The Algebra of Connectors — Structuring Interaction in BIP

Simon Bliudze, Joseph Sifakis

***Abstract*— We provide an algebraic formalization of *connectors* in the BIP component framework. A connector relates a set of typed ports. Types are used to describe different modes of synchronization: rendezvous and broadcast, in particular.**

Connectors on a set of ports P are modeled 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) synchronization types — *trigger* or *synchron* — that determine modes of synchronization. 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 methods for incremental model decomposition, and efficient implementation by using symbolic techniques.

***Index Terms*— C.3.d Real-time and embedded systems, C.0.e System architectures, integration and modeling, C.0.f Systems specification methodology, B.4.3 Interconnections, C.2.0.a Architecture**

I. Introduction

A key idea in systems engineering is that complex systems are built by assembling components. Components are systems characterized by an abstraction, which is adequate for composition and reuse. 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 analyzing coordination between components. Such a paradigm would allow system designers and implementers to formulate their solutions in terms of tangible, well-founded and organized 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.

We propose the *Algebra of Connectors* for modeling interaction in component-based systems.

Both authors are with VERIMAG, Centre Équation, 2 av de Vignate, 38610, Gières, France

The term “connector” is widely used in the component frameworks literature with a number of different interpretations. These interpretations can be loosely separated in two main categories: in the *data flow* setting, connectors define the way data is transferred between components; alternatively, in what we call *control flow* setting, connectors rather define synchronization constraints, while pushing to the second plan or completely abstracting the data flow.

Control flow connectors are often specified in an operational setting, usually a process algebra. In [1], 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 [2], a connector is defined as a set of processes, with one process for each role of the connector, plus one process for the “glue” that describes how all the roles are bound together. A similar approach is developed by J. Fiadeiro and his colleagues in a categorical framework for CommUnity [3].

All the above models define connectors that can exhibit complex behavior. That is, computation is not limited to the components, but can be partly performed in the connectors. In [4], an algebra of connectors is developed that allows, in particular, an algebraic translation of the categorical approach used in CommUnity. This algebra allows to construct *stateless* connectors from a number of basic ones.

In the data flow setting, *Reo* [5] is a channel-based exogenous coordination model for multi-agent systems. It uses connectors compositionally built out of different types of channels formalized in data-stream semantics and interconnected by using nodes. The connectors in Reo allow computation, but their computational aspects are limited to the underlying channels.

Our approach is closest to that of [4], as it focuses on stateless connectors in a control flow setting. We consider connectors as relations between ports with synchronization types, which allows to describe complex coordination patterns with a small set of basic primitives.

The Algebra of Connectors allows the description of coordination between components in terms of structured connectors involving communication ports. It formalizes mechanisms and concepts that have been implemented in the *Behavior-Interaction-Priority* (BIP) component framework developed at Verimag [6], [7]. BIP distinguishes between three basic entities: 1) Behavior, described as extended automata, including a set of transitions labeled 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 parametrized by a set of interactions.

We present an algebraic formalization of the concept of *connector*, introduced in [8], [9] as a set of communication ports belonging to different components that may be involved in some interaction. To express different types of synchronization, the

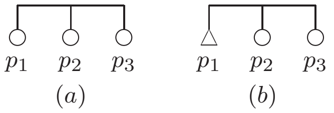


Fig. 1. Connectors for rendezvous (a) and broadcast (b)

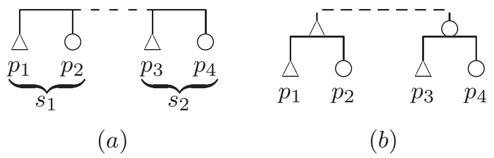


Fig. 2. Fusion (a) and structuring (b) of connectors

ports of a connector have a type (attribute) *trigger* or *synchron*. Given a connector involving a 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 Fig. 1, we show two connectors modeling 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 this 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. Fig. 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 Fig. 2(b), where the connectors involving $p_1\,p_2$ 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 modeling 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. Sec. II provides a succinct presentation of the basic semantic model for BIP and in particular, its composition parametrized by sets of interactions. In Sec. III, we present the Algebra of Interactions. It is a simple algebra used to introduce the Algebra of Connectors presented in Sec. IV. 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.

II. BIP COMPONENT FRAMEWORK

A. Basic semantic model

BIP is a component framework for constructing systems by superposing three layers of modeling: Behavior, 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 synchronization 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 modeling and composing layered components. Atomic components are communicating automata extended with C functions and data. Their transitions are labeled with sets of communication ports. The BIP language also allows composition of components parametrized 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 [6], [10].

We provide a succinct formalization 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.

1) The Behavior layer in BIP is modeled by labeled transition systems.
   Definition 2.2: A labeled 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 labeled by an interaction.
   For any pair of states $q, q' \in Q$ and an interaction $a \in 2^P$, we write $q \xrightarrow{a} q'$, iff $(q, a, q') \in \rightarrow$.
   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 modeled by transition systems $B_i = (Q_i, P_i, \rightarrow_i)$, 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.
2) The second layer, Interaction, is modeled by the *composition operator* parametrized by the set of allowed interactions.
   Definition 2.3: The composition of components $\{B_i\}_{i=1}^{n}$, parametrized 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

TABLE I

INTERACTION SETS FOR BASIC COORDINATION SCHEMES

	Set of interactions γ
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	$\{s,\ s\,r_1,\ s\,r_1\,r_2,\ s\,r_1\,r_2\,r_3\}$

the least set of transitions satisfying the rule

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

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. Observe also that γ defines the set of *allowed* interactions: it is possible that an interaction $a \in \gamma$ is never enabled in $\gamma(B_1,\ldots,B_n)$.

3) The third layer, Priorities, allows to restrict non-determinism in the product behavior due to the fact that several distinct interactions can be enabled at the same time.

Definition 2.4: 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 lower 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

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

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

Example 2.5 (Sender/Receivers): Fig. $3(a)$ 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. Tab. I specifies γ for four different coordination schemes.

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

Broadcast: this allows all interactions 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: a message is either 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 a rendezvous.

Causality chain: 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 the maximal number of ports has higher priority, that is we take $\pi = \{(a,a') \,|\, a \subset a'\}$.

Throughout the paper, the above *maximal progress* rule is applied. In other words, amongst the enabled interactions, are preferred the ones involving the maximal number of ports. Notice that, by enforcing maximal progress, priorities allow to express broadcast.

Example 2.6 (Modulo-8 counter): Fig. $3(b)$ shows a model for the Modulo-8 counter, presented in [11], 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.

B. Modeling parallel composition operators in BIP

The composition operator, introduced in the previous section, can express usual parallel composition operators, such as the ones used in CSP [12] and CCS [13]. Indeed, both CCS and CSP can be given SOS-style operational semantics: individual processes are represented by labeled transition systems, whereas parallel composition operators restrict the transitions of product transition systems. In BIP, parallel composition operators of these process algebras are expressed as presented below.

1) Communicating Sequential Processes (CSP): In CSP [12], components can communicate over a set of *channels*, common to the system. Full semantics of CSP can be found in [14, Chapter 7]. We will limit ourselves to the most essential case.

Atomic components (processes) in CSP can be considered as labeled transition systems defined as triples $(Q,C,\rightarrow)$, 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 labeled 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 $i = 1,2$. For any $c \in C'$,

$$\frac{q_1 \overset{c}{\rightarrow}_1 q_1' \quad \wedge \quad q_2 \overset{c}{\rightarrow}_2 q_2'}{q_1 ||_{C'} q_2 \overset{c}{\rightarrow} q_1' ||_{C'} q_2'},$$

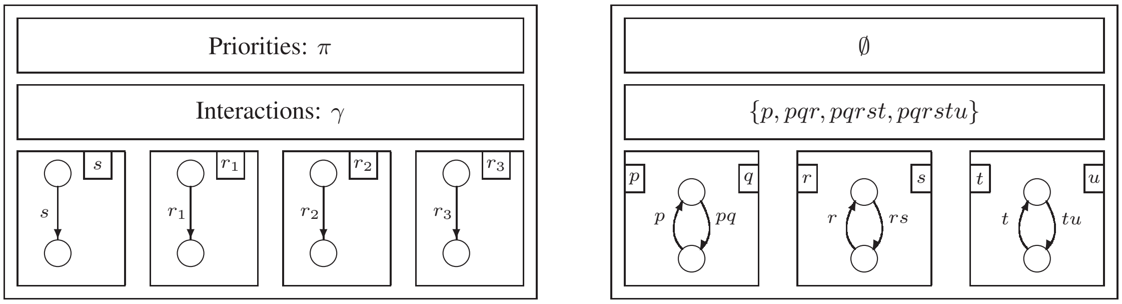
whereas, for any $c \in C \setminus C'$,

$$\frac{q_1 \overset{c}{\rightarrow}_1 q_1'}{q_1 ||_{C'} q_2 \overset{c}{\rightarrow} q_1' ||_{C'} q_2} \quad \text{and} \quad \frac{q_2 \overset{c}{\rightarrow}_2 q_2'}{q_1 ||_{C'} q_2 \overset{c}{\rightarrow} q_1 ||_{C'} q_2'},$$

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

$$\begin{aligned}\gamma_{CSP} \quad &= \quad \Big\{\{B_1.c, B_2.c\} \,\Big|\, c \in C'\Big\} \ \cup\ \Big\{\{B_1.c\} \,\Big|\, c \notin C'\Big\} \\ &\quad \cup \Big\{\{B_2.c\} \,\Big|\, c \notin C'\Big\}.\end{aligned}$$

2) Calculus of Communicating Systems (CCS): In CCS [13], all communication is performed by binary interactions between complementary actions a and $\overline{a}$. Let A be the set of actions, and $L = A \cup \overline{A} \cup \{\tau\}$ be the set of labels, where τ represents an internal (non-observable transition).



(a) (b)

Fig. 3. A system with four atomic components (a) and a modulo-8 counter (b)

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

$$\frac{q_2 \ \xrightarrow{l}_2 \ q_2', \quad l \in \{a, \overline{a}\}}{q_1 \mid\mid q_2 \ \xrightarrow{l} \ q_1 \mid\mid q_2'}, \qquad \frac{q_1 \ \xrightarrow{l}_1 \ q_1', \quad l \in \{a, \overline{a}\}}{q_1 \mid\mid q_2 \ \xrightarrow{l} \ q_1' \mid\mid q_2},$$

and

$$\frac{q_1 \ \xrightarrow{a}_1 \ q_1' \quad \wedge \quad q_2 \ \xrightarrow{\overline{a}}_2 \ q_2'}{q_1 \mid\mid q_2 \ \xrightarrow{\tau} \ q_1' \mid\mid q_2'}.$$

Another important operator in CCS is restriction $B \setminus a$, which excludes a given action from communication. In $(B_1||B_2) \setminus a$, restriction enforces synchronization on a 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

$$\begin{aligned} \gamma_{CCS,1} \quad &= \quad \Big\{\{B_i.a, B_{3-i}.\overline{a}\} \,\Big|\, i = 1, 2\Big\} \ \cup \ \Big\{\{B_1.l\} \,\Big|\, l \in L\Big\} \\ &\cup \quad \Big\{\{B_2.l\} \,\Big|\, l \in L\Big\}. \end{aligned}$$

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

$$\begin{aligned} \gamma_{CCS,2} &= \Big\{\{B_i.a, B_{3-i}.\overline{a}\} \,\Big|\, i = 1, 2\Big\} \\ \cup \quad &\Big\{\{B_1.l\} \,\Big|\, l \in L \setminus \{a, \overline{a}\}\Big\} \ \cup \ \Big\{\{B_2.l\} \,\Big|\, l \in L \setminus \{a, \overline{a}\}\Big\}. \end{aligned}$$

III. The Algebra of Interactions

A. Syntax, axiomatization, and semantics

Consider a family of components, indexed by I and equipped with sets of ports P_i, for $i \in I$, through which they can interact. The communication model considered implies atomic synchronization of all ports participating in a given interaction. Therefore, each interaction is represented by the set of the 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

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

with $p \in P$ an arbitrary port, and where '+' and '·' are binary operators, respectively called *union* and *synchronization*. Synchronization binds stronger than union.

TABLE II
$\mathcal{AI}(P)$ AND $\mathcal{AC}(P)$ REPRESENTATION OF BASIC COORDINATION SCHEMES

	$\mathcal{AI}(P)$	$\mathcal{AC}(P)$
Rendezvous	$s\, r_1\, r_2\, r_3$	$s\, r_1\, r_2\, r_3$
Broadcast	$s\,(1 + r_1)\,(1 + r_2)\,(1 + r_3)$	$s'\, r_1\, r_2\, r_3$
Atomic Broadcast	$s\,(1 + r_1\, r_2\, r_3)$	$s'\,[r_1\, r_2\, r_3]$
Causality Chain	$s\,(1 + r_1\,(1 + r_2\,(1 + r_3)))$	$s'\,[r_1'\,[r_2'\, r_3]]$

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) Synchronization '·' is idempotent, associative, and commutative, has an identity element 1, and an absorbing element 0; synchronization 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 $\|\cdot\| : \mathcal{AI}(P) \rightarrow 2^{2^P}$, defined by

$$\begin{aligned} \|0\| \ = \ \emptyset, \quad \|1\| \ = \ \{\emptyset\}, \quad \|p\| \ &= \ \Big\{\{p\}\Big\}, \ \text{for any } p \in P, \\ \|x_1 + x_2\| \quad &= \quad \|x_1\| \cup \|x_2\|\,, \\ \|x_1 \cdot x_2\| \quad &= \quad \Big\{a_1 \cup a_2 \,\Big|\, a_1 \in \|x_1\|, a_2 \in \|x_2\|\Big\}\,, \\ \|(x)\| \quad &= \quad \|x\|\,. \end{aligned} \tag{4}$$

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

Proposition 3.1: The axiomatization of $\mathcal{AI}(P)$ is sound and complete. 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 flattening the elements, applying the distributivity, 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 sets for the four coordination schemes of Ex. 2.5 are represented by the elements in the first column of Tab. II. 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.

B. 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:

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

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 Tab. III.

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:

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) Synchronization 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 $\beta(q) = \overline{p}\,q\,\overline{r}\,\overline{s}$.

IV. The Algebra of Connectors

We provide an algebraic formalization of the concept of connector, supported by the BIP language [6]. Connectors can express complex coordination schemes combining synchronization by rendezvous and broadcast.

A. 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

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

for $p \in P$, and where '$+$' and '$\cdot$' are binary operators, respectively called *union* and *fusion*, brackets '$[\cdot]$' and '$[\cdot]'$' are unary *typing* operators. Fusion binds stronger than union.

Union has the same meaning as union in $\mathcal{AI}(P)$. Fusion is a generalization of the synchronization 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 synchronization 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 operator.

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 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 operator.

Definition 4.3: For any term x of the form $\prod_{i\in I}[x_i]^*$, we denote by $\#x$ the number of its trigger elements, 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 $\#x \stackrel{def}{=} \max\{\#x_i \mid i \in [1, n]\}$. We say that x has *strictly positive degree* iff $\min\{\#x_i \mid i \in [1, n]\} > 0$.

The algebraic structure on $\mathcal{AC}(P)$ inherits most of the axioms from $\mathcal{AI}(P)$. However, it should be noted that associativity of fusion does not hold, when typing is applied instead of simple grouping, e.g. equality does not hold for any pair of $[x]\,[y]\,[z]$, $[x]\,\Big[[y]\,[z]\Big]$, and $\Big[[x]\,[y]\Big]\,[z]$, for $x, y, z \in \mathcal{AC}(P)$.

Axioms (AC1): The operators satisfy the following axioms.

1) Union '$+$' is associative, commutative, idempotent, and has an 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) $\Big[[x]^\alpha\Big]^\beta = [x]^\beta$,
   c) $[x+y]^\alpha = [y]^\alpha + [x]^\alpha$,
   d) $[x]'\,[y]' = [x]'\,[y] + [x]\,[y]'$.

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 is a direct consequence of axiom (3d). ■

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: Clearly, any element of $\mathcal{AC}(P)$ can be uniquely rewritten, by associativity and distributivity, in a form without parentheses. The semantics of $\mathcal{AC}(P)$ is then given in terms of the Algebra of Interactions $\mathcal{AI}(P)$ by the function $|\cdot| : \mathcal{AC}(P) \to \mathcal{AI}(P)$, defined by the rules

$$|[p]| = p\,, \text{ for } p \in P \cup \{0, 1\} \tag{6}$$

$$|x_1 + x_2| = |x_1| + |x_2|\,, \tag{7}$$

$$\left|\prod_{i=1}^n [x_i]\right| = \prod_{i=1}^n |x_k|\,, \tag{8}$$

$$\left|\prod_{i=1}^n [x_i]' \cdot \prod_{j=1}^m [y_j]\right| = \sum_{i=1}^n |x_i| \cdot \left(\prod_{k\neq i}\Big(1 + |x_k|\Big) \cdot \right. \tag{9}$$

$$\left. \cdot \prod_{j=1}^m \Big(1 + |y_j|\Big)\right)\,, \tag{10}$$

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

TABLE III

CORRESPONDENCE BETWEEN $\mathcal{AI}(\{p,q\})$ AND BOOLEAN FUNCTIONS WITH TWO VARIABLES

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

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 (9) can be decomposed in two steps: 1) the application of Lem. 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 Ex. 4.8 in the following section with Ex. 2.5 and Ex. 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.

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

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 axiomatization of $\mathcal{AC}(P)$ is sound, that is, for $x, y \in \mathcal{AC}(P)$,

$$x = y \Longrightarrow |x| = |y| \, . \tag{11}$$

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. $x \simeq y \overset{def}{\Longleftrightarrow} |x| = |y|$.

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

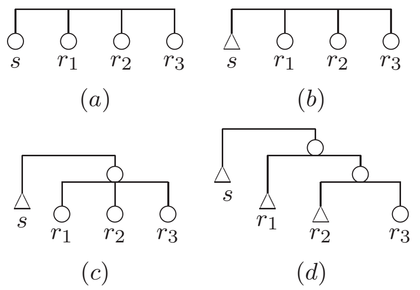


Fig. 4. Graphical representation of connectors.

B. Examples

The typing operator induces a hierarchical structure. Connectors can be represented as sets of trees, having ports at their leaves. Indeed, by distributivity of fusion and typing over union, each connector can be represented as a union of monomial connectors. The latter can be, in turn, represented by their parse trees. We use triangles and circles to represent types: triggers and synchrons, respectively.

Example 4.8 (Sender/Receiver continued): In $\mathcal{AC}(P)$, the interaction sets for the four coordination schemes of Ex. 2.5 are represented by the elements in the second column of Tab. II.

Notice that $\mathcal{AC}(P)$ allows compact representation of interactions, and, moreover, explicitly captures the difference between broadcast and rendezvous. The graphical representations of the four connectors are shown in Fig. 4.

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

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

Example 4.10 (Broadcast): For the broadcast connector $s' \, r_1 \, r_2 \, r_3$ (Fig. 4(b), reproduced in Fig. 5(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 $|s' \, r_1| \; = \; s \, (1 + r_1)$.

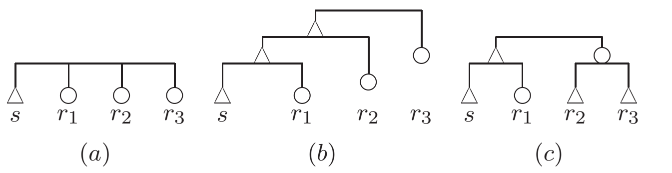


Fig. 5. Three connectors realizing a broadcast

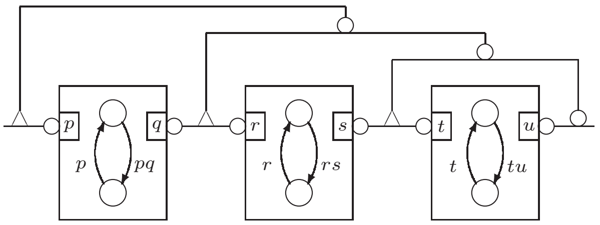


Fig. 6. Modulo-8 counter.

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 $[[s' \, r_1]' \, r_2]' \, r_3$ (Fig. 5(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 $[s' \, r_1]' \, [r_2' \, r_3']$ (Fig. 5(c)).

Example 4.11 (Modulo-8 counter): In the model shown in Fig. 6, the causality chain pattern (cf. Fig. 4(d)) is applied to connectors p, $q\,r$, $s\,t$, and u. Thus the interactions of the Modulo-8 counter in Fig. 3(b) are also modeled by a single structured connector $p' \left[[q\,r]' \, [[s\,t]' \, u] \right]$:

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

C. 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.

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 (5)

$$\begin{array}{rcl} s & ::= & [0] \mid [1] \mid [p] \mid [x] \\ x & ::= & s \mid x \cdot x \mid x + x \mid (x)\,. \end{array} \tag{12}$$

$\mathcal{AC}_S(P)$ inherits the axioms of $\mathcal{AC}(P)$, and consequently fusion of typed connectors is also non-associative. The *Algebra of Synchrons*, $\mathcal{AS}(P)$, is obtained by adding to the axioms of $\mathcal{AC}_S(P)$ the following axiom:

$$\Big[[x]\,[y] \Big]\,[z] \;=\; [x]\,[y]\,[z] \;=\; [x]\,\Big[[y]\,[z] \Big] \quad (ASSOC)\,. \tag{13}$$

Thus, in $\mathcal{AS}(P) \stackrel{def}{=} \mathcal{AC}_S(P)/ASSOC$ fusion is associative also on typed connectors (i.e. when brackets are used for grouping instead of parentheses) and satisfies the same axioms as synchronization in $\mathcal{AI}(P)$. Consequently, dropping the brackets in the elements of $\mathcal{AS}(P)$ immediately provides an isomorphism with $\mathcal{AI}(P)$.

Proposition 4.12: The axiomatization of $\mathcal{AS}(P)$ is sound and complete.

Proof: This proposition follows from the associativity of synchronization in $\mathcal{AI}(P)$ and the rule (8) in the definition of the semantics of $\mathcal{AC}(P)$. ■

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 (5) to triggers:

$$\begin{array}{rcl} t & ::= & [0]' \mid [1]' \mid [p]' \mid [x]' \\ x & ::= & t \mid x \cdot x \mid x + x \mid (x)\,. \end{array} \tag{14}$$

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 that has to be observed here. Indeed, as $[1] \not\in \mathcal{AC}_T(P)$, the identity element for fusion is $[0]'$ (cf. Cor. 5.5 in Sec. V-A).

As in the case of $\mathcal{AC}_S(P)$, fusion of typed connectors is non-associative in $\mathcal{AC}_T(P)$. Again, we consider a quotient algebra $\mathcal{AT}(P) \stackrel{def}{=} \mathcal{AC}_T(P)/ASSOC'$ obtained by adding to the axioms of $\mathcal{AC}_T(P)$ the axiom

$$\Big[[x]'\,[y]' \Big]'\,[z]' \;=\; [x]'\,[y]'\,[z]' \;=\; [x]'\,\Big[[y]'\,[z]' \Big]' \quad (ASSOC')\,.$$

where $x, y, z \in \mathcal{AC}_T(P)$.

Proposition 4.13: The axiomatization of $\mathcal{AT}(P)$ is sound. It becomes complete with the additional axiom

$$[x]'\,y = [x]'\,y + [x]'\,. \tag{15}$$

Proof: The soundness of this axiomatization follows from Cor. 5.5 and Cor. 5.7(2), obtained in Sec. V-A, the idempotence of union and synchronization in $\mathcal{AI}(P)$, and the rule (9). The completeness is proved by showing that the associativity of fusion and the absorption axiom (15) 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 $\mathcal{AC}(P)$ related as shown in Fig. 7. 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 arrows from $\mathcal{AC}(P)$ to $\mathcal{AI}(P)$ define the semantics of $\mathcal{AC}(P)$ in terms of interactions. The remaining arrows ($\mathcal{AT}(P) \rightarrow \mathcal{AI}(P)$ and the dashed ones) are obtained by commutative closure of the existing ones.

V. CONGRUENCE RELATION ON $\mathcal{AC}(P)$

Observe that, in general, two equivalent terms are not congruent in the sense that one cannot be substituted for another in certain contexts. For example, $p' \simeq p$, but $p'\,q \not\simeq p\,q$, for $p, q \in P$.

Definition 5.1: 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 $z \not\in P$,

$$x \cong y \Longrightarrow \forall E \in \mathcal{AC}(P \cup \{z\}), \quad E(x/z) \simeq E(y/z)\,, \tag{16}$$

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

In the following subsections, we provide a characterization of the congruence relation in terms of semantic equivalence, as well as its complete axiomatization.

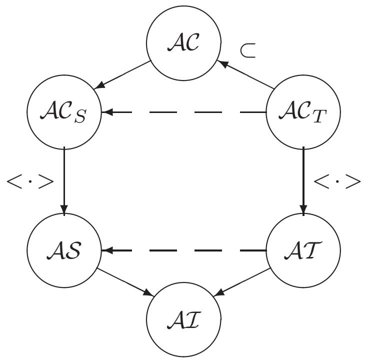


Fig. 7. Hierarchy of algebras.

A. Characterization of the congruence relation

Proposition 5.2: 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

$$x \simeq y \Longleftrightarrow [x]^\alpha \cong [y]^\alpha \,. \tag{17}$$

Proof: The right-to-left implication is trivial. To prove the left-to-right implication, 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

$$x \simeq y \Longrightarrow [x]^\alpha \cdot w \;\simeq\; [y]^\alpha \cdot w \tag{18}$$

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 (18) 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) $\alpha = 0, \#w = 0$, 2) $\alpha = 1, \#w = 0$, 3) $\alpha = 0, \#w > 0$, and 4) $\alpha = 1, \#w > 0$.

Notice that the case 4 follows from the cases 2 and 3 by application of the reduction axiom (3d) (or equivalently Lem. 4.4). The proofs of the three other cases are similar. Here, we prove the case 3. The other two cases can be treated in the same way. Taking $\alpha = 0$ and $\#w > 0$, 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. ■

Lemma 5.3: For $x, y \in \mathcal{AC}(P)$, $x \cong y$ iff $xz \simeq yz$, for all monomials $z \in \mathcal{AC}(P)$.

Proof: Follows directly from Def. 5.1 and Prop. 5.2. ■

Theorem 5.4: Let $x, y \in \mathcal{AC}(P)$ be two non-zero monomial connectors, we then have

$$x \cong y \Longleftrightarrow \begin{cases} x \simeq y \\ x \cdot 1' \simeq y \cdot 1' \\ \#x > 0 \Leftrightarrow \#y > 0\,. \end{cases} \tag{19}$$

Proof: The left side obviously implies the right side: 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 participating neither in x nor in y. Therefore, we only have to show that the three conditions on the right-hand side imply $x \cong y$.

By Lem. 5.3, 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 monomials, 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 typing $\{\alpha_i\}_{i=1}^n$, $\{\beta_i\}_{i=1}^m$, and $\{\gamma_i\}_{i=1}^l$ in $\{0,1\}$,

$$\begin{gathered} x \equiv [x_1]^{\alpha_1}\cdot\ldots\cdot[x_n]^{\alpha_n}\,, \quad y \equiv [y_1]^{\beta_1}\cdot\ldots\cdot[y_m]^{\beta_m}\,, \\ z \equiv [z_1]^{\gamma_1}\cdot\ldots\cdot[z_l]^{\gamma_l}\,. \end{gathered} \tag{20}$$

As in Prop. 5.2, 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 $\|x\,z\| = \|y\,z\|$. However, by symmetry, it is sufficient to show that $\|x\,z\| \subseteq \|y\,z\|$, i.e. that for an arbitrary interaction $a \in \|x\,z\|$, we also have $a \in \|y\,z\|$.

Case 1. ($\#x = \#y = \#z = 0$) By the definition of 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 $a \in \|y\,z\|$.

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 $a \in \|y\,z\|$, as we also have $\#y > 0$.

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 $a = a_0 \cup \bigcup_{i\in I} a_i$. Rewriting $x \cdot 1'$ as a sum of terms of degree one (cf. Lem. 4.4), one has

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

and similarly for $y \cdot 1'$.

By the choice of a_i, we have

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

and there exists $J \subset [1,m]$ such that $\bigcup_{i\in I} a_i \in \left\|\prod_{j\in J}[y_j]\right\|$. Hence, $a = a_0 \cup \bigcup_{i\in I} a_i \in \left\|z \cdot \prod_{j\in J}[y_j]\right\| \subset \|y \cdot z\|$. ■

The following two corollaries are used for the axiomatization of the Algebra of Triggers, defined in Sec. IV-C.2.

Corollary 5.5: For $x \in \mathcal{AC}(P)$, such that $\deg(x) > 0$, we have $x \cdot 0' \cong x$.

Proof: For monomials, 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 the general case, we apply distributivity observing that all monomials also have non-zero degrees. ■

Note 5.6: 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 (20). Th 5.4 can also be easily generalized to arbitrary x and y having strictly positive degree (recall Def. 4.3).

Corollary 5.7: For any $x, y, z \in \mathcal{AC}(P)$.

1) $[x]'\,[y]\,[z] \;\cong\; [x]'\left[[y]'\,[z]'\right]$.
2) $[x]'\,[y]' \;\cong\; \left[[x]'\,[y]'\right]'$,

B. Complete axiomatization of the congruence relation

Congruence relation $\cong$ is the largest congruence relation respecting the semantics of $\mathcal{AI}(P)$. Therefore, it is clear that the axiomatization given in Sec. IV-A is sound with respect to this relation (i.e. we have $x = y \Rightarrow x \cong y$, for any $x, y \in \mathcal{AC}(P)$). However, it is easy to verify that it is not complete, e.g. by considering Prop. 5.2. We now extend this axiomatic system to provide a complete axiomatization of $\cong$.

Axioms (AC2): The operators 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) $\Big[[x]^\alpha\Big]^\beta = [x]^\beta$,
   c) $[x+y]^\alpha = [x]^\alpha + [y]^\alpha$,
4) For $x, y, z \in \mathcal{AC}(P)$ and $\alpha \in \{0,1\}$:
   a) $[x]'\,[0] = [x]'$,
   b) $\Big[[x]\,[0]\Big]^\alpha = [0]$,
   c) $[x]' + [x] = [x]'$,
   d) $[x]'\,[y]' = [x]'\,[y] + [y]'$,
   e) $[x]'\,[y]\,[z] = [x]'\,\Big[[y]'\,[z]'\Big]$,
   f) $[x]'\,[y] = \Big[[x]'\,[y]\Big]' + [0]'\,[y]$,
   g) $[x]\,[y] = \Big[[x]\,[y]\Big] + [0]'\,[x]\,[y]$,
   h) $\Big[[x]'\,[y]\Big]^\alpha = [x]^\alpha + \Big[[x]\,[y]\Big]^\alpha$,
   i) $\Big[\Big[[x]\,[y]\Big]\,[z]\Big]^\alpha = \Big[[x]\,[y]\,[z]\Big]^\alpha$.

Lemma 5.8: For $x, y \in \mathcal{AC}(P)$ and $\alpha \in \{0,1\}$, holds $[x]'\,[y]^\alpha + [x]' = [x]'\,[y]^\alpha$.

Proof: For $\alpha = 0$, we have

$$\begin{aligned}
[x]'\,[y] + [x]' &\overset{(4f)}{=} \Big[[x]'\,[y]\Big]' + [0]'\,[y] + [x]' \\
&\overset{(4h)}{=} [x]' + \Big[[x]\,[y]\Big]' + [0]'\,[y] + [x]' \\
&= [x]' + \Big[[x]'\,[y]\Big]' + [0]'\,[y] \\
&\overset{(4h)}{=} \Big[[x]'\,[y]\Big]' + [0]'\,[y] \overset{(4f)}{=} [x]'\,[y]\,.
\end{aligned}$$

For $\alpha = 1$, we have $[x]'\,[y]' + [x]' \overset{(4d)}{=} [x]\,[y]' + [x]' + [x]' = [x]\,[y]' + [x]' \overset{(4d)}{=} [x]'\,[y]'$. ∎

Axs. (4a) and (4e) correspond respectively to Cor. 5.5 and 5.7(1). Ax. (4f) generalizes Cor. 5.7(2):

$$\begin{aligned}
&[x]'\,[y]' \overset{(4f)}{=} \Big[[x]'\,[y]'\Big]' + [0]\,[y]' \overset{(4a)}{=} \Big[[x]'\,[y]'\Big]' + [y]' \overset{(3b)}{=} \\
&\overset{(3b)}{=} \Big[[x]'\,[y]'\Big]' + \Big[[y]'\Big]' \overset{(3c)}{=} \Big[[x]'\,[y]' + [y]'\Big]' = \Big[[x]'\,[y]'\Big]',
\end{aligned}$$

TABLE IV

REWRITING SYSTEM FOR THE PROOF OF COMPLETENESS OF $AC2$

distributivity:					
$x\,(y+z)$	$\rightsquigarrow$	$x\,y + x\,z$,	$[x+y]^\alpha$	$\rightsquigarrow$	$[x]^\alpha + [y]^\alpha$,
absorption:					
$x + [0]^*$	$\rightsquigarrow$	x,	$[x]^\alpha + [x]\,[0]^*$	$\rightsquigarrow$	$[x]^\alpha$,
$x\,[1]$	$\rightsquigarrow$	x,	$[x]' + [x]$	$\rightsquigarrow$	$[x]'$,
$[x]'\,[0]^*$	$\rightsquigarrow$	$[x]'$	$\Big[[x]\,[0]\Big]^\alpha$	$\rightsquigarrow$	$[0]$,
idempotence:					
$x + x$	$\rightsquigarrow$	x,	$[w]^\alpha\,[w]^\beta$	$\rightsquigarrow$	$[w]^{\alpha\vee\beta}$,
external typing:					
$[x]\,[y]$	$\rightsquigarrow$	$\Big[[x]\,[y]\Big] + [x]\,[y]\,[0]'$,	$\Big[[x]^\alpha\Big]^\beta$	$\rightsquigarrow$	$[x]^\beta$,
$[x]'\,[y]$	$\rightsquigarrow$	$\Big[[x]'\,[y]\Big]' + [y]\,[0]'$,			
grouping of synchrons:			reduction:		
$[x]'\,[y]\,[z]$	$\rightsquigarrow$	$[x]'\,\Big[[y]'\,[z]'\Big]$,	$[x]'\,[y]'$	$\rightsquigarrow$	$[x]'\,[y] + [y]'$,
typed equivalence:					
$\Big[\Big[[x]\,[y]\Big]\,[z]\Big]^\alpha$	$\rightsquigarrow$	$\Big[[x]\,[y]\,[z]\Big]^\alpha$,			
$\Big[[x]'\,[y]\Big]^\alpha$	$\rightsquigarrow$	$[x]^\alpha + \Big[[x]\,[y]\Big]^\alpha$.			

where the last equality follows from Lem. 5.8. Similarly, axiom (4d) replaces (3d) from $AC1$:

$$\begin{aligned}
[x]'\,[y]' &= [x]'\,[y]' + [x]'\,[y]' \overset{(4d)}{=} \\
&\overset{(4d)}{=} [x]'\,[y] + [y]' + [x]\,[y]' + [x]' = [x]'\,[y] + [x]\,[y]'\,.
\end{aligned} \tag{21}$$

Axioms (4i) and (4h) correspond to Prop. 5.2 (cf. also (8), (9)).

Proposition 5.9: The following properties hold in $\mathcal{AC}(P)$ for the equality defined by $AC2$. For $x \in \mathcal{AC}(P)$ and $\alpha \in \{0,1\}$,

1) $[x]^\alpha + [x]\,[0]^* = [x]^\alpha$,
2) $[w]'\,[w] = [w]'$, if w is monomial.

Proof: 1. For $\alpha = 0$, we have $[x] + [x]\,[0] = [x]\,([1] + [0]) = [x]\,[1] = [x]$. For $\alpha = 1$, the equality is obtained by substituting 0 in axiom (4d): $[x]' + [x]\,[0]' \overset{(4d)}{=} [x]'\,[0]' \overset{(4a)}{=} [x]'$.

2. For monomial w, from (21), we have $[w]' = [w]'\,[w]' = [w]'\,[w] + [w]\,[w]' = [w]'\,[w]$. ∎

Theorem 5.10: The system $AC2$ is a sound and complete axiomatization for the congruence $\cong$ in $\mathcal{AC}(P)$: for $x, y \in \mathcal{AC}(P)$, $x = y \iff x \cong y$.

Proof: For axioms (4a), (4b), (4e), (4i), and (4h), the soundness is a direct consequence of Cor. 5.5 and Cor. 5.7, and Prop. 5.2, whereas for axiom (4d) it follows directly from Th 5.4 (cf. also Note 5.6). Finally, the soundness of axioms (4c), (4f), and (4g) follows immediately from Lem. 5.3.

To prove the completeness of $AC2$, we consider the transformation rules shown in Tab. IV, where $x, y, z \in \mathcal{AC}(P)$, $\alpha, \beta \in \{0,1\}$, and $w \in \mathcal{AC}(P)$ is a monomial. These rules assign direction to some axioms of $AC2$ and equalities of Prop. 5.9. By limiting the application of the first two external typing rules to the maximal fusion (top level) terms and non-zero x and y, we obtain a terminating and confluent rewriting system.

To verify that this system is terminating, notice that the only transformations, which decrease neither the depth of the hierarchy, nor the number of triggers in a given term, are the first two external typing transformations and the grouping of synchrons. However, it is clear that with the limitations imposed above on the external typing, these three transformations can only be applied a finite number of times.

To verify that this system is confluent, it is sufficient to observe that in the final configuration, we obtain the following form, for $x \in \mathcal{AC}(P)$,

$$x \;=\; \sum_{i=1}^{n} [x_i]' \;+\; \sum_{i=1}^{m} [y_i] \;+\; [0]' \cdot \sum_{i=1}^{l} [z_i]\,, \tag{22}$$

where all x_i, y_i, and z_i are synchronizations of individual ports from P. The terms of the first summand correspond to interactions that can be triggered, i.e. that are obtained by synchronization of interactions from sub-terms of x, at least one of which is a trigger. The terms of the second summand correspond to interactions that can only be obtained by a maximal synchronization of interactions from synchron sub-terms of x. Finally, the terms of the third summand represent interactions that are not possible in x, but contained in some of its sub-terms, and which become possible in the context of a larger connector (e.g. $x\,[1]'$).

The same observations imply that, if two connectors are congruent, after rewriting, their forms (22) coincide. ■

The following example illustrates the application of the rewriting rules used in the proof above.

Example 5.11: Consider the connector $[p\,q]'\,r \,+\, p\,r \,\in\, \mathcal{AC}(\{p,q,r\})$. We then have

$$\begin{aligned}
[p\,q]'\,r + p\,r \;&\rightsquigarrow\; \Big[[p\,q]'\,r\Big]' + 0'\,r + [p\,r] + 0'\,p\,r\\
&\rightsquigarrow\; [p\,q]' + \Big[[p\,q]\,r\Big]' + 0'\,r + [p\,r] + 0'\,[p'\,r']\\
&\rightsquigarrow\; [p\,q]' + [p\,q\,r]' + 0'\,r + [p\,r] + 0'\,[p'\,r] + 0'\,r\\
&\rightsquigarrow\; [p\,q]' + [p\,q\,r]' + [p\,r] + 0'\,p + 0'\,[p\,r] + 0'\,r\\
&\rightsquigarrow\; [p\,q]' + [p\,q\,r]' + [p\,r] + 0'\,p + 0'\,r\,.
\end{aligned}$$

VI. Applications

The Algebra of Connectors formalizes 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.

A. 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 labeling 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 componentization 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 (cf. Ex. 6.1 below).

Step 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 $i = 1, \ldots, k$).

Step 2. In the resulting connector, erase all primes to obtain a term $\widetilde{K} \in \mathcal{AI}(P)$.

Step 3. Consider $\widetilde{K}$ as a star-free regular expression and build the associated (acyclic) automaton with states labeled by sub-interactions of a. As the synchronization operator in $\mathcal{AI}(P)$ is commutative, in the resulting automaton, there is only one state for all sub-interactions that coincide up reordering of ports.

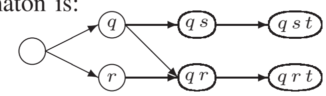
Step 4. The final states of the obtained automaton correspond to maximal enabled interactions within K.

Example 6.1: 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

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

which becomes $\Big[q\,[s+r] + r\,q\Big]\,t$ 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.

Proposition 6.2: For any term $K \in \mathcal{AC}(P)$ and a set $a \subseteq P$ of active ports, the algorithm above computes exactly the set of maximal enabled interactions in K.

Proof: Clearly, substituting 0 for a given port eliminates exactly the interactions, where this port participates. Thus the first step of the algorithm computes the term representing exactly all the enabled interactions in K.

In a monomial (sub-)term, any active port can participate in the interaction. As we are interested in maximal interactions, and the inactive ports have been eliminated at the previous step, the trigger/synchron typing becomes irrelevant.

The union operator in $\mathcal{AI}(P)$ represents a choice between its two alternatives, which corresponds to branching in the constructed automaton. Similarly, synchronization implies that both sub-interactions involved must participate, which corresponds to a transition between two states of this automaton. Consequently, it is also clear that the final states are exactly those labeled by the maximal interactions in their respective branches, and therefore together form the set of maximal enabled interactions in K. ■

B. The multi-shot semantics

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.

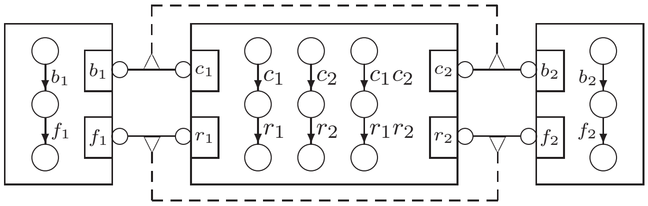


Fig. 8. Modeling a joint call of two functions.

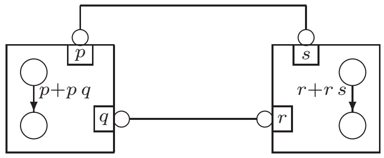


Fig. 9. Causality loop.

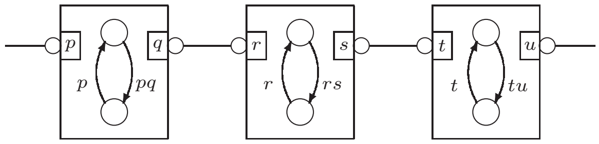


Fig. 10. Multi-shot modulo-8 counter

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 the notion of d-shot semantics discussed below.

To motivate the proposed extension, we model *joint function call* inspired from constructs found in languages such as nesC and Polyphonic C# [15], [16]. A function call for a function F_i, involves two strong synchronizations 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 Fig. 8 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 Fig. 8, 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 6.3: 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 $P = \bigcup_{i=1}^n P_i$.

For an integer parameter $0 < d \leq m$, the *d-shot semantics* of $(\{B_i\}_{i=1}^n, \{K_j\}_{j=1}^m)$ is the system $\gamma_d(B_1, \dots, B_n)$ defined by applying the rule (1) with $\gamma = \gamma_d$, where $\gamma_d = \sum \prod_{i \in I} [K_i]'$, with the summation performed over all subsets $I \subseteq [1, m]$ of cardinality d.

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

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

Note 6.4: In the rest of this subsection, we implicitly associate the $\mathcal{AC}(P)$-connectors with the corresponding sets of interactions, obtained by applying the semantic function $\|\cdot\| : \mathcal{AC}(P) \rightarrow 2^{2^P}$ (cf. Sec. IV-A).

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

Proposition 6.5: Let $S = (\{B_i\}_{i=1}^n, \{K_j\}_{j=1}^m)$ be an interconnected system. For $i \in [1, n]$, we denote $G_i = \sum_{q_i \in Q_i} G_{q_i}$ with $G_{q_i} = \sum_{q_i \xrightarrow{a}} a$ the set of all interactions offered by the component i alone. The set of the possible interactions for d-shot semantics of S is $\prod_{i=1}^n [G_i]' \cap \gamma_d$.

Notice that $\prod_{i=1}^n [G_i]'$ is the set of all the interactions offered by the components, whereas γ_d is the set of the interactions allowed by the d-synchronized connectors. Therefore, the intersection of the two sets characterizes all the possible interactions in the system.

Example 6.6 (Causality loop): Consider the interconnected system shown in Fig. 9, where the transition labeled $p + pq$ (resp. $r{+}rs$) is an abbreviation for two transitions labeled p and pq (resp. r and rs). For $d = 2$ (multi-shot semantics), the only possible interaction is

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

which corresponds to a causality loop, in the synchronous languages terminology [17], [18].

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

$$[p + pq]'\, [r + rs]' \cap \Big(q\, r + p\, s\Big) = 0\,.$$

Example 6.7 (Modulo-8 counter): For multi-shot semantics the system in Fig. 10 is equivalent to the modulo-8 counter given in Ex. 4.11 of Sec. IV-B. The multi-shot model is a more natural representation of this system. Its interactions can be computed by application of Prop. 6.5:

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

As shown in the above examples, it is important to compute efficiently the interactions of a system for d-shot semantics with $d > 1$. To avoid costly enumerative 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 multi-shot semantics, without having to enumerate them explicitly.

For the Modulo-8 counter, the interconnected system in Fig. 10 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 synchronize, as well as s and t. All these relations together, are represented by the graph

$$p \rightarrow q\, r \rightarrow s\, t \rightarrow u\,. \tag{23}$$

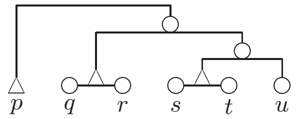


Fig. 11. Synthesized connector for the multi-shot modulo-8 counter

Observe that each path in such a dependency graph represents a causality chain. The graph in (23) represents the connector $p'\,[[q\,r]'[[s\,t]'u]]$, shown in Fig. 11 (cf. also Fig. 6). In general, this technique allows the synthesis of the connectors of a 1-shot model which is equivalent to a given multi-shot model.

C. Incremental decomposition of connectors

In [7], [9], 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.

In Ex. 4.10, for instance, we have presented the following incremental construction for the broadcast connector: $s'\,r_1\,r_2\,r_3 \simeq [s'\,r_1\,r_2]'\,r_3 \simeq \Big[[s'\,r_1]'\,r_2\Big]'\,r_3$.

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 6.8 (Decomposition of connectors): Given a connector $K \in \mathcal{AC}(P)$ and a subset of ports $P_0 \subset P$, construct a connector $\widetilde{K} \equiv \sum_{i=1}^n K_i \cdot \overline{K}_i$, 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 6.9: We require that $K \simeq \widetilde{K}$. Indeed, the 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. Prop. 5.2).

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.

1) Decomposition by rewriting rules: In the context presented above, the connector $\widetilde{K}$ can be constructed by pushing the group of ports $\{p_i\}_{i=1}^k$ through the hierarchical levels iteratively. This procedure can be separated into the following three steps illustrated in Fig. 12.

Step 1. Grouping the ports $\{p_i\}_{i=1}^k$ into a single typed connector (transition from (a) to (b) in Fig. 12). More precisely, we transform a connector of the form

$$[p_1 \ldots p_k p_{k+1} \ldots p_n][y][z] \tag{24}$$

into another one of the form

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

where in both cases (as well as in Fig. 12) we simplify notation by omitting synchron/trigger typing. Observe that this transformation is a congruence, as all the changes are made inside an typed connector (cf. Prop. 5.2).

Step 2. Grouping the sibling connectors into a single typed one (transition from (b) to (c) in Fig. 12). Here, we continue the transformation by replacing the connector of the form (25) by an equivalent one of form

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

Step 3. A rotation pushing the sub-connector containing ports $\{p_i\}_{i=1}^k$ one level up (transition from (c) to (d) in Fig. 12). This is obtained by substituting the connector constructed in the previous step by an equivalent one of the form

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

In the case where the initial connector considered in Step 1 above is itself a 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 (25), 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 finalize this procedure, we state the three following lemmas that describe in a formal way the transformations of the steps enumerated above. The proofs of these lemmas consist in a straightforward verification that the corresponding sets of interactions coincide. Therefore, we skip them here.

Lemma 6.10 (Step 1: Grouping 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 $0 \leq k \leq m$

$$\begin{array}{rl} p'_1 \ldots p'_n\, q_1 \ldots q_m & \simeq \\ \simeq & [p'_1 \ldots p'_l\, q_1 \ldots q_k]' \cdot [p'_{l+1} \ldots p'_n\, q_{k+1} \ldots q_m]' \\ + & [p'_1 \ldots p'_l\, q_1 \ldots q_k]' \cdot [q'_{k+1} \ldots q'_m] \\ + & [q'_1 \ldots q'_k] \cdot [p'_{l+1} \ldots p'_n\, q_{k+1} \ldots q_m]'\,. \end{array} \tag{28}$$

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

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

Lemma 6.11 (Step 2: Grouping 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

$$[x_0] \prod_{i=1}^n [x_i] \simeq [x_0] \left[\prod_{i=1}^n [x_i]\right], \tag{29}$$

$$[x_0]' \prod_{i=1}^n [x_i] \simeq [x_0]' \left[\prod_{i=1}^n [x_i]'\right], \tag{30}$$

$$[x_0] \prod_{i=1}^n [x_i]^{\alpha_i} \simeq [x_0] \left[\prod_{i=1}^n [x_i]^{\alpha_i}\right]', \tag{31}$$

$$[x_0]' \prod_{i=1}^n [x_i]^{\alpha_i} \simeq [x_0]' \left(\left[\prod_{i=1}^n [x_i]^{\alpha_i}\right]' + \left[\prod_{i \in S} [x_i]'\right]\right), \tag{32}$$

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

Lemma 6.12 (Step 3: Rotation): For arbitrary connectors $x, y, z \in \mathcal{AC}(P)$ and types $\alpha, \beta, \gamma, \delta \in \{0, 1\}$, holds the

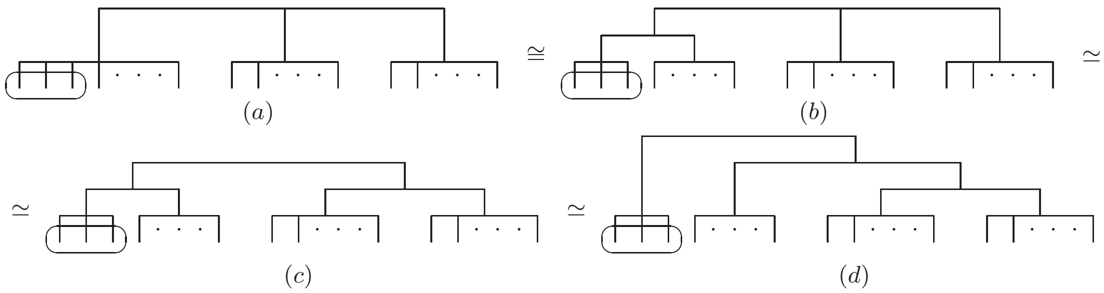


Fig. 12. Hierarchical connector transformation

equivalence

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

with

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

2) Decomposition by derivation:

Theorem 6.13: 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

$$K \;\simeq\; p \cdot \left[\frac{dK}{dp}\right] \;+\; K(0/p)\,, \tag{33}$$

where $K(0/p)$ denotes the connector obtained by substituting all occurrences of p in K by 0 (to simplify the notation, below we write $K(0)$ instead of $K(0/p)$).

Proof: It is, indeed, sufficient to consider the flattened term $|K| \in \mathcal{AI}(P)$ of the connector K. This term $|K|$ 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 6.14: We call the connector dK/dp in (33) the *derivative of K with respect to p.*

Observe that Th 6.13 ensures the uniqueness of such decomposition in $\mathcal{AI}(P)$.

Proposition 6.15: For $K, K_1, K_2 \in \mathcal{AI}(P)$, $\alpha, \beta \in \{0, 1\}$, and $p \in P$, derivative has the following properties.

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

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 Prop. 5.2,

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

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

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

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

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

thus proving the required property for $\alpha = \beta = 0$. For the case $\alpha = 1$ and $\beta = 0$, the proposition follows from the equivalence $[K_1]'[K_2] \simeq [K_1] + [K_1][K_2]$, and, similarly, for $\alpha = \beta = 1$, from the equivalence $[K_1]'[K_2]' \simeq [K_1] + [K_2] + [K_1][K_2]$. ■

The last property above can be generalized to a fusion of any number of typed connectors.

Proposition 6.16: Let $\{K_i\}_{i=1}^n$ and $\{L_j\}_{j=1}^m$ be two families of connectors from $\mathcal{AC}(P)$.

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

Proof: The proof is by induction on $n + m$, with property 4 in Prop. 6.15 constituting its base. The complete proof of the induction step is straightforward and can be found in [19]. ■

Example 6.17: Consider the connector $K = p' \cdot [qr]' \cdot [rs]$. This connector models coordination between two components with ports q and s, which require a certain resource to operate. This resource is accessed by interacting with another component with a port r. Once this resource available, both of these components are connected with a third one, which communicates through port p.

Suppose now that we want to restructure K in order to separate the component providing the considered resource from the ones that utilize it. To do so, we differentiate K by r, applying Prop. 6.16

$$\begin{aligned} \frac{dK}{dr} \;&\simeq\; \left[\tfrac{dp}{dr}\right]'[q][s] + \left[\tfrac{d(qr)}{dr}\right]'[p][s] + \left[\left[\tfrac{d(sr)}{dr}\right]'\right][p'q'] \\ &\simeq\; q'ps + s[p'q']\,. \end{aligned} \tag{34}$$

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

$$K(0) \;\equiv\; p' \cdot [q \cdot 0]' \cdot [0 \cdot s] \;\cong\; p' \cdot [0]' \cdot [0] \;\cong\; p' \;\simeq\; p\,. \tag{35}$$

Substituting (34) and (35) into (33), we obtain the following decomposition $K \simeq r \cdot \left[q'ps + s[p'q']\right] + p$, which can now be easily verified.

VII. Conclusion

$\mathcal{AC}(P)$ provides an abstract and powerful framework for modeling control flow between components. It allows the structured combination of two basic synchronization 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 [20], 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 formalizing the multi-shot semantics for component models. Axiomatization 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.

$\mathcal{AC}(P)$ is a simple and powerful algebraic framework for modeling interaction. It can be a semantic model for formalisms used for architecture description languages and provides a basis for comparing coordination mechanisms.

References

[1] M. Bernardo, P. Ciancarini, and L. Donatiello, "On the formalization of architectural types with process algebras." in *SIGSOFT FSE*, 2000, pp. 140–148.

[2] B. Spitznagel and D. Garlan, "A compositional formalization of connector wrappers." in *ICSE*. IEEE Computer Society, 2003, pp. 374–384.

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

[4] R. Bruni, I. Lanese, and U. Montanari, "A basic algebra of stateless connectors," *Theor. Comput. Sci.*, vol. 366, no. 1, pp. 98–120, 2006.

[5] F. Arbab, "Reo: a channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2004.

[6] 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)*, Sept. 2006, pp. 3–12.

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

[8] G. Gößler and J. Sifakis, "Component-based construction of deadlock-free systems: Extended abstract." in *FSTTCS*, ser. Lecture Notes in Computer Science, P. K. Pandya and J. Radhakrishnan, Eds., vol. 2914. Mumbai, India: Springer, Dec. 2003, pp. 420–433.

[9] ——, "Composition for component-based modeling." *Science of Computer Programming*, vol. 55, no. 1–3, pp. 161–183, 2005.

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

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

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

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

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

[15] "Cω," http://research.microsoft.com/comega/.

[16] "nesC: A programming language for deeply networked systems," http://nescc.sourceforge.net/.

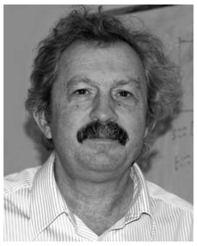
[17] G. Berry and G. Gonthier, "The ESTEREL synchronous programming language: Design, semantics implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87–152, Nov. 1992.

[18] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous dataflow programming language LUSTRE," in *Proceedings of the IEEE*, vol. 79, Sept. 1991, pp. 1305–1320.

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

[20] 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," VERIMAG, Tech. Rep. TR-2007-5, 2007, http://www-verimag.imag.fr/index.php?page=techrep-list.

Simon Bliudze has obtained an MSc in Mathematics from the St. Petersburg State University (Russia) in 1998, an MSc in Computer Science from Université Paris 6 (France) in 2001, and a PhD in Computer Science from École Polytechnique (Paris, France) in 2006. He is presently holding a post-doctoral position at VERIMAG (Grenoble, France), where he is working in the domain of formal methods for component-based construction of real-time systems.



Joseph Sifakis is CNRS researcher and the Founder of Verimag laboratory, in Grenoble, France. He studied Electrical Engineering at the Technical University of Athens and Computer Science at the University of Grenoble.

Joseph Sifakis is recognized for his pioneering work on both theoretical and practical aspects of concurrent systems specification and verification. He contributed to emergence of the area of model-checking. His current research activities include component-based construction of real-time systems with focus on correct-by-construction techniques.

Joseph Sifakis is the Scientific Coordinator of the European Network of Excellence ARTIST2 on Embedded Systems Design. He is a member of the editorial board of several journals, a co-founder of the International Conference on Computer Aided Verification (CAV) and a member of the Steering Committee Board of EMSOFT.