The Algebra of Connectors - Structuring Interaction in BIP

Simon Bliudze, Joseph Sifakis

Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

The Algebra of Connectors — Structuring Interaction in BIP

Simon Bliudze and Joseph Sifakis

Technical Report n^o TR-2007-3

April 19, 2007

The Algebra of Connectors — Structuring Interaction in BIP

Simon Bliudze and Joseph Sifakis

April 19, 2007

Abstract

We provide an algebraic formalisation of *connectors* in BIP. These are used to structure *interactions* in a component-based system. A connector relates a set of typed ports. Types are used to describe different modes of synchronisation: rendezvous and broadcast, in particular.

Connectors on a set of ports P are modelled as terms of the algebra $\mathcal{AC}(P)$, generated from P by using an n-ary *fusion* operator and a unary *typing* operator. Typing associates with terms (ports or connectors) synchronisation types — *trigger* or *synchron* —, which determine modes of synchronisation. Broadcast interactions are initiated by triggers. Rendezvous is a maximal interaction of a connector including only synchrons.

The semantics of $\mathcal{AC}(P)$ associates with a connector the set of its interactions. It induces on connectors an equivalence relation which is not a congruence as it is not stable for fusion. We provide a number of properties of $\mathcal{AC}(P)$ used to symbolically simplify and handle connectors. We provide examples illustrating applications of $\mathcal{AC}(P)$, including a general component model encompassing synchrony, methods for incremental model decomposition, and efficient implementation by using symbolic techniques.

How to cite this report:

@techreport {TR-2007-3,
title = {The Algebra of Connectors — Structuring Interaction in {BIP}},
authors = {Simon Bliudze and Joseph Sifakis},
institution = {VERIMAG},
number = {TR-2007-3} ,
year = {2007}
}

Contents

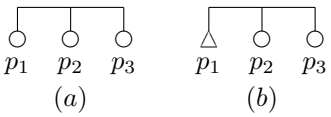


Figure 1: Graphical representation of rendezvous (a) and broadcast (b) connectors.

1 Introduction

A key idea in systems engineering is that complex systems are built by assembling components (building blocks). Components are systems characterised by an abstraction, which is adequate for composition and re-use. Large components are obtained by composing simpler ones. Component-based design confers many advantages such as reuse of solutions, modular analysis and validation, reconfigurability, controllability etc.

Component-based design relies on the separation between coordination and computation. Systems are built from units processing sequential code insulated from concurrent execution issues. The isolation of coordination mechanisms allows a global treatment and analysis.

One of the main limitations of the current state-of-the-art is the lack of a unified paradigm for describing and analysing the coordination between components. Such a paradigm would allow system designers and implementers to formulate their solutions in terms of tangible, well-founded and organised concepts instead of using dispersed low-level coordination mechanisms including semaphores, monitors, message passing, remote call, protocols etc. A unified paradigm should allow a comparison and evaluation of otherwise unrelated architectural solutions, as well as derivation of implementations in terms of specific coordination mechanisms.

A number of paradigms for unifying interaction in heterogeneous systems have been proposed in [Arb05, BWH+03, BGK+06, EJL+03]. In these works unification is achieved by reduction to a common low-level semantic model. Interaction mechanisms and their properties are not studied independently of behaviour. Coordination languages also offer mechanisms for unified and implementation-independent interaction specification, e.g. [BPE, nes]. Nonetheless, these are defined on an ad hoc basis, and there is no underlying theoretical framework.

We propose the *algebra of connectors* for modelling interaction in component-based systems. The algebra allows the description of coordination between components in terms of structured connectors involving communication ports. It formalises mechanisms and concepts that have been implemented in the *Behaviour-Interaction-Priority* (BIP) component framework developed at Verimag [BBS06, Sif05]. BIP distinguishes between three basic entities: 1) Behaviour, described as extended automata, including a set of transitions labelled with communication ports. 2) Interaction, described by structured connectors relating communication ports. 3) Dynamic priorities, used to model simple control policies, allowing selection amongst possible interactions. BIP uses a powerful composition operator parametrised by a set of interactions.

We present an algebraic formalisation of the concept of *connector*, introduced in [GS03, GS05] as a set of communication ports belonging to different components that may be involved in some interaction. To express different types of synchronisation, the ports of a connector have a type (attribute) *trigger* or *synchron*. Given a connector involving as set of ports $\{p_1, \ldots, p_n\}$, the set of its interactions is defined by the following rule: *an interaction is any non empty subset of* $\{p_1, \ldots, p_n\}$ *which contains some port that is a trigger; otherwise, (if all the ports are synchrons) the only possible interaction is the maximal one that is,* $\{p_1, \ldots, p_n\}$.

In Figure 1, we show two connectors modelling respectively rendezvous and broadcast between ports p_1, p_2, p_3. For rendezvous, all the involved ports are synchrons (represented by bullets) and the only possible interaction is $p_1p_2p_3$. As usual, we simplify notation by writing $p_1p_2p_3$ instead of the set $\{p_1, p_2, p_3\}$. For broadcast, p_1 is a trigger (represented by a triangle). The possible interactions are p_1, p_1p_2, p_1p_3, and $p_1p_2p_3$. A connector may have several triggers. For instance, if both p_1 and p_2 are triggers in the above connector, then p_2 and p_2p_3 should be added to the list of possible interactions.

The main contributions of this paper are the following:

- The algebra of connectors extends the notion of connectors to terms built from a set of ports by using

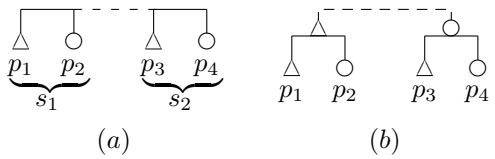


Figure 2: Fusion (a) and structuring (b) of connectors.

a n-ary fusion operator and a unary typing operator (trigger or synchron). Given two connectors involving sets of ports s_1 and s_2, it is possible to obtain by *fusion* a new connector involving the set of ports $s_1 \cup s_2$ (cf. Figure $2(a)$). Ports preserve their types except for the case where some port occurs in both connectors with different types. In this case, the port in the new connector is a trigger. It is also possible to structure connectors hierarchically as shown in Figure $2(b)$, where terms $p_1\, p_2$ and $p_3\, p_4$ are typed and then fused to obtain a new connector.

- The semantics of the algebra of connectors associates with a connector (a term) the set of its interactions. This induces an equivalence on terms. We show that this equivalence is not a congruence as it is not preserved by fusion. This fact has deep consequences on composability of interaction models investigated in the paper. We show that for the subset of the terms where all the connectors have the same type (synchron or trigger) the semantic equivalence is a congruence.
- The algebra and its laws can be used to represent and handle symbolically complex interaction patterns. The number of interactions of a connector can grow exponentially with its size. We provide applications of the algebra in modelling languages, such as BIP, and show that the use of symbolic instead of enumerative techniques can drastically enhance efficiency in execution and transformation.

The paper is structured as follows. Section 2 provides a succinct presentation of the basic semantic model for BIP and in particular, its composition parametrised by interactions. In Section 3, we present the Algebra of Interactions. It is a simple algebra used to introduce the Algebra of Connectors presented in Section 4. The last section discusses possible applications of the algebra of connectors to efficient design, analysis, and execution of languages with complex interaction structure, such as BIP.

2 BIP component framework

2.1 Basic semantic model

BIP is a component framework for constructing systems by superposing three layers of modelling: Behaviour, Interaction, and Priority. The lower layer consists of a set atomic components representing transition systems. The second layer models interactions between components, specified by connectors. These are relations between ports equipped with synchronisation types. Priorities are used to enforce scheduling policies applied to interactions of the second layer.

The BIP component framework has been implemented in a language and a tool-set. The BIP language offers primitives and constructs for modelling and composing layered components. Atomic components are communicating automata extended with C functions and data. Their transitions are labelled with sets of communication ports. The BIP language also allows composition of components parametrised by sets of interactions as well as application of priorities.

The BIP tool-set includes an editor and a compiler for generating from BIP programs, C++ code executable on a dedicated platform (see [BBS06, bip]).

We provide a succinct formalisation of the BIP component model focusing on the operational semantics of component interaction and priorities.

Definition 2.1 For a set of ports P, an *interaction* is a non-empty subset $a \subseteq P$ of ports.

Definition 2.2 A labelled transition system is a triple $B = (Q, P, \rightarrow)$, where Q is a set of *states*, P is a set of *communication ports*, and $\rightarrow \subseteq Q \times 2^P \times Q$ is a set of *transitions*, each labelled by an interaction.

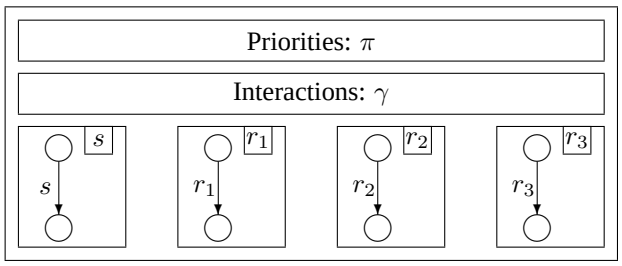


Figure 3: A system with four atomic components

For any pair of states $q, q' \in Q$ and an interaction $a \in 2^P$, we write $q \xrightarrow{a} q'$, iff $(q, a, q') \in \rightarrow$. When the interaction is irrelevant, we simply write $q \rightarrow q'$.

An interaction a is *enabled* in state q, denoted $q \xrightarrow{a}$, iff there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. A port P is *active*, iff it belongs to an enabled interaction.

In BIP, a system can be obtained as the composition of n components, each modelled by a transition system $B_i = (Q_i, P_i, \rightarrow_i)$, for $i \in [1, n]$, such that their sets of ports are pairwise disjoint: for $i, j \in [1, n]$ $(i \neq j)$, we have $P_i \cap P_j = \emptyset$. We take $P = \bigcup_{i=1}^{n} P_i$, the set of all ports in the system.

The *composition* of components $\{B_i\}_{i=1}^{n}$, parametrised by a set of interactions $\gamma \subset 2^P$ is the transition system $B = (Q, P, \rightarrow_\gamma)$, where $Q = \bigotimes_{i=1}^{n} Q_i$ and $\rightarrow_\gamma$ is the least set of transitions satisfying the rule

$$\frac{a \in \gamma \quad \wedge \quad \forall i \in [1, n],\ (a \cap P_i \neq \emptyset \Rightarrow q_i \xrightarrow{a \cap P_i}_i q_i')}{(q_1, \ldots, q_n) \xrightarrow{a}_\gamma (q_1', \ldots, q_n')}, \tag{1}$$

where $q_i = q_i'$ for all $i \in [1, n]$ such that $a \cap P_i = \emptyset$. We write $B = \gamma(B_1 \ \ldots, B_n)$.

Notice that an interaction $a \in \gamma$ is enabled in $\gamma(B_1, \ldots, B_n)$, only if, for each $i \in [1, n]$, the interaction $a \cap P_i$ is enabled in B_i; the states of components that do not participate in the interaction remain unchanged.

Several distinct interactions can be enabled at the same time, thus introducing non-determinism in the product behaviour, which can be restricted by means of priorities.

Definition 2.3 Given a system $B = \gamma(B_1, \ldots, B_n)$, a *priority model* π is a strict partial order on γ. For $a, a' \in \gamma$, we write $a \prec a'$ iff $(a, a') \in \pi$, meaning that interaction a has less priority than interaction a'.

For $B = (Q, P, \rightarrow)$, and a priority model π, the transition system $\pi(B) = (Q, P, \rightarrow_\pi)$, is defined by the rule

$$\frac{q \xrightarrow{a} q' \quad \wedge \quad \nexists\, a' : (a \prec a' \ \wedge \ q \xrightarrow{a'})}{q \xrightarrow{a}_\pi q'}. \tag{2}$$

Notice that an interaction is enabled in $\pi(B)$ only if it is enabled in B, and maximal according to π.

Example 2.4 (Sender/Receivers)
Figure 3 shows a component $\pi\,\gamma(S, R_1, R_2, R_3)$ obtained by composition of four atomic components: a sender, S, and three receivers, R_1, R_2, R_3. The sender has a port s for sending messages, and each receiver has a port r_i $(i = 1, 2, 3)$ for receiving them. The following table specifies γ and π for four different coordination schemes.

	Set of interactions γ	Priority model π
Rendezvous	$\{s\,r_1\,r_2\,r_3\}$	$\emptyset$
Broadcast	$\{s,\ s\,r_1,\ s\,r_2,\ s\,r_3,\ s\,r_1\,r_2,\ s\,r_1\,r_3, s\,r_2\,r_3,\ s\,r_1\,r_2\,r_3\}$	
Atomic Broadcast	$\{s,\ s\,r_1\,r_2\,r_3\}$	$\{(a, a') \mid a \subset a'\}$
Causality Chain	$\{s,\ s\,r_1,\ s\,r_1\,r_2,\ s\,r_1\,r_2\,r_3\}$	

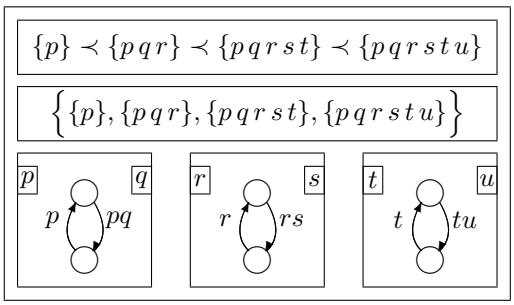


Figure 4: Modulo-8 counter.

Rendezvous means strong synchronisation between S and all R_i. This is specified by a single interaction involving all the ports. This interaction can occur only if all the components are in states enabling transitions labelled respectively by s, r_1, r_2, r_3.

Broadcast means weak synchronisation, that is a synchronisation involving S and any (possibly empty) subset of R_i. This is specified by the set of all interactions containing s. These interactions can occur only if S is in a state enabling s. Each R_i participates in the interaction only if it is in a state enabling r_i.

Atomic broadcast means that either a message is received by all R_i, or by none. Two interactions are possible: s, when at least one of the receiving ports is not enabled, and the interaction $s\,r_1\,r_2\,r_3$, corresponding to strong synchronisation.

Causality chain means that for a message to be received by R_i it has to be received at the same time by all R_j, for $j < i$. This coordination scheme is common in reactive systems.

For rendezvous, the priority model is empty. For all other coordination schemes, whenever several interactions are possible, the interaction involving a maximal number of ports has higher priority, that is we take $\pi = \{(a, a') \mid a \subset a'\}$. ■

Throughout the paper, the above rule is applied. In other words, amongst the enabled interactions, are preferred the ones involving a maximal number of ports.

Example 2.5 (Modulo-8 counter)
Figure 4 shows a model for the Modulo-8 counter, presented in [MR01], obtained by composition of three Modulo-2 counter components. Ports p, r, and t correspond to inputs, whereas q, s, and u correspond to outputs. It can be easily verified that the interactions $p\,q\,r$, $p\,q\,r\,s\,t$, and $p\,q\,r\,s\,t\,u$ happen, respectively, on every second, fourth, and eighth occurrence of an input interaction through the port p. ■

2.2 Modeling parallel composition operations in BIP

The composition operator, introduced in the previous section, can express usual parallel composition operators, such as the ones used in CSP [Hoa85] and CCS [Mil89]. By enforcing maximal progress, priorities allow to express broadcast.

2.2.1 Communicating Sequential Processes (CSP)

In CSP [Hoa85], components can communicate over a set of *channels*, common to the system. Full semantics of CSP can be found for example in [Ros97, Chapter 7], whereas, as announced above, we will limit ourselves to the most essential case.

Atomic components (processes) in CSP can be considered as labelled transition systems consisting of a triple $(Q, C, \rightarrow)$, where Q is the set of states, C is the set of communication channels, and $\rightarrow \subset Q \times C \times Q$ is the set of state transitions labelled by channels from C.

Thus, for two components $B_i = (Q_i, C, \rightarrow_i)$ with $i = 1, 2$ and a subset $C' \subset C$, parallel composition $B_1||_{C'}B_2$ can be defined by the following rules, where we assume $q_i, q_i' \in Q_i$ for $i = 1, 2$ and $c \in C$.

$$\frac{q_1 \xrightarrow{c}_1 q_1' \quad \wedge \quad q_2 \xrightarrow{c}_2 q_2'}{q_1 \,||_{C'}\, q_2 \xrightarrow{c} q_1' \,||_{C'}\, q_2'}, \qquad \text{for any } c \in C' \,, \tag{3}$$

$$\frac{q_1 \xrightarrow{c}_1 q_1'}{q_1 \,||_{C'}\, q_2 \xrightarrow{c} q_1' \,||_{C'}\, q_2} \quad \text{and} \quad \frac{q_2 \xrightarrow{c}_2 q_2'}{q_1 \,||_{C'}\, q_2 \xrightarrow{c} q_1 \,||_{C'}\, q_2'}, \quad \text{for any } c \notin C' \,.$$

To construct an equivalent system in BIP, we consider two components $\widetilde{B_i} = (Q_i, B_i.C, \rightarrow_i)$ with $B_i.C \stackrel{def}{=} \{B_i.c \,|\, c \in C\}$ for $i = 1, 2$. An interaction model corresponding to (3) is then defined by taking

$$\gamma_{CSP} = \left\{ \{B_1.c, B_2.c\} \,\middle|\, c \in C' \right\} \cup \left\{ \{B_1.c\} \,\middle|\, c \notin C' \right\} \cup \left\{ \{B_2.c\} \,\middle|\, c \notin C' \right\}.$$

2.2.2 Calculus of Communicating Systems (CCS)

In CSS [Mil89], all communication is performed by binary interactions between complementary actions a and $\overline{a}$. Denoting by A the set of actions, one considers the set of labels $L = A \cup \overline{A} \cup \{\tau\}$, where τ represents an internal (non-observable transition).

Thus, for two components $B_i = (Q_i, L, \rightarrow_i)$ with $i = 1, 2$, a parallel composition $B_1||B_2$ can be defined by the following rules, where we assume $q_i, q_i' \in Q_i$ for $i = 1, 2$ and $a \in A$.

$$\frac{q_1 \xrightarrow{a}_1 q_1' \quad \wedge \quad q_2 \xrightarrow{\overline{a}}_2 q_2'}{q_1 \,||\, q_2 \xrightarrow{\tau} q_1' \,||\, q_2'}, \quad \frac{q_1 \xrightarrow{l}_1 q_1', \quad l \in \{a, \overline{a}\}}{q_1 \,||\, q_2 \xrightarrow{l} q_1' \,||\, q_2} \quad \text{and} \quad \frac{q_2 \xrightarrow{l}_2 q_2', \quad l \in \{a, \overline{a}\}}{q_1 \,||\, q_2 \xrightarrow{l} q_1 \,||\, q_2'}. \tag{4}$$

Another important operation in CSS is restriction $B \setminus a$, which excludes a given action from communication. Thus, in $(B_1||B_2) \setminus a$, restriction enforces synchronisation between B_1 and B_2.

As in the previous section, we model this by considering two components $\widetilde{B_i} = (Q_i, B_i.L, \rightarrow_i)$ with $B_i.L \stackrel{def}{=} \{B_i.l \,|\, l \in L\}$ for $i = 1, 2$. A set of interactions corresponding to $B_1||B_2$ is then defined by putting

$$\gamma_{CCS,1} = \left\{ \{B_i.a, B_{3-i}.\overline{a}\} \,\middle|\, i = 1, 2 \right\} \cup \left\{ \{B_1.l\} \,\middle|\, l \in L \right\} \cup \left\{ \{B_2.l\} \,\middle|\, l \in L \right\}.$$

The only modification to do, in order to account for restriction in $(B_1||B_2) \setminus a$, is then to exclude a and $\overline{a}$ from possible singleton interactions. Thus we put

$$\gamma_{CCS,2} = \left\{ \{B_i.a, B_{3-i}.\overline{a}\} \,\middle|\, i = 1, 2 \right\} \cup \left\{ \{B_1.l\} \,\middle|\, l \in L \setminus \{a, \overline{a}\} \right\} \cup \left\{ \{B_2.l\} \,\middle|\, l \in L \setminus \{a, \overline{a}\} \right\}.$$

3 The algebra of interactions

We define the algebra of interactions that will serve as a base for building the algebra of connectors.

3.1 Syntax, axiomatisation, and semantics

Consider a family of components, indexed by I and equipped with a set of ports P_i, for $i \in I$, through which it can communicate with the others. The communication model considered implies atomic synchronisation of all ports participating in a given interaction. Therefore, each interaction is represented by the set of ports it involves. Accordingly, each element in the algebra of interactions, which we define below, should be considered as a set of possible interactions.

Syntax
Let $P = \cup_{i\in I} P_i$ be a set of all ports of the system, and assume that $0, 1 \notin P$. The syntax of the *algebra of interactions*, $\mathcal{AI}(P)$, is defined by

$$x \quad ::= \quad 0 \mid 1 \mid p \mid x \cdot x \mid x + x \mid (x)\,, \tag{5}$$

with $p \in P$ an arbitrary port, and where '$+$' and '$\cdot$' are binary operators, respectively called *union* and *synchronisation*. Synchronisation has a higher order of precedence than union.

Axioms
The operations satisfy the following axioms.

1. Union '$+$' is idempotent, associative, commutative, and has an identity element 0, i.e. $(\mathcal{AI}(P), +, 0)$ is a commutative monoid;

2. Synchronisation '$\cdot$' is idempotent, associative, and commutative, has an identity element 1, and an absorbing element 0; synchronisation distributes over union, i.e. $(\mathcal{AI}(P), +, \cdot, 0, 1)$ is a commutative semi-ring.

Semantics
The semantics of $\mathcal{AI}(P)$ is defined by the function $|\cdot| : \mathcal{AI}(P) \to 2^{2^P}$, defined by

$$\begin{array}{lcl}
|0| & = & \emptyset, \\
|1| & = & \{\emptyset\}, \\
|p| & = & \Big\{\{p\}\Big\}, \text{ for any } p \in P, \\
|x_1 + x_2| & = & |x_1| \cup |x_2|, \text{ for any } x_1, x_2 \in \mathcal{AI}(P), \\
|x_1 \cdot x_2| & = & \Big\{x_1 \cup x_2 \,\Big|\, x_1 \in |x_1|, x_2 \in |x_2|\Big\}, \text{ for any } x_1, x_2 \in \mathcal{AI}(P), \\
|(x)| & = & |x|, \text{ for any } x \in \mathcal{AI}(P)\,,
\end{array} \tag{6}$$

for arbitrary $p \in P$, $x, x_1, x_2 \in \mathcal{AI}(P)$. Terms of $\mathcal{AI}(P)$ represent sets of interactions between the ports of P.

Proposition 3.1 *The axiomatisation of $\mathcal{AI}(P)$ is sound and complete, that is, for any $x, y \in \mathcal{AI}(P)$,*

$$x = y \Longleftrightarrow \|x\| = \|y\|\,.$$

Example 3.2 (Sender/Receiver continued)
In $\mathcal{AI}(P)$, the interaction for the four coordination schemes of Example 2.4 are:

	Set of interactions
Rendezvous	$s\, r_1\, r_2\, r_3$
Broadcast	$s\,(1 + r_1)\,(1 + r_2)\,(1 + r_3)$
Atomic Broadcast	$s\,(1 + r_1\, r_2\, r_3)$
Causality Chain	$s\,(1 + r_1\,(1 + r_2\,(1 + r_3)))$

Clearly, this representation is more compact and exhibits more information: e.g. the expression $(1 + r_i)$ suggests that the port r_i is optional. ■

$\mathcal{AI}(P)$	$\mathbb{B}[P]$
0	$false$
$1 \quad p \quad q \quad p\,q$	$\overline{p}\,\overline{q} \quad p\,\overline{q} \quad \overline{p}\,q \quad p\,q$
$p+1 \quad q+1 \quad p\,q+1 \quad p+q \quad p+p\,q \quad q+p\,q$	$\overline{q} \quad \overline{p} \quad \overline{p}\,\overline{q} \vee p\,q \quad p\,\overline{q} \vee \overline{p}\,q \quad p \quad q$
$p+q+1 \quad p\,q+p+1 \quad p\,q+q+1 \quad p\,q+p+q$	$\overline{p} \vee \overline{q} \quad p \vee \overline{q} \quad \overline{p} \vee q \quad p \vee q$
$p\,q+p+q+1$	$true$

Figure 5: Correspondence between $\mathcal{AI}(\{p,q\})$ and boolean functions with two variables.

3.2 Correspondence with boolean functions

$\mathcal{AI}(P)$ can be bijectively mapped to the free boolean algebra $\mathbb{B}[P]$ generated by P. We define a mapping $\beta : \mathcal{AI}(P) \to \mathbb{B}[P]$ by setting:

$$\begin{aligned} \beta(0) &= false\,, \qquad \beta(1) = \bigwedge_{p\in P} \overline{p}\,, \\ \beta(p_{i_1}\dots p_{i_k}) &= \bigwedge_{j=1}^{k} p_{i_j} \wedge \bigwedge_{i\neq i_j} \overline{p_i}\,, \\ \beta(x+y) &= \beta(x) \vee \beta(y)\,, \end{aligned}$$

for $p_{i_1},\dots p_{i_k} \in P$, and $x, y \in \mathcal{AI}(P)$, where in the right-hand side the elements of P are considered to be boolean variables. For example, consider the correspondence table for $P = \{p, q\}$ shown in Figure 5.

The mapping β is an order isomorphism, and each expression $x \in \mathcal{AI}(P)$ represents exactly the set of interactions corresponding to boolean valuations of P satisfying $\beta(x)$.

Although techniques specific to boolean algebras can be applied to the boolean representation of $\mathcal{AI}(P)$ (e.g. BDDs), $\mathcal{AI}(P)$ provides a more natural representation of interactions for two reasons.

1. Representation in $\mathcal{AI}(P)$ is more intuitive as it gives directly all the interactions. For example, the term $p + p\,q$ of $\mathcal{AI}(P)$ represents the set of interactions $\{p, p\,q\}$ for any set of ports P containing p and q. The boolean representation of $p + p\,q$ depends on P: if $P = \{p, q\}$ then $\beta(p + pq) = p$, whereas if $P = \{p, q, r, s\}$ then $\beta(p + pq) = p\,\overline{r}\,\overline{s}$.

2. Synchronisation of two interactions in $\mathcal{AI}(P)$ is by simple concatenation, whereas for their boolean representation there is no simple context-independent composition rule, e.g. to obtain the representation of $p\,q$ from $\beta(p) = p\,\overline{q}\,\overline{r}\,\overline{s}$ and $\beta(q) = \overline{p}\,q\,\overline{r}\,\overline{s}$.

4 The algebra of connectors

We provide an algebraic formalisation of the concept of connector, supported by the BIP language [BBS06]. Connectors can express complex coordination schemes combining synchronisation by rendezvous and broadcast.

4.1 Syntax, axioms, and semantics

Syntax
Let P be a set of ports, such that $0, 1 \notin P$. The syntax of the *algebra of connectors*, $\mathcal{AC}(P)$, is defined by

$$\begin{array}{rcll} s & ::= & [0] \mid [1] \mid [p] \mid [x] & (synchrons) \\ t & ::= & [0]' \mid [1]' \mid [p]' \mid [x]' & (triggers) \\ x & ::= & s \mid t \mid x \cdot x \mid x + x \mid (x)\,, & \end{array} \tag{7}$$

for $p \in P$, and where '$+$' is binary operator called *union*, '$\cdot$' is an n-ary operator called *fusion*, and brackets '$[\cdot]$' and '$[\cdot]'$' are unary *typing* operators. Fusion has a higher order of precedence than union.

Union has the same meaning as union in $\mathcal{AI}(P)$. Fusion is a generalisation of the synchronisation in $\mathcal{AI}(P)$. It is not associative. Typing is used to form typed connectors: '$[\cdot]'$' defines *triggers* (which can initiate an interaction), and '$[\cdot]$' defines *synchrons* (which need synchronisation with other ports in order to interact).

Definition 4.1 A term $x \in \mathcal{AC}(P)$ is a *monomial*, iff it does not involve union operators.

Notation 4.2
We write $[x]^\alpha$, for $\alpha \in \{0, 1\}$, to denote a typed connector. When $\alpha = 0$, the connector is a synchron, otherwise it is a trigger. When the exact type is irrelevant, we write '$[\cdot]^*$'.

In order to simplify notation, we will omit brackets on 0, 1, and ports $p \in P$, as well as '$\cdot$' for the fusion operation.

The algebraic structure on $\mathcal{AC}(P)$ inherits most of the axioms from $\mathcal{AI}(P)$ except for the associativity of fusion.

Axioms
The operations satisfy the following axioms.

1. Union '$+$' is associative, commutative, idempotent, and has the identity element $[0]$.
2. Fusion '$\cdot$' is commutative, distributive, and has an identity element $[1]$. It is idempotent on monomial connectors, i.e. for any monomial $x \in \mathcal{AC}(P)$ we have $x \cdot x = x$.
3. Typing '$[\cdot]^*$' satisfies the following axioms, for $x, y, z \in \mathcal{AC}(P)$ and $\alpha, \beta \in \{0, 1\}$:
   (a) $[0]' = [0]$,
   (b) $\left[[x]^\alpha\right]^\beta = [x]^\beta$,
   (c) $[x + y]^\alpha = [y]^\alpha + [x]^\alpha$,
   (d) $[x]'\,[y]' = [x]'\,[y] + [x]\,[y]'$.

Let us now give some basic properties of $\mathcal{AC}(P)$.

Lemma 4.3 *For any monomial $x \in \mathcal{AC}(P)$ and any typing $\alpha, \beta \in \{0, 1\}$ holds the following equality*

$$[x]^\alpha \cdot [x]^\beta = [x]^{\alpha \vee \beta}\,,$$

where $\alpha \vee \beta = \max\{\alpha, \beta\}$.

Proof — If $\alpha = \beta$, then, $[x]^\alpha$ also being monomial, $[x]^\alpha \cdot [x]^\alpha = [x]^\alpha$ follows automatically from axioms above. If $\alpha = 0$ and $\beta = 1$, we have by axiom (3d), and by the idempotence of union and fusion for $\alpha = \beta$,

$$[x]' \cdot [x] = [x]' \cdot [x] + [x] \cdot [x]' = [x]' \cdot [x]' = [x]'\,.$$

■

Lemma 4.4 *For an arbitrary family $\{x_i\}_{i=1}^n \subset \mathcal{AC}(P)$, holds the following equality*

$$\prod_{i=1}^{n}[x_i]' = \sum_{i=1}^{n}\left([x_i]' \cdot \prod_{i\neq j}^{n}[x_j]\right) .$$

Proof — This lemma is a direct consequence of axiom (3d) and the idempotence of the union operation. ■

Notice that, by application of the above lemma, it is possible to reduce the degree of the terms to one. For example, consider a connector between two independent senders and three receivers $s_1'\, s_2'\, [r_1 + r_2\, r_3]$. This connector is equal to $s_1'\, s_2\, [r_1 + r_2\, r_3] + s_1\, s_2'\, [r_1 + r_2\, r_3]$.

Semantics

The semantics of $\mathcal{AC}(P)$ is given by the function $|\cdot| : \mathcal{AC}(P) \to \mathcal{AI}(P)$, defined by the rules

$$|[p]| = p\,, \tag{8}$$
$$|x_1 + x_2| = |x_1| + |x_2|\,, \tag{9}$$
$$\Big|\prod_{i=1}^{n}[x_i]\Big| = \prod_{i=1}^{n}|x_k|\,, \tag{10}$$
$$\Big|\prod_{i=1}^{n}[x_i]' \cdot \prod_{j=1}^{m}[y_j]\Big| = \sum_{i=1}^{n}|x_i| \cdot \left(\prod_{k\neq i}\Big(1 + |x_k|\Big) \cdot \prod_{j=1}^{m}\Big(1 + |y_j|\Big)\right), \tag{11}$$

for $x, x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathcal{AC}(P)$ and $p \in P \cup \{0, 1\}$. Rules (10) and (11) are applied to the maximal fusion terms.

Notice that, through the semantics of $\mathcal{AI}(P)$, connectors represent sets of interactions. The interaction semantics $\|\cdot\| : \mathcal{AC}(P) \to 2^{2^P}$ is defined by composing $|\cdot| : \mathcal{AC}(P) \to \mathcal{AI}(P)$ and $\|\cdot\| : \mathcal{AI}(P) \to 2^{2^P}$.

Rule (11) can be decomposed in two steps: 1) the application of Lemma 4.4, to reduce the degree of all terms to one; 2) the application of rule (11) for $n = 1$, expressing the fact that the single trigger in each term must participate in all interactions, while synchrons are optional. Compare Example 4.8 in the following section with Examples 2.4 and 3.2.

Example 4.5

Consider a system consisting of two Senders with ports s_1, s_2, and three Receivers with ports r_1, r_2, r_3. The meaning of the connector $s_1'\, s_2'\, [r_1 + r_2\, r_3]$ is computed as follows.

$$\begin{aligned}
&|s_1'\, s_2'\, [r_1 + r_2\, r_3]| = \\
&\overset{(11)}{=} \quad |s_1|\,(1 + |s_2|)\,(1 + |r_1 + r_2\, r_3|) \\
&\qquad\quad +|s_2|\,(1 + |s_1|)\,(1 + |r_1 + r_2\, r_3|) \\
&\overset{(9)}{=} \quad |s_1|\,(1 + |s_2|)\,(1 + |r_1| + |r_2\, r_3|) \\
&\qquad\quad +|s_2|\,(1 + |s_1|)\,(1 + |r_1| + |r_2\, r_3|) \\
&\overset{(10)}{=} \quad |s_1|\,(1 + |s_2|)\,(1 + |r_1| + |r_2|\,|r_3|) \\
&\qquad\quad +|s_2|\,(1 + |s_1|)\,(1 + |r_1| + |r_2|\,|r_3|) \\
&\overset{(8)}{=} \quad s_1\,(1 + s_2)\,(1 + r_1 + r_2\, r_3) \\
&\qquad\quad +s_2\,(1 + s_1)\,(1 + r_1 + r_2\, r_3)\,,
\end{aligned}$$

which corresponds to exactly the set of all possible interactions containing at least one of s_1 and s_2, and possibly either r_1 or both r_2 and r_3. ■

Proposition 4.6 *The axiomatisation of $\mathcal{AC}(P)$ is sound, that is, for $x, y \in \mathcal{AC}(P)$,*

$$x = y \Longrightarrow |x| = |y|\,. \tag{12}$$

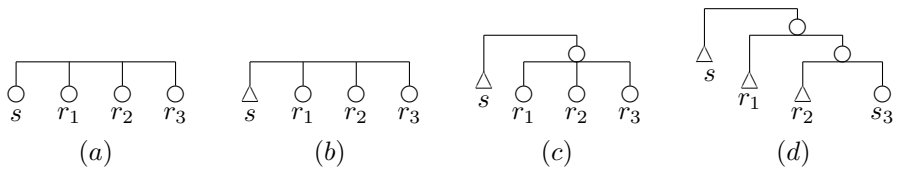


Figure 6: Graphic representation of connectors.

Definition 4.7 Two connectors $x, y \in \mathcal{AC}(P)$ are *equivalent* (denoted $x \simeq y$), iff they have the same sets of interactions, i.e.

$$x \simeq y \stackrel{def}{\Longleftrightarrow} |x| = |y| \,. \tag{13}$$

In Section 4.4, we show that this equivalence relation is not a congruence, which implies that there is no complete axiomatisation for the semantics '$|\cdot|$'.

4.2 Examples

Example 4.8 (Sender/Receiver continued)
In $\mathcal{AC}(P)$, the interactions for the four coordination schemes of Example 2.4 are:

	Set of interactions
Rendezvous	$s\, r_1\, r_2\, r_3$
Broadcast	$s'\, r_1\, r_2, r_3$
Atomic Broadcast	$s'\, [r_1\, r_2\, r_3]$
Causality Chain	$s'\, [r'_1\, [r'_2\, r_3]]$

Notice that $\mathcal{AC}(P)$ allows compact representation of interactions, and, moreover, explicitly captures the difference between broadcast and rendezvous. The four connectors are shown in Figure 6. The typing operator induces a hierarchical structure. Connectors can be represented as sets of trees, having ports at their leaves. We use triangles and circles to represent types: triggers and synchrons, respectively. ■

The distinction between parentheses '$(\cdot)$' and the typing operator '$[\cdot]^*$' is important, as shown by the following example.

Example 4.9
Consider two terms $p'\,(a'\,c + b)$ and $p'\,[a'\,c + b]$ of $\mathcal{AC}(P)$. For the first term we have

$$\begin{aligned} |p'\,(a'\,c + b)| &= |p'\,a'\,c + p'\,b| \\ &= p\,(1 + a)\,(1 + c) + a\,(1 + p)\,(1 + c) + p\,(1 + b) \\ &= p + p\,a + p\,c + p\,a\,c + a + a\,c + p\,b\,, \end{aligned}$$

whereas for $p'\,[a'\,c + b]$ we have

$$|p'\,[a'\,c + b]| \;=\; |p|\,(1 + |a'\,c + b|) \;=\; p\,(1 + a + a\,c + b) \;=\; p + p\,a + p\,a\,c + p\,b\,.$$

■

Example 4.10 (Broadcast)
For the broadcast connector $s'\,r_1\,r_2\,r_3$ (Figure 6(b), reproduced in Figure 7(a)), we have

$$|s'\,r_1\,r_2\,r_3| \;=\; s\,(1 + r_1)\,(1 + r_2)\,(1 + r_3)\,.$$

This connector can be constructed incrementally. For example, one can start from the connector $s'\,r_1$, having $|s'\,r_1| \;=\; s\,(1 + r_1)$. By typing this connector as a trigger and adding the synchron r_2, we obtain

$$|[s'\,r_1]'\,r_2| \;=\; |s'\,r_1|\,(1 + |r_2|) \;= s\,(1 + r_1)\,(1 + r_2)\,.$$

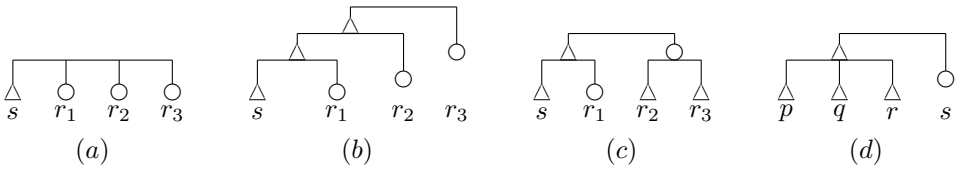


Figure 7: Some examples of $\mathcal{AC}(P)$ connectors.

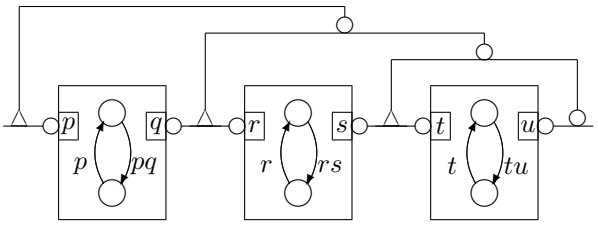


Figure 8: Modulo-8 counter.

Connecting r_3 in a similar manner gives $[[s' \, r_1]' \, r_2]' \, r_3$ (Figure 7(b)). The two connectors are equivalent:

$$|[[s' \, r_1]' \, r_2]' \, r_3| \; = \; s \, (1 + r_1) \, (1 + r_2) \, (1 + r_3)$$

It is easy to verify that another incremental construction results in the equivalent connector $[s' \, r_1]' \, [r_2' \, r_3']$ (Figure 7(c)). ■

Example 4.11 (An observer component)
Another simple example consists in adding an observer component to an existing connector. Suppose for example that we have three ports p, q, and r connected in an ethernet-like configuration $p' \, q' \, r'$. (In this connector all interactions are possible, except the empty one, and any port can initiate an interaction.)

Let us now introduce an observer, registering the interactions on this connector through a given port s. In order to do so, it is sufficient to join our existing connector (typed as trigger) and s typed as synchron, thus obtaining $[p' \, q' \, r']' \, s$ (Figure 7(d)).

As intended, s cannot itself initiate an interaction, but can participate in any one of the original connector. ■

Example 4.12 (Modulo-8 counter)
In the model shown in Figure 8, the causality chain pattern (cf. Figure 6(d)) is applied to connectors p, $q \, r$, $s \, t$, and u. Thus interactions are modelled by a single structured connector $p' \left[[q \, r]' \, [[s \, t]' \, u] \right]$:

$$\left| p' \, \left[[q \, r]' \left[[s \, t]' \, u \right] \right] \right| = p + p \, q \, r + p \, q \, r \, s \, t + p \, q \, r \, s \, t \, u \, .$$

These are exactly the interactions of the Modulo-8 counter of Figure 4. ■

Example 4.13 (Ethernet)
Consider n components, each equipped with a send port, s_i, and a receive port r_i, for $i \in [1, n]$. We model two types of interactions:

- successful communication, where some component k sends data through the port s_k, and all the others listen on their respective receive ports r_i for $i \neq k$;
- collision, where several components try to send data on their respective send ports $\{s_i\}_{i \in I}$ for some $I \subseteq [1, n]$, while the others listen on $\{r_i\}_{i \notin I}$.

Thus, the connector modelling the possible interactions is

$$\sum_{i=1}^{n} s'_k \prod_{i \neq k} (s'_i + r_i)\,.$$

■

4.3 Normal form of connectors

In this section, we define the normal form representation of connectors from $\mathcal{AC}(P)$. First, we define the normal form representation of monomial connectors, and we extend it to arbitrary connectors by distributivity of union.

A monomial connector is in normal form, if it does not contain synchron constants ([0] and [1]), and at each hierarchical level it is a fusion of typed elements, such that at most one of them is a trigger.

Definition 4.14 For any term x of the form $\prod_{i \in I} [x_i]^*$, we denote by $\#x$ the number of its trigger components, which we call the *degree* of x.

In general, for an expression $x = \sum_{i=1}^{n} x_i$, where all x_i have the form above, we define

$$\#x \stackrel{def}{=} \max_{i=1,n} \#x_i\,.$$

We say that x has *strictly positive degree* iff $\min\{\#x_i | i = 1, \ldots, n\} > 0$.

Definition 4.15 Let $x \in \mathcal{AC}(P)$ be a monomial connector, we say that it is in *normal form*, iff it is either $[p]^*$ for some $p \in P \cup \{0, 1\}$, or $x \equiv \prod_{i=1}^{n} [x_i]^{\alpha_i}$ (where '$\equiv$' denotes syntactic equality) and hold the following conditions

1. $\#x \leq 1$,
2. for all $i \in [1, n]$, we have both $x_i \neq 0$ and $[x_i]^{\alpha_i} \neq [1]$,
3. all x_i are pairwise distinct normal form monomials.

We say that an arbitrary connector is in normal form, iff it is a union of a family of pairwise distinct non-zero normal form monomials.

Example 4.16
Consider the connector $x \equiv p' \cdot [q' \cdot r + s]'$. This term is not in normal form for two reasons: firstly, $\#x > 1$, as both p' and $[q' \cdot r + s]'$ are triggers, and secondly $q' \cdot r + s$ is not a monomial. We can, however, represent x in a normal form by transforming it in the following way (we apply the distributivity of fusion and typing over union, and the reduction axiom (3d))

$$x \;\equiv\; p' \cdot [q' \cdot r + s]' \;=\; p' \cdot [q' \cdot r]' + p' \cdot s' \;=\; p' \cdot [q' \cdot r] + p \cdot [q' \cdot r]' + p' \cdot s + p \cdot s'\,.$$

Another example of the term that is not in normal form is $y \equiv p' \cdot [q' \cdot r']$. This time, it is due to the fact that $\#(q' \cdot r') > 1$. We obtain a normal form representation by applying the same transformations

$$y \;\equiv\; p' \cdot [q' \cdot r'] \;=\; p' \cdot [q' \cdot r + q \cdot r'] \;=\; p' \cdot [q' \cdot r] + p' \cdot [q \cdot r']\,.$$

■

Proposition 4.17 *For an arbitrary connector $x \in \mathcal{AC}(P)$ there exists a unique normal form connector $x^n \in \mathcal{AC}(P)$, such that $x = x^n$.*

Proof — Normal form of $\mathcal{AC}(P)$ connectors is defined by the four transformation rules below. These correspond to the idempotence, distributivity and absorption axioms (and also Lemma 4.3), as well the reduction axiom (3d): for any monomial $w \in \mathcal{AC}(P)$, arbitrary $x, y, z \in \mathcal{AC}(P)$, and $\alpha, \beta \in \{0, 1\}$ we put

$$x + x \overset{IDP}{\rightsquigarrow} x\,, \qquad x \cdot (y + z) \overset{DIST}{\rightsquigarrow} x \cdot y + x \cdot z\,, \qquad [0] + x \overset{ABS}{\rightsquigarrow} x\,,$$

$$[w]^\alpha \cdot [w]^\beta \overset{IDP}{\rightsquigarrow} [w]^{\alpha \vee \beta}\,, \qquad [x + y]^\alpha \overset{DIST}{\rightsquigarrow} [x]^\alpha + [y]^\alpha\,, \qquad [1] \cdot x \overset{ABS}{\rightsquigarrow} x\,,$$

$$[x]' \cdot [y]' \overset{RED_1}{\rightsquigarrow} [x]' \cdot [y] + [x] \cdot [y]'\,.$$

It can be easily verified that the rewriting system defined by these rules is terminating and confluent. For example, let us consider $[x]' \cdot [x]'$ for some monomial $x \in \mathcal{AC}(P)$. We then have essentially two possible chains of transformations:[1]

$$[x]' \cdot [x]' \overset{IDP}{\rightsquigarrow} [x]'\,,$$

and

$$[x]' \cdot [x]' \overset{RED_1}{\rightsquigarrow} [x]' \cdot [x] + [x] \cdot [x]' \overset{IDP}{\rightsquigarrow} [x]' \cdot [x] \overset{IDP}{\rightsquigarrow} [x]'\,.$$

■

4.4 Congruence relation

Definition 4.18 We denote by '$\cong$' the largest congruence relation contained in '$\simeq$', that is the largest relation satisfying, for $x, y \in \mathcal{AC}(P)$, and $z \notin P$,

$$x \cong y \Longrightarrow \forall E \in \mathcal{AC}(P \cup \{z\}), \quad E(x/z) \simeq E(y/z)\,, \tag{14}$$

where e.g. $E(x/z)$ denotes the expression, obtained from E by replacing all occurrences of z by x.

Notice that, in general, two equivalent terms are not congruent. For example, $p' \simeq p$, but $p' \ncong p$ as $p'\,q \not\simeq p\,q$, for $p, q \in P$.

Lemma 4.19 *Similarly typed semantically equivalent elements are congruent, i.e. for any two connectors $x, y \in \mathcal{AC}(P)$, and any $\alpha \in \{0, 1\}$, we have*

$$x \simeq y \Longrightarrow [x]^\alpha \cong [y]^\alpha\,. \tag{15}$$

Proof — By definition of the congruence '$\cong$', we have to show that for any expression $E \in \mathcal{AC}(P \cup \{z\})$ we have $E(x/z) \simeq E(y/z)$. Without loss of generality we can assume that z only occurs once in E. Due to distributivity of fusion and typing over union, it is sufficient to prove the implication

$$x \simeq y \Longrightarrow [x]^\alpha \cdot w \;\simeq\; [y]^\alpha \cdot w \tag{16}$$

for any monomial $w \in \mathcal{AC}(P)$ and any typing $\alpha \in \{0, 1\}$. Applying this argument iteratively, we will obtain the required equivalence $E(x/z) \simeq E(y/z)$.

Let us now prove (16) under the assumptions above. By symmetry, it is sufficient to prove that $\| [x]^\alpha \cdot w \| \subseteq \| [y]^\alpha \cdot w \|$, i.e. that for any interaction $a \in \| [x]^\alpha \cdot w \|$ we also have $a \in \| [y]^\alpha \cdot w \|$.

We have to consider four different cases according to the value of α and the degree $\#w$:

1. $\alpha = 0, \#w = 0$,

[1] Indeed, although both these chains can be eventually interleaved in various ways with transformations induced by the fusion context (e.g. $[x]' \cdot [x]' \cdot y$), it is clear that these would have the same effect on the final result (cf. also Lemma 4.4).

2. $\alpha = 1, \#w = 0$,
3. $\alpha = 0, \#w > 0$,
4. $\alpha = 1, \#w > 0$.

First of all, observe that case 4 follows from cases 3 and 2 by application of the reduction axiom (3d) (or equivalently Lemma 4.4). Below we provide the proof of case 3, and the other two cases can be treated in the same way.

We assume now that $\alpha = 0$ and $\#w > 0$, and we have to show that, for any $a \in \| [x] \cdot w \|$, we also have $a \in \| [y] \cdot w \|$.

From the definition of the interaction semantics '$\| \cdot \|$' of $\mathcal{AC}(P)$ we deduce that a can be decomposed as $a = a_1 \cup a_2$ for some $a_1 \in \|x\|$ or $a_1 = \emptyset$ and $a_2 \in \|w\|$. If $a_1 \neq \emptyset$, we deduce from $x \simeq y$ that $a_1 \in \|y\|$, and consequently $a = a_1 \cup a_2 \in \| [y] \cdot w \|$, which ends the proof. ■

Note 4.20 Clearly the converse implication in (15) is also true.

Lemma 4.21 *For $x, y \in \mathcal{AC}(P)$,*

$$x \cong y \Longleftrightarrow \forall z \in \mathcal{AC}(P),\ (z \textit{ is monomial} \Rightarrow x \cdot z \simeq y \cdot z)\,.$$

Proof — This lemma is a direct consequence of the Definition 4.18 of semantic congruence, Lemma 4.19, and distributivity laws. ■

Theorem 4.22 *Let $x, y \in \mathcal{AC}(P)$ be two non-zero monomial connectors, we then have*

$$x \cong y \Longleftrightarrow \begin{cases} x \simeq y \\ x \cdot 1' \simeq y \cdot 1' \\ \#x > 0 \Leftrightarrow \#y > 0\,. \end{cases} \tag{17}$$

Proof — One side of the implication is obvious. (The third condition is obtained by comparing $x \cdot p \cdot q$ and $y \cdot p \cdot q$, where $p, q \in P$ are two ports that do not participate in neither x nor y.[2]). Therefore, we only have to show that the three conditions on the right-hand side imply $x \cong y$.

By Lemma 4.21, it is sufficient to show that for any monomial term $z \in \mathcal{AC}(P)$ we have $x \cdot z \simeq y \cdot z$. As both x and y are also monomial, we have for some $\{x_i\}_{i=1}^n$, $\{y_i\}_{i=1}^m$, and $\{z_i\}_{i=1}^l$ in $\mathcal{AC}(P)$ and some $\{\alpha_i\}_{i=1}^n$, $\{\beta_i\}_{i=1}^m$, and $\{\gamma_i\}_{i=1}^l$ in $\{0, 1\}$

$$\begin{array}{rcl} x & \equiv & [x_1]^{\alpha_1} \cdot \ldots \cdot [x_n]^{\alpha_n}\,, \\ y & \equiv & [y_1]^{\beta_1} \cdot \ldots \cdot [y_m]^{\beta_m}\,, \\ z & \equiv & [z_1]^{\gamma_1} \cdot \ldots \cdot [z_l]^{\gamma_l}\,. \end{array} \tag{18}$$

Similarly to the proof of Lemma 4.19, we have to consider four cases according to the degrees of x, y, and z. However, the case, where all three degrees are positive, can be derived from the other three.

In each case, we have to show that $\|x \cdot z\| = \|y \cdot z\|$. However, by symmetry, it is sufficient to show that $\|x \cdot z\| \subseteq \|y \cdot z\|$, i.e. that for an arbitrary interaction $a \in \|x \cdot z\|$, we also have $a \in \|y \cdot z\|$.

Case 1. ($\#x = \#y = \#z = 0$) By the definition of the interaction semantics '$\| \cdot \|$' of $\mathcal{AC}(P)$, there exist $a_0 \in \|x\|$ and $a_1 \in \|z\|$ such that $a = a_0 \cup a_1$. Recall now that $x \simeq y$, and consequently we also have $a_0 \in \|y\|$, which immediately implies $a \in \|y \cdot z\|$.

Case 2. ($\#x, \#y > 0, \#z = 0$) There exist $a_0 \in \|x\|$ and a family $\{a_i \in \|z_i\|\}_{i \in I}$ indexed by some $I \subset [1, l]$ such that $a = a_0 \cup \bigcup_{i \in I} a_i$. As above, we deduce that $a_0 \in \|y\|$, and consequently $a \in \|y \cdot z\|$, as we also have $\#y > 0$.

[2] We assume here that there is always a port in P that does not participate in the considered expression, which is rather reasonable considering that new components can be potentially added to the system offering new communication ports.

Case 3. ($\#x = \#y = 0$, $\#z > 0$) Similarly to the previous case, there exist $a_0 \in \|z\|$ and a family $\{a_i \in \|x_i\|\}_{i\in I}$ indexed by some $I \subset [1,n]$ such that $a = a_0 \cup \bigcup_{i\in I} a_i$. Let us now consider $x \cdot 1'$, and rewrite it in the normal form. We obtain

$$x \cdot 1' \;=\; \sum_{k\in T(x)} [x_k]' \cdot \prod_{\substack{j=1\\ j\neq k}}^{n} [x_j] \;+\; 1' \cdot \prod_{j=1}^{n} [x_j] \;\simeq\; \sum_{J\subset[1,n]} \prod_{j\in J} [x_j]\,.$$

In the same manner, we obtain the corresponding equivalence for $y \cdot 1'$

$$y \cdot 1' \;\simeq\; \sum_{J\subset[1,m]} \prod_{j\in J} [y_j]\,.$$

By the choice of a_i, we then have

$$\bigcup_{i\in I} a_i \in \left\| \prod_{i\in I} [x_i] \right\| \subset \|x \cdot 1'\|\,,$$

which implies $\bigcup_{i\in I} a_i \in \|y \cdot 1'\|$, and consequently there exists $J \subset [1,m]$ such that

$$\bigcup_{i\in I} a_i \in \left\| \prod_{j\in J} [y_j] \right\|\,,$$

and we conclude the proof by observing that this implies

$$a = a_0 \cup \bigcup_{i\in I} a_i \in \left\| z \cdot \prod_{j\in J} [y_j] \right\| \subset \|y \cdot z\|\,.$$

■

The following two corollaries are used for the axiomatisation of the algebra of triggers, defined in the next section.

Corollary 4.23 *For $x \in \mathcal{AC}(P)$, such that $\deg(x) > 0$, we have $x \cdot 0' \cong x$.*

Proof — For monomial elements, the proof is straightforward and consists in applying the procedure described in the previous sections to verify that both $x \cdot 0' \simeq x$ and $x \cdot 0' \cdot 1' \simeq x \cdot 1'$. The condition that the degrees of both sides be non-zero simultaneously is guaranteed by the assumption of the lemma. In general case, we apply distributivity observing that all monomials also have non-zero degrees. ■

Note 4.24 Notice that the proof above does not make use of the fact that x and y are monomials. Indeed, it is sufficient to require that they have the form (18).

Theorem 4.22 can also be easily generalised to arbitrary x and y having strictly positive degree (recall Definition 4.14).

Corollary 4.25 *For any $x, y, z_1, \ldots, z_n, \in \mathcal{AC}(P)$.*

1. $[x]' \cdot [y]' \;\cong\; \Big[[x]' \cdot [y]'\Big]'$,
2. $[x]' \cdot \prod_{i=1}^{n} [z_i] \;\cong\; [x]' \cdot \left[\prod_{i=1}^{n} [z_i]'\right]$.

To finalise, let us give an example of the situation, where Theorem 4.22 is not applicable.

Example 4.26
Consider two ports $p, q \in P$. We then clearly have $p' \cdot q \ncong p' + p \cdot q$, as for any $r, s \in P$ the interaction $\{p, q, r\}$ is possible in $p' \cdot q \cdot r \cdot s$, but not in $(p' + p \cdot q) \cdot r \cdot s$.

At the same time, one can easily verify that hold all three conditions in the right-hand side of (17). Indeed, we have

$$\begin{array}{rcccl} p' \cdot q & \simeq & p + p \cdot q & \simeq & p' + p \cdot q \\ p' \cdot q \cdot 1' & \simeq & 1 + p + q & \simeq & p' \cdot 1' + p \cdot q \cdot 1' \\ \#(p' \cdot q) & = & 1 & = & \#(p' + p \cdot q)\,. \end{array}$$

This situation is explained by the fact that neither $p' + p \cdot q$ is a monomial, nor it has a strictly positive degree. ■

4.5 Sub-algebras

The subsets of the terms of $\mathcal{AC}(P)$, involving only triggers or synchrons, define two sub-algebras: the *algebra of triggers*, $\mathcal{AT}(P)$, and the *algebra of synchrons*, $\mathcal{AS}(P)$. The terms of these algebras model, respectively, coordination by rendezvous and by broadcast.

4.5.1 The algebra of synchrons

First, we consider the sub-algebra $\mathcal{AC}_S(P) \subset \mathcal{AC}(P)$ generated by the restriction to synchrons of the syntax (7)

$$\begin{array}{rcl} s & ::= & [0] \mid [1] \mid [p] \mid [x] \\ x & ::= & s \mid x \cdot x \mid x + x \mid (x)\,. \end{array} \tag{19}$$

$\mathcal{AC}_S(P)$ inherits the axioms of $\mathcal{AC}(P)$, and consequently fusion is also non-associative. The *algebra of synchrons*, $\mathcal{AS}(P)$, is obtained by adding the associativity axiom

$$\Big[[x]\,[y]\Big]\,[z] \;=\; [x]\,[y]\,[z] \;=\; [x]\,\Big[[y]\,[z]\Big]\,, \tag{20}$$

to those of $\mathcal{AC}_S(P)$. Thus, $\mathcal{AS}(P) \stackrel{def}{=} \mathcal{AC}_S(P)/ASSOC$, where $ASSOC$ is the reflexive transitive closure of (20), is an associative quotient algebra, which satisfies the same axioms as $\mathcal{AI}(P)$. Consequently, dropping the brackets in the elements of $\mathcal{AS}(P)$ immediately provides an isomorphism with $\mathcal{AI}(P)$.

Proposition 4.27 *The axiomatisation of $\mathcal{AS}(P)$ is sound and complete.*

Proof — This proposition follows from the associativity of synchronisation in $\mathcal{AI}(P)$ and the rule (10) in the definition of the semantics of $\mathcal{AC}(P)$. ■

4.5.2 The algebra of triggers

Similarly to the previous section we consider the sub-algebra $\mathcal{AC}_T(P) \subset \mathcal{AC}(P)$ generated by the restriction of syntax (7) to triggers:

$$\begin{array}{rcl} t & ::= & [0]' \mid [1]' \mid [p]' \mid [x]' \\ x & ::= & t \mid x \cdot x \mid x + x \mid (x)\,. \end{array} \tag{21}$$

Although the algebraic structure on $\mathcal{AC}_T(P)$ is inherited from $\mathcal{AC}(P)$ in very much the same way as that of $\mathcal{AC}_S(P)$, there is a slight but important difference in the structure that has to be observed here. Indeed, as $[1] \notin \mathcal{AC}_T(P)$, the identity element for fusion is $[0]'$ (cf. Corollary 4.23).

As in the case of $\mathcal{AC}_S(P)$, the algebra $\mathcal{AC}_T(P)$ is non-associative. Again, we consider a quotient algebra $\mathcal{AT}(P)$ obtained by adding the associativity axiom

$$\Big[[x]'\,[y]'\Big]'\,[z]' \;=\; [x]'\,[y]'\,[z]' \;=\; [x]'\,\Big[[y]'\,[z]'\Big]'\,.$$

for arbitrary trigger connectors $x, y, z \in \mathcal{AC}_T(P)$, to those of $\mathcal{AC}_T(P)$.

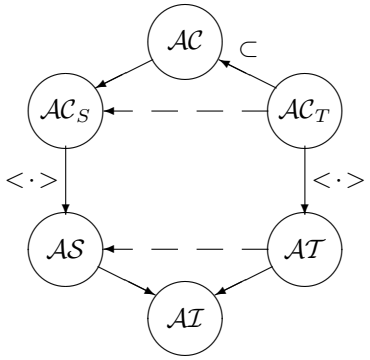


Figure 9: Hierarchy of algebras.

Proposition 4.28 *The axiomatisation of $\mathcal{AT}(P)$ is sound. It becomes complete with the additional axiom*

$$[x]' y = [x]' y + [x]' . \tag{22}$$

Proof — The soundness of this axiomatisation follows from Corollaries 4.23 and 4.25(1), the idempotence of union and synchronisation in $\mathcal{AI}(P)$, and the rule (11). The completness is proven by showing that the associativity of fusion and the absorbtion axiom (22) allow to define a normal form, coinciding for equivalent terms. ■

$$- o -$$

We have so far defined six algebras on a given set of ports P: $\mathcal{AI}(P)$, $\mathcal{AS}(P)$, $\mathcal{AT}(P)$, $\mathcal{AC}_S(P)$, $\mathcal{AC}_T(P)$, and $\mathcal{AC}(P)$, which can be arranged in a diagram shown in Figure 9. In this diagram, '$\subset$' is the set inclusion, '$< \cdot >$' represent the projections of $\mathcal{AC}_S(P)$ (resp. $\mathcal{AC}_T(P)$) on $\mathcal{AS}(P)$ (resp. $\mathcal{AT}(P)$) induced by associativity. The left branch can be associated with the construction of $\mathcal{AI}(P)$-semantics of $\mathcal{AC}(P)$. The remaining arrows are obtained by composing the existing ones. Observe that the definition of *monomials*, can be naturally formulated for elements of all six of these algebras.

5 Applications

The algebra of connectors formalises the concept of structured connector already used in the BIP language. It finds multiple applications in improving both the language and its execution engine. The three applications presented in this section, show its expressive power and analysis capabilities.

5.1 Efficient execution of BIP

The proposed algebraic framework can be used to enhance performance of the BIP execution Engine. The Engine drives the execution of (the C++ code generated from) a BIP program. A key performance issue is the computation of the set of the possible interactions of the BIP program from a given state. The Engine has access to the set of the connectors and the priority model of the program. From a given global state, each atomic component of the BIP program, waits for an interaction through a set of active ports (ports labelling enabled transitions) communicated to the Engine. The Engine computes from the set of all the active ports and the connectors, the set of the maximal interactions involving active ports. It chooses one of them, computes associated data transformations and notifies the components involved in the chosen interaction.

Currently, the computation of the maximal set of interactions involves a costly exploration of enumerative representations for connectors. This leads to a considerable overhead in execution times. For instance, for an MPEG4 encoder in BIP obtained by componentisation of a monolithic C program of 11,000 lines of code, we measured almost 100% of overhead in execution time. We provide below the principle of a not yet implemented, symbolic method which could be used to drastically reduce this overhead.

Given a set a of active ports, we use the following algorithm to find the maximal interactions contained in a connector K.

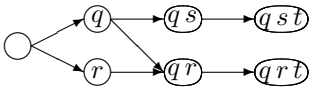
1. Let $\{p_1, \ldots, p_k\}$ be the set of ports that do not belong to a. Compute $K(0/p_1, \ldots, 0/p_k)$ (substitute 0 for all p_i, with $i = 1, \ldots, k$).
2. In the resulting connector, erase all primes to obtain a term $\widetilde{K} \in \mathcal{AI}(P)$.
3. Consider $\widetilde{K}$ as a star-free regular expression and build the associated (acyclic) automaton with states labelled by sub-interactions of a.
4. The final states of the obtained automaton correspond to maximal enabled interactions within K.

Example 5.1
Suppose that only ports q, r, s, and t are enabled, and compute the maximal interactions of the connector $p'\,[q\,[s+r]+r\,q']'[t+u]$.

Substitute 0 for p and u to obtain

$$0'\left[q\,[s+r]+r\,q'\right]'[t+0] \;=\; \left[q\,[s+r]+r\,q'\right]'t\,,$$

which becomes $\left[q\,[s+r]+r\,q\right]t$ by erasing the primes. The associated automaton is:



The final states of this automaton correspond to two interactions, $q\,r\,t$ and $q\,s\,t$, and it can be easily verified that those are, indeed, the two maximal interactions in the given connector, when ports p and u are disabled. ■

5.2 d-Synchronous component model

The evaluation of the BIP language on complex case studies, has shown that some coordination schemes need a number of connectors increasing exponentially with the number of ports. Nonetheless, these connectors can be obtained by combination of a reasonably small number of basic connectors.

To avoid tedious and error prone enumerative specification, we propose an extension of the current component model where a transition of the product component may involve synchronous execution of interactions from several connectors. This leads to a d-synchronous extension of the BIP component model discussed below.

To motivate the proposed extension, we model *joint function call* inspired from constructs found in languages such as nesC and Polyphonic C# [Com, nes]. A function call for a function f_i, involves two strong synchronisations between the $Caller$ and the $Callee_i$: 1) through the connector $K_i = c_i\,b_i$ to begin the execution of f_i; 2) through the connector $L_i = r_i\,f_i$ for finish and return (see Figure 10 for an example with two Callees).

Joint function calls involve the computation in parallel of several functions. The $Caller$ awaits for all the invoked functions to complete their execution. For instance, modeling a joint function call for functions f_1 and f_2, entails a modification of existing connectors by adding the links in dashed lines, shown in Figure 10, to obtain

$$[b_1\,c_1]'\,[b_2\,c_2]' \;\simeq\; b_1\,c_1 + b_2\,c_2 + b_1\,c_1\,b_2\,c_2\,.$$

Depending on the number of ports involved in the call, an exponential number of connectors can be required. To avoid connector explosion, we extend the composition operator of BIP in the following manner.

Definition 5.2 An *interconnected system* is given by a pair $(\{B_i\}_{i=1}^n, \{K_j\}_{j=1}^m)$, where $B_i = (Q_i, P_i, \rightarrow_i)$ with $\rightarrow_i \subseteq Q_i \times 2^{P_i} \times Q_i$, are components, and $K_j \in \mathcal{AC}(P)$ with $P = \bigcup_{i=1}^n P_i$.

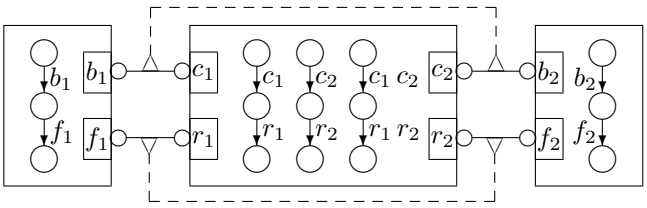


Figure 10: Modeling a joint call of two functions.

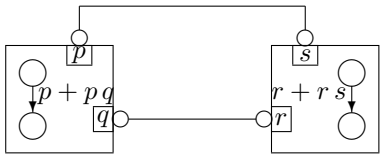


Figure 11: Causality loop.

For a integer parameter $0 < d \leq m$, the *d-synchronous semantics* of $(\{B_i\}_{i=1}^n, \{K_j\}_{j=1}^m)$ is the system $\gamma_d(B_1, \ldots, B_n)$ defined by applying the rule (1) with $\gamma = \gamma_d$, where

$$\gamma_d = \sum_{\substack{I \subseteq [1,m] \\ |I|=d}} \prod_{i \in I} [K_i]' \,.$$

The *synchronous semantics* corresponds to the case, where d is maximal (i.e. $d = m$).

Notice that γ_d contains all the interactions obtained by synchronisation of at most d connectors. Thus, in particular, we have $\gamma_1 \subseteq \gamma_2 \subseteq \ldots \subseteq \gamma_m$.

The application of rule (1) for the d-synchronous semantics with $d > 1$, requires the nontrivial computation of all the possible interactions. For this the following proposition can be used.

Proposition 5.3 *Let $(\{B_i\}_{i=1}^n, \{K_j\}_{j=1}^m)$ be an interconnected system. The set of possible interactions for its d-synchronous semantics is*

$$\prod_{i=1}^{n} [G_i]' \cap \gamma_d \,, \tag{23}$$

where, for $i \in [1, n]$, we put $G_i = \sum_{q_i \in Q_i} G_{q_i}$ with $G_{q_i} = \sum_{q_i \xrightarrow{a}} a$.

Notice that G_i, in (23), is the set of all interactions offered by the component i alone. Thus, $\prod_{i=1}^n [G_i]'$ is the set of all the interactions offered by the components, whereas γ_d is the set of the interactions allowed by the d-synchronised connectors. Therefore, the intersection of the two sets characterises all the possible interactions in the d-synchronous semantics.

Example 5.4 (Causality loop)
Consider the interconnected system shown in Figure 11. For $d = 2$ (synchronous semantics), the only possible interaction is

$$[p'\,q]'\,[r'\,s]' \cap [q\,r]'\,[p\,s]' = p\,q\,r\,s\,,$$

which corresponds to a causality loop, in the synchronous languages terminology [BG92, HCRP91].

Notice that, for $d = 1$, the set of possible interactions is empty:

$$[p'\,q]'\,[r'\,s]' \cap \Big(q\,r + p\,s\Big) = \emptyset\,.$$

■

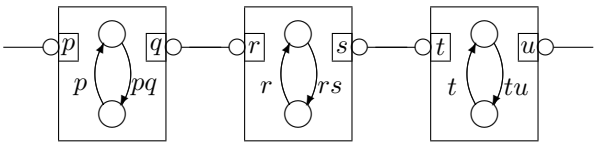


Figure 12: Synchronous modulo-8 counter.

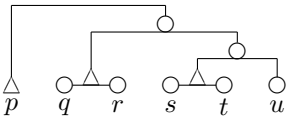


Figure 13: Synthesised connector for mod-8 counter.

Example 5.5 (Modulo-8 counter)
The synchronous semantics of the system in Figure 12 is equivalent to the modulo-8 counter given in Example 4.12 of Section 4.2. The synchronous model is a more natural representation of this system. Its interactions can be computed by application of Proposition 5.3:

$$\begin{aligned}&[p + p\,q]'\,[r + r\,s]'\,[t + t\,u]' \;\cap\; p'\,[q\,r]'\,[s\,t]'\,u' \;=\\ &\quad=\; p + p\,q\,r + p\,q\,r\,s\,t + p\,q\,r\,s\,t\,u\,.\end{aligned}$$

■

This example illustrates the importance of being able to compute the interactions of a system for the d-synchronous semantics with $d > 1$. As shown in the above examples, the execution of d-synchronous models with $d > 1$, requires a computation of intersection of connectors. To avoid costly enumeratuve techniques we have developed an alternative technique, based on dependency graph analysis. We illustrate this technique below, by applying it to the Modulo-8 counter.

The dependency graph analysis consists in building a directed acyclic graph, based on relations induced by connectors between the components of an interconnected system and labels of the transitions of these components. The resulting graph allows to determine the set of the possible interactions in the synchronous semantics, without having to enumerate them explicitly.

For the modulo-8 counter, the interconnected system in Figure 12 provides the following relations: $p \to q$ (p can trigger q), $r \to s$, $t \to u$, $q = r$ (q and r must synchronise), and $s = t$. All these relations together, are represented by the graph

$$p \to q\,r \to s\,t \to u\,. \tag{24}$$

Observe that each path in such dependency graph represents a causality chain. The graph in (24) represents the connector $p'\,[[q\,r]'[[s\,t]'u]]$, shown in Figure 13 (cf. also Figure 8). In general, this technique allows the synthesis of the connectors of a 1-synchronous model which is equivalent to a given synchronous model.

5.3 Incremental decomposition of connectors

In [GS05, Sif05], it has been argued that incrementality, which means that models can be constructed by adding and removing components in such a way that the resulting system is not affected by the order of operations, is an important property of the system composition.

For instance, the following incremental construction for the broadcast connector $s'\,r_1\,r_2\,r_3$ is provided in Example 4.10.

$$s'\,r_1\,r_2\,r_3 \;\simeq\; [s'\,r_1\,r_2]'\,r_3 \;\simeq\; \Big[[s'\,r_1]'\,r_2\Big]'\,r_3\,.$$

We studied techniques for computing incremental decompositions for connectors. These techniques are based on the iterative application of decompositions as defined by the following problem.

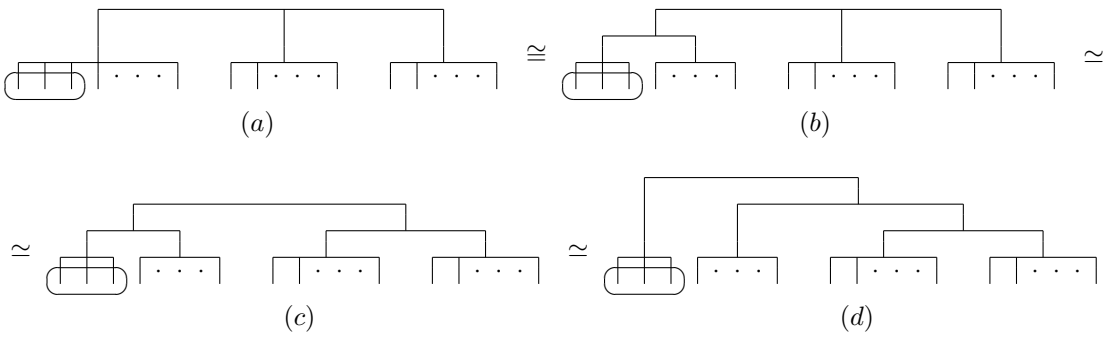


Figure 14: Hierarchical connector transformation

Problem 5.6 (Decomposition of Connectors) Given a connector $K \in \mathcal{AC}(P)$ and a subset of ports $P_0 \subset P$, construct a connector $\widetilde{K}$

$$K \equiv \sum_{i=1}^{n} K_i \cdot \overline{K}_i \,,$$

with $K_i \in \mathcal{AC}(P_0)$ and $\overline{K}_i \in \mathcal{AC}(P \setminus P_0)$, for $i = 1, \ldots, n$, such that $K \simeq \widetilde{K}$.

Note 5.7 We require that $K \simeq \widetilde{K}$. Indeed, the semantic congruence '$\cong$' is too strong to allow interesting transformations. However, semantic equivalence '$\simeq$' is sufficient in a large number of applications as it is transformed into congruence by typing (cf. Lemma 4.19).

Clearly, it is possible to solve this problem by computing explicitly all the interactions of K, and, for each interaction, separating the ports of P_0. This involves exhaustive enumeration of possible interactions, and thus leads to a combinatorial explosion of terms. We have developed two techniques for decomposing connectors, avoiding this explosion.

Both techniques, involve an iterative application of decompositions. The first technique is based on term rewriting rules, whereas the second uses the notion of derivation.

5.3.1 Decomposition by rewriting rules

In the context presented above, the required connector $\widetilde{K}$ can be constructed taking the group of ports $\{p_i\}_{i=1}^{k}$ up the hierarchical levels iteratively. This procedure can be separated into the following three steps illustrated in Figure 14.

Step 1. Regrouping the ports $\{p_i\}_{i=1}^{k}$ into a single typed connector (transition from (a) to (b) in Figure 14). More precisely, we transform a connector of the form

$$[p_1 \cdot \ldots \cdot p_k \cdot p_{k+1} \cdot \ldots \cdot p_n] \cdot [y] \cdot [z] \tag{25}$$

into another one of the form

$$\Big[[p_1 \cdot \ldots \cdot p_k] \cdot [p_{k+1} \cdot \ldots \cdot p_n]\Big] \cdot [y] \cdot [z] \,, \tag{26}$$

where in both cases (as well as in Figure 14) we skip all synchron/trigger typing to avoid overcharging the expressions. Observe that this transformation is a congruence, as all the changes are made inside an typed connector (cf. Lemma 4.19).

Step 2. Regrouping the sibling connectors into a single typed one (transition from (b) to (c) in Figure 14). Here, we continue the transformation by replacing the connector of the form (26) by an equivalent one of form

$$\Big[[p_1 \cdot \ldots \cdot p_k] \cdot [p_{k+1} \cdot \ldots \cdot p_n]\Big] \cdot \Big[[y] \cdot [z]\Big] \,. \tag{27}$$

Step 3. Finally, we perform a rotation taking the connector containing ports $\{p_i\}_{i=1}^k$ outside (transition from (c) to (d) in Figure 14). This is obtained by substituting the connector constructed in the previous step by an equivalent one of the form

$$[p_1 \cdot \ldots \cdot p_k] \cdot \Big[[p_{k+1} \cdot \ldots \cdot p_n] \cdot \Big[[y] \cdot [z]\Big]\Big] \,. \tag{28}$$

In the case where the initial connector considered in Step 1 above is itself an typed sub-connector of a more complex one, we consider the next hierarchical level of the connector obtained by these transformations. This level then automatically has the form (26), and therefore we can continue by iteratively applying Steps 2 and 3 until we reach the top level of the hierarchy, at which moment we obtain the required connector $\widetilde{K}$.

To finalise this procedure, we state the three following lemmas that describe in a formal way the transformations of the steps enumerated above.

Lemma 5.8 (Step 1: Regrouping of ports) *The following decomposition rule holds for any ports $p_i, q_j \in P$, where $i \in [1,n]$ and $j \in [1,m]$, and for any $1 \leq l < n$ and $0 \leq k \leq m$*

$$\begin{array}{rcl} p'_1 \ldots p'_n\, q_1 \ldots q_m & \simeq & [p'_1 \ldots p'_l\, q_1 \ldots q_k]' \cdot [p'_{l+1} \ldots p'_n\, q_{k+1} \ldots q_m]' \\ & + & [p'_1 \ldots p'_l\, q_1 \ldots q_k]' \cdot [q'_{k+1} \ldots q'_m] \\ & + & [q'_1 \ldots q'_k] \cdot [p'_{l+1} \ldots p'_n\, q_{k+1} \ldots q_m]' \,. \end{array} \tag{29}$$

For the two cases, where $l = n$ or $n = 0$, we have respectively the following two equivalences

$$p'_1 \ldots p'_n\, q_1 \ldots q_m \simeq [p'_1 \ldots p'_n\, q_1 \ldots q_k]' \cdot [q'_{k+1} \ldots q'_m] \,, \tag{30}$$

and

$$q_1 \ldots q_m \simeq [q_1 \ldots q_k] \cdot [q_{k+1} \ldots q_m] \,. \tag{31}$$

Note 5.9 Clearly, this lemma remains valid if we replace any number of ports p_i or q_i by typed connectors, i.e. $[x]$ for some $x \in \mathcal{AC}(P)$.

Lemma 5.10 (Step 2: Regrouping of siblings) *Let $\{x_i\}_{i=0}^n$ be a family of arbitrary elements of $\mathcal{AC}(P)$, and $\{\alpha_i\}_{i=1}^n$ be a corresponding $\{0,1\}$-typing such that $\alpha_k = 1$ for at least one $k \in [1,n]$. Then hold the following four properties*

$$[x_0] \cdot \prod_{i=1}^n [x_i] \simeq [x_0] \cdot \left[\prod_{i=1}^n [x_i]\right] , \tag{32}$$

$$[x_0]' \cdot \prod_{i=1}^n [x_i] \simeq [x_0]' \cdot \left[\prod_{i=1}^n [x_i]'\right] , \tag{33}$$

$$[x_0] \cdot \prod_{i=1}^n [x_i]^{\alpha_i} \simeq [x_0] \cdot \left[\prod_{i=1}^n [x_i]^{\alpha_i}\right]' , \tag{34}$$

$$[x_0]' \cdot \prod_{i=1}^n [x_i]^{\alpha_i} \simeq [x_0]' \cdot \left(\left[\prod_{i=1}^n [x_i]^{\alpha_i}\right]' + \left[\prod_{i\in S} [x_i]'\right]\right) , \tag{35}$$

where, in the last one, we put $S = \{i \in [1,n] \,|\, \alpha_i = 0\}$.

Lemma 5.11 (Step 3: Rotation) *For arbitrary connectors $x, y, z \in \mathcal{AC}(P)$ and types $\alpha, \beta, \gamma, \delta \in \{0,1\}$, holds the equivalence*

$$\Big[[x]^\alpha [y]^\beta\Big]^\delta [z]^\gamma \simeq [x]^{\alpha\delta} \Big[[y]^\delta [z]^\alpha\Big] + w \,, \tag{36}$$

where $w \in \mathcal{AC}(P)$ is defined by

$$w = \begin{cases} 0, & \text{if } \beta = 0 \text{ and } \gamma = 0 \\ z, & \text{if } \beta = 0 \text{ and } \gamma = 1 \\ [y]^\delta [z]^\gamma, & \text{if } \beta = 1\,. \end{cases}$$

5.3.2 Decomposition by derivation

Theorem 5.12 *Let $p \in P$ be an arbitrary port, and $K \in \mathcal{AC}(P)$ be a connector. Then there exist a unique $dK/dp \in \mathcal{AI}(P \setminus \{p\})$ such that*

$$K \;\simeq\; p \cdot \left[\frac{dK}{dp}\right] \;+\; K(0/p)\,, \tag{37}$$

where $K(0/p)$ denotes the connector obtained by substituting all occurrences of p in K by 0.

Proof — It is, indeed, sufficient to consider the flattening $|K| \in \mathcal{AI}(P)$ of the connector K. This flattening $|K|$ is a union of interactions that can be regrouped in two parts according to whether they contain p or not, thus proving the theorem. ■

Definition 5.13 We call the connector dK/dp in (37) the *derivative of K by p*.

Observe that Theorem 5.12 only states the uniqueness of such decomposition in $\mathcal{AI}(P)$. In $\mathcal{AC}(P)$, it is possible to have several distinct representations of dK/dp, which are, however, all semantically equivalent. Thus by restricting the discussion to semantic equivalence, we can consider any representation of dK/dp in $\mathcal{AC}(P)$.

Proposition 5.14 *The derivative possesses the following basic properties.*

1. $K(1) \simeq \dfrac{dK}{dp} + K(0)$,
2. $\forall K \in \mathcal{AI}(P \setminus \{p\}), \quad \dfrac{d(p \cdot K)}{dp} \simeq K \quad \text{and} \quad \dfrac{d(p' \cdot K)}{dp} \simeq 1' \cdot K$,
3. $\dfrac{d}{dp}\Big(K_1 + K_2\Big) \;\simeq\; \dfrac{dK_1}{dp} \;+\; \dfrac{dK_2}{dp}$,
4. $\forall \alpha, \beta \in \{0,1\}, \quad \dfrac{d}{dp}\Big([K_1]^\alpha \cdot [K_2]^\beta\Big) \;\simeq\; \left[\dfrac{dK_1}{dp}\right]^\alpha \cdot [K_2(1)] \;+\; [K_1(1)] \cdot \left[\dfrac{dK_2}{dp}\right]^\beta$,

where $K(1) \stackrel{def}{=} K(1/p)$.

Proof — Only property 4 is non-trivial. First, let us prove it for the case $\alpha = 0$ and $\beta = 0$. We have, by definition of derivative and by Lemma 4.19,

$$[K_1] \cdot [K_2] \;\simeq\; \left[p \cdot \left[\frac{dK_1}{dp}\right] + K_1(0)\right] \cdot \left[p \cdot \left[\frac{dK_2}{dp}\right] + K_2(0)\right]\,,$$

which transforms, by distributivity of fusion into

$$[K_1] \cdot [K_2] \;\simeq\; p \cdot \left[\frac{dK_1}{dp}\right] \cdot \left[\frac{dK_2}{dp}\right] + p \cdot \left[\frac{dK_1}{dp}\right] \cdot [K_2(0)] + p \cdot \left[\frac{dK_2}{dp}\right] \cdot [K_1(0)] + [K_1(0)] \cdot [K_2(0)]\,,$$

adding, by idempotence of the union, a second copy of the first summand in the right-hand side and re-grouping again, we obtain

$$[K_1] \cdot [K_2] \;\simeq\; p \cdot \left[\frac{dK_1}{dp}\right] \cdot \left[\left[\frac{dK_2}{dp}\right] + [K_2(0)]\right] + p \cdot \left[\frac{dK_2}{dp}\right] \cdot \left[\left[\frac{dK_1}{dp}\right] + [K_1(0)]\right] + [K_1(0)] \cdot [K_2(0)]\,,$$

which, by the first property above, results in the equivalence

$$\begin{aligned}[K_1]\cdot[K_2] &\simeq p\cdot\left[\frac{dK_1}{dp}\right]\cdot[K_2(1)]+p\cdot\left[\frac{dK_2}{dp}\right]\cdot[K_1(1)]+[K_1(0)]\cdot[K_2(0)]\\ &\simeq p\cdot\left[\left[\frac{dK_1}{dp}\right]\cdot[K_2(1)]+\left[\frac{dK_2}{dp}\right]\cdot[K_1(1)]\right]+[K_1(0)]\cdot[K_2(0)]\,,\end{aligned}$$

thus proving the required property for $\alpha=\beta=0$.

For the case $\alpha=1$ and $\beta=0$ we have

$$\begin{aligned}\frac{d}{dp}\Big([K_1]'\cdot[K_2]\Big) &\simeq \frac{d}{dp}\Big([K_1]+[K_1]\cdot[K_2]\Big)\\ &\simeq \left[\frac{dK_1}{dp}\right]+\left[\frac{dK_1}{dp}\right]\cdot[K_2(1)]+\left[\frac{dK_2}{dp}\right]\cdot[K_1(1)]\\ &\simeq \left[\frac{dK_1}{dp}\right]'\cdot[K_2(1)]+\left[\frac{dK_2}{dp}\right]\cdot[K_1(1)]\,.\end{aligned}$$

The proof for the case $\alpha=\beta=1$ is obtained in the same way. ■

The last property in the proposition above can be generalised to a fusion of any number of typed connectors.

Proposition 5.15 *Let $\{K_i\}_{i=1}^n$ and $\{L_j\}_{j=1}^m$ be two families of arbitrary connectors from $\mathcal{AC}(P)$. We then have the following equivalence for the derivative of the connectors formed by the fusion inside each of these families.*

$$\begin{aligned}&\frac{d}{dp}\left(\prod_{i=1}^n[K_i]'\cdot\prod_{j=1}^m[L_j]\right)\simeq\\ &\simeq\sum_{i=1}^n\left[\frac{dK_i}{dp}\right]'\cdot\prod_{k\neq i}[K_k(1)]\cdot\prod_{j=1}^m[L_j(1)]+\sum_{j=1}^m\left[\left[\frac{dL_j}{dp}\right]'\cdot\prod_{k\neq j}[L_k(1)]\right]\cdot\left[\prod_{i=1}^n[K_i(1)]'\right]\end{aligned}$$

Proof — We prove this proposition by induction on $n+m$. Property 4 in Proposition 5.14 constitutes its base. We prove the induction step for the case $m,n>0$. The cases $m=0$ or $n=0$ are treated in the similar manner. We then have, by Corollary 4.25 of Theorem 4.22,

$$\frac{d}{dp}\left(\prod_{i=1}^n[K_i]'\cdot\prod_{j=1}^m[L_j]\right)\simeq\frac{d}{dp}\left(\left[\prod_{i=1}^n[K_i]'\right]'\cdot\left[\prod_{j=1}^m[L_j]'\right]\right),$$

which, by property 4 in Proposition 5.14, can be developed to

$$\frac{d}{dp}\left(\prod_{i=1}^n[K_i]'\cdot\prod_{j=1}^m[L_j]\right)\simeq\left[\frac{d}{dp}\prod_{i=1}^n[K_i]'\right]'\cdot\left[\prod_{j=1}^m[L_j(1)]'\right]+\left[\frac{d}{dp}\prod_{j=1}^m[L_j]'\right]\cdot\left[\prod_{i=1}^n[K_i(1)]'\right],$$

and subsequently, by induction assumption,

$$\begin{aligned}&\frac{d}{dp}\left(\prod_{i=1}^n[K_i]'\cdot\prod_{j=1}^m[L_j]\right)\simeq\\ &\simeq\left[\sum_{i=1}^n\left[\frac{dK_i}{dp}\right]'\cdot\prod_{k\neq i}[K_i(1)]\right]'\cdot\left[\prod_{j=1}^m[L_j(1)]'\right]+\left[\sum_{j=1}^m\left[\frac{dL_j}{dp}\right]'\cdot\prod_{k\neq j}[L_j(1)]\right]\cdot\left[\prod_{i=1}^n[K_i(1)]'\right].\end{aligned}$$

Once again, applying Corollary 4.25 of Theorem 4.22 to the first summand, and distributing fusion over union in the second one, we obtain the desired result. ■

Example 5.16
Consider the connector $K = p' \cdot [qr]' \cdot [rs]$. This corresponds to a situation, where the components providing ports q and s require a certain resource to operate. This resource is provided by another component during a communication over the port r. Once this resource available, both these components are connected with a third one, which communicates over port p.

Suppose now that we want to restructure K in order to separate the component providing the resource in question from the ones that utilise it. To do so, we differentiate K by r, applying Proposition 5.15

$$\frac{dK}{dr} \simeq \left[\frac{dp}{dr}\right]' \cdot [q] \cdot [s] + \left[\frac{d(qr)}{dr}\right]' \cdot [p] \cdot [s] + \left[\left[\frac{d(sr)}{dr}\right]'\right] \cdot [p'q'] \simeq q'ps + s[p'q']\,. \tag{38}$$

We calculate also $K(0)$, by substituting 0 instead of r in K,

$$K(0) \equiv p' \cdot [q \cdot 0]' \cdot [0 \cdot s] \cong p' \cdot [0]' \cdot [0] \cong p' \simeq p\,. \tag{39}$$

Substituting (38) and (39) into (37), we obtain the following decomposition

$$K \simeq r \cdot \Big[q'ps + s[p'q']\Big] + p\,,$$

which can now be easily verified. ■

In the above example, port r is only present in K as a synchron, and therefore, when we calculate the derivative, it is absorbed directly, whereas, if it was a trigger, we would obtain an expression containing $1'$, which has to be eliminated in the final decomposition. To do so, we apply one of the two following equivalences.[3] For any $x, y \in \mathcal{AC}(P)$, we have

$$x \cdot [1' \cdot y] \simeq [x]' \cdot y\,, \tag{40}$$
$$x \cdot [1' \cdot y]' \simeq [x]' \cdot y + 1' \cdot y\,. \tag{41}$$

Example 5.17
Consider the connector $K = \Big[[p'q]'r\Big]'s$, and assume that we want to decompose it with respect to port p. Clearly, we have $K(0) \simeq 0$ and

$$\frac{dK}{dp} \simeq \Big[[1'q]'r\Big]'s\,.$$

Substituting these equivalences into (37) and applying (41), we obtain

$$K \simeq p \cdot \Big[\Big[[1'q]'r\Big]'s\Big] \simeq p \cdot \Big[\Big[r'q + 1'q\Big]'s\Big] \simeq p \cdot \Big[[r'q]'s + s'q + 1'q\Big] \simeq p \cdot \Big[[r'q]'s + s'q\Big] + p'q\,.$$

■

$$-\ o\ -$$

Clearly, both decomposition techniques presented above are sub-optimal. For instance, both connectors of Examples 5.16 and 5.17 can also be represented in the following — less complex — forms

$$p' \cdot [qr]' \cdot [rs] \simeq r \cdot \Big[q'ps + ps\Big] + p\,,$$
$$\Big[[p'q]'r\Big]'s \simeq p' \cdot [q'r's']\,.$$

However, in general, to obtain such representations, one has to flatten completely — to the level of $\mathcal{AI}(P)$ interactions — the connector in question and regroup subsequently the resulting terms using ad hoc methods.

On the contrary, both presented techniques have the advantage of being straightforward and providing decompositions that preserve some trigger/synchron typing without making explicit reference to the semantics of the decomposed connectors.

[3] These two equivalences are special cases of Lemma 5.11.

6 Conclusion

$\mathcal{AC}(P)$ provides an abstract and powerful framework for modelling control flow between components. It allows the structured combination of two basic synchronisation protocols: rendezvous and broadcast. It is powerful enough to represent any kind of coordination by interaction, avoiding combinatorial explosion inherent to broadcast.

Connectors are constructed by using two operators having a very intuitive interpretation. Triggers initiate asymmetric interactions; they are sources of causal interaction chains. Synchrons are passive ports which either can be activated by triggers or can be involved in some maximal symmetric interaction. Fusion allows the construction of new connectors by assembling typed connectors. Typing induces a hierarchical structuring, naturally represented by trees.

The concept of structured connectors is directly supported by the BIP language where connectors describe a set of interactions as well as associated data transformations. Its interest has been demonstrated in many case studies including an autonomous planetary robot, wireless sensor networks [BMP+07], and adaptive data-flow multimedia systems. The BIP language is used in the framework of industrial projects, as a semantic model for the HRC component model (IST/SPEEDS integrated project), and for AADL (ITEA/SPICES project).

We believe that $\mathcal{AC}(P)$ provides an elegant mathematical framework to deal with interactions. The comparison with boolean algebra shows its interest: fusion becomes a context-sensitive and rather complicated operation on boolean functions. Boolean algebra representation allows the use of existing powerful decision techniques, e.g. to decide that an interaction belongs to a connector or equivalence between connectors. The relations between $\mathcal{AC}(P)$ and boolean algebra should be further investigated.

The notation has been instrumental for formalising the semantics of the synchronous component model. Axiomatisation and properties of derivatives in $\mathcal{AC}(P)$ allow an efficient incremental decomposition of connectors avoiding enumeration of interactions. Finally, algebraic representation is a basis for symbolic manipulation and transformation of connectors which is essential for efficient implementation of the BIP framework.

To our knowledge, $\mathcal{AC}(P)$ is the first algebraic framework for modelling interaction. It can be a semantic model for formalisms used for modelling architecture, and provides a basis for comparing coordination mechanisms supported by existing languages, such as coordination languages.

Appendix: Idempotent regular expressions on ports

In this paper, we have presented an algebra of connectors $\mathcal{AC}(P)$, which provides a flexible and powerful instrument to represent interaction models, i.e. sets of possible interactions between components of an integrated system.

Although connectors from $\mathcal{AC}(P)$ can always be transformed into terms of the algebra of interactions $\mathcal{AI}(P)$, the former have an advantage of representing the same interactions in a compact manner, preserving an important semantic aspect — the trigger/synchron typing.

We will now present succinctly another way of representing interactions in a more compact way than that provided by $\mathcal{AI}(P)$. We do so by assigning a different kind of types to ports participating in a given connector: *required* or *optional*. We adopt the notation used in regular expressions, and write p^* if this port is optional, whereas p without an asterisk represents an obligatory port.[4] On the axiomatic level, this corresponds to adding the equality

$$p^* = p + 1$$

to the axioms of $\mathcal{AI}(P)$ presented in Section 3.

Thus, an expression $p_1 \dots p_n q_1^* \dots q_m^*$ denotes a connector, where ports $p_1, \dots, p_n$ are required (obligatory) and $q_1, \dots, q_m$ are optional. It is easy to verify that the resulting set of possible interactions is exactly that corresponding to an $\mathcal{AI}(P)$-expression

$$p_1 \dots p_n (1 + q_1) \dots (1 + q_m)\,. \tag{42}$$

Connectors can be converted in a straightforward manner between the regular expression notation and that of $\mathcal{AC}(P)$. It is sufficient to observe that for a flat $\mathcal{AC}(P)$ connector in normal form we have the following equivalence

$$p' q_1 \dots q_n \simeq p\, q_1^* \dots q_n^*\,.$$

The conversion in the other direction is based on a similar equivalence

$$p_1 \dots p_n\, q_1^* \dots q_m^* \simeq [p_1 \dots p_n]' q_1 \dots q_m\,.$$

Although the two representations of connectors appear to be similar, it is important to observe that, while $\mathcal{AC}(P)$ connectors reflect the trigger/synchron typing and, consequently, allow a natural representation of both broadcast and rendezvous communication, the regular expression form is just an abbreviation of $\mathcal{AI}(P)$, and, therefore, it cannot capture this difference in a natural manner.

Example .1
Consider a situation, where an emitter p is broadcasting information to receivers q_1, q_2. This can be easily represented in both approaches, by writing respectively $p' q_1 q_2$ and $p\, q_1^* q_2^*$.

Suppose, now, that another emitter r has to be connected to the system. In terms of $\mathcal{AC}(P)$, this operation is trivial: we add r' to the initial expression, thus obtaining $p' q_1 q_2 r'$. At the same time, in terms of regular expressions, we have to modify the existing connector to reflect the fact that, among p and r, it is sufficient that only one participates in the interaction. The simplest way to express this is to write $(p^* r + p\, r^*) q_1^* q_2^*$. ■

The advantage of the regular expressions approach is that it allows us to consider equations of the type

$$x = A \cdot x + B\,,$$

where all three of $x, A, B \in \mathcal{AI}(P)$ are sets of interactions. Due to the idempotence of synchronisation, the minimal solution of such equations is finite.

Example .2
It is easy to verify that the minimal solution of the equation $x = A \cdot x$, with $A = a_1 + a_2$, is $a_1^* a_2 + a_1\, a_2^*$. ■

[4] Due to the idempotence of synchronisation in $\mathcal{AI}(P)$, the regular expression p^* is reduced to "zero or one occurrence of p", which is equivalent to saying that p is optional.

References

[Arb05] Farhad Arbab. Abstract behavior types: a foundation model for components and their composition. *Sci. Comput. Program.*, 55(1-3):3–52, 2005. 1

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In 4^{th} *IEEE International Conference on Software Engineering and Formal Methods (SEFM06)*, pages 3–12, September 2006. Invited talk. 1, 2.1, 4

[BG92] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: Design, semantics implementation. *Science of Computer Programming*, 19(2):87–152, November 1992. 5.4

[BGK+06] K. Balasubramanian, A.S. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema. Developing applications using model-driven design environments. *IEEE Computer*, 39(2):33–40, 2006. 1

[bip] BIP. `http://www-verimag.imag.fr/~async/index.php?view=components`. 2.1

[BMP+07] Ananda Basu, Laurent Mounier, Marc Poulhiès, Jacques Pulou, and Joseph Sifakis. Using BIP for modeling and verification of networked systems — A case study on TinyOS-based networks. Technical Report TR-2007-5, VERIMAG, 2007. `http://www-verimag.imag.fr/index.php?page=techrep-list`. 6

[BPE] Business process execution language for web services. `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`. 1

[BWH+03] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A.L. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003. 1

[Com] Cω. `http://research.microsoft.com/comega/`. 5.2

[EJL+03] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003. 1

[GS03] Gregor Gößler and Joseph Sifakis. Component-based construction of deadlock-free systems: Extended abstract. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 420–433, Mumbai, India, December 2003. Springer. 1

[GS05] Gregor Gößler and Joseph Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1–3):161–183, 2005. 1, 5.3

[HCRP91] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language LUSTRE. In *Proceedings of the IEEE*, volume 79, pages 1305–1320, September 1991. 5.4

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall, April 1985. 2.2, 2.2.1

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989. 2.2, 2.2.2

[MR01] Florence Maraninchi and Yann Rémond. Argos: an automaton-based synchronous language. *Computer Languages*, 27:61–92, 2001. 2.5

[nes] nesC: A programming language for deeply networked systems. `http://nescc.sourceforge.net/`. 1, 5.2

[Ros97] A. W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1997. 2.2.1

[Sif05] Joseph Sifakis. A framework for component-based construction. In 3^{rd} *IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, September 2005. Keynote talk. 1, 5.3