Form Methods Syst Des (2010) 36: 167–194
DOI 10.1007/s10703-010-0091-z

Causal semantics for the algebra of connectors

Simon Bliudze · Joseph Sifakis

Published online: 15 April 2010

Abstract The Algebra of Connectors $\mathcal{AC}(P)$ is used to model structured interactions in the BIP component framework. Its terms are *connectors*, relations describing synchronization constraints between the ports of component-based systems. Connectors are structured combinations of two basic synchronization protocols between ports: *rendezvous* and *broadcast*.

In a previous paper, we have studied interaction semantics for $\mathcal{AC}(P)$ which defines the meaning of connectors as sets of interactions. This semantics reduces broadcasts into the set of their possible interactions and thus blurs the distinction between rendezvous and broadcast. It leads to exponentially complex models that cannot be a basis for efficient implementation. Furthermore, the induced semantic equivalence is not a congruence.

For a subset of $\mathcal{AC}(P)$, we propose a new *causal* semantics that does not reduce broadcast into a set of rendezvous and explicitly models the causal dependency relation between ports. The Algebra of Causal Interaction Trees $\mathcal{T}(P)$ formalizes this subset. It is the set of the terms generated from interactions on the set of ports P, by using two operators: a *causality* operator and a *parallel composition* operator. Terms are sets of trees where the successor relation represents causal dependency between interactions: an interaction can participate in a global interaction only if its father participates too. We show that causal semantics is consistent with interaction semantics; the semantic equivalence on $\mathcal{T}(P)$ is a congruence. Furthermore, it defines an isomorphism between $\mathcal{T}(P)$ and a subset of $\mathcal{AC}(P)$.

Finally, we define for causal interaction trees a boolean representation in terms of *causal rules*. This representation is used for their manipulation and simplification as well as for synthesizing connectors.

Keywords BIP · Component · Connectors · Connector synthesis · Interaction · Causal semantics · Causal interaction trees

S. Bliudze (✉)
CEA, LIST, 91191 Gif-sur-Yvette, France
e-mail: Simon.Bliudze@cea.fr

J. Sifakis
VERIMAG, Centre Équation, 2 av de Vignate, 38610 Gières, France
e-mail: Joseph.Sifakis@imag.fr

1 Introduction

Component-based design is based 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 information flow 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 coordination mechanisms such as semaphores, monitors, message passing, remote call, protocols etc. A unified paradigm should allow a comparison of otherwise unrelated architectural solutions and could be a basis for evaluating them and deriving implementations in terms of specific coordination mechanisms. Furthermore, it should be expressive enough to directly encompass various coordination as discussed in [11].

Taking the *unbuffered synchronous communication* approach (see, for example, [18] or [21, Chap. 1]) allows one to separate the model of a system in three layers:

1. **Behavior**, representing the sequential computation of individual components,
2. **Coordination**, defining the synchronization between these components,
3. **Data Transfer**, specifying how the data is exchanged among the components upon synchronization.

Although each layer depends on the layers underneath, it can only restrict the overall behavior of the system, thus inducing an abstraction/refinement relation. The *separation of concerns* principle means that the three layers are modeled explicitly and can be analyzed separately from one another. We are particularly interested in studying the Coordination layer responsible for modeling scheduling constraints such as synchronization and mutual exclusion, while abstracting away from the Data Transfer layer.

A number of paradigms for unifying interaction in heterogeneous systems have been studied in [3, 4, 14]. In these works, unification is achieved by reduction to a common low-level semantic model. Coordination mechanisms and their properties are not studied independently of behavior.

Similarly, although numerous compositional [1, 15, 25, 27] and algebraic [7, 12, 13, 18, 23, 28] frameworks exist for system modeling and design, they all fail to respect the separation of concerns by mixing at least two of the three layers mentioned above. In particular, the term "connector" is widely used in the component frameworks literature with a number of different interpretations, often combining aspects of the three layers. This failure to respect the separation of concerns results in formalisms that are too complex, restricting the potential analysis and applications.

Connectors are often specified in an operational setting, e.g., a process algebra. In [7] and [19], process algebras are used to define *architectural types* as a set of component/connector instances related by a set of attachments among their interactions. In [27], 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 [15]. *Reo* [1] is a channel-based exogenous coordination model. 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 it is limited to the underlying channels. The nodes of connectors realize coordination between these channels. All these 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.

Our approach is closest to that of [12], where an algebra of connectors is developed that allows, in particular, an algebraic translation of the categorical approach used in CommUnity. This algebra allows one to construct *stateless* connectors from a number of basic ones.

In this paper, we propose a new *causal semantics* for the *Algebra of Connectors* $\mathcal{AC}(P)$, studied in [9]. Terms of $\mathcal{AC}(P)$ are *connectors*—the basic concept for modeling coordination between components. Connectors are relations between ports with synchronization types, allowing description of complex coordination patterns with an extremely small set of basic primitives.

In [9], we have studied an *interaction semantics* for $\mathcal{AC}(P)$, used to model interactions in the BIP (Behavior, Interaction, Priority) component framework [5, 26]. In [11], we propose a notion of expressiveness for component-based systems and show that any type of coordination can be expressed in BIP. In particular, most of the frameworks listed above can be modeled in BIP.

The interaction semantics defines the meaning of a connector as the set of the interactions it allows. $\mathcal{AC}(P)$ is defined from a set P of ports. Its terms represent sets of interactions which are non-empty sets of ports. Within a connector, an interaction can take place in two situations: either an interaction is fired when all involved ports are ready to participate (strong synchronization), or some subset of ports triggers the interaction without waiting for other ports. Thus, connectors are generated from the ports of P by using a binary *fusion* operator and a unary *typing* operator. Typing associates with terms (ports or connectors) synchronization types: *trigger* or *synchron*.

A *Simple* (or *flat*) connector is an expression of the form $p_1' \dots p_k' p_{k+1} \dots p_n$, where primed ports p_i' are triggers, and unprimed ports p_j are synchrons. For a flat connector involving the set of ports $\{p_1, \dots, p_n\}$, interaction semantics defines the set of its interactions by the following rule: *an interaction is any non-empty subset of* $\{p_1, \dots, p_n\}$ *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* $p_1 \dots p_n$. As usual, we abbreviate $\{p_1, \dots, p_n\}$ to $p_1 \dots p_n$.

In particular, two basic synchronization protocols can be modeled naturally: (1) *rendezvous*, when all the related ports are synchrons, and the only possible interaction is the maximal one containing all ports of the connector; (2) *broadcast*, when the transmitting port is a trigger, receiving ports are synchrons, and possible interactions are those containing the trigger. Connectors, representing these two protocols for a sender s and receivers r_1, r_2, r_3, are shown in Fig. 1(a, b). Triangles represent triggers, whereas bullets represent synchrons.

Hierarchical connectors are expressions composed of typed ports and/or typed sub-connectors. Figure 1(c) shows a connector realizing an atomic broadcast from a port s to ports r_1, r_2, r_3. The sender port s is a trigger, and the three receiver ports are strongly synchronized in a sub-connector itself typed as a synchron. The corresponding $\mathcal{AC}(P)$ term is $s'[r_1r_2r_3]$, and the possible interactions are: s and $sr_1r_2r_3$. Here the term in brackets $[\cdot]$ is a sub-connector typed as a synchron. Primed brackets $[\cdot]'$ denote a sub-connector typed as a trigger. The connector shown in Fig. 1(d) is a causal chain of interactions initiated by the port s. The corresponding $\mathcal{AC}(P)$ term is $s'[r_1'[r_2'r_3]]$, and the possible interactions are s, sr_1, sr_1r_2, $sr_1r_2r_3$: a trigger s alone or combined with some interaction from the sub-connector $r_1'[r_2'r_3]$, itself a shorter causal chain.

As shown in the above examples, interaction semantics reduces a connector into the set of its interactions. This leads to exponentially complex representations. Furthermore, it blurs

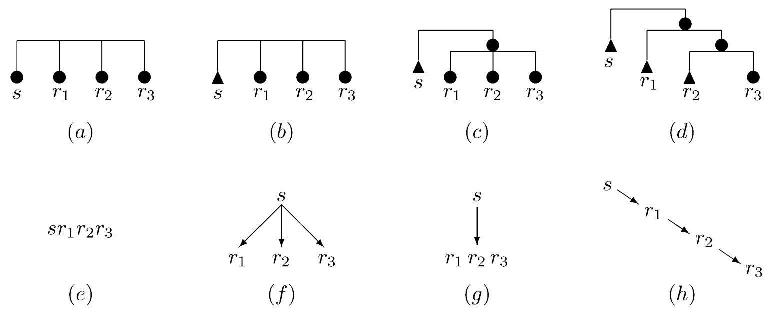


Fig. 1 Connectors and causal interaction trees representing a rendezvous (**a**, **e**), a broadcast (**b**, **f**), an atomic broadcast (**c**, **g**), and a causal chain (**d**, **h**)

the distinction between rendezvous and broadcast as each interaction of a broadcast can be realized by a rendezvous. In [9], we have shown that this also has deep consequences on the induced semantic equivalence: broadcasts may be equivalent to sets of rendezvous but they are not congruent (with respect to the $\mathcal{AC}(P)$ operators). In [10], we have further studied the maximal congruence relation on $\mathcal{AC}(P)$ and presented its complete axiomatization.

The deficiencies of interaction semantics have motivated the investigation of a new *causal* semantics for a subset of connectors of $\mathcal{AC}(P)$, formalized as the Algebra of Causal Interaction Trees $\mathcal{T}(P)$. This semantics distinguishes broadcast and rendezvous by explicitly modeling the causal dependency relation between triggers and synchrons in broadcasts. The terms of $\mathcal{T}(P)$ represent sets of interactions, generated from atomic interactions on the set of ports P, by using two operators:

- A *causality* operator $\rightarrow$ which defines the causal relationship. The term $a_1 \rightarrow a_2 \rightarrow a_3$ is a causal chain meaning that interaction a_1 may trigger interaction a_2 which may trigger interaction a_3. The possible interactions for this chain are a_1, a_1a_2, $a_1a_2a_3$.
- An associative and commutative *parallel composition* operator $\oplus$. A causal interaction tree can be considered as the parallel composition of all its causal chains. For instance, the term $a_1 \rightarrow (a_2 \oplus a_3)$ is equivalent to $(a_1 \rightarrow a_2) \oplus (a_1 \rightarrow a_3)$ (both describing the set of four interactions: a_1, a_1a_2, a_1a_3, and $a_1a_2a_3$).

Terms of $\mathcal{T}(P)$ are naturally represented as sets of trees where '$\rightarrow$' corresponds to the parent/son relation. Figure 1(e–h) shows the causal interaction trees for the four connectors discussed above.

Causality has already been studied in several contexts such as the *Causal Trees* by Darondeau and Degano [13], the step semantics of Statecharts [20], and synchronous systems such as Lustre [17] or Signal [6] in general. The main accent of these studies is different from that of our paper: causal relations are considered between subsequent actions of a process rather than between the participation of different ports in the same interaction. Although structural similarities can be found between the concepts considered so far and causal interaction trees, their study is beyond the scope of the present paper.

The main results of the paper are the following:

- We define causal semantics for $\mathcal{AC}(P)$ in terms of causal trees, as a function $\mathcal{AC}(P) \rightarrow \mathcal{T}(P)$. Causal semantics is sound with respect to interaction semantics. An important result is that the algebra of causal interaction trees $\mathcal{T}(P)$ is isomorphic to classes of *causal connectors* $\mathcal{AC}_c(P)$ and *causal sets of interactions* $\mathcal{AI}_c(P)$. A causal set of interactions is closed under synchronization. Furthermore, we show that interaction equivalence on $\mathcal{T}(P)$ is a congruence.

- We define for causal interaction trees, $\mathcal{T}(P)$ a boolean representation by using *causal rules*. Terms are represented by boolean expressions on P. The boolean valuation of port p is interpreted as the presence/absence of a port in an interaction. This representation is used for their symbolic manipulation and simplification as well as for performing boolean operations on connectors. It is applied for the efficient implementation of BIP, in particular, to compute the possible interactions for a given state.
- We also present a method for synthesizing $\mathcal{AC}(P)$ connectors for a given set of atomic components characterized by their behavior. Connectors are synthesized from a set of boolean constraints on the set P of ports of the components, expressing their possible interactions.

  The examples of Sect. 6 show how certain safety properties, such as mutual exclusion, can be expressed by this kind of boolean constraints. Connectors are then synthesized that enforce these properties.

The paper is structured as follows. Section 2 provides a succinct presentation of the basic semantic model for BIP and in particular, its composition parametrized by interactions. Section 3 presents the Algebra of Connectors and its global interaction semantics. Section 4 introduces the Algebra of Causal Interaction Trees and its properties. It then develops a causal semantics for $\mathcal{AC}(P)$ and shows a way to efficiently compute a boolean representation for connectors. Section 5 studies the opposite transformation, i.e., from boolean functions to causal interaction trees and $\mathcal{AC}(P)$ connectors. Section 6 provides examples of connector synthesis based on this transformation. Finally, Sect. 7 concludes the paper.

2 The BIP component framework

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 of 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 (see [5, 8]).

The execution of a BIP program is driven by a dedicated engine, which has access to the set of the connectors and the priority model of the program. In a given global state, each atomic component waits for an interaction through a set of active ports (i.e., ports labeling 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 transfer and notifies the components involved in the chosen interaction. Below we provide a succinct formalization of this system-level operational semantics, focusing on 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. To simplify notation we represent an interaction $\{p_1, p_2, \dots, p_n\}$ as $p_1 p_2 \dots p_n$.

Definition 2.2 A transition system is a triple $B = (Q, P, \rightarrow)$, where Q is a set of *states*, P is a set of *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 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 $q \xrightarrow{a} q'$.

In BIP, a system can be obtained as the composition of n components, each modeled 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 $\{B_i\}_{i=1}^n$, parametrized by a set of interactions $\gamma \subset 2^P$ is the transition system $B = (Q, P, \rightarrow_\gamma)$, where $Q = \prod_{i=1}^n Q_i$ and $\rightarrow_\gamma$ is the least set of transitions satisfying the rule

$$\frac{a \in \gamma \qquad \forall i \in [1, n],\ \left(q_i \xrightarrow{a \cap P_i}_i q_i' \vee (a \cap P_i = \emptyset \wedge q_i = q_i')\right)}{(q_1, \dots, q_n) \xrightarrow{a}_\gamma (q_1', \dots, q_n')}. \tag{1}$$

We write $B = \gamma(B_1, \dots, B_n)$.

Notice that an interaction $a \in \gamma$ is enabled in $\gamma(B_1, \dots, B_n)$, 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 behavior. This can be restricted by means of priorities [8, 9]. Throughout this paper, whenever two interactions, a and a', such that $a \subset a'$, are possible, we always choose a'.

Example 2.3 (Sender/Receivers) Figure 2 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 with a set of interactions γ and priorities π. The sender has a port s for sending messages, and each receiver has a port r_i $(i = 1, 2, 3)$ for receiving them. Table 1 specifies γ for four different interaction schemes.

Rendezvous means strong synchronization 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 labeled respectively by s, r_1, r_2, r_3.

Broadcast means weak synchronization, that is a synchronization involving S and any (possibly empty) subset of R_i. This is specified by the set of all interactions containing s. These

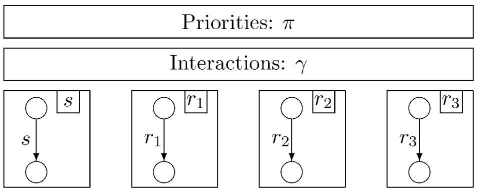


Fig. 2 A system with four atomic components

Table 1

Interaction scheme	Interactions
Rendezvous	$sr_1r_2r_3$
Broadcast	$s, sr_1, sr_2, sr_3, sr_1r_2, sr_1r_3, sr_2r_3, sr_1r_2r_3$
Atomic broadcast	$s, sr_1r_2r_3$
Causal chain	$s, sr_1, sr_1r_2, sr_1r_2r_3$

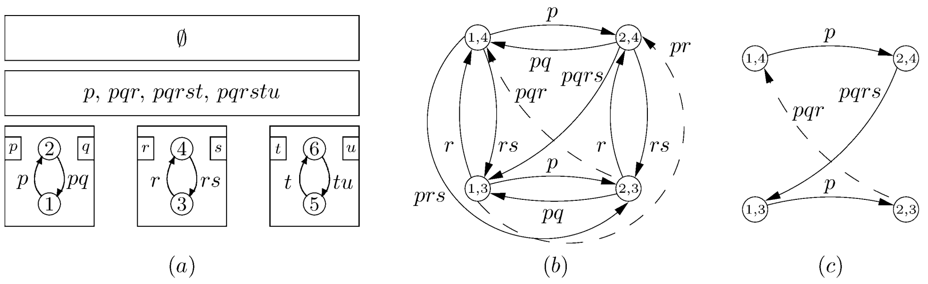


Fig. 3 A BIP model for the Modulo-8 counter (**a**), parallel composition of two Modulo-2 counter LTS (**b**), and the Modulo-4 counter LTS (**c**)

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 $sr_1r_2r_3$, corresponding to strong synchronization.

Causal chain means that for a message to be received by R_i it has to be received by all R_j, for $j < i$. This interaction scheme is common in reactive systems.

Example 2.4 (Modulo-8 counter) Figure 3(a) shows a BIP model for the Modulo-8 counter presented in [22]. It is obtained by composing three Modulo-2 counter components. Ports p, r, and t correspond to inputs, whereas q, s, and u correspond to outputs. One can easily verify that the interactions pqr, $pqrst$, and $pqrstu$ happen, respectively, on every second, fourth, and eighth occurrence of an interaction through the port p. This can be done by computing the semantics of this model according to (1). For simplicity, we only consider here the first two Modulo-2 counter components, but the observations are extended to the complete model in a straightforward manner.

Figure 3(b) shows an LTS obtained by parallel composition of the first two Modulo-2 counter components in Fig. 3(a), whereas its restriction under the interaction model $\{p, pqr, pqrs\}$ is shown in Fig. 3(c).

Notice that the composition operator can express usual parallel composition operators [9], such as the ones used in CSP [18] and CCS [23]. By enforcing maximal progress, priorities allow to express broadcast.

3 The algebra of connectors

In this section, we introduce the *algebra of connectors* $\mathcal{AC}(P)$, which formalizes the concept of connector, supported by the BIP language [5].

Table 2 $\mathcal{AI}(P)$, $\mathcal{AC}(P)$, and $\mathcal{T}(P)$ representations of four basic interaction schemes

	$\mathcal{AI}(P)$	$\mathcal{AC}(P)$	$\mathcal{T}(P)$
Rendezvous	$sr_1r_2r_3$	$sr_1r_2r_3$	$sr_1r_2r_3$
Broadcast	$s(1+r_1)(1+r_2)(1+r_3)$	$s'r_1r_2r_3$	$s \to (r_1 \oplus r_2 \oplus r_3)$
Atomic broadcast	$s(1+r_1r_2r_3)$	$s'[r_1r_2r_3]$	$s \to r_1r_2r_3$
Causal chain	$s(1+r_1(1+r_2(1+r_3)))$	$s'[r_1'[r_2'r_3]]$	$s \to r_1 \to r_2 \to r_3$

3.1 The algebra of interactions

Let P be a set of ports, such that $0, 1 \notin P$. Recall (Definition 2.1) that an *interaction* is a non-empty subset $a \subseteq P$. In the following, to simplify the presentation, we lift the non-emptiness restriction.

In [9], we have introduced the *algebra of interactions* $\mathcal{AI}(P)$, used to define the interaction semantics of $\mathcal{AC}(P)$. The elements of this algebra can be bijectively mapped to sets of interactions. The additive *union* operator in $\mathcal{AI}(P)$ corresponds to the set union in 2^{2^P}, whereas the multiplicative *synchronization* operator corresponds to the element-wise union: the synchronization of two sets of interactions is the set consisting of interactions obtained by taking the union of two interactions, one from each of the operands. For instance, the term $p+q$ in $\mathcal{AI}(\{p,q,r\})$, represents the set of interactions $\{p,q\}$, whereas $(p+q)r$ represents the set $\{pr, qr\}$. The additive and multiplicative identity elements 0 and 1 correspond respectively to the sets $\emptyset$ and $\{\emptyset\}$.

$\mathcal{AI}(P)$ provides a clear and convenient notation for manipulation of sets of interactions.

Example 3.1 (Sender/Receiver continued) The second column of Table 2 shows the representation in $\mathcal{AI}(P)$ for the four interaction schemes of Example 2.3.

Any interaction $a \in 2^P$ defines a boolean valuation on P with, for each $p \in P$, $p = \mathit{true}$ iff $p \in a$. In other words, a valuation on P determines for each port whether it participates in the interaction or no. Notice that the constant valuation *false* is associated to the interaction 1, which corresponds to the empty set of ports $\emptyset \in 2^P$.

Below, we denote by $\mathbb{B}[P]$ the free boolean algebra generated by the set P. Each element of $\mathbb{B}[P]$ is a formula defining a set of interactions in a system with the set of ports P.

Definition 3.2 An interaction $a \in 2^P$ *satisfies* a formula $R \in \mathbb{B}[P]$ (denoted $a \models R$) iff the corresponding boolean valuation satisfies R. A term $x \in \mathcal{AI}(P)$ satisfies R (denoted $x \models R$) iff all interactions belonging to x satisfy R, that is

$$x \models R \quad \overset{def}{\Longleftrightarrow} \quad \forall a \in x, \quad a \models R.$$

3.2 Syntax and interaction semantics for $\mathcal{AC}(P)$

In this section, we introduce the Algebra of Connectors, $\mathcal{AC}(P)$. We follow the mathematical style of defining algebraic structures. First, we provide the grammar generating a set of connector terms from the set P of ports of the system by using two operations: *fusion* and *typing*. Then, we introduce the axioms satisfied by these operators, which results in the algebraic structure that is the quotient of the set of connector terms by the equivalence relation induced by the axioms. Finally, we give the interaction semantics of connectors by defining $|x| \in \mathcal{AI}(P)$ for each term x generated by the grammar, and show that this definition is unambiguous, i.e., the function $|\cdot|$ respects the axioms.

For the sake of simplicity, we consider the subset of terms of $\mathcal{AC}(P)$ that do not involve union, that is the subset of *monomial connectors* (cf. [9]).

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

$$\begin{aligned} s &::= [0] \mid [1] \mid [p] \mid [x] \quad \text{(synchrons)} \\ t &::= [0]' \mid [1]' \mid [p]' \mid [x]' \ \text{(triggers)} \\ x &::= s \mid t \mid x \cdot x, \end{aligned} \tag{2}$$

for $p \in P$, and where '$\cdot$' is a binary operator called *fusion*, and brackets '$[\cdot]$' and '$[\cdot]'$' are unary *typing* operators.

Fusion is a generalization of synchronization in $\mathcal{AI}(P)$. Typing is used to form connectors: $[\cdot]'$ defines *triggers* (which can initiate an interaction), and $[\cdot]$ defines *synchrons* (which need synchronization with other ports).

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 3.3 In a system with a set of ports P, *connectors* are elements of $\mathcal{AC}(P)$.

The algebraic structure of $\mathcal{AC}(P)$ inherits most of the axioms of $\mathcal{AI}(P)$.

Axioms

1. Fusion is associative, commutative, idempotent, and has an identity element $[1]$.
2. Typing satisfies the following axioms, for $x \in \mathcal{AC}(P)$:
   a) $[0]' = [0]$,
   b) $[[x]]' = [[x]']' = [x]'$ and $[[x]] = [[x]'] = [x]$.

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

$$|p| = p, \tag{3}$$

$$\left|\prod_{i=1}^{n}[x_i]\right| = \prod_{i=1}^{n}|x_i|, \tag{4}$$

$$\left|\prod_{i=1}^{n}[x_i]' \cdot \prod_{j=1}^{m}[y_j]\right| = \sum_{i=1}^{n}|x_i| \prod_{k\neq i}\big(1 + |x_k|\big) \prod_{j=1}^{m}\big(1 + |y_j|\big), \tag{5}$$

for $p \in P \cup \{0, 1\}$ and $x, x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathcal{AC}(P)$. In (5), $\sum$ and $\prod$ are, respectively, the union and synchronization operators of $\mathcal{AI}(P)$.

Example 3.4 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 $s_1' s_2' r_1 [r_2 r_3]$ is

$$\begin{aligned}
&|s_1's_2'r_1[r_2r_3]| \\
&\overset{(5)}{=} |s_1|(1+|s_2|)(1+|r_1|)(1+|r_2r_3|) + |s_2|(1+|s_1|)(1+|r_1|)(1+|r_2r_3|) \\
&\overset{(4)}{=} \big(|s_1|(1+|s_2|) + |s_2|(1+|s_1|)\big)(1+|r_1|)(1+|r_2||r_3|) \\
&\overset{(3)}{=} \big(s_1(1+s_2) + s_2(1+s_1)\big)(1+r_1)(1+r_2r_3) \\
&= (s_1+s_2+s_1s_2)(1+r_1+r_2r_3+r_1r_2r_3),
\end{aligned}$$

which corresponds to the set of the interactions containing at least one of s_1 and s_2, and possibly r_1 and a synchronization of both r_2 and r_3.

Proposition 3.5 ([9]) *The axiomatization of $\mathcal{AC}(P)$ is sound, that is, for $x, y \in \mathcal{AC}(P)$, the equality $x = y$ implies $|x| = |y|$.*

Example 3.6 (Sender/Receiver continued) The third column of Table 2 shows the connectors for the four interaction schemes of Example 2.3.

Notice that $\mathcal{AC}(P)$ allows compact representation of interactions and, moreover, explicitly captures the difference between broadcast and rendezvous. The typing operator induces a hierarchical structure.

Example 3.7 (Modulo-8 counter continued) In the model shown in Fig. 4, the causal chain pattern is applied to connectors p, qr, st, and u. Interactions are modeled by a single structured connector $p'[[qr]'[[st]'u]]$:

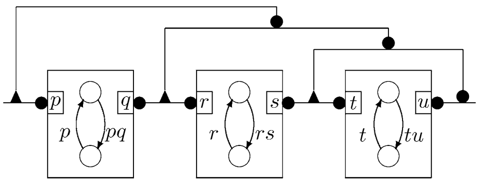
$$\big|p'\big[[qr]'\big[[st]'u\big]\big]\big| = p + pqr + pqrst + pqrstu.$$

These are exactly the interactions of the Modulo-8 counter of Fig. 3.

Definition 3.8 Two connectors $x, y \in \mathcal{AC}(P)$ are *equivalent* (denoted $x \simeq y$), iff they have the same sets of interactions, i.e., $x \simeq y$ if and only if $|x| = |y|$.

Note 3.9 Notice that, in general, two equivalent terms are not congruent. For example, $p' \simeq p$, but $p'q \simeq p + pq \not\simeq pq$, for $p, q \in P$. Similarly, the following terms are equivalent, but not congruent: pqr, $p[qr]$, and $[pq]r$, as different sets of interactions are obtained, when these terms are fused with a trigger. For instance, $s'[pq]r \simeq s + spq + sr + spqr$, whereas $s'p[qr] \simeq s + sp + sqr + spqr$. This is due to the fact that the semantics of the fusion operator is not defined in the same way according to whether the operands have triggers or not (cf. (4), (5)).

Fig. 4 Modulo-8 counter



Definition 3.10 We denote by '$\cong$' the largest congruence relation contained in $\simeq$, that is the largest relation satisfying

$$x \cong y \quad \Longrightarrow \quad \forall E \in \mathcal{AC}(P \cup \{z\}),\ E(x/z) \simeq E(y/z), \tag{6}$$

where $x, y \in \mathcal{AC}(P)$, $z \notin P$, $E(x/z)$, and (resp. $E(y/z)$) denotes the expression obtained from E by replacing all occurrences of z by x (resp. y).

Theorem 3.11 ([9]) *For $x, y \in \mathcal{AC}(P)$, we have $x \cong y$ iff the following three conditions hold simultaneously*

1. $x \simeq y$,
2. $x \cdot 1' \simeq y \cdot 1'$,
3. $\#x > 0 \Leftrightarrow \#y > 0$,

where we denote by $\#x$ the number of non-zero triggers in x, that is, for $x = \prod_{i=1}^{k}[x_i]' \times \prod_{i=k+1}^{n}[x_i]$, $\#x \overset{def}{=} \#\{i \in [1,k] | x_i \not\simeq 0\}$.

The last two conditions in this theorem ensure that the two connectors "behave" similarly when fused with other connectors. For instance, to illustrate the application of the third condition, consider $x[p][q]$ and $y[p][q]$. If $\#x = 0 \neq \#y$, all interactions of $x[p][q]$ contain both ports p and q, whereas interactions in $y[p][q]$ can contain any combination of these two ports (p and/or q, or neither).

Corollary 3.12 *For $x, y \in \mathcal{AC}(P)$, it holds $[x]'[y]' \cong [[x]'[y]']'$.*

Note 3.13 A complete axiomatization of the congruence relation $\cong$ is given in [10].

4 Causal semantics for connectors

In this section, we propose a new *causal* semantics for $\mathcal{AC}(P)$ connectors, which is consistent with that given in terms of interactions. This semantics allows efficient computation of a boolean representation for connectors.

Indeed, in [9], we have shown that efficient computation of boolean operations (e.g., intersection, complementation) is crucial for efficient implementation of some classes of systems, e.g., synchronous systems. In this section, we present a method for computing boolean representations for $\mathcal{AC}(P)$ connectors through a translation into the algebra of causal interaction trees $\mathcal{T}(P)$. The terms of the latter have a natural boolean representation as sets of causal rules (implications). In particular, this boolean representation allows the use of existing techniques such as BDDs avoiding the complex enumeration of the interactions of connectors.

The key idea for causal semantics is to render explicit the causal relations between different parts of the connector. In a fusion of typed connectors, triggers are mutually independent, and can be considered *parallel* to each other. Synchrons participate in an interaction only if it is initiated by a trigger. This introduces a causal relation: the trigger is a *cause* that can provoke an *effect*, which is the participation of a synchron in an interaction.

There are essentially three possibilities for connectors involving ports p and q:

1. A strong synchronization pq.
2. One trigger $p'q$, i.e., p is the cause of an interaction and q a potential effect, which we will denote in the following by $p \to q$.
3. Two triggers $p'q'$, i.e., p and q are independent (parallel), which we will denote in the following by $p \oplus q$.

This can be further extended to chains of causal relations between interactions. For example, $(p \oplus q) \to (rs \to t)$ corresponds to the connector $p'q'[[rs]'t]$. It means that any combination of p and q (i.e., p, q, or pq) can trigger an interaction in which both r and s may participate (thus, the corresponding interactions are p, q, pq, prs, qrs, and $pqrs$). Moreover, if r and s participate then t may do so, which adds the interactions $prst$, $qrst$, and $pqrst$.

Causal interaction trees constructed with these two operators provide a compact and clear representation for connectors that shows explicitly the atomic interactions (p, q, rs, and t in the above example) and the dependencies between them. They also allow to exhibit the boolean *causal rules*, which define the necessary conditions for a given port to participate in an interaction. Intuitively, this corresponds to expressing arrows in the causal interaction trees by implications.

A causal rule is a boolean formula over P, which has the form $p \Rightarrow \bigvee_{i=1}^{n} a_i$, where p is a port and a_i are interactions that can provoke p. Thus, in the above example, the causal rule for the port r is $r \Rightarrow ps \vee qs$, which means that for the port r to participate in an interaction of this connector, it is necessary that this interaction contain either ps or qs.

A set of causal rules uniquely describes the set of interactions that satisfy it (cf. Sect. 3.1), which provides a simple and efficient way for computing boolean representations for connectors by transforming them first into causal interaction trees and then into a conjunction of the associated causal rules.

In the following sub-sections we formalize these ideas.

4.1 Causal interaction trees

In this section, we introduce the Algebra of Causal Interaction Trees, $\mathcal{T}(P)$. We adopt the same approach as in Sect. 3.2, that is we proceed by introducing the syntax of the elements, followed by the axioms for the operators and the semantics of the algebra. We also show that the semantic equivalence respects the axioms, i.e., these axioms are sound (but not complete) with respect to the semantic equivalence.

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

$$t ::= a | a \to t | t \oplus t, \tag{7}$$

where a is an interaction, and '$\to$' and '$\oplus$' are respectively the *causality* and the *parallel composition* operators. Causality binds stronger than parallel composition.

Although the causality operator is not associative, for interactions $a_1, \ldots, a_n$, we abbreviate $a_1 \to (a_2 \to (\cdots \to a_n) \cdots))$ to $a_1 \to a_2 \to \cdots \to a_n$. We call this construction a *causal chain*.

Notice that causality is an operator transforming a pair consisting of an interaction and a causal interaction tree into a causal interaction tree, that is $\to : 2^P \times \mathcal{T}(P) \to \mathcal{T}(P)$. In the end of this section we extend causality to a total operator on $\mathcal{T}(P)$.

Axioms

1. Parallel composition, '$\oplus$', is associative, commutative, idempotent, and its identity element is 0.
2. Causality, '$\rightarrow$', satisfies the following axioms:
   (a) $a \rightarrow 1 = a$,
   (b) $a \rightarrow (1 \rightarrow t) = a \rightarrow t$,
   (c) $a \rightarrow 0 = a$,
   (d) $0 \rightarrow t = 0$.
3. The following axiom relates the two operators:

$$a \rightarrow (t_1 \oplus t_2) = a \rightarrow t_1 \oplus a \rightarrow t_2.$$

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

$$|a| = a, \tag{8}$$

$$|a \rightarrow t| = a\big(1 + |t|\big), \tag{9}$$

$$|t_1 \oplus t_2| = |t_1| + |t_2| + |t_1||t_2|, \tag{10}$$

where a is an interaction and $t, t_1, t_2 \in \mathcal{T}(P)$.

Example 4.1 (Causal chain) Consider interactions $a_1, \ldots, a_n \in 2^P$ and a causal chain $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n$. Iterating rule (9), we then have

$$\begin{aligned}|a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n| &= a_1\big(1 + |a_2 \rightarrow \cdots \rightarrow a_n|\big)\\ &= a_1 + a_1a_2\big(1 + |a_3 \rightarrow \cdots \rightarrow a_n|\big)\\ &= \cdots\\ &= a_1 + a_1a_2 + \cdots + a_1a_2\cdots a_n.\end{aligned}$$

Definition 4.2 Two causal interaction trees $t_1, t_2 \in \mathcal{T}(P)$ are *equivalent*, denoted $t_1 \sim t_2$, iff $|t_1| = |t_2|$.

Proposition 4.3 *The axiomatization of $\mathcal{T}(P)$ is sound with respect to the semantic equivalence, i.e., for $t_1, t_2 \in \mathcal{T}(P)$, $t_1 = t_2$ implies $t_1 \sim t_2$.*

Sketch This proposition is proved by verifying that the semantics of left- and right-hand sides coincide for all axioms above. For most axioms, this trivially follows from the properties of $\mathcal{AI}(P)$, such as, in particular, the idempotence of both union and synchronization. Let us show this, for instance, for axiom 3. We compute the semantics of both sides:

$$|a \rightarrow (t_1 \oplus t_2)| = a\big(1 + |t_1 \oplus t_2|\big) = a\big(1 + |t_1| + |t_2| + |t_1||t_2|\big),$$

and

$$\begin{aligned}|a \rightarrow t_1 \oplus a \rightarrow t_2| &= |a \rightarrow t_1| + |a \rightarrow t_2| + |a \rightarrow t_1||a \rightarrow t_2|\\ &= a\big(1 + |t_1|\big) + a\big(1 + |t_2|\big) + a\big(1 + |t_1|\big)a\big(1 + |t_2|\big)\\ &= a\big(1 + |t_1| + |t_2| + |t_1||t_2|\big),\end{aligned}$$

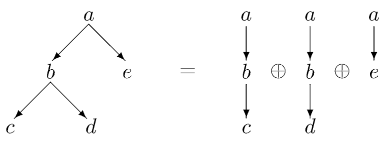


Fig. 5 A causal interaction tree is the parallel composition of its causal chains

where the last equation follows from the idempotence of operations on $\mathcal{AI}(P)$. □

Note 4.4 Axiom 3 of $\mathcal{T}(P)$ implies that any causal interaction tree can be represented as a parallel composition of its causal chains (see Fig. 5). Thus an interaction belonging to a causal interaction tree is a synchronization of any number of prefixes (cf. Example 4.1) of the corresponding causal chains, i.e., branches of this tree.

Example 4.5 (Sender/Receiver continued) The fourth column of Table 2 shows the causal interaction trees for the four interaction schemes of Example 2.3.

Proposition 4.6 *The equivalence relation $\sim$ on $\mathcal{T}(P)$ is a congruence.*

Sketch We have to show that for $t_1, t_2 \in \mathcal{T}(P)$ and a context $C(z) \in \mathcal{T}(P \cup \{z\})$, the equivalence $t_1 \sim t_2$ implies $C(t_1/z) \sim C(t_2/z)$, where $C(t_i/z)$ $(i = 1, 2)$ is the expression obtained, as in Definition 3.10, by replacing in C all occurrences of z by t_i. As the semantics of $\mathcal{T}(P)$ operators does not depend on the syntactic structure of their operands (cf. Note 3.9), structural induction on the context $C(z)$ proves the proposition. □

Note 4.7 Notice that the system of axioms for $\mathcal{T}(P)$ presented above is not complete with respect to $\sim$. For instance, for $a, b \not\in \{0, 1\}$ we have $a \to ab \sim a \to b$, but $a \to ab \neq a \to b$.

Finally, we recursively define a total causality operator on $\mathcal{T}(P)$, by putting

$$t \to t' \stackrel{def}{=} \begin{cases} a \to (t_1 \oplus t'), & \text{if } t = a \to t_1, \\ t_1 \to t' \oplus t_2 \to t', & \text{if } t = t_1 \oplus t_2. \end{cases}$$

This definition simplifies the presentation, in the next section, of the τ function transforming $\mathcal{AC}(P)$ connectors into causal interaction trees (see (12)).

Proposition 4.8 *For all $t, t_1, t_2 \in \mathcal{T}(P)$, hold the following properties of the total causality operator on $\mathcal{T}(P)$.*

1. $t \to 1 = t$,
2. $t \to (1 \to t_1) = t \to t_1$,
3. $t \to 0 = t$,
4. $t \to (t_1 \oplus t_2) = t \to t_1 \oplus t \to t_2$.

Sketch All these properties follow directly from the corresponding $\mathcal{T}(P)$ axioms by structural induction on t. □

4.2 Correspondence with $\mathcal{AC}(P)$

We define the function $\tau : \mathcal{AC}(P) \to \mathcal{T}(P)$ associating a causal interaction tree with a connector. By Corollary 3.12 and the associativity of both fusion in $\mathcal{AC}(P)$ and parallel composition in $\mathcal{T}(P)$, the following equations are sufficient to define τ:

$$\tau(p) = p, \tag{11}$$

$$\tau\left([x]'\prod_{i=1}^{n}[y_i]\right) = \tau(x) \to \bigoplus_{i=1}^{n}\tau(y_i), \tag{12}$$

$$\tau\big([x_1]'[x_2]'\big) = \tau(x_1) \oplus \tau(x_2), \tag{13}$$

$$\tau\big([y_1][y_2]\big) = \bigoplus_{i=1}^{m_1}\bigoplus_{j=1}^{m_2} a_i^1 a_j^2 \to \big(t_i^1 \oplus t_j^2\big), \tag{14}$$

where $x, x_1, x_2, y_1, \dots, y_n \in \mathcal{AC}(P)$, $p \in P \cup \{0, 1\}$, and, in (14), we assume $\tau(y_k) = \bigoplus_{i=1}^{m_k} a_i^k \to t_i^k$, for $k = 1, 2$.

Example 4.9 Consider $P = \{p, q, r, s, t, u\}$ and $p'q'[[r's][t'u]] \in \mathcal{AC}(P)$. We have

$$\begin{aligned}\tau\left(p'q'\big[[r's][t'u]\big]\right) &= \tau\left(\big[p'q'\big]'\big[[r's][t'u]\big]\right) = \tau(p'q') \to \tau\big([r's][t'u]\big)\\ &= (p \oplus q) \to \big(rt \to (s \oplus u)\big)\\ &= \big(p \to rt \to (s \oplus u)\big) \oplus \big(q \to rt \to (s \oplus u)\big).\end{aligned}$$

We also define the function $\sigma : \mathcal{T}(P) \to \mathcal{AC}(P)$, which associates with a causal interaction tree a connector:

$$\sigma(a) = a, \tag{15}$$

$$\sigma(a \to t) = [a]'[\sigma(t)], \tag{16}$$

$$\sigma(t_1 \oplus t_2) = [\sigma(t_1)]'[\sigma(t_2)]'. \tag{17}$$

Proposition 4.10 *The functions $\sigma : \mathcal{T}(P) \to \mathcal{AC}(P)$ and $\tau : \mathcal{AC}(P) \to \mathcal{T}(P)$, satisfy the following properties*

1. $\forall x \in \mathcal{AC}(P)$, $|x| = |\tau(x)|$,
2. $\forall t \in \mathcal{T}(P)$, $|t| = |\sigma(t)|$,
3. $\tau \circ \sigma = id$,
4. $\sigma \circ \tau \simeq id$, *that is* $\forall x \in \mathcal{AC}(P)$, $\sigma(\tau(x)) \simeq x$.

Sketch The first three properties can be demonstrated by comparing definitions (3)–(5) and (8)–(10) of the semantic function $|\cdot|$ and (12)–(17) for functions τ and σ. The fourth property then follows trivially from the first two. □

The above proposition says that the diagram shown in Fig. 6 is commutative except for the loop $\mathcal{AC}(P) \xrightarrow{\tau} \mathcal{T}(P) \xrightarrow{\sigma} \mathcal{AC}_c(P) \hookrightarrow \mathcal{AC}(P)$.

In this diagram, $\mathcal{AC}_c(P) \subset \mathcal{AC}(P)$ is the set of *causal connectors*, which is the image of $\mathcal{T}(P)$ by σ. Note that any connector has an equivalent representation in $\mathcal{AC}_c(P)$. Similarly, $\mathcal{AI}_c(P) \subset \mathcal{AI}(P)$ is the set of *causal interactions*, the image of $\mathcal{T}(P)$ by the semantic function $|\cdot|$. Proposition 4.12 provides a characteristic property of the set of causal interactions.

Fig. 6 A diagram relating the algebras

$$\begin{tikzcd}
\mathcal{AC} \arrow[rr, "|\cdot|"] \arrow[dr, "\tau"] & & \mathcal{AI} \\
& \mathcal{T} \arrow[dl, "\sigma"'] \arrow[dr, "|\cdot|"] & \\
\mathcal{AC}_c \arrow[uu, hook] \arrow[rr, "|\cdot|"] & & \mathcal{AI}_c \arrow[uu, hook]
\end{tikzcd}$$

Note 4.11 Notice that σ together with the restriction of τ to $\mathcal{AC}_c(P)$ define an isomorphism between $\mathcal{T}(P)$ and $\mathcal{AC}_c(P)$. Indeed, consider $x \in \mathcal{AC}_c(P)$. By definition of $\mathcal{AC}_c(P)$, there exists $t \in \mathcal{T}(P)$ such that $x = \sigma(t)$. By Proposition 4.10.3, we then have

$$\sigma(\tau(x)) = \sigma(\tau(\sigma(t))) = \sigma(t) = x,$$

which shows that $(\sigma \circ \tau)|_{\mathcal{AC}_c(P)} = id$, that is $\forall x \in \mathcal{AC}_c(P)$, $\sigma(\tau(x)) = x$. This equality, together with Proposition 4.10.4 proves the isomorphism.

Proposition 4.12 *The set of the causal interactions is closed under synchronization, that is* $x \in \mathcal{AI}_c(P)$ *iff* $\forall a, b \in x$, $ab \in x$.

Proof Consider $x \in \mathcal{AI}_c(P)$ and interactions $a, b \in x$. There exists $t \in \mathcal{T}(P)$ such that $x = |t|$. Hence, according to Note 4.4, both a and b can be represented as unions of a number of prefixes of branches of t, which implies automatically that ab can also be represented in this form, and therefore $ab \in x$.

To prove that the condition of the proposition is sufficient, consider $x \in \mathcal{AI}(P)$ satisfying this property, and take $t = \bigoplus_{a\in x} a$. Clearly, $|t| = x$, which, by definition, implies $x \in \mathcal{AI}_c(P)$. □

4.3 Boolean representation of connectors

We now introduce *causal rules* that provide a mechanism for operations on causal trees and, consequently, a simple and efficient way for computing boolean representations for $\mathcal{AC}(P)$ connectors.

Definition 4.13 A *causal rule* is a $\mathbb{B}[P]$ formula $E \Rightarrow C$, where E (the *effect*) is either a constant, *true*, or a port variable $p \in P$, and C (the *cause*) is either a constant, *true* or *false*, or a disjunction of interactions, i.e., $\bigvee_{i=1}^n a_i$ where, for all $i \in [1, n]$, a_i are conjunctions of port variables.

Causal rules without constants can be rewritten as formulas of the form $\overline{p} \vee \bigvee_{i=1}^n a_i$ and, consequently, are conjunctions of dual Horn clauses, i.e., disjunctions of variables whereof at most one is negative.

In line with Definition 3.2, an interaction $a \in 2^P$ satisfies the rule $p \Rightarrow \bigvee_{i=1}^n a_i$, iff $p \in a$ implies $a_i \subseteq a$, for some $i \in [1, n]$, that is for a port to belong to an interaction at least one of the corresponding causes must belong there too.

Example 4.14 Let $p \in P$, $a \in 2^P$, and $x \in \mathcal{AI}(P)$. Three particular types of causal rules can be set apart:

1. For an interaction to satisfy the rule *true* $\Rightarrow a$, it is necessary that it contain a.

2. Rules of the form $p \Rightarrow \mathit{true}$ are satisfied by all interactions.
3. An interaction can satisfy the rule $p \Rightarrow \mathit{false}$ only if it does not contain p.

Note 4.15 Notice that $a_1 \vee a_1a_2 = a_1$, and therefore causal rules can be simplified accordingly:

$$(p \Rightarrow a_1 \vee a_1a_2) \rightsquigarrow (p \Rightarrow a_1). \tag{18}$$

We assume that all the causal rules are simplified by using (18).

Definition 4.16 A *system of causal rules* is a set $R = \{p \Rightarrow x_p\}_{p \in P^t}$, where $P^t \overset{def}{=} P \cup \{\mathit{true}\}$. An interaction $a \in 2^P$ satisfies the system R (denoted $a \models R$), iff $a \models \bigwedge_{p \in P^t}(p \Rightarrow x_p)$. We denote by $|R|$ the union (in the sense of $\mathcal{AI}(P)$) of the interactions satisfying R:

$$|R| \overset{def}{=} \sum_{a \models R} a.$$

A causal interaction tree $t \in \mathcal{T}(P)$ is equivalent to a system of causal rules R iff $|t| = |R|$.

We associate with $t \in \mathcal{T}(P)$ the system of causal rules

$$R(t) \overset{def}{=} \{p \Rightarrow c_p(t)\}_{p \in P^t}, \tag{19}$$

where, for $p \in P^t$, the function $c_p : \mathcal{T}(P) \to \mathbb{B}[P]$ is defined as follows. For $a \in 2^P$ (with $p \notin a$) and $t, t_1, t_2 \in \mathcal{T}(P)$, we put

$$c_p(0) = \mathit{false}, \tag{20}$$

$$c_p(p \to t) = \mathit{true}, \tag{21}$$

$$c_p(pa \to t) = a, \tag{22}$$

$$c_p(a \to t) = a \wedge c_p(t), \tag{23}$$

$$c_p(t_1 \oplus t_2) = c_p(t_1) \vee c_p(t_2). \tag{24}$$

Similarly, we define $c_{\mathit{true}}(t)$ by

$$\begin{aligned} c_{\mathit{true}}(0) &= \mathit{false}, \\ c_{\mathit{true}}(1 \to t) &= \mathit{true}, \\ c_{\mathit{true}}(a \to t) &= a, \\ c_{\mathit{true}}(t_1 \oplus t_2) &= c_{\mathit{true}}(t_1) \vee c_{\mathit{true}}(t_2). \end{aligned}$$

Note 4.17 It is important to observe that, for any $t \in \mathcal{T}(P)$, the system of causal rules $R(t)$, defined by (19), contains exactly one causal rule for each $p \in P^t$ (i.e., each $p \in P$ and *true*). For ports that do not participate in t, the rule is $p \Rightarrow \mathit{false}$. For ports that do not have any causality constraints, the rule is $p \Rightarrow \mathit{true}$.

Example 4.18 Consider the causal interaction tree $t = p \to (q \to r \oplus qs)$ shown in Fig. 7. The associated system $R(t)$ of causal rules is

$$\{\mathit{true} \Rightarrow p,\ p \Rightarrow \mathit{true},\ q \Rightarrow p,\ r \Rightarrow pq,\ s \Rightarrow pq\}.$$

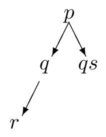


Fig. 7 Graphical representation of the causal interaction tree $t = p \to (q \to r \oplus qs)$

Notice that $c_q(t) = p(c_q(q \to r) \vee c_q(qs)) = p \vee ps = p$.

The corresponding boolean formula is then

$$(true \Rightarrow p) \wedge (p \Rightarrow true) \wedge (q \Rightarrow p) \wedge (r \Rightarrow pq) \wedge (s \Rightarrow pq) = pq \vee p\overline{r}\,\overline{s}.$$

Proposition 4.19 *For any causal interaction tree $t \in \mathcal{T}(P)$, $|t| = |R(t)|$.*

Sketch This proposition follows from Note 4.4 and a similar observation for the system $R(t)$. Indeed, according to the rules (20)–(24) and the simplification rule (18), the cause in a causal rule $p \Rightarrow c_p(t)$ is the union of all the shortest prefixes in t, containing p. □

5 Synthesis of causal interaction trees

5.1 Expressing boolean functions as causal rules

In the previous section, we have introduced the causal rules providing a straightforward way for computing boolean representations of connectors (and causal interaction trees). This section deals with the reverse procedure: given a boolean function on P, we construct a causal interaction tree model (and consequently an $\mathcal{AC}(P)$ connector). We proceed in two steps: given a boolean formula, we translate it into an equivalent one, which is the disjunction of the corresponding causal rules, from which we construct an equivalent causal interaction tree. This translation leads, in particular, to the definition of a normal form for causal interaction trees. We also derive a simple algorithm for computing the intersection of causal interaction trees directly on the corresponding systems of causal rules.

In order to compute the causal rules for a given boolean function $\varphi \in \mathbb{B}[P]$, we take its conjunctive normal form (CNF) $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ with, for $k \in [1, n]$, $C_k = \bigvee_{i \in I_k} p_i \vee \bigvee_{j \in J_k} \overline{p_j}$, where $I_k \cap J_k = \emptyset$, and $p_i, p_j \in P$ for all $i \in I_k$ and $j \in J_k$. We can now rewrite every clause C_k, with $J_k \neq \emptyset$, as a disjunction of dual Horn clauses $C_k = \bigvee_{j \in J_k} (\overline{p_j} \vee \bigvee_{i \in I_k} p_i)$. By distributivity, we obtain a representation of φ as a disjunction of dual Horn formulas and, after combining the clauses with the same negative variable, $\varphi = R_1 \vee R_2 \vee \cdots \vee R_m$ with, for $k \in [1, m]$,

$$R_k = \bigwedge_{i \in \widetilde{I}_k} \Bigg(\overline{p_i} \vee \bigvee_{j \in \widetilde{J}_{k,i}} a_j \Bigg) = \bigwedge_{i \in \widetilde{I}_k} \Bigg(p_i \Rightarrow \bigvee_{j \in \widetilde{J}_{k,i}} a_j \Bigg),$$

where, for all $i \in \widetilde{I}_k$, $p_i \in P^t$ and, for all $j \in \widetilde{J}_{k,i}$, a_j is *false*, *true*, or a conjunction of positive variables. Recall (Example 4.14) that for a positive clause C_k we have $C_k = (true \Rightarrow C_k)$, whereas $\overline{p} = (p \Rightarrow false)$. Thus, each R_k is a system of causal rules as defined in Sect. 4.3.

Proposition 5.1 *For $\varphi \in \mathbb{B}[P]$, the above representation is defined uniquely.*

The next section presents an algorithm for constructing a causal interaction tree for a system of causal rules, thus completing the chain of transformations necessary for constructing the causal interaction tree corresponding to a boolean function.

This algorithm also allows one to normalize and compute intersections of causal interaction trees by transforming them into systems of causal rules and back. These two operations are also presented in the subsequent sections.

5.2 Constructing causal interaction trees from causal rules

Definition 5.2 A system of causal rules $\{p_i \Rightarrow x_i\}_{i=1}^n$ is *saturated* iff, for all $i \in [1,n]$, $x_i = x_i[(x_j p_j)/p_j]$, where $x_i[(x_j p_j)/p_j]$ is obtained by substituting $(x_j p_j)$ for p_j in x_i, for all $j \neq i$. We denote by $\mathcal{CR}(P)$ the set of saturated systems of causal rules over P.

For a given system of causal rules $R = \{p_i \Rightarrow x_i\}_{i=1}^n$, we denote by $R_{sat} = \{p_i \Rightarrow x_i^*\}_{i=1}^n$, the saturated system of rules, where $\{x_i^*\}_{i=1}^n$ is the unique fixpoint iteratively computed by

$$x_i^0 = x_i, \qquad x_i^{k+1} = x_i^k[(p_j x_j^k)/p_j], \quad \text{for } i = 1, \dots, n.$$

Clearly, this computation terminates within a bounded number of iterations.

Lemma 5.3 *Let R be a system of causal rules, and R_{sat} be the corresponding saturated system. Then $|R| = |R_{sat}|$.*

Sketch This lemma follows directly from the fact that the substitution, used to compute the fixpoint in the definition of saturation, preserves boolean equivalence of systems of causal rules. □

Example 5.4 Consider the system of causal rules $\{p \Rightarrow a_p,\ q \Rightarrow p a_q\}$, where $p, q \in P$ are two ports, and $a_p, a_q \in 2^P$. To saturate it, we substitute $p a_p$ for p in the second rule to obtain $\{p \Rightarrow a_p,\ q \Rightarrow p a_p a_q\}$.

Clearly, the corresponding boolean formulas are equivalent:

$$\begin{aligned}(p \Rightarrow a_p) \wedge (q \Rightarrow p a_q) &= (\overline{p} \vee a_p) \wedge (\overline{q} \vee p a_q)\\ &= \overline{pq} \vee a_p \overline{q} \vee p a_p a_q = (\overline{p} \vee a_p) \wedge (\overline{q} \vee p a_p a_q)\\ &= (p \Rightarrow a_p) \wedge (q \Rightarrow p a_p a_q).\end{aligned}$$

Note 5.5 For any $t \in \mathcal{T}(P)$, the system of causal rules $R(t)$, defined by (19), is saturated.

Lemma 5.6 *Let $X = \{p \Rightarrow x_p\}_{p \in P'}$ be a saturated system of causal rules simplified by absorption (18), with $x_p = \bigvee_{i=1}^{m_p} a_i^p$. The set $Y = \{p a_i^p \mid p \in P, i \in [1, m_p]\} \cup \{a_i^{true} \mid i \in [1, m_{true}]\}$ consists exactly of all interactions satisfying X and minimal in the following sense:*

1. *Any interaction a, such that $a \models X$, can be decomposed as $a = b_1 \dots b_k$, with $b_1, \dots, b_k \in Y$.*
2. *No $a \in Y$ can be further decomposed in this way, i.e., $a = b_1 \dots b_k$, with $1 \neq b_j \in Y$ for $j \in [1,k]$, implies $k = 1$.*

```
Input:   A saturated system of causal rules $X = \{p \Rightarrow x_p\}_{p \in P^t}$ with $x_p = \bigvee_{i=1}^{m_p} a_i^p$.
Output:  A causal interaction tree $t \in \mathcal{T}(P)$ equivalent to $X$.
// Initialisation phase
1.  $Y := \{pa_i^p \mid p \in P, i \in [1, m_p]\} \cup \{a_i^{true} \mid i \in [1, m_{true}]\}$;
2.  $Z, Z_0 := \{1\}$; // 1 corresponds to the empty interaction $\emptyset \in 2^P$
3.  $t := 1$;        // $t \in \mathcal{T}(P)$—the resulting tree
4.  $n := 0$;        // counter variable for iteration
// Computation phase
5.  $n := n + 1$;
6.  $Z_n := \min(Y \setminus Z)$; // the subset of interactions of minimal cardinality in $Y \setminus Z$
7.  for each $w \in Z_n$
8.      $k := \max\{j \in [0, n-1] \mid \exists z \in Z_j : z \subset w\}$; // k is defined, as $\forall w \subset P, 1 \subset w$
9.      for each $z \in Z_k$ such that $z \subset w$
10.         // add a son labeled by w to the node z in t
            replace the sub-tree $z \to t_z$ of $t$ rooted in $z$ by $z \to (t_z \oplus w)$;
11. $Z := Z \cup Z_n$;
12. if $Y \neq Z$, goto Step 5;
// Clean-up phase
13. if $1 \notin Y$, $t := \bigoplus_i t_i$ // we had $t = 1 \to \bigoplus_i t_i$ due to the initialisation at Step 3.
14. starting from the leaves of $t$, for all nodes $a_1 \to a_2$ replace $a_2$ by $a_2 \setminus a_1$.
```

Fig. 8 Algorithm for constructing a causal interaction tree from a saturated system of causal rules

Proof 1. Consider $a \models X$, and a port $p_1 \in a$. We then have $a \models \bigwedge_{p \in P^t}(p \Rightarrow \bigvee_{i=1}^{m_p} a_i^p)$, and therefore, for some $i_1 \in [1, m_{p_1}]$, $a_{i_1}^{p_1} \subseteq a$. Hence, $a = b_1 a_1$, with $b_1 = p_1 a_{i_1}^{p_1}$ and $a_1 = a \setminus b_1$. Idempotence of synchronization in $\mathcal{AI}(P)$ allows us to proceed by picking some $p_2 \in a_1$ and applying the same reasoning to obtain $a = b_1 b_2 a_2$, where $b_j = p_j a_{i_j}^{p_j}$, for $j = 1, 2$, and $a_2 = a_1 \setminus b_2$, and so on. As at each step we select $p_j \in a_{j-1}$, we have $a_j \subsetneq a_{j-1}$, and therefore, for some k, $a_k = 1$, which implies $a = b_1 \dots b_k$, with $b_j = p_j a_{i_j}^{p_j} \in Y$, for $j \in [1, k]$.

2. Consider $a = b_1 \dots b_k \in Y$. As $a \in Y$, for some $p \in P^t$, we have $a = pa_p$, where a_p is a summand in x_p. If $k > 1$, there exists $l \in [1, k]$ such that $p \in b_l$ and $b_l \subsetneq a$. As $b_l \in Y$, we have $b_l = qa_q$, for some $q \in P^t$ and a_q a summand in x_q. The assumption that all rules are simplified by absorption (18) implies that $p \neq q$. As X is saturated, we have

$$x_q = x_q\big[(px_p)/p\big] = \bigvee_{i=1}^{m_q} \bigvee_{j=1}^{m_p} a_i^q[(pa_j^p)/p] = \bigvee_{i \in I}\left(\bigvee_{j=1}^{m_p} a_i^q a_j^p\right) \vee \bigvee_{i \notin I} a_i^q,$$

where $I \subseteq [1, m_q]$ is the subset indexing the summands of x_q containing p, i.e., $i \in I$ iff $p \in a_i^q$. As a_q is a summand in x_q and $p \in a_q$, there exist $i \in [1, m_q]$ and $j \in [1, m_p]$, such that $a_q = a_i^q a_j^p$. Hence, $a_j^p \subseteq a_q \subsetneq a_p$ (recall that $q \in a_p$, but $q \notin a_q$). However, both a_j^p and a_p are summands in x_p, which contradicts the assumption that all rules are simplified by absorption (18). □

Theorem 5.7 *Given a saturated system of causal rules $X = \{p \Rightarrow x_p\}_{p \in P^t}$, the algorithm shown in Fig. 8, constructs an equivalent causal interaction tree t.*

Proof Consider the set Y defined in Lemma 5.6. By Lemma 5.6(2), the elements of Y correspond exactly to all the prefixes in the causal interaction tree constructed by the algorithm

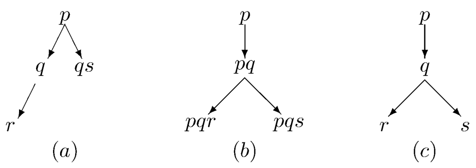


Fig. 9 Construction of the normal form of causal interaction trees example: initial (**a**), intermediate (**b**), and normal form (**c**) trees

shown in Fig. 8. Thus, Lemma 5.6(1) implies $|X| \subseteq |t|$. As X is saturated, $Y \subset |X|$. Therefore, by Proposition 4.12, $|t| \subseteq |X|$, which finalizes the proof. □

5.3 Normal form for causal interaction trees

Lemma 5.8 *Let R_1, R_2 be two saturated systems of causal rules. Then $|R_1| = |R_2|$ implies $R_1 = R_2$ (equal as sets of causal rules that cannot be simplified by absorption).*

Proof Suppose that $R_1 \neq R_2$, and all rules are simplified by absorption (18). Then there exists $p \in P$ such that the rules $(p \Rightarrow \bigvee_{i=1}^n a_i) \in R_1$ and $(p \Rightarrow \bigvee_{j=1}^m b_j) \in R_2$ do not coincide. Without loss of generality, for some $i \in [1, n]$, we have $b_j \not\subseteq a_i$ simultaneously for all $j \in [1, m]$. This implies that the interaction pa_i does not satisfy the rule $(p \Rightarrow \bigvee_{j=1}^m b_j) \in R_2$, and therefore $pa_i \notin |R_2|$. At the same time $pa_i \in |R_1|$, as R_1 is saturated. □

Corollary 5.9 *Let $t_1, t_2 \in \mathcal{T}(P)$ be two equivalent causal interaction trees. The corresponding systems of causal rules $R(t_1)$ and $R(t_2)$ (cf. (19)) are identical.*

Proof $t_1 \sim t_2$ implies $|R(t_1)| = |t_1| = |t_2| = |R(t_2)|$. Hence, by Lemma 5.8, $R(t_1) = R(t_2)$. □

Definition 5.10 Let $t \in \mathcal{T}(P)$ be a causal interaction tree. The *normal form* of t is the causal interaction tree obtained by applying the algorithm in Fig. 8 to the system $R(t)$.

Proposition 4.19, Theorem 5.7 and Corollary 5.9 guarantee that the definition above is well founded and, indeed, defines a normal form.

Example 5.11 Consider again the causal interaction tree $t = p \rightarrow (q \rightarrow r \oplus qs)$ from Example 4.18 (also shown in Fig. 9(a)) and the associated system $R(t)$ of causal rules

$$\{true \Rightarrow p,\ p \Rightarrow true,\ q \Rightarrow p,\ r \Rightarrow pq,\ s \Rightarrow pq\}.$$

The set Y defined in Step 1 of the algorithm in Fig. 8 is therefore $\{p, pq, pqr, pqs\}$. The sets of minimal cardinality interactions (represented by Z_n) at subsequent iterations at Step 6 are then respectively $\{p\}$, $\{pq\}$, and $\{pqr, pqs\}$. Thus, the intermediate tree constructed in Steps 2–13 is that shown in Fig. 9(b), whereas the final tree is shown in Fig. 9(c).

6 Examples

6.1 Multi-shot semantics

For the examples of this section we will need the notions of *interconnected systems* and *multi-shot semantics* introduced in [9].

```
Input:   Causal interaction trees $t_1, t_2 \in \mathcal{T}(P)$.
Output:  A causal interaction tree $t \in \mathcal{T}(P)$ such that $|t| = |t_1| \cap |t_2|$.
1.  compute systems of causal rules $R(t_1)$ and $R(t_2)$ defined by (19);
2.  $X := \{p \Rightarrow c_p(t_1) \wedge c_p(t_2) | p \in P^t\}$;
3.  compute $X_{sat}$ by saturating $X$;
4.  apply the algorithm in Fig. 8 to $X_{sat}$;
```

Fig. 10 Algorithm for computing an intersection of two causal interaction trees

Definition 6.1 An *interconnected system* is a pair $(\{B_i\}_{i=1}^n, \{C_j\}_{j=1}^m)$, 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 $C_j \in \mathcal{AC}(P)$ with $P = \bigcup_{i=1}^n P_i$.

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

Multi-shot semantics corresponds to the case, where d is maximal (i.e., $d = m$), and, in particular, $\gamma_m = |\prod_{i=1}^m [C_i]'|$.

Notice that, for an interconnected system, d-shot semantics entails simultaneous firing of interactions from at most d connectors.

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

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

To compute efficiently the intersection of two causal connectors we use their boolean representation. It follows trivially from Proposition 4.12 that an intersection of two causal sets of interactions is itself causal. To simplify presentation, we reason on causal interaction trees. The mapping between $\mathcal{AC}(P)$ and $\mathcal{T}(P)$ is given by functions σ and τ defined in Sect. 4.2.

Lemma 6.3 *For $t_1, t_2 \in \mathcal{T}(P)$, the algorithm in Fig. 10 computes the unique causal interaction tree in normal form $t \in \mathcal{T}(P)$ (denoted $t_1 \cap t_2$), such that $|t| = |t_1| \cap |t_2|$.*

Example 6.4 Consider two causal interaction trees with opposite causal relations: $t_1 = p \rightarrow q$ (possible interactions: p and pq) and $t_2 = q \rightarrow p$ (possible interactions: q and pq). Let us compute $t_1 \oplus t_2$ and $t_1 \cap t_2$.

The systems of causal rules corresponding to t_1 and t_2 are respectively $\{true \Rightarrow p,\ p \Rightarrow true,\ q \Rightarrow p\}$ and $\{true \Rightarrow q,\ q \Rightarrow true,\ p \Rightarrow q\}$.

1. To compute $t_1 \oplus t_2$, we apply (24) from the definition of the function c_p to obtain the system of causal rules $\{true \Rightarrow p \vee q,\ p \Rightarrow true \vee q,\ q \Rightarrow true \vee p\}$, which is simplified

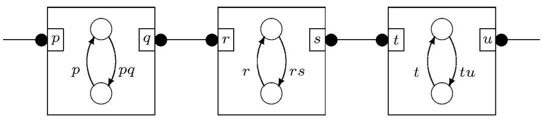


Fig. 11 Multi-shot modulo-8 counter

Table 3 Causal interaction trees and rules for Example 6.5

G			γ_m			
$(p \to q) \oplus (r \to s) \oplus (t \to u)$			$p \oplus qr \oplus st \oplus u$			
$true \Rightarrow p + r + t$			$true \Rightarrow p + qr + st + u$			
$p \Rightarrow true$	$r \Rightarrow true$	$t \Rightarrow true$	$p \Rightarrow true$			
$q \Rightarrow p$	$s \Rightarrow r$	$u \Rightarrow t$	$q \Rightarrow r$	$r \Rightarrow q$	$s \Rightarrow t$	$t \Rightarrow s$

to $\{true \Rightarrow p \vee q,\ p \Rightarrow true,\ q \Rightarrow true\}$ by absorption (18). This corresponds to the causal interaction tree $p \oplus q$.

2. To compute $t_1 \cap t_2$, we apply Lemma 6.3 to obtain the system $\{true \Rightarrow pq, p \Rightarrow q, q \Rightarrow p\}$, which saturates to $\{true \Rightarrow pq, p \Rightarrow pq, q \Rightarrow pq\}$, and produces therefore the causal interaction tree with a single node pq.

Example 6.5 (Modulo-8 counter continued) Consider the interconnected system in Fig. 11. The set of the possible interactions for multi-shot semantics is the intersection of $G = [p'q]'[r's]'[t'u]'$ and $\gamma_m = p'[qr]'[st]'u'$, and corresponds to the modulo-8 counter from Example 2.4. As in Proposition 6.2, G represents the interactions offered by the components, whereas γ_m represents those offered by atomic connectors. The causal interaction trees for G and γ_m are shown in Table 3, as well as the corresponding systems of causal rules.

Applying Lemma 6.3 and absorption (18), we compute the system of causal rules for $G \cap \gamma_m$:

$$\{true \Rightarrow p + qr + st + ru + tu,\ p \Rightarrow true,\ q \Rightarrow pr,\ r \Rightarrow q,\ s \Rightarrow rt,\ t \Rightarrow s,\ u \Rightarrow t\}.$$

After saturation, we obtain

$$\{true \Rightarrow p,\ p \Rightarrow true,\ q \Rightarrow pqr,\ r \Rightarrow pqr,\ s \Rightarrow pqrts,\ t \Rightarrow pqrts,\ u \Rightarrow pqrts\}.$$

By applying the algorithm of Fig. 8, we obtain the causal interaction tree $p \to qr \to st \to u$, which represents the connector of Example 3.7.

6.2 Two tasks with preemption

Let T_1 and T_2 be two tasks running on a single processor. We assume that each one of these tasks can preempt the other. No interactions other than preemption are possible.

Tasks can be modeled by the generic atomic component shown in Fig. 12(a). Its behavior has three states: 1—the task is running, 2—the task is waiting to begin computation, and 3—the task has been preempted and is waiting to resume computation. The transitions are labeled b, f, p, and r for *begin*, *finish*, *preempt*, and *resume* respectively, and can be synchronized with external events through the corresponding ports of the behavior.

Mutual preemption is described by two statements:

1. A running task is preempted, when the other one begins computation.
2. A preempted task resumes computation, when the other one finishes.

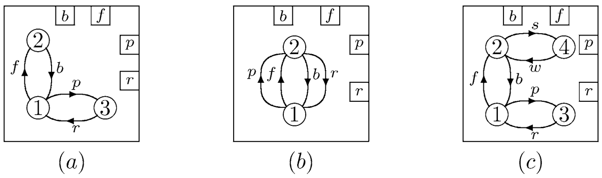


Fig. 12 Three behaviors modeling a preemptable task

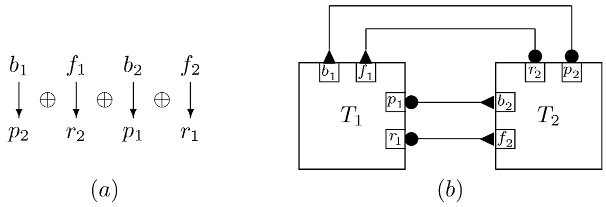


Fig. 13 A causal interaction tree (**a**) and an interconnected system (**b**) modeling two mutually preempting tasks

In order to compute the connectors ensuring these interactions, we rewrite these statements as causal rules on $\{b_i, f_i, p_i, r_i\}_{i=1,2}$:

$$\begin{aligned} &true \Rightarrow b_1 \vee f_1 \vee b_2 \vee f_2, \\ &p_1 \Rightarrow b_2, \qquad p_2 \Rightarrow b_1, \\ &r_1 \Rightarrow f_2, \qquad r_2 \Rightarrow f_1. \end{aligned} \tag{25}$$

The first rule in (25) means that at any moment at least one task must execute a begin or finish action. It can be easily verified that this system of causal rules is saturated. Applying the algorithm in Fig. 8, we obtain the causal interaction tree shown in Fig. 13(a), and, furthermore,

$$\sigma\big(b_1 \to p_2 \oplus f_1 \to r_2 \oplus b_2 \to p_1 \oplus f_2 \to r_1\big) = \big[b_1' p_2\big]'\big[f_1' r_2\big]'\big[b_2' p_1\big]'\big[f_2' r_1\big]'. \tag{26}$$

Figure 13(b) shows an interconnected system consisting of two tasks and four connectors forming the right-hand side of (26). The multi-shot semantics of this system realizes the desired interaction model.

Different behaviors can be used to model a preemptable task. In addition to behavior in Fig. 12(a), other possible behaviors for tasks are given in Fig. 12(b, c).

1. The behavior in Fig. 12(b) has two states: 1—the task is running, 2—the task is waiting, with the four transitions labeled by b, f, p, and r.
2. The behavior in Fig. 12(c) has four states. States 1–3 are the same as those of the behavior in Fig. 12(a), whereas in state 4 the task is sleeping, and the two additional transitions s and w correspond respectively to actions *sleep* and *wake-up*.

Independently of which behavior in Fig. 12 is used to model the tasks, the interactions offered by each task are $G_i = b_i + f_i + p_i + r_i$, for $i = 1, 2$ (cf. Proposition 6.2). Thus, the interactions possible in the multi-shot semantics of the system in Fig. 13(b) are described by

$$\begin{aligned} &\big[b_1 + f_1 + p_1 + r_1\big]'\big[b_2 + f_2 + p_2 + r_2\big]' \cap \big[b_1' p_2\big]'\big[f_1' r_2\big]'\big[b_2' p_1\big]'\big[f_2' r_1\big]' \\ &\quad \simeq b_1' p_2 + f_1' r_2 + b_2' p_1 + f_2' r_1. \end{aligned}$$

Hence, the multi-shot and single-shot semantics of this system coincide.

Notice, however, that the connectors computed above define the set of allowed interactions. The actual interactions, as well as the order of their execution, depend on the behavior that is used to model the tasks. For example, when the behavior in Fig. 12(b) is used, the following trace is acceptable in the composed system

$$b_1(b_2p_1)(b_1p_2)(b_2p_1)\cdots,$$

whereas, when the behavior in Fig. 12(a) is used, an interaction f_2r_1 must follow b_2p_1:

$$b_1(b_2p_1)(f_2r_1)\cdots.$$

6.3 Sequential execution of two tasks

In a setting, similar to that of the previous section—that is two tasks T_1 and T_2 running on a single processor and mutually preempting each other—we now additionally require that each execution of T_1 be followed by one of T_2, and, conversely, each execution of T_2 be preceded by one of T_1. In other words, we impose the sequential execution $T_1; T_2$.

The corresponding causal rules can therefore be obtained by adding to (25) the boolean constraint $b_2 = f_1$, expressed as the conjunction of two causal rules $f_1 \Rightarrow b_2$ and $b_2 \Rightarrow f_1$. The resulting system is shown in Fig. 14(a) and the saturated one in Fig. 14(b).

The causal interaction tree computed by the algorithm in Fig. 8 is shown in Fig. 15(a), and the corresponding $\mathcal{AC}(P)$ connector is

$$\sigma\big(b_1 \to p_2 \oplus b_2f_1 \to (r_2 \oplus p_1) \oplus f_2 \to r_1\big) = \big[b_1'p_2\big]'\big[[b_2f_1]'r_2p_1\big]'\big[f_2'r_1\big]',$$

which corresponds to the multi-shot semantics of the interconnected system (with three connectors) in Fig. 15(b). (Notice that the ports b_2 and r_2 are interchanged in this figure, as compared to Fig. 13(b).)

The above systems of causal rules represent boolean constraints on the interactions of the composed system, derived from the required interaction model (i.e., sequential execution with preemption). Other boolean constraints can also be considered. When we model the two tasks as atomic components given in Fig. 12, we can derive additional constraints

Fig. 14 Causal rules for the sequential execution of two tasks example

$$\begin{array}{ll} \multicolumn{2}{l}{true \Rightarrow b_1 \vee f_1 \vee b_2 \vee f_2} \\ p_1 \Rightarrow b_2 & p_2 \Rightarrow b_1 \\ r_1 \Rightarrow f_2 & r_2 \Rightarrow f_1 \\ f_1 \Rightarrow b_2 & b_2 \Rightarrow f_1 \\ \multicolumn{2}{c}{(a)} \end{array} \qquad \begin{array}{ll} \multicolumn{2}{l}{true \Rightarrow b_1 \vee b_2f_1 \vee f_2} \\ p_1 \Rightarrow b_2f_1 & p_2 \Rightarrow b_1 \\ r_1 \Rightarrow f_2 & r_2 \Rightarrow b_2f_1 \\ f_1 \Rightarrow b_2 & b_2 \Rightarrow f_1 \\ \multicolumn{2}{c}{(b)} \end{array}$$

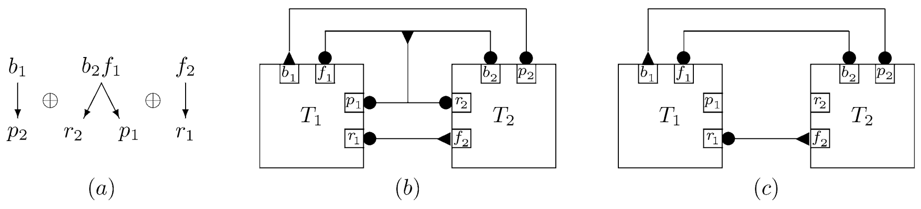


Fig. 15 A causal interaction tree (**a**) and two interconnected systems (**b**, **c**) Medellin a sequential execution of two mutually preempting tasks

from their behavior: two ports from the same component cannot participate together in the same interaction. Hence one can, for instance, add to the system in Fig. 14(b) the boolean constraints $p_1 \Rightarrow \overline{b_1}\,\overline{f_1}\,\overline{r_1}$ and $r_2 \Rightarrow \overline{b_2}\,\overline{f_2}\,\overline{p_2}$, which modifies the existing causal rules for p_1 and r_2, giving $p_1 \Rightarrow \mathit{false}$ and $r_2 \Rightarrow \mathit{false}$. Thus, the resulting causal interaction tree is $b_1 \to p_2 \oplus b_2 f_1 \oplus f_2 \to r_1$, and the corresponding $\mathcal{AC}(P)$ connector is (see Fig. 15(c))

$$\sigma\big(b_1 \to p_2 \oplus b_2 f_1 \oplus f_2 \to r_1\big) = \big[b_1' p_2\big]'\big[b_2 f_1\big]'\big[f_2' r_1\big]'.$$

6.4 Three sequential tasks running on two processors

We now further develop the example of the previous sections, by introducing one more task, although running on a different processor. Thus our system is composed of three tasks T_1, T_2, and T_3, with T_1 and T_3 running on the same processor and preempting each other as in Sect. 6.2 (cf. Fig. 16(a)). The second task is running on a separate processor and, consequently, cannot be preempted. Therefore ports p and r are irrelevant for T_2, and will not be considered in the sequel.

We want to ensure the following execution protocol:

1. Each execution of T_2 must be immediately preceded by an execution of T_1. However, T_1 can be executed without being immediately followed by an execution of T_2 (dashed arrow in Fig. 16(a)). (One can assume, for example, that T_1 represents a producer serving multiple consumers, whereof only one, represented by T_2, has to be considered. Thus several executions of T_1 can happen before an execution of T_2 is triggered.)
2. Each execution of T_2 must be immediately followed by an execution of T_3, and, conversely, each execution of T_3 must be immediately preceded by an execution of T_2 (solid arrow in Fig. 16(a)).

As in the previous sections, we represent the constraints characterizing this protocol as causal rules on ports of the three components. The corresponding system of causal rules is shown in Fig. 17(a) and the saturated one in Fig. 17(b).

The corresponding causal interaction tree is shown in Fig. 16(b) and the interconnected system in Fig. 16(c).

7 Conclusion

The paper provides a causal semantics for the algebra of connectors. This semantics leads to simpler and more intuitive representations which can be used for efficient implementation

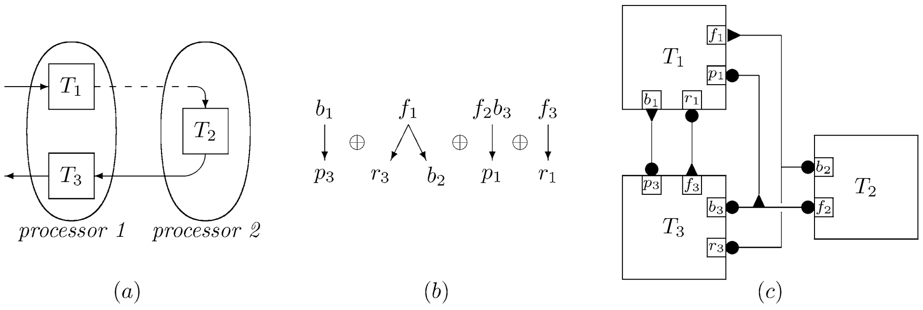


Fig. 16 An illustration (**a**), a causal interaction tree (**b**), and an interconnected system (**c**) for the example of three sequential tasks

$$\begin{array}{ll} \multicolumn{2}{l}{true \Rightarrow b_1 \vee f_1 \vee b_2 \vee f_2 \vee b_3 \vee f_3} \\ p_1 \Rightarrow b_3 & p_3 \Rightarrow b_1 \\ r_1 \Rightarrow f_3 & r_3 \Rightarrow f_1 \\ b_2 \Rightarrow f_1 & \\ f_2 \Rightarrow b_3 & b_3 \Rightarrow f_2 \\ & (a) \end{array} \qquad \begin{array}{ll} \multicolumn{2}{l}{true \Rightarrow b_1 \vee f_1 \vee f_2 b_3 \vee f_3} \\ p_1 \Rightarrow f_2 b_3 & p_3 \Rightarrow b_1 \\ r_1 \Rightarrow f_3 & r_3 \Rightarrow f_1 \\ b_2 \Rightarrow f_1 & \\ f_2 \Rightarrow b_3 & b_3 \Rightarrow f_2 \\ (b) & \end{array}$$

Fig. 17 Causal rules for the sequential execution of three tasks example

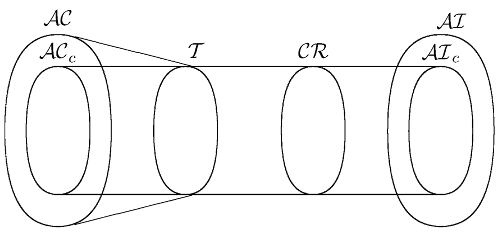


Fig. 18 A graphical representation of the relations between different algebras

of operations on connectors in BIP. Causal semantics allows a nice characterization of the set of causal connectors, which is isomorphic to the set of causal interaction trees. The set of causal connectors also corresponds to the set of causal interactions, which are closed under synchronization. The relation between the different algebras is shown in Fig. 18.

$\mathcal{T}(P)$ breaks with the reductionist view of interaction semantics as it distinguishes between symmetric and asymmetric interaction. It allows structuring global interactions as the parallel composition of chains of interactions. This is a very intuitive and alternate approach to interaction modeling especially for broadcast-based languages such as synchronous languages. Causal interaction trees are very close to structures used to represent dependencies between signals in synchronous languages, e.g., [24]. This opens new possibilities for unifying asynchronous and synchronous semantics.

$\mathcal{T}(P)$ is a basis for computing boolean representations for connectors, adequate for their symbolic manipulation and computation of boolean operations. These can be used for efficient implementations of component-based languages such as BIP. The examples provided in the previous section show that causal rules can be used for the specification of interactions from which connectors can be synthesized by using the algorithm given in Fig. 8.

Although several approaches to connector synthesis can be found in the literature (e.g., [2, 16, 19]), all of them are developed in a rather different context. Indeed, as discussed above, connectors in these settings can be viewed as special types of components and comprise all three layers among Behavior, Coordination, and Data transfer. In [16, 19], connectors are synthesized that enforce specific properties in rather restrained contexts. From this point of view our approach is rather novel, as it allows one to synthesize connectors in a very generic setting and respects the separation of concerns principle; it is similar to synthesis of circuits from boolean specifications. This last idea seems very interesting and deserves further investigation. In a new paper currently in preparation, we show how the coordination layer of BIP (connectors and priorities) can be used to synthesize controllers enforcing any present-state safety property.

Acknowledgements The authors would like to thank the anonymous reviewers for their comments that have greatly contributed to the improvement of the original draft.

References

1. Arbab F (2004) Reo: a channel-based coordination model for component composition. Math Struct Comput Sci 14(3):329–366
2. Arbab F, Meng S (2008) Synthesis of connectors from scenario-based interaction specifications. In: CBSE'08. LNCS, vol 5282. Springer, Berlin, pp 114–129
3. Balarin F, Watanabe Y, Hsieh H, Lavagno L, Passerone C, Sangiovanni-Vincentelli A (2003) Metropolis: an integrated electronic system design environment. IEEE Comput 36(4):45–52
4. Balasubramanian K, Gokhale A, Karsai G, Sztipanovits J, Neema S (2006) Developing applications using model-driven design environments. IEEE Comput 39(2):33–40
5. Basu A, Bozga M, Sifakis J (2006) Modeling heterogeneous real-time components in BIP. In: 4th IEEE international conference on software engineering and formal methods (SEFM06), September 2006, pp 3–12. Invited talk
6. Benveniste A, Guernic PL, Jacquemot C (1991) Synchronous programming with events and relations: the SIGNAL language and its semantics. Sci Comput Program 16(2):103–149
7. Bernardo M, Ciancarini P, Donatiello L (2000) On the formalization of architectural types with process algebras. In: SIGSOFT FSE, pp 140–148
8. BIP. http://www-verimag.imag.fr/~async/BIP/bip.html
9. Bliudze S, Sifakis J (2007) The algebra of connectors—structuring interaction in BIP. In: Proceedings of the EMSOFT'07, pp 11–20. ACM SigBED, October 2007
10. Bliudze S, Sifakis J (2008) The algebra of connectors—structuring interaction in BIP. IEEE Trans Comput 57(10):1315–1330
11. Bliudze S, Sifakis J (2008) A notion of glue expressiveness for component-based systems. In: van Breugel F, Chechik M (eds) CONCUR 2008. LNCS, vol 5201. Springer, Berlin, pp 508–522
12. Bruni R, Lanese I, Montanari U (2006) A basic algebra of stateless connectors. Theor Comput Sci 366(1):98–120
13. Darondeau P, Degano P (1989) Causal trees. In: Ausiello G, Dezani-Ciancaglini M, Rocca SRD (eds) ICALP. LNCS, vol 372. Springer, Berlin, pp 234–248
14. Eker J, Janneck J, Lee E, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S, Xiong Y (2003) Taming heterogeneity: the Ptolemy approach. Proc IEEE 91(1):127–144
15. Fiadeiro JL (2004) Categories for software engineering. Springer, Berlin
16. Galik O, Bureš T (2005) Generating connectors for heterogeneous deployment. In: SEM'05. ACM, New York, pp 54–61
17. Halbwachs N, Caspi P, Raymond P, Pilaud D (1991) The synchronous dataflow programming language LUSTRE. Proc IEEE 79:1305–1320
18. Hoare CAR (1985) Communicating sequential processes. Prentice Hall international series in computer science. Prentice Hall, New York
19. Inverardi P, Tivoli M (2001) Automatic synthesis of deadlock free connectors for com/dcom applications. In: ACM proceedings of the joint 8th ESEC and 9th FSE, Vienna, September 2001. ACM, New York
20. Maggiolo-Schettini A, Peron A, Tini S (2003) A comparison of Statecharts step semantics. Theor Comput Sci 290(1):465–498
21. Manna Z, Pnueli A (1992) The temporal logic of reactive and concurrent systems: specification, vol 1. Springer, New York
22. Maraninchi F, Rémond Y (2001) Argos: an automaton-based synchronous language. Comput Lang 27:61–92
23. Milner R (1989) Communication and concurrency. Prentice Hall international series in computer science. Prentice Hall, New York
24. Nowak D (2006) Synchronous structures. Inf Comput 204(8):1295–1324
25. Ray A, Cleaveland R (2003) Architectural interaction diagrams: AIDs for system modeling. In: ICSE'03: proceedings of the 25th international conference on software engineering. IEEE Computer Society, Washington, pp 396–406
26. Sifakis J (2005) A framework for component-based construction. In: 3rd IEEE international conference on software engineering and formal methods (SEFM05), September 2005, pp 293–300. Keynote talk
27. Spitznagel B, Garlan D (2003) A compositional formalization of connector wrappers. In: ICSE. IEEE Computer Society, Los Alamitos, pp 374–384
28. Stefănescu G (2000) Network algebra. Springer, New York