widehat{b}}{\Rightarrow} f' \mid g''$ and $f' \mid g'' \approx F' * \|_{1,0,v} * G''$.

(ii.3) $F' * \|_{1,0,v} * G' \xrightarrow{b} F' * \|_{1,0,0} * G''$.
Since $G' \in \mathsf{G}^{-I}$ and the combinator $\|_{1,0,v}$ evolves to $\|_{1,0,0}$, then the input node is fired, *i.e.*, $G'' \in \mathsf{G}^{+I}$. Since $G' \xrightarrow{b} G''$, then $g' \overset{\widehat{b}}{\Rightarrow} g''$ and $g'' \approx_g G''$. Therefore $f' \mid g' \overset{\widehat{b}}{\Rightarrow} f' \mid g''$ and $f' \mid g'' \approx F' * \|_{1,0,0} * G''$.

(ii.4) $F' * \|_{1,0,v} * G' \xrightarrow{a,b} F'' * \|_{1,0,v} * G''$.
Analogous to proofs in (ii.1) and (ii.2).

(ii.5) $F'' * \|_{1,0,v} * G' \xrightarrow{a,b} F'' * \|_{1,0,0} * G''$.
Analogous to proofs in (ii.1) and (ii.3).

– The remaining cases are analogous.

Before proving the cases when $h = f >x> g$ and when $h = g$ **where** $x :\in f$, we introduce another necessary lemma.

Lemma A.1. *(The same as Lemma 2.9.) Let $h \in \mathcal{O}\mathsf{rc}^-$ and $h_v \overset{def}{=} [v/x].h$. We claim that substitution does not change the behaviour of the translation,* i.e.,

$$\textit{If } h \sim [\![h]\!] \textit{ and } [\![h]\!] \xrightarrow{X(v)} H_v \textit{ then } h_v \sim H_v,$$

where x is a free variable in h, v is a data value, and H_v is obtained by sending value v in node X.

Proof. We need to prove that $h_v \sim H_v$, where h_v is obtained by substituting variable x by value v, and H_v is obtained by sending value v in node X. Recall that we are assuming that each variable name is unique, and node X is associated with variable x. Using induction on the structure of h, we can easily verify that, for $h = f \mid g$, $h = f >y> g$, or $h = g$ **where** $y :\in f$, where $x \neq y$, the result follows directly. The node X can only exist in $[\![f]\!]$ and $[\![h]\!]$ (and can be fired), and if the property holds for f and g, then it also holds for h. Therefore, it is enough to consider the case when $h = M(x_1, .., x_n, v_1, .., v_m)$. Furthermore, the only relevant case is when $x \in \{x_1, .., x_n\}$.

We now consider, without lost of generality, that $h = M(x_1, .., x_n, x, v_1, .., v_m)$. Then $h_v = M(x_1, .., x_n, v, v_1, .., v_m)$, and $[\![h_v]\!] = \mathbb{M}_{0,\langle 0,..,0\rangle,\langle v,v_1,..,v_m\rangle} * \mathsf{M}_\mathsf{k}$. Since h and h_v are calls to primitive sites, then we can conclude, by the beginning of the proof of Theorem 2.10, that $h \sim [\![h]\!]$ and $h_v \sim [\![h_v]\!]$. Consider now the connector $[\![h]\!]$, obtained by rule (d) in Fig. 6. Connector H_v corresponds to the same connector after data v flows in input node X, *i.e.*, $H_v = \mathbb{M}_{0,\langle 0,..,0,v\rangle,\langle v_1,..,v_n\rangle}$. It is now enough to prove that the possible behaviour of $[\![h_v]\!]$ and H_v is the same, since the assumption that $h_v \sim [\![h_v]\!]$ will guarantee that also $h_v \sim H_v$.

The only difference between $[\![h_v]\!]$ and H_v is that in the former the value v is in *FIFO1* channel whose input end is connected to a primitive that never returns data, while in the latter the value v is in a *FIFO1* channel whose input end is connected to a One Time node labelled by one. The One Time node, although also connected to node X, will always guarantee that no flow will occur on any of its ends, which corresponds to the behaviour of the primitive that never returns data and a node that can never flow data again. Therefore $[\![h_v]\!]$ and H_v have the same behaviour. □

- $h = f >x> g$
  Let $n = \#f$, and $1 \leq j \leq n$. Also let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G}_j = [\![g]\!]$. We know by the induction hypothesis that there exist $n+1$ bisimulations, $\approx_f$ and $\approx_{gj}$, such that $f \approx_f \mathsf{F}$ and $g \approx_{gj} \mathsf{G_j}$. Based on these, we define $\approx$ such that $h \approx [\![h]\!]$:

$$
\begin{aligned}
\approx \quad = \quad & \{(f' >x> g', F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'n) \\
& \qquad \mid f' \approx_f F' \wedge g' \approx_{gj} G'_j \wedge F' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}\} \\
\cup \quad & \{(f' >x> g' \mid [v_1/x].g' \mid \ \ldots \ \mid [v_r/x].g', F' * \lfloor x \rfloor_{1,\langle \alpha_1,..,\alpha_n\rangle} * \\
& \qquad G'_1 * \ldots * G'_n) \\
& \qquad \mid f' \approx_f F' \quad \wedge \quad F' \in \mathsf{F}^{+I} \quad \wedge \quad v_1, .., v_n \text{ are values} \\
& \qquad \wedge \quad \forall_{m \in \{1,..,r\}}.([v_m/x].g' \approx_{gs} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \\
& \qquad \wedge \quad \forall_{m \in \{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0)\}
\end{aligned}
$$

  We now prove that $\approx$ is a weak bisimulation as defined in Definition 2.8, by proving that the two implications, numbered by (i) and (ii), are verified for every element in $\approx$.

  - $f' >x> g' \approx F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * G'_n$ where $f' \approx_f F'$, $g' \approx_{gj} G'_j$, $F' \in \mathsf{F}^{-I}$, and $G'_j \in \mathsf{G}^{-I}$.
    (i) Since $F' \in \mathsf{F}^{-I}$, then by Corollary 2.6 F' cannot produce any observation of the form $!v$. This and the fact that $f' \approx_f F'$ implies that possible reduction steps of f' can only be $f' \xrightarrow{a} f''$, where $!v \notin a$. Therefore the only possible reduction step of $f' >x> g'$ is by a to $f'' >x> g'$.

    Since $f' \approx_f F'$, we know that exists F'' and a' such that $\widehat{a'} = a$, $F' \overset{a'}{\Rightarrow} F''$ and $f'' \approx_f F''$.With respect to F'' we can consider two cases:

    (i.1) $F'' \in \mathsf{F}^{-I}$: The input node is not fired, so $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a'} F'' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ and $f'' >x> g' \approx F'' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge g' \approx_g G'_j \wedge F'' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}$).

    (i.2) $F'' \in \mathsf{F}^{+I}$: The input node is fired, so $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a'} F'' * \lfloor x \rfloor_{1,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$, and $f'' >x> g' \approx F'' * \lfloor x \rfloor_{1,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge g' \approx_g G'_j \wedge F'' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}$).

    (ii) The behavour of $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ depends mainly on F'. The possible reduction steps of $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ are:

    (ii.1) If $F' \xrightarrow{a} F''$, then, with respect to F'', the possible behaviour of the connector is:

    (ii.1.1) $F'' \in \mathsf{F}^{-I}$: The input node is not fired, so $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a} F'' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$. This corresponds to the reduction step $f' >x> g' \overset{\widehat{a}}{\Rightarrow} f'' >x> g'$, and $f'' >x> g' \approx F'' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge g' \approx_g G'_j \wedge F'' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}$).

    (ii.1.2) $F'' \in \mathsf{F}^{+I}$: The input node is fired, so $F' * \lfloor x \rfloor_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a} F'' * \lfloor x \rfloor_{1,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$. This corresponds to the reduction step $f' >x> g' \overset{\widehat{a}}{\Rightarrow} f'' >x> g'$, and $f'' >x> g' \approx F'' * \lfloor x \rfloor_{1,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge g' \approx_g G'_j \wedge F'' \in \mathsf{F}^{+I} \wedge G'_j \in \mathsf{G}^{-I}$).

    (ii.2) Let $F' \xrightarrow{a} F''$, and let the arity of each G'_j be $\{I_{gj}\} \cup \vec{V_g} \rightarrow \vec{O_{gj}}$. As described in (i), $!v \notin a$, and none of the output nodes are fired (so the buffers will not change their values). The input node of each G'_j has not been fired yet ($G'_j \in \mathsf{G}^{-I}$), therefore $\vec{O_{gj}}$ cannot be fired. The input node cannot be fired because the associated FIFO1 channel is empty. The only possible behaviour is then $G'_j \xrightarrow{X} G''_j$, where $\mathsf{nodes}(X) \subseteq \vec{V_g}$. Note that, since each node in $\vec{V_g}$ is common to every connector G_j, then they will be all triggered in the same step. We conclude that the possible behaviour, with respect to G_j, is: $F' * \langle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle} * G'_1 * \ldots * G'_n \xrightarrow{X} F' * \rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle} * G''_1 * \ldots * G''_n$. Note that $g' \approx_{gj} G'_j$ and $G'_j \xrightarrow{X} G''_j$, where $\widehat{X} = \{\tau\}$, therefore $g' \xrightarrow{\tau} g''$, and $g'' \approx_{gj} G''_j$. We can then conclude that $f' >x> g'' \approx F' * \rangle\!\rangle x \rangle\!\rangle_{0,\langle 0,..,0\rangle} * G''_1 * \ldots * G''_n$ (because $f' \approx_f F' \wedge g'' \approx_g G''_j \wedge F'' \in \mathsf{F}^{-I} \wedge G'_j \in \mathsf{G}^{-I}$).

– $f' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g \approx F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n$, where $f' \approx_f F' \wedge F' \in \mathsf{F}^{+I} \wedge v_1, .., v_n$ are values $\wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0)$. Let h' and H' be these two elements of the composition. We have to prove that for every possible behaviour of h' and H', the bisimulation conditions still apply.

(i) Since h' is a parallel composition of several $\mathcal{O}\mathsf{rc}$ expressions, the possible behaviour of h' is:

(i.1) if $f' \xrightarrow{!v_{r+1},..,!v_s,a} f''$, where $!w \notin a$, then $h' \xrightarrow{a} f'' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g \mid [v_{r+1}/x].g \mid \ldots \mid [v_s/x].g$. Since $f' \approx_f F'$, then exists a' such that $\widehat{a'} = \{!v_{r+1}, .., !v_s, a\}$ and $F \xrightarrow{a'} F''$. Furthermore, a' must be equal to $\{!\mathtt{k}_1(v_{r+1}), .., !\mathtt{k}_{\mathtt{s-r+1}}(v_s), a''\}$, where $\{!\mathtt{k}_1, .., !\mathtt{k}_{s-r+1}\}$ are output nodes of F', and $\widehat{a''} = a$. Let then $\langle \alpha'_1, .., \alpha'_n\rangle$ be the new buffer content after the values $\{v_{r+1}, .., v_s\}$ flow into the corresponding buffer. Let also, for each output v_t, G''_t be such that $G'_t \xrightarrow{X_t(v_t)} G''_t$, where X_t corresponds to the variable x in $[v_t/x].g$. We know that $F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a''} F'' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha'_1,..,\alpha'_n\rangle} * G'_1 * \ldots * G'_r * G''_{r+1} * \ldots * G''_s * \ldots * G''_n$. We now need to show that the resulting connector is bisimilar to $f'' > x > g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g \mid [v_{r+1}/x].g \mid \ldots \mid [v_s/x].g$. Since $F' \in \mathsf{F}^{+I}$, then also $F'' \in \mathsf{F}^{+I}$. Since the input node of none of G'_j was fired, and $\forall_{m\in\{r+1,..,s\}}.G''_m \in \mathsf{G}^{-I}$, then the restriction regarding the α's is stil valid. Finally, since for every $m \in \{r+1, .., s\}$ we have that $g' \approx_{gm} G'_m$, then by Lemma A.1 we conclude that $[v_m/x].g' \approx_{gm} G''_m$, which guarantees that they are bisimilar.

(i.2) if $[v_m/x].g' \xrightarrow{b} [v_m/x].g''$, then $h' \xrightarrow{b} f' > x > g' \mid [v_1/x].g' \mid \ldots \mid [v_m/x].g'' \mid \ldots \mid [v_r/x].g$. Since $[v_m/x].g' \approx_{gm} G'_m$, then exists b' such that $\widehat{b'} = a$, $G'_m \xrightarrow{b'} G''_m$, and $[v_m/x].g'' \approx_{gm} G''_m$. There are two cases with respect to G''_m.

(i.2.1) if $G''_m \in \mathsf{G}^{-I}$, then also $G'_m \in \mathsf{G}^{-I}$ (by Corolary 2.6). Therefore $F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{b'} F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$. Furthermore, $f' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_m/x].g'' \mid \ldots \mid [v_r/x].g \approx F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$ (because $f' \approx_f F' \wedge F' \in \mathsf{F}^{+I} \wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0)$).

(i.2.2) if $G''_m \in \mathsf{G}^{+I}$, then there are two more cases with respect to G'_m: *(i.2.2.1)* – if $G'_m \in \mathsf{G}^{-I}$, then $F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{b'} F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$. Furthermore, $f' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_m/x].g'' \mid \ldots \mid [v_r/x].g \approx F' * \rangle\!\rangle x \rangle\!\rangle_{1,\langle \alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$ (because $f' \approx_f F' \wedge F' \in \mathsf{F}^{+I} \wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0)$).
(i.2.2.1) – if $G' \in \mathsf{G}^{+I}$, then $I \in \mathsf{nodes}(b')$, where I is the main input node of G'_m. The firing of node I makes the FIFO1 channel attached to it to become empty, *i.e.*, the corresponding α value becomes zero. The FIFO1 must be full since the firing

of node I is guaranteed by assuming that $\approx_{gm}$ is a bisimulation. Let $\langle\alpha'_1, ..\alpha'_n\rangle$ be the new α values after replacing α_m by zero. We can then conclude that $F' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{b'} F' * \|x\|_{1,\langle\alpha'_1,..,\alpha'_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$. Furthermore, $f' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_m/x].g'' \mid \ldots \mid [v_r/x].g \approx F' * \|x\|_{1,\langle\alpha'_1,..,\alpha'_n\rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$ (because $f' \approx_f F' \wedge F' \in \mathsf{F}^{+I} \wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0))$.

(i.3) any combination of (i.1) and (i.2), for which the prove is identical.

(ii) The possible behaviour of $F' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n$ is:

(ii.1) With respect to F', let $F' \xrightarrow{a} F''$. We know that $F' \in \mathsf{F}^{+I}$, therefore $F'' \in \mathsf{F}^{+I}$. Let $\vec{O_f}$ be the output nodes of F'. There are two possible cases:

(ii.1.1) If $\vec{O_f} \cap \mathsf{nodes}(a) = \emptyset$, then we know that $F' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a} F'' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n$. Since $f' \approx_f F'$, then $f' \overset{\widehat{a}}{\Rightarrow} f''$ and $f'' \approx_f F''$. Therefore $f' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g \overset{\widehat{a}}{\Rightarrow} f'' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g$. We conclude that $f'' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_r/x].g \approx F'' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge F'' \in \mathsf{F}^{+I} \wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0))$.

(ii.1.2) If $\vec{O_f} \cap \mathsf{nodes}(a) \neq \emptyset$, then some output nodes of F' are fired. Let $a = a' \uplus \vec{X_f}$ correspond to the partition of a into the output nodes (a') and the input variable nodes ($\vec{X_f}$). Note that, by Lemma 2.3, the input node cannot be fired a second time. Let $a' = \{O_{r+1}(v_{r+1}), .., O_s(v_s)\}$. In this case $\widehat{a} = \widehat{a'} \cup \{\tau\}$. Then we know that $f' \overset{\widehat{a}}{\Rightarrow} f''$. The firing of the output nodes will fill some FIFO1 channels. Let $\langle\alpha'_1, .., \alpha'_n\rangle$ be the values of the FIFO1 channels after the output node are fired. Since for each connector G_m connected to these FIFO1 channels $G_m \in \mathsf{G}^{-I}$, then the conditions over the α's will still hold. Furthermore, the firing of the output nodes will also trigger the actions $X = \{X_{r+1}(v_{r+1}), .., X_s(v_s)\}$ corresponding to the variable x in the connectors $G'_{r+1}, .., G'_s$, who evolve to $G''_{r+1}, .., G''_s$, respectively. We can then conclude that $F' * \|x\|_{1,\langle\alpha_1,..,\alpha_n\rangle} * G'_1 * \ldots * G'_n \xrightarrow{a\cup X} F'' * \|x\|_{1,\langle\alpha'_1,..,\alpha'_n\rangle} * G'_1 * \ldots * G''_{r+1} * \ldots * G''_s * \ldots * G'_n$. We also conclude that $\forall_{m\in\{r+1,..,s\}}.(g' \approx_{gm} G'_m)$, and by Lemma A.1 $[v_m/x].g' \approx_{gm} G''_m$. Therefore $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_r/x].g' \overset{\widehat{a\cup X} = \widehat{a'}\cup\{\tau\}}{\Rightarrow} f'' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_s/x].g'$, and $f'' >x> g' \mid [v_1/x].g \mid \ldots \mid [v_s/x].g \approx F'' * \|x\|_{1,\langle\alpha'_1,..,\alpha'_n\rangle} * G'_1 * \ldots * G''_{r+1} * \ldots * G''_s * \ldots * G'_n$ (because $f'' \approx_f F'' \wedge F'' \in \mathsf{F}^{+I} \wedge \forall_{m\in\{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m\in\{r+1,..,s\}}.([v_m/x].g' \approx_{gm} G''_m \wedge G'_m \in \mathsf{G}^{-I}) \wedge \forall_{m\in\{s+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0))$.

(ii.2) With respect to G'_m, where $1 \leq m \leq r$, we have two different cases:

(ii.2.1) If $G'_m \in \mathsf{G}^{-I}$, then $G'_m \xrightarrow{a} G''_m$, where a, with respect to G''_m, can be:

(ii.2.1.1) – If $G''_m \in \mathsf{G}^{-I}$, then a corresponds to the firing of input nodes associated with variables. If the variable is attached to one of the outputs of F', then this case is the same as (ii.1.2). If is not associated to an output node of F', then the variable is not x, which means it is common to all $F', G_1, .., G_n$. Let $F'', G''_1, .., G''_n$ be the connectors after firing a, and let $a = \{X_1(v'_1), .., X_t(v'_t)\}$. This corresponds to the substitution $\sigma = [v'_1/x_1, .., v'_t/x_t]$ to $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_n/x].g'$, which

corresponds to the application of that sustitution of all occurences of f' and g'. Since $f' \approx_f F'$ and $\forall_{m \in \{1,..,n\}}.(g' \approx_{gm} G'_m)$, then, by Lemma A.1, $\sigma.f' \approx_f F''$ and $\forall_{m \in \{1,..,n\}}.(\sigma.g' \approx_{gm} G''_m)$. We can conclude that $\sigma.f' >x> \sigma.g' \mid [v_1/x].\sigma.g' \mid \ldots \mid [v_n/x].\sigma.g' \approx F'' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G''_1 * \ldots * G''_n$. Note that the application of the substitution is considered to be an internal transition.

(ii.2.1.2) – If $G''_m \in \mathsf{G}^{+I}$, then $I_{gm} \in \mathsf{nodes}(a)$, where I_{gm} is the main input node of G'_m. This also triggers the FIFO1 channel connected to I_{gm}, changing α_m to zero. Let $\langle \alpha'_1, .., \alpha'_n \rangle$ the α values after the step. Then $F' * \rangle x \rangle_{1,\langle \alpha'_1,..,\alpha'_n \rangle} * G'_1 * \ldots * G'_n \xrightarrow{a} F' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$. Since $g' \approx_{gm} G'_m$, then $g' \overset{\widehat{a}}{\Rightarrow} g''$, and we conclude that $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_r/x].g' \overset{\widehat{a}}{\Rightarrow} f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_m/x].g'' \mid [v_r/x].g'$, and $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_m/x].g'' \mid [v_r/x].g' \approx F' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$.

(ii.2.2) If $G'_m \in \mathsf{G}^{+I}$, then $G'_m \xrightarrow{a} G''_m$ and $G''_m \in \mathsf{G}^{+I}$. If a contains a node corresponding to an input variable other than x, then the situation is equivalent to case (ii.2.1.1). Otherwise, since $[v_m/x].g' \approx_{gm} G'_m$, then $[v_m/x].g' \overset{\widehat{a}}{\Rightarrow} [v_m/x].g''$, and $[v_m/x].g'' \approx_{gm} G''_m$. Therefore $F' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G'_1 * \ldots * G'_n \xrightarrow{a} F' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$, $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_r/x].g' \overset{\widehat{a}}{\Rightarrow} f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_m/x].g'' \mid [v_r/x].g'$, and $f' >x> g' \mid [v_1/x].g' \mid \ldots \mid [v_m/x].g'' \mid [v_r/x].g' \approx F' * \rangle x \rangle_{1,\langle \alpha_1,..,\alpha_n \rangle} * G'_1 * \ldots * G''_m * \ldots * G'_n$ (because $f' \approx_f F' \wedge F' \in \mathsf{F}^{+I} \wedge \forall_{m \in \{1,..,r\}}.([v_m/x].g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{+I} \Leftrightarrow \alpha_m = 0) \wedge \forall_{m \in \{r+1,..,n\}}.(g' \approx_{gm} G'_m \wedge G'_m \in \mathsf{G}^{-}I \wedge \alpha_m = 0)$).

(ii.3) With respect to G'_m, where $r+1 \leq m \leq n$, we know that $G'_m \in \mathsf{G}^{-I}$. Let $G'_m \xrightarrow{a} G''_m$. Since $\alpha_m = 0$, then the input node cannot be fired, and therefore $G''_m \in \mathsf{G}^{-I}$. The possible behaviour for G'_m is then to fire input nodes associated with variables. If node X, associated with variable x, is fired, then the corresponding output node of F' is also fired, which corresponds to the case proven in (ii.1.1). If another node is fired, then this corresponds to the case proven in (ii.2.1.1). Combination of these cases follow a similar prove.

(ii.4) Any combination of the (ii.1), (ii.2) and (ii.3), for which the proofs are identicals.

- $h = g$ **where** $x :\in f$

Let $n = \#f$, and $1 \leq j \leq n$. Also let $\mathsf{F} = [\![f]\!]$ and $\mathsf{G} = [\![g]\!]$. We know by the induction hypothesis that there exist 2 bisimulations, $\approx_f$ and $\approx_g$, such that $f \approx_f \mathsf{F}$ and $g \approx_g \mathsf{G}$. Let v be any data value. Based on these bisimulations, we define $\approx$ such that $h \approx [\![h]\!]$:

$$
\begin{aligned}
\approx \quad = \quad & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{0,0,0,0} * G') \\
& ,(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,v,v,0} * G') \\
& \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{-I}\} \\
\cup \quad & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,v,0,0} * G') \\
& \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{-I} \wedge G' \in \mathsf{G}^{+I}\} \\
\cup \quad & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,0,v,\delta} * G') \\
& \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{-I} \wedge \delta \in \{0,1\}\} \\
\cup \quad & \{(g' \textbf{ where } x :\in f', F' * \mathbb{W}^x_{1,0,0,\delta} * G') \\
& \qquad \mid f' \approx_f F' \wedge g' \approx_g G' \wedge F' \in \mathsf{F}^{+I} \wedge G' \in \mathsf{G}^{+I} \wedge \delta \in \{0,1\}\}
\end{aligned}
$$

The proof that $\approx$ is in fact a bisimulation follow similar lines to the previous cases, and is omitted.

□