The Algebra of Connectors – Structuring Interaction in BIP

Simon Bliudze, Joseph Sifakis
VERIMAG, Centre Équation
2 av de Vignate
38610, Gières, France
{bliudze, sifakis}@imag.fr

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 a binary *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.

Categories and Subject Descriptors

C.0 [**General**]: System architectures; Systems specification methodology; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design, Theory

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 [1, 2, 3, 12]. 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.

We propose the *algebra of connectors* for modelling interaction in component-based systems. This algebra considers connectors as the basic concept for modeling coordination between components. Different formalisations for connectors in component frameworks have been proposed. In most of them, connectors are specified in an operational setting, usually a process algebra. In [21], a connector is defined as a set of processes: there is one process for each role of the connector, plus one process for the "glue" that describes how all the roles are bound together. In [7], a process algebra is used to define an *architectural type* as a set of component/connector instances related by a set of attachments among their interactions. In [1], *Reo* is a channel-based exogenous coordination model for multi-agent systems. It uses connectors compositionally built out of different types of channels formalised in data-stream semantics. Our approach considers connectors as relations between ports with synchronisation types. It is close to [10, 13], where the notion of "higher-order" connectors is investigated in a categorical framework for component composition. Nonetheless, the categorical semantic underpinnings of their work gives a very different framework.

The algebra of connectors allows the description of coor-

EMSOFT'07, September 30–October 3, 2007, Salzburg, Austria.

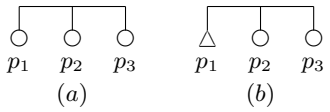


Figure 1: Graphical representation of rendezvous (a) and broadcast (b) connectors.

dination between components in terms of structured stateless connectors involving communication ports. It formalises mechanisms and concepts that have been implemented in the *Behaviour-Interaction-Priority* (BIP) component framework developed at Verimag [4, 20]. 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 [14, 15] 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, and 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 a binary 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.

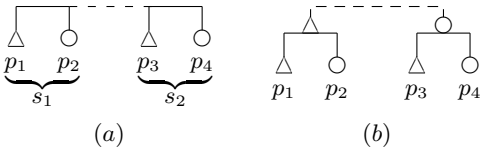


Figure 2: Fusion (a) and structuring (b) of connectors.

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

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 [4, 8]).

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.

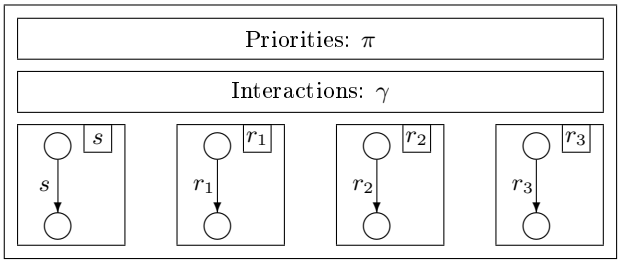


Figure 3: A system with four atomic components

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 $B = \gamma(B_1 \ldots, B_n)$.

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 for four different coordination schemes.

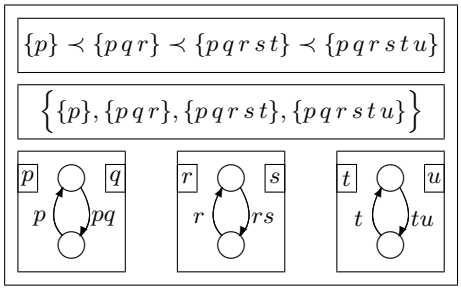


Figure 4: Modulo-8 counter.

	Set of interactions
Rendezvous	$\{s\, r_1\, r_2\, r_3\}$
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	$\{s,\ s\, r_1\, r_2\, r_3\}$
Causality Chain	$\{s,\ s\, r_1,\ s\, r_1\, r_2,\ s\, r_1\, r_2\, r_3\}$

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 [17], 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.

Notice that the composition operator can express usual parallel composition operators [9], such as the ones used in CSP [16] and CCS [18]. By enforcing maximal progress, priorities allow to express broadcast.

3. THE ALGEBRA OF INTERACTIONS

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

3.1 Syntax, axioms, and semantics

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

$$x \quad ::= \quad 0 \mid 1 \mid p \in P \mid x \cdot x \mid x + x \mid (x)\,, \tag{3}$$

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 given by the function $\|\cdot\| : \mathcal{AI}(P) \to 2^{2^P}$, defined by

$$\begin{array}{lcl} \|0\| = \emptyset, \quad \|1\| = \{\emptyset\}, \quad \|p\| = \Big\{\{p\}\Big\}, \\ \|x_1 + x_2\| & = & \|x_1\| \cup \|x_2\|, \\ \|x_1 \cdot x_2\| & = & \Big\{a_1 \cup a_2 \,\Big|\, a_1 \in \|x_1\|,\ a_2 \in \|x_2\|\Big\}, \\ \|(x)\| & = & \|x\|, \end{array} \tag{4}$$

for $p \in P$, $x, x_1, x_2 \in \mathcal{AI}(P)$. 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 $x, y \in \mathcal{AI}(P)$,*

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

PROOF. Both the soundness and completeness proofs are straightforward. The latter is obtained by applying distributivity to flatten the elements and verifying that the normal forms, obtained in this way for elements having same sets of interactions, coincide. □

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.

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:

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

for $p_{i_1}, \ldots p_{i_k} \in P$, 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 [4]. 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

$$\begin{array}{lcll} s & ::= & [0] \mid [1] \mid [p] \mid [x] & (\mathit{synchrons}) \\ t & ::= & [0]' \mid [1]' \mid [p]' \mid [x]' & (\mathit{triggers}) \\ x & ::= & s \mid t \mid x \cdot x \mid x + x \mid (x)\,, & \end{array} \tag{5}$$

for $p \in P$, and where '$+$' is binary operator called *union*, '$\cdot$' is a binary 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 $\mathcal{AI}(P)$. 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.

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

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

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.

Definition 4.3. The *degree* of a term $x \in \mathcal{AC}(P)$ of the form $\prod_{i\in I}[x_i]^*$, denoted by $\#x$, is the number of its trigger sub-terms.

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 associative, 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) $[x]'\,[y]' = [x]'\,[y] + [x]\,[y]'$.

Lemma 4.4. *For $x_i \in \mathcal{AC}(P)$, where $i = 1, \ldots, n$,*

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

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

$$|[p]| = p, \tag{6}$$
$$|x_1 + x_2| = |x_1| + |x_2|, \tag{7}$$
$$\left|\prod_{i=1}^{n}[x_i]\right| = \prod_{i=1}^{n}|x_k|, \tag{8}$$
$$\left|\prod_{i=1}^{n}[x_i]' \cdot \prod_{j=1}^{m}[y_j]\right| = \sum_{i=1}^{n}|x_i| \cdot \left(\prod_{k\neq i}\left(1+|x_k|\right) \cdot \prod_{j=1}^{m}\left(1+|y_j|\right)\right), \tag{9}$$

for $x, x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathcal{AC}(P)$ and $p \in P \cup \{0,1\}$. Rules (8) and (9) are applied to the maximal fusion terms.

Notice that, through the semantics of $\mathcal{AI}(P)$, connectors represent sets of interactions.

Rule (9) 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 (9) 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.

$$\begin{aligned}
&|s_1'\,s_2'\,[r_1 + r_2\,r_3]| = \\
&\overset{(9)}{=} |s_1|\,(1+|s_2|)\,(1+|r_1 + r_2\,r_3|) \\
&\quad +|s_2|\,(1+|s_1|)\,(1+|r_1 + r_2\,r_3|) \\
&\overset{(7)}{=} |s_1|\,(1+|s_2|)\,(1+|r_1| + |r_2\,r_3|) \\
&\quad +|s_2|\,(1+|s_1|)\,(1+|r_1| + |r_2\,r_3|) \\
&\overset{(8)}{=} |s_1|\,(1+|s_2|)\,(1+|r_1| + |r_2|\,|r_3|) \\
&\quad +|s_2|\,(1+|s_1|)\,(1+|r_1| + |r_2|\,|r_3|) \\
&\overset{(6)}{=} s_1\,(1+s_2)\,(1+r_1 + r_2\,r_3) \\
&\quad +s_2\,(1+s_1)\,(1+r_1 + r_2\,r_3),
\end{aligned}$$

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 axioms of $\mathcal{AC}(P)$ are sound with respect to the semantics defined by (6)–(9), that is, for $x, y \in \mathcal{AC}(P)$, $x = y$ implies $|x| = |y|$.*

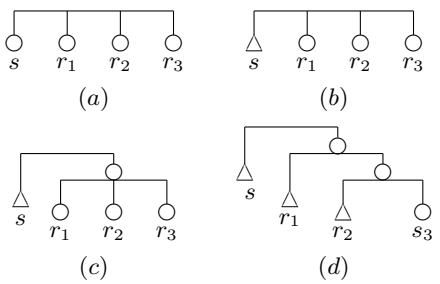


Figure 6: Graphic representation of connectors.

PROOF. As rules (8) and (9) are applied to maximal fusion terms, to prove this proposition, we have to verify that all the axioms preserve the semantics in any fusion context, i.e. for an axiom $x = y$ and arbitrary $z \in \mathcal{AC}(P)$, we have to verify that $|x\,z| = |y\,z|$. However, it is clear that it is sufficient to verify this property only for monomial z, which is straightforward. □

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.

$$x \simeq y \stackrel{def}{\Longleftrightarrow} |x| = |y|\,. \tag{10}$$

In Section 4.3, we show that this equivalence relation is not a congruence.

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 following example illustrates the distinction between parentheses '$(\cdot)$' and the typing operator '$[\cdot]^*$'.

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

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

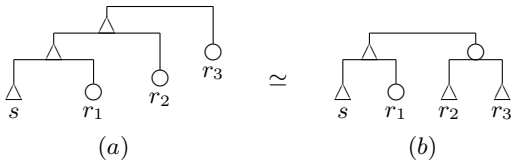


Figure 7: Two connectors realising a broadcast.

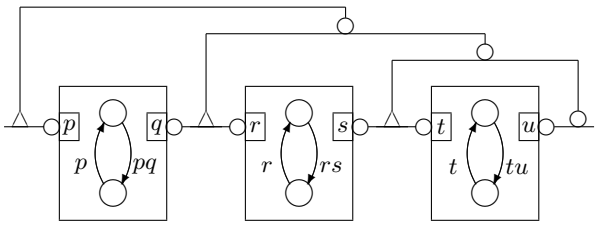


Figure 8: Modulo-8 counter.

Example 4.10 (Broadcast)**.** For the broadcast connector $s'\,r_1\,r_2\,r_3$ (Figure 6(b)), 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

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

Connecting r_3 in a similar manner gives $[[s'\,r_1]'\,r_2]'\,r_3$ (Figure 7(a)). 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 (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 $p'\,\Big[[q\,r]'\,[[s\,t]'\,u]\Big]$:

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

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

Example 4.12 (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 Congruence relation on $\mathcal{AC}(P)$

Definition 4.13. 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 $z \notin P$,

$$x \cong y \Longrightarrow \forall E \in \mathcal{AC}(P \cup \{z\}), \quad E(x/z) \simeq E(y/z)\,, \quad (11)$$

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 $p, q \in P$.

Proposition 4.14. *Similarly typed equivalent terms are congruent, i.e. for $x, y \in \mathcal{AC}(P)$, and $\alpha \in \{0, 1\}$, we have*

$$x \simeq y \Longrightarrow [x]^\alpha \cong [y]^\alpha\,. \quad (12)$$

Note 4.15. Clearly, the converse implication in (12) is also true.

Lemma 4.16. *For $x, y \in \mathcal{AC}(P)$,*

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

Theorem 4.17. *For two non-zero monomial connectors $x, y \in \mathcal{AC}(P)$, we have*

$$x \cong y \Longleftrightarrow \begin{cases} x \simeq y \\ x \cdot 1' \simeq y \cdot 1' \\ \#x > 0 \Leftrightarrow \#y > 0\,. \end{cases} \quad (13)$$

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

Corollary 4.18. *For $x \in \mathcal{AC}(P)$ such that $\#x > 0$, we have $x \cdot 0' \cong x$.*

Corollary 4.19. *For $x, y \in \mathcal{AC}(P)$, $[x]'\,[y]' \cong [[x]'\,[y]']'$.*

4.4 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.

It can be shown [9] that, for $\mathcal{AS}(P)$, fusion of typed connectors is also associative, that is for $x, y, z \in \mathcal{AS}(P)$

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

It follows that dropping the brackets immediately provides an isomorphism between $\mathcal{AS}(P)$ and $\mathcal{AI}(P)$.

Corollary 4.19 shows that fusion of typed connectors is equally associative in $\mathcal{AT}(P)$, that is for $x, y, z \in \mathcal{AT}(P)$

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

Notice that $[1] \notin \mathcal{AT}(P)$. The identity element for fusion in $\mathcal{AT}(P)$ is $[0]'$ (cf. Corollary 4.18).

Proposition 4.20.

1. *The axiomatisation of $\mathcal{AS}(P)$ is sound and complete.*
2. *The axiomatisation of $\mathcal{AT}(P)$ is sound. It becomes complete with the additional axiom*

$$[x]'\,y = [x]'\,y + [x]'\,. \quad (14)$$

PROOF. 1. This affirmation follows from the associativity of synchronisation in $\mathcal{AI}(P)$ and the rule (8) in the definition of the semantics of $\mathcal{AC}(P)$.

2. The soundness of the axiomatisation of $\mathcal{AT}(P)$ follows from Corollary 4.18 and Corollary 4.19, the idempotence of union and synchronisation in $\mathcal{AI}(P)$, and the rule (9). The completness is proven by showing that the associativity of fusion and the absorbtion axiom (14) allow to define a normal form, coinciding for equivalent terms. □

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 connectors of the BIP program and the set of all the active ports, 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 and 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 interactions contained in 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 active, and compute the maximal interactions of the connector $p'\,[q\,[s + r] + r\,q']'[t + u]$.

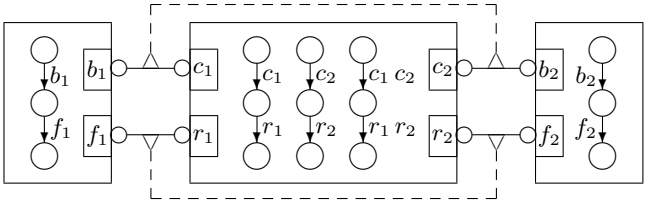
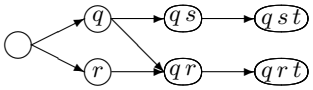


Figure 9: Modelling a joint call of two functions.

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 these are, indeed, the two maximal interactions in the given connector, when ports p and u are not active.

5.2 d-Synchronous component model

Modelling heterogeneous models in BIP, and in particular synchronous models, 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# [11, 19]. 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 9 for an example with two Callees).

Joint function calls involve the parallel computation of several functions. The *Caller* awaits for all the invoked functions to complete their execution. For instance, modelling 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 9, 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 $P = \bigcup_{i=1}^n P_i$.

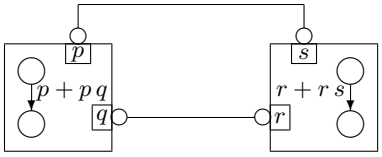


Figure 10: Causality loop.

For an integer parameter $d \in [1, m]$, 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

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

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 $\gamma_1 \subseteq \gamma_2 \subseteq \cdots \subseteq \gamma_m$.

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*

$$\prod_{i=1}^{n} [G_i]' \cap \gamma_d\,, \tag{15}$$

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 (15), 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 for the d-synchronous semantics.

Example 5.4 (Causality loop)**.** Consider the interconnected system shown in Figure 10. 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 terminology of synchronous languages [6].

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

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

Example 5.5 (Modulo-8 counter)**.** For synchronous semantics the system in Figure 11 is equivalent to the Modulo-8 counter given in Example 4.11 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:

$$[p + p\,q]'\,[r + r\,s]'\,[t + t\,u]' \;\cap\; p'\,[q\,r]'\,[s\,t]'\,u' \;=$$
$$=\; p + p\,q\,r + p\,q\,r\,s\,t + p\,q\,r\,s\,t\,u\,.$$

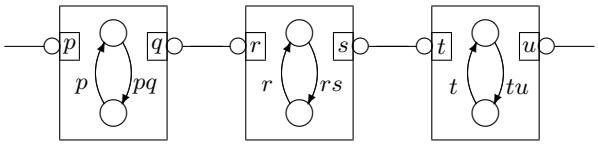


Figure 11: Synchronous modulo-8 counter.

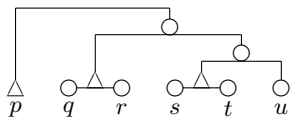


Figure 12: Synthesised connector for mod-8 counter.

As shown in the above examples, it is important to compute efficiently the interactions of a system for d-synchronous semantics with $d > 1$. To avoid costly enumeration, 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 11 provides the following relations: $p \rightarrow q$ (p can trigger q, i.e. p is a necessary condition for q), $r \rightarrow s$, and $t \rightarrow u$; on the other hand, q and r must synchronise, as well as s and t. All these relations together, are represented by the graph

$$p \rightarrow q\, r \rightarrow s\, t \rightarrow u\,. \tag{16}$$

Each path in such dependency graph represents a causality chain. The graph shown in (16) represents the connector $p'\,[[q\,r]'\,[[s\,t]'\,u]]$, shown in Figure 12 (cf. also Figure 8). In general, this technique allows the synthesis of the connectors of a 1-synchronous model equivalent to a given synchronous model.

5.3 Incremental decomposition of connectors

In [15, 20], 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 $s'r_1r_2r_3$ is given 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.

Problem 5.6 (Decomposition of Connectors)**.** Given a connector $K \in \mathcal{AC}(P)$ and a subset of ports $P_0 \subset P$, construct two families $K_i \in \mathcal{AC}(P_0)$ and $\widetilde{K}_i \in \mathcal{AC}(P \setminus P_0)$, for $i = 1, \ldots, n$, such that

$$K \;\simeq\; \sum_{i=1}^{n} K_i \cdot \widetilde{K}_i\,.$$

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 [9] is based on term rewriting rules, whereas the second technique, presented below, uses the notion of derivation.

Theorem 5.7. *For $p \in P$ and $K \in \mathcal{AC}(P)$ there exists a unique, up to equivalence,* derivative $dK/dp \in \mathcal{AC}(P \setminus \{p\})$ *such that*

$$K \;\simeq\; p \cdot \left[\frac{dK}{dp}\right] \;+\; K(0/p)\,. \tag{17}$$

Derivatives can be computed by applying the axioms of $\mathcal{AC}(P)$ and the following rules.

Proposition 5.8. *For $K \in \mathcal{AC}(P)$ and $\alpha, \beta \in \{0, 1\}$,*

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

$$\left[\frac{dK_1}{dp}\right]^{\alpha} [K_2(1)] \;+\; [K_1(1)] \left[\frac{dK_2}{dp}\right]^{\beta}.$$

Example 5.9. Consider the connector $K = \Big[[s'\,r_1]'\,r_2\Big]'\,r_3$ modelling a broadcast. Let us decompose it with respect to s. We have

$$\frac{dK}{ds} \;\simeq\; \Big[[1'\,r_1]'\,r_2\Big]'\,r_3 \quad \text{and} \quad K(0) \simeq 0\,. \tag{18}$$

Substituting (18) into (17), and applying the equivalence $x\,[1'\,y] \simeq [x]'\,y$, we obtain

$$\begin{aligned} K \;&\simeq\; s\left[\Big[[1'r_1]'r_2\Big]'r_3\right] \;\simeq\; s\left[\Big[r_2'r_1 + 1'r_1\Big]'r_3\right] \\ &\simeq\; s\Big[[r_2'r_1]'r_3 + r_3'r_1 + 1'r_1\Big] \\ &\simeq\; s\Big[[r_2'r_1]'r_3 + r_3'r_1\Big] + s'r_1. \end{aligned}$$

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 [5], 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.

Due to space limitations, we could not provide detailed results about applications of $\mathcal{AC}(P)$. 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 independently from computation. 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.

Acknowledgements

The authors would like to thank Ananda Basu, Marius Bozga, Paul Caspi, Susanne Graf and Yassine Lakhnech for constructive remarks and valuable discussion.

7. REFERENCES

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

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

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

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In 4^{th} *IEEE International Conference on Software Engineering and Formal Methods (SEFM06)*, pages 3–12, Sept. 2006. Invited talk.

[5] A. Basu, L. Mounier, M. Poulhiès, J. Pulou, and J. 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] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone. The synchronous languages twelve years later. *Proc. of the IEEE, Special Issue on Embedded Systems*, 91(1):64–83, 2003.

[7] M. Bernardo, P. Ciancarini, and L. Donatiello. On the formalization of architectural types with process algebras. In *SIGSOFT FSE*, pages 140–148, 2000.

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

[9] S. Bliudze and J. Sifakis. The algebra of connectors — structuring interaction in BIP. Technical Report TR-2007-3, VERIMAG, 2007. `http://www-verimag.imag.fr/index.php?page=techrep-list`.

[10] R. Bruni, J. L. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari. New insights on architectural connectors. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *IFIP TCS*, pages 367–380. Kluwer, 2004.

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

[12] J. Eker, J. Janneck, E. 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.

[13] J. L. Fiadeiro. *Categories for Software Engineering*. Springer-Verlag, Apr. 2004.

[14] G. Gößler and J. Sifakis. Component-based construction of deadlock-free systems: Extended abstract. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 420–433, Mumbai, India, Dec. 2003. Springer.

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

[16] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall, Apr. 1985.

[17] F. Maraninchi and Y. Rémond. Argos: an automaton-based synchronous language. *Computer Languages*, 27:61–92, 2001.

[18] R. Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.

[19] nesC: A programming language for deeply networked systems. `http://nescc.sourceforge.net/`.

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

[21] B. Spitznagel and D. Garlan. A compositional formalization of connector wrappers. In *ICSE*, pages 374–384. IEEE Computer Society, 2003.