Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 366 (2006) 98–120
Theoretical Computer Science
www.elsevier.com/locate/tcs

A basic algebra of stateless connectors ☆

Roberto Bruni[a,*], Ivan Lanese[b], Ugo Montanari[a]

[a] *Dipartimento di Informatica, Università di Pisa, Italy*
[b] *Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy*

Abstract

The conceptual separation between computation and coordination in distributed computing systems motivates the use of peculiar entities commonly called *connectors*, whose task is managing the interaction among distributed components. Different kinds of connectors exist in the literature at different levels of abstraction. We focus on an algebra of connectors that exploits five kinds of basic connectors (plus their duals), namely symmetry, synchronization, mutual exclusion, hiding and inaction. Basic connectors can be composed in series and in parallel. We first define the operational, observational and denotational semantics of connectors, then we show that the observational and denotational semantics coincide and finally we give a complete normal-form axiomatization. The expressiveness of the framework is witnessed by the ability to model all the (stateless) connectors of the architectural design language CommUnity and of the coordination language Reo.

Keywords: Connectors; Synchronization; Axiomatization; Normal form

1. Introduction

The advent of modern communication technologies shifted the focus of computer science researchers from isolated computing systems to massively distributed communicating systems, in which *interaction* plays the prominent role when developing large and complex applications. In Milner's words [28], "computing has grown into informatics and Turing's logical computing machines are matched by a logic of interaction". In this perspective, the analysis of global computing systems is facilitated by approaches, techniques and paradigms that follow the spirit of component-based software development and that exploit a clean conceptual separation between *computation* and *coordination*. This is much evident at several levels of abstraction (architecture, software, processes), where issues like reusability, maintenance and heterogeneity call for modular specifications, theories and models.

When separating coordination from computation, the notion of a *connector* has emerged in different contexts, with slightly different meanings, expressiveness and functionalities. On the one hand, we have separately developed components, that can be seen as offering certain small functionalities and services, out of which larger and complete applications can be designed and deployed. On the other hand, we must synthesize the appropriate *glue code*, necessary

☆ Research supported by the FET-GC II Project Sensoria.
* Corresponding author. Tel.: +39 050 2212785; fax: +39 050 2212726.
E-mail addresses: bruni@di.unipi.it (R. Bruni), lanese@cs.unibo.it (I. Lanese), ugo@di.unipi.it (U. Montanari).
URLs: http://www.di.unipi.it/∼bruni (R. Bruni), http://www.cs.unibo.it/∼lanese (I. Lanese), http://www.di.unipi.it/∼ugo (U. Montanari).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.005

for components to connect, interact and interoperate. Thus, connectors must take care of all those aspects that lie outside of the scopes of individual components. Having rigorous mathematical foundations is crucial for the analysis of coordinated distributed systems. Several different kinds of connectors have been studied in the literature, relying e.g. on the study of observational semantics of process contexts [18,19,23,24,31], or on the analysis of suitable equational theories and reduction to normal forms [6,11,22,32]. The common trait is the role of a connector: a (software, architectural, process) component that mediates the interaction of other computational components and connectors. In particular, connectors have been studied within both *algebraic* and *categorical approaches*, two important frameworks for system modeling.

The algebraic approach [21,27] models systems as terms in a suitable algebra, with constants modeling basic components that can be composed through the other operators, e.g., parallel composition. Operational and abstract semantics are then usually based on inductively defined labeled transition systems.

The categorical approach [20] models systems as objects in a category, with morphisms defining relations such as subsystem or refinement. Complex software architectures can be modeled as diagrams in the category, with universal constructions, such as colimit, building an object in the same category that behaves as the whole system and that is uniquely determined up to isomorphisms. The case of architectural connectors is well exemplified by CommUnity [15,16].

In this paper, we concentrate on the algebraic approach by promoting a small algebra of connectors, for which we define suitable operational, observational and denotational semantics. The operational semantics is expressed using the Tile Model [18]. The observational semantics we select is tile bisimilarity (see Definition 4), which also coincides with tile trace equivalence for the special case of the algebra under inspection. The denotational semantics is based on (an algebra of) suitable boolean matrices, called tick-tables. We first show that the observational and denotational semantics coincide and then give a complete normal-form axiomatization for them, which is the main result of the paper.

Our connectors are rather simple: they essentially model basic synchronization, mutual exclusion and hiding and they are all stateless. Nevertheless, we think that the analysis of these connectors is quite interesting, since they allow to build a wide range of coordination connectors. For instance, they are expressive enough to model the multiple-action synchronization mechanism of CommUnity which uses morphisms and complex architectural connectors. More details about the modeling of CommUnity can be found in a joint work with Fiadeiro and Lopes [4], where we have defined an encoding from CommUnity into the Tile Model. One of the main results of [4] is that the translation of a diagram is tile bisimilar to the translation of its colimit. The main result of this paper, i.e., the complete axiomatization of abstract semantics, improves the work in [4] by showing that, for the part of action coordination, tile bisimilarity can be axiomatized as a suitable equational theory, where equivalence classes have standard representatives. While in the algebraic approach equivalence classes are usually abstract entities, having a normal form gives a concrete representation that matches a nice feature of the categorical approach, namely that the colimit of a diagram is its best concrete representative. The research initiated in [4] and extended in this paper represents a first step towards a more general reconciliation between the categorical and the algebraic approach, of which CommUnity and the Tile Model are just two selected representatives.

A second example addresses the modeling of Reo [1], an emerging paradigm in the field of coordination that comes equipped with a powerful mechanism for constructing connectors based on channel composition. A wide range of examples illustrating the expressive power of Reo can be found, e.g., in [1,2]. In this case, the modeling shows that our algebra of connectors can be easily extended by adding some typing information to the interfaces of connectors and by considering an arbitrary alphabet of actions, instead of a boolean (absence/presence of action) set. Interestingly, the axioms identified for the simpler case are still sufficient to deal with the more general setting.

With respect to other approaches to the modeling of synchronization connectors existing in the literature [8,11,22,32], our main contribution is the introduction of the mutual exclusion connector, which allows to specify a wider range of possible synchronization policies. In this sense, the mutual exclusion connector can be seen as a non-trivial extension of [5,6]. Furthermore, the semantics based on matrices is new and it provides a concrete mathematical definition of the behavior of connectors. Finally, we also provide a characterization of the classes of matrices that can be specified, both with and without mutual exclusion.

This paper is the full version of [7], which is extended here with all proofs of main results. The case study of Reo is also original to this contribution.

1.1. Structure of the paper

Section 2 contains some background on the symmetric monoidal structure of connectors and on the Tile Model, although we assume the reader has some familiarity with the basics of category theory. Section 3 presents syntax and semantics of our connectors, showing the correspondence between the observational and denotational semantics. Section 4 contains the main results of the paper, namely the axiomatization of connectors and the theorems for semantic equivalence and normal form: we consider the case without mutual exclusion first (Section 4.1) and the general case later (Section 4.2). Sections 5 and 6 are devoted to two important case studies, namely the application of our algebra of connectors to the modeling of: (i) architectural connectors of CommUnity and (ii) coordination connectors of Reo. Conclusion and future work are in Section 7. Some of the longer proofs are moved to Appendix A.

2. Background

2.1. Symmetric monoidal categories for connectors

It has been highlighted in the literature that distributed systems can be conveniently modeled as graphs [13,14,29] that straightforwardly account for the network distribution of processes, mobile agents, etc. The advantage of using (freely generated) symmetric monoidal categories for representing configuration graphs is three-fold. First, it introduces a suitable notion of (observable) interfaces for configurations. Second, it introduces two key operations for composing graphs, namely sequential and parallel compositions. Third, the natural isomorphism defined by symmetries allows to take graphs up to interface-preserving graph isomorphisms.

We recall that a (*strict*) *monoidal category* [25] $(\mathcal{C}, \otimes, e)$ is a category $\mathcal{C}$ together with a functor $\otimes\colon \mathcal{C}\times\mathcal{C} \to \mathcal{C}$ called the *tensor product* and an object e called the *unit*, such that for any arrows $\alpha_1, \alpha_2, \alpha_3 \in \mathcal{C}$ we have $(\alpha_1 \otimes \alpha_2) \otimes \alpha_3 = \alpha_1 \otimes (\alpha_2 \otimes \alpha_3)$ and $\alpha_1 \otimes id_e = \alpha_1 = id_e \otimes \alpha_1$. The tensor product has higher precedence than the categorical composition ;. Note that we focus only on "strict" monoidal categories, where the monoidal axioms hold as equalities and not just up to natural isomorphisms. By functoriality of $\otimes$ we have, e.g., $\alpha_1 \otimes \alpha_2 = \alpha_1 \otimes id_{a_2}; id_{b_1} \otimes \alpha_2 = id_{a_1} \otimes \alpha_2; \alpha_1 \otimes id_{b_2}$ for any $\alpha_i\colon a_i \to b_i, i \in \{1, 2\}$.

Definition 1. A *symmetric* (*strict*) *monoidal category* $(\mathcal{C}, \otimes, e, \gamma)$ is a (strict) monoidal category $(\mathcal{C}, \otimes, e)$ together with a family $\{\gamma_{a,b}\colon a \otimes b \to b \otimes a\}_{a,b}$ of arrows, called *symmetries*, indexed by pairs of objects in $\mathcal{C}$ such that for any two arrows $\alpha_1, \alpha_2 \in \mathcal{C}$ with $\alpha_i\colon a_i \to b_i$, we have $\alpha_1 \otimes \alpha_2; \gamma_{b_1,b_2} = \gamma_{a_1,a_2}; \alpha_2 \otimes \alpha_1$ (that is, γ is a natural isomorphism) that satisfies the coherence equalities (for any objects a, b, c):

$$\gamma_{a,b}; \gamma_{b,a} = id_{a\otimes b} \quad \gamma_{a\otimes b,c} = id_a \otimes \gamma_{b,c}; \gamma_{a,c} \otimes id_b.$$

The categories we are interested in are those freely generated from an unsorted (hyper)signature Σ, i.e., from a ranked family of operators $f\colon n \to m$. The objects are just natural numbers expressing the arities of the interfaces, i.e., the number of "attach points", with $n \otimes m = n + m$ and $e = 0$. The operators $\sigma \in \Sigma$ are seen as basic arrows with source and target defined accordingly to the arity of σ. Symmetries can always be expressed in terms of the basic symmetry $\gamma_{1,1}\colon 2 \to 2$. Intuitively, symmetries can be used to rearrange the input–output interfaces of graph-like configurations. We call *permutation* any composition of identities and symmetries. A generic arrow can always be expressed as a suitable composition of id_1, $\gamma_{1,1}$ and $\sigma \in \Sigma$.

Lemma 2. *Any arrow α can be decomposed as $id_{n_1} \otimes \sigma_1 \otimes id_{m_1}; \ldots; id_{n_k} \otimes \sigma_k \otimes id_{m_k}$ for some natural numbers $k, n_1, \ldots, n_k, m_1, \ldots, m_k$ and $\sigma_1, \ldots, \sigma_k \in \{\gamma_{1,1}\} \cup \Sigma$.*

A term written using only identities and (possibly multiple instances of) one particular $\sigma \in \{\gamma_{1,1}\} \cup \Sigma$ is called a *layer of σs*.

2.2. Tile model

In this paper, we choose the Tile Model for defining the operational and observational semantics of connectors. In fact, tile configurations are particularly suitable to represent the above concept of connector, which includes input

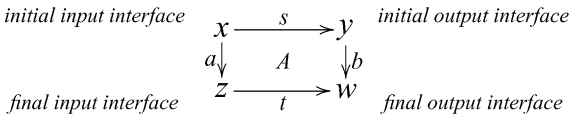

Fig. 1. Graphical representation of a tile A.

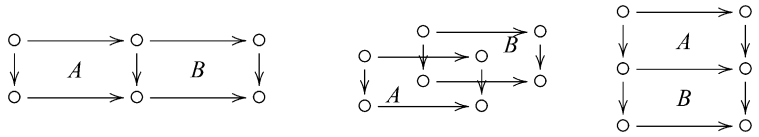

Fig. 2. Horizontal, parallel and vertical tile compositions.

$$\frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[c]{b} f}{s;h \xrightarrow[c]{a} t;f}\ (\texttt{hor}) \qquad \frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[d]{c} f}{s \otimes h \xrightarrow[b\otimes d]{a\otimes c} t \otimes f}\ (\texttt{par}) \qquad \frac{s \xrightarrow[b]{a} t \quad t \xrightarrow[d]{c} h}{s \xrightarrow[b;d]{a;c} h}\ (\texttt{ver})$$

Fig. 3. Inference rules for tile logic.

and output interfaces where actions can be observed and that can be used to compose configurations and also to coordinate their local behaviors.

The Tile Model [18] is a rule-based framework whose main ingredients are rewrite rules with side effects, called *basic tiles* that combine inspirations from SOS rules, *context systems* [24], *structured transition systems* [12] and *rewriting logic* [26].

A tile $A : s \xrightarrow[b]{a} t$ is a rewrite rule stating that the *initial configuration* s can evolve to the *final configuration* t via A, producing the *effect* b; but the step is allowed only if the 'arguments' of s can contribute by producing a, which acts as the *trigger* of A (see Fig. 1). Triggers and effects are called *observations* and tile vertices are called *interfaces*.

Tiles can be composed horizontally, in parallel, or vertically to generate larger steps (see Fig. 2). Horizontal composition $A; B$ coordinates the evolution of the initial configuration of A with that of B, yielding the 'synchronization' of the two rewrites. Horizontal composition is possible only if the initial configurations of A and B interact cooperatively: the effect of A must provide the trigger for B. The parallel composition $A \otimes B$ builds concurrent steps. Vertical composition $A * B$ is sequential composition of computations.

The operational semantics of concurrent systems can be expressed via tiles if system configurations form a monoidal category $\mathcal{H}$, and observations form a monoidal category $\mathcal{V}$ with the same underlying set of objects as $\mathcal{H}$. Abusing the notation, we denote by $\_ \otimes \_$ both monoidal functors of $\mathcal{H}$ and $\mathcal{V}$ and by $\_;\_$ both sequential compositions in $\mathcal{H}$ and $\mathcal{V}$.

Definition 3. A *tile system* is a tuple $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ where $\mathcal{H}$ and $\mathcal{V}$ are monoidal categories with the same set of objects $O_{\mathcal{H}} = O_{\mathcal{V}}$, N is the set of rule names and $R\colon N \to \mathcal{H} \times \mathcal{V} \times \mathcal{V} \times \mathcal{H}$ is a function such that for all $A \in N$, if $R(A) = \langle s, a, b, t\rangle$, then the arrows s, a, b, t can form a tile like in Fig. 1.

Like rewrite rules in rewriting logic, tiles can be seen as sequents of *tile logic*: the sequent $s \xrightarrow[b]{a} t$ is *entailed* by the tile logic associated with $\mathcal{R}$, written $\mathcal{R} \vdash s \xrightarrow[b]{a} t$, if it can be obtained by composing horizontally, in parallel and vertically some basic tiles in R (possibly also using auxiliary tiles, like identities $id \xrightarrow[a]{a} id$ propagating observations). The "borders" of composed sequents are defined in Fig. 3.

The main feature of tiles is their double labeling with triggers and effects, allowing to observe the input–output behavior of configurations. By taking ⟨trigger, effect⟩ pairs as labels one can see tiles as a labeled transition system. In this context, the usual notion of bisimilarity is called *tile bisimilarity*.

Definition 4. Let $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ be a tile system. A symmetric relation $\sim_t$ on configurations is called a *tile bisimulation* if whenever $s \sim_t t$ and $\mathcal{R} \vdash s \xrightarrow[b]{a} s'$, then t' exists such that $\mathcal{R} \vdash t \xrightarrow[b]{a} t'$ and $s' \sim_t t'$.

The maximal tile bisimulation is called *tile bisimilarity* and it is denoted by $\simeq_t$. Note that $s \simeq_t t$ only if s and t have the same input–output interfaces.

Given an equation between tile configurations we say that the equation "bisimulates" iff for all the instances $t_1 = t_2$ of the equation $t_1 \simeq_t t_2$ holds.

We shall focus on tile systems of *stateless* connectors, meaning that in all basic tiles the final configuration is equal to the initial one. Operationally, this means that the behavior of a connector is history independent. An easy consequence is that $\simeq_t$ coincides with tile trace equivalence.

3. Algebra of connectors

We present here a rich algebra of connectors for action coordination. We have developed such an algebra aiming to model systems where multiple actions can be executed at each time, either independently or synchronized. Connectors are used to ensure that the distributed components behave properly, i.e. to guarantee the global consistency of local evolutions. For instance, in the translation of CommUnity, the basic connectors are used in conjunction with other operators representing the computational entities. Roughly these have n attach points associated with actions and according to the computed action they emit 1 tick (action performed) and $n - 1$ unticks (forced inactivity) on their interfaces.

We remark that all structures that we are going to present are based on the symmetric strict monoidal structure given by symmetries γ, tensor product $\otimes$ and unit 0, for which the ordinary coherence, naturality and functoriality axioms hold.

The complete list of connectors is in Fig. 4. The ordinary basic connectors are in the leftmost part of the table, while their duals are on the right (symmetry is self-dual). The term mex stands for "mutual exclusion". We also speak about synch connectors (∇ and Δ), choice connectors ($\overline{\nabla}$ and $\overline{\Delta}$), hiding connectors (! and i) and inaction connectors ($\mathbf{0}$ and $\overline{\mathbf{0}}$).

We now define the tile semantics for our connectors. As usual for the Tile Model, we first define the categories of configurations and of observations (sharing the same set of objects) and then we give the basic tiles.

As explained in Section 2, the objects of our categories are natural numbers.

The horizontal category of configurations is the free symmetric strict monoidal category generated by the basic connectors. The basic connector γ is the symmetry $\gamma_{1,1}$. We call *connector* any arrow in the horizontal category. Given a connector $\alpha\colon n \to m$ we denote by $\alpha^c\colon m \to n$ its dual, defined in the obvious way for basic connectors (see Fig. 4) and then inductively by $(\alpha; \beta)^c = \beta^c; \alpha^c$ and $(\alpha \otimes \beta)^c = \alpha^c \otimes \beta^c$.

The vertical category is the free monoidal category generated by the arrows $\mathsf{tick} : 1 \to 1$ and $\mathsf{untick} : 1 \to 1$.

The tiles defining the semantics of ordinary connectors are in Fig. 5. The first rule specifies that a symmetry can accept any input pair which is swapped in the output. Then there are the two rules for the duplicator, where the constraint is that all the actions must coincide. The last rule in the first row defines the only allowed behavior for zero, which allows just untick on its interface. Rules in the second row specify the behavior of bang, which hides any action on its interface, and mex: if the trigger is untick, then the effects are two unticks, otherwise the trigger tick is propagated to exactly one effect.

Dual connectors have symmetric tiles. For instance, the tiles for Δ are

$$\Delta \xrightarrow[\mathsf{tick}]{\mathsf{tick}\otimes\mathsf{tick}} \Delta \qquad \Delta \xrightarrow[\mathsf{untick}]{\mathsf{untick}\otimes\mathsf{untick}} \Delta.$$

From the tile system we can derive an observational semantics using tile bisimilarity. This semantics is compositional, as proved by the following theorem.

Theorem 5. *In all the tile systems built using only the above tiles for connectors,* $\simeq_t$ *is a congruence* (*w.r.t. parallel and sequential composition*).

Ordinary structure			Dual structure		
name	symbolic	graphical	name	symbolic	graphical
symmetry	$\gamma\colon 2 \to 2$		**symmetry**	$\gamma\colon 2 \to 2$	
duplicator	$\nabla\colon 1 \to 2$		**coduplicator**	$\Delta\colon 2 \to 1$	
bang	$!\colon 1 \to 0$		**cobang**	$¡\colon 0 \to 1$	
mex	$\nabla\colon 1 \to 2$		**comex**	$\Delta\colon 2 \to 1$	
zero	$\mathbf{0}\colon 1 \to 0$		**cozero**	$\overline{\mathbf{0}}\colon 0 \to 1$	

Fig. 4. Syntax of basic connectors.

$$\gamma \xrightarrow[y\otimes x]{x\otimes y} \gamma \text{ with } x,y \in \{\mathsf{tick},\mathsf{untick}\} \qquad \nabla \xrightarrow[\mathsf{tick}\otimes\mathsf{tick}]{\mathsf{tick}} \nabla \qquad \nabla \xrightarrow[\mathsf{untick}\otimes\mathsf{untick}]{\mathsf{untick}} \nabla \qquad \mathbf{0} \xrightarrow[id_0]{\mathsf{untick}} \mathbf{0}$$

$$! \xrightarrow[id_0]{\mathsf{tick}} ! \qquad ! \xrightarrow[id_0]{\mathsf{untick}} ! \qquad \nabla \xrightarrow[\mathsf{untick}\otimes\mathsf{untick}]{\mathsf{untick}} \nabla \qquad \nabla \xrightarrow[\mathsf{tick}\otimes\mathsf{untick}]{\mathsf{tick}} \nabla \qquad \nabla \xrightarrow[\mathsf{untick}\otimes\mathsf{tick}]{\mathsf{tick}} \nabla$$

Fig. 5. Tiles for ordinary connectors.

Proof. For this proof we need some results about tile systems from the literature. The *basic source property* requires that $\mathcal{H}$ is generated from a (hyper)signature Σ and that the initial configuration of each basic tile consists of a basic operator in Σ. A result in [18] guarantees that whenever a tile system enjoys the basic source property then tile bisimilarity is a congruence. This happens in our case. □

It is worth noting that the coherence and naturality axioms for symmetry (see Definition 1) bisimulate, and thus the left hand side and the right hand side of each axiom are equated by the observational semantics.

The coordination policy of a connector $\alpha\colon n \to m$ can be represented as a $2^n \times 2^m$ tick-table whose cells contain boolean values. Each row represents a combination of tick/untick values (denoted as 1 or 0 in the tick-tables) for the n inputs, while each column represents a combination of tick/untick values for the m outputs. If a cell is true (i.e., marked), then the corresponding combination of inputs and outputs is admissible, otherwise (the cell is false, i.e., empty, unmarked) the corresponding combination of inputs and outputs is forbidden. The list of tick-tables for basic connectors is in Fig. 6.

We denote with $T(\alpha)$ the tick-table associated to connector α. Furthermore, given a position $[i, j]$ in a tick-table T we denote with $d_T([i, j])$ its *domain*, that is the set of elements in its input and output interfaces on which tick actions are performed.

A connector $\alpha\colon n \to m$ can be seen as a hypergraph where basic connectors are edges and elements of interfaces are nodes. Each edge constrains the behavior of the attached nodes. The solution of the network of constraints S associated with a connector $\alpha\colon n \to m$ is the set of consistent assignments of tick/untick values to all the nodes appearing in the graph denoted by α in such a way that a corresponding "tiling" can be found. Note that such an assignment concerns not only the $n + m$ nodes in the external interfaces of α, but also all the internal nodes of the network, not observable from the outside. However, this semantics is too *concrete* when one is not interested in knowing the way in which all constraints of the network are satisfied. A more abstract semantics of α is the solution of the network, where all the internal nodes (i.e., neither inputs nor outputs) have been existentially quantified, that is the projection of the concrete semantics on the interfaces. Thus tick-tables can be seen as the *denotational semantics* of connectors.

The next lemma shows the operations on tick-tables that correspond to sequential and parallel composition of connectors (see Appendix A for the proof).

Lemma 6. *For any two connectors* $\alpha\colon n \to h$, $\beta\colon h \to m$, *the* tick*-table* $T(\alpha;\beta)$ *is the product matrix* $T(\alpha) \times T(\beta)$, *i.e.*, $T(\alpha;\beta)[i, j] = \bigvee_k (T(\alpha)[i, k] \wedge T(\beta)[k, j])$. *For any two connectors* $\alpha\colon n \to h$, $\beta\colon l \to m$, *the* tick*-table* $T(\alpha \otimes \beta)$

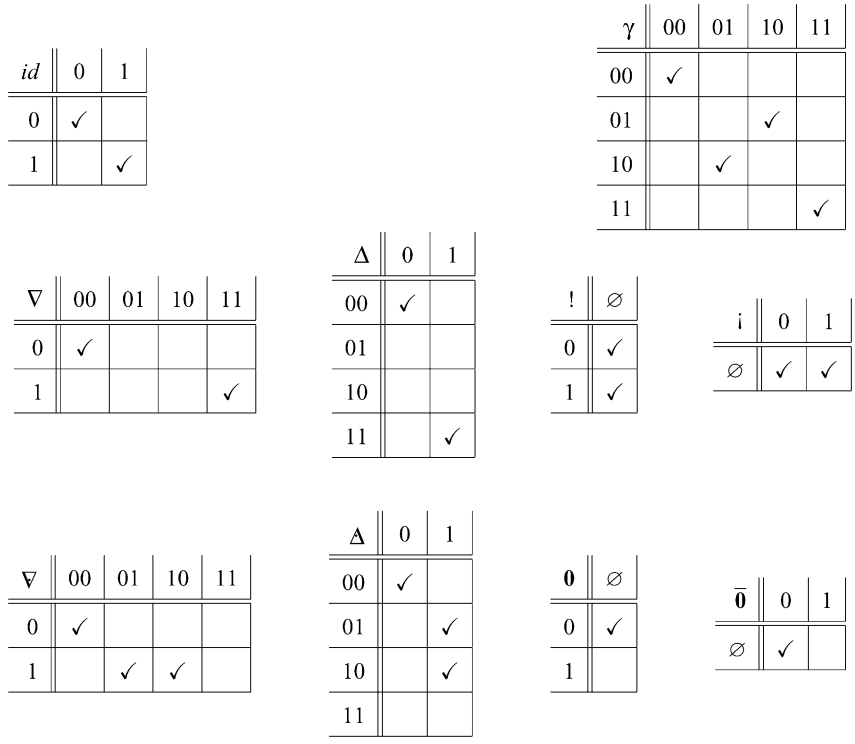

Fig. 6. Denotational semantics.

is obtained by refining each marked entry of $T(\beta)$ *by a copy of* $T(\alpha)$, *and each unmarked entry of* $T(\beta)$ *by the empty table with the same dimension as* $T(\alpha)$. *Moreover, for any connector* α, *the* tick*-table* $T(\alpha^c)$ *is the transposition of* $T(\alpha)$.

The denotational semantics of connectors given by tick-tables agrees with the observational semantics defined by tile bisimilarity, that is two connectors are tile bisimilar iff they have the same associated tick-table.

Theorem 7. *For each pair of connectors* α *and* β, $\alpha \simeq_t \beta$ *iff* $T(\alpha) = T(\beta)$.

Proof. Since all connectors are stateless and each tile can be decomposed as a vertical composition of horizontal compositions of basic tiles, then two connectors are tile bisimilar iff their allowed combinations of tick and untick on the interfaces are equal, that is, iff they have the same tick-table. □

4. Normal form

We first show an axiomatization of connectors which is correct and complete w.r.t. their denotational semantics, and then we show an algorithm to derive a standard representative for each equivalence class, i.e., a normal form. From the categorical point of view, this corresponds to compute the colimit of a diagram (such as an architectural diagram in CommUnity). We perform the task incrementally, by studying different *classes of connectors*.

Definition 8 (*Classes of connectors*). Given a set of connectors S we denote with $\mathbf{CC}(S)$ the class of connectors generated by connectors in S. Symmetries are always included in $\mathbf{CC}(S)$, even when $S = \emptyset$.

Several axioms for connectors have been proposed, studied and applied in the literature, see e.g. [3,6,8,11,32]. Axioms over connectors are usually aimed at characterizing a suitable category of *links between objects* as the equational term

algebra freely generated from a restricted set of basic connectors, like those in Fig. 4. Usually the axioms have just to consider the few possible ways in which two or three basic connectors can be composed together. However, our algebra is very rich and therefore a few more complex patterns need to be considered.

The consistency of all the axioms we present w.r.t. the denotational semantics can be verified by looking at the tables associated to each term. More precisely, for each axiom $\alpha = \beta$ that we propose, it is easy to check that $T(\alpha) = T(\beta)$.

We start by introducing some terminology and notation that will be useful in the remainder of this paper and in the proofs.

Notation. As already mentioned, a connector can be seen as a hypergraph where basic connectors are edges and elements of interfaces are nodes. An edge is *adjacent* to any node in its interfaces. Two edges are *adjacent* if they share a node. A *path* in the graph is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that for each $i \in \{1, \ldots, k-1\}$ the node n_i is an element of the input interface of a basic connector and n_{i+1} is an element of the output interface of the same basic connector if the connector is not a symmetry. As suggested by their graphical representation, for symmetries the path can only enter in the first element of the input interface and exit from the second one in the output interface, or enter from the second one and exit from the first one. The *components* of a path are all its nodes and all the edges traversed. We say that two components of a graph are *linearly connected* iff there exists a path of which they are both components. The relation of *connectedness* is the transitive closure of the relation of linear connectedness.

We let ∇^n denote the "tree" of ∇ connectors with n leaves, inductively defined as $\nabla^0 =!$ and $\nabla^{n+1} = \nabla; id \otimes \nabla^n$. Note that $\nabla^1 = id$. We also define connectors for structured objects in terms of connectors defined for smaller objects

$$\begin{array}{ll} \nabla_0 = id_0 & \nabla_{n+1} = \nabla \otimes \nabla_n; id \otimes \gamma_{1,n} \otimes id_n \\ !_0 = id_0 & !_{n+1} = !\otimes !_n \end{array}$$

Note that $\nabla_1 = \nabla$ and $!_1 =!$. Similar notations are used for the other connectors.

4.1. Connectors for synchronization

First, we focus on the class of connectors $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$. The tick-tables associated to these connectors can be characterized as below.

Proposition 9. *Let $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$. Then $T(\alpha)$ satisfies the following properties.*

- $T(\alpha)[\vec{0}, \vec{0}] = ✓$;
- *suppose* $T(\alpha)[i_1, j_1] = ✓$ *and* $T(\alpha)[i_2, j_2] = ✓$;
  - *if* $d_{T(\alpha)}([i, j]) = d_{T(\alpha)}([i_1, j_1]) \cup d_{T(\alpha)}([i_2, j_2])$ *then* $T(\alpha)[i, j] = ✓$,
  - *if* $d_{T(\alpha)}([i, j]) = d_{T(\alpha)}([i_1, j_1]) \cap d_{T(\alpha)}([i_2, j_2])$ *then* $T(\alpha)[i, j] = ✓$,
  - *if* $d_{T(\alpha)}([i, j]) = d_{T(\alpha)}([i_1, j_1]) \backslash d_{T(\alpha)}([i_2, j_2])$ *then* $T(\alpha)[i, j] = ✓$,
  - *if* $d_{T(\alpha)}([i, j]) = \overline{d_{T(\alpha)}([i_1, j_1])}$ *then* $T(\alpha)[i, j] = ✓$.

Proposition 9 (see Appendix A for the proof) says that the cell with empty domain is always enabled and that table entries are closed under domain union, intersection, difference and complement. For example, it is an easy consequence that, for any $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$, $T(\alpha)[\vec{1}, \vec{1}] = ✓$. We call *synch-tables* the tables that satisfy these properties. As we will show in Theorem 18, any synch-table can be implemented by a connector in $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$.

Definition 10 (*Base*). Given a synch-table T, its base $b(T)$ is the set of the non-empty domains of its marked cells that are minimal w.r.t. set inclusion.

As an example, the base of the table $T(\gamma)$ contains just the domains of the two marked cells where only one element in each interface has value 1.

The synch-tables are uniquely identified by their bases.

Lemma 11. *Let T_1 and T_2 be any two synch-tables with the same dimension. Then $T_1 = T_2$ iff $b(T_1) = b(T_2)$.*

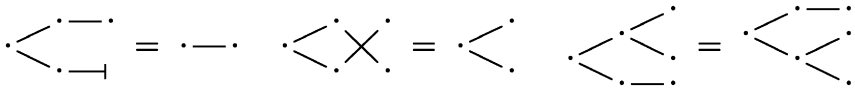

Fig. 7. Gs-monoidal axioms for ∇ and !.

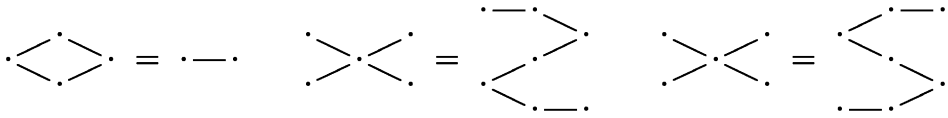

Fig. 8. Match-share axioms.

Proof. Marked cells are all the possible unions of elements of the base. Thus, tables with the same base have the same marked cells and thus are equal. □

Intuitively the previous lemma says that connectors built of synch and hiding connectors individuate equivalence classes on the elements of the interfaces, and that different equivalence classes act independently.

Analogous structures have been already studied in the literature [6,8,11,32]. If we inspect which equalities are satisfied among those in [6], then according to the terminology therein, we have a gs-monoidal structure $(\nabla, !)$, a cogs-monoidal structure $(\Delta, ¡)$, a match-share structure (∇, Δ) and a new-bang structure $(!, ¡)$. The whole structure is called a p-monoidal structure. Interestingly, the p-monoidal axioms characterize exactly tile bisimilarity and allow for normal-form reduction. This is explained below in detail.

As far as the gs-structure is concerned there are three axioms expressing the "associativity", "commutativity" and "unit" for the ∇ (with ! as "unit"). The quoted terminology can be easily understood by looking at their graphical representation in Fig. 7.

$$\nabla; id \otimes ! = id \quad \nabla; \gamma = \nabla \quad \nabla; \nabla \otimes id = \nabla; id \otimes \nabla$$

A cogs-monoidal structure is just a gs-monoidal structure in the dual category. Therefore the axioms are obtained by reversing the order of composition:

$$id \otimes ¡; \Delta = id \quad \gamma; \Delta = \Delta \quad \Delta \otimes id; \Delta = id \otimes \Delta; \Delta.$$

The axioms of match-share categories have been proposed in [6], where the free algebra of match-share connectors has been shown to model partition relations between non-empty source and target objects. There are three match-share axioms (see Fig. 8):

$$\nabla; \Delta = id \quad \Delta; \nabla = id \otimes \nabla; \Delta \otimes id \quad \Delta; \nabla = \nabla \otimes id; id \otimes \Delta.$$

The leftmost axiom essentially says that the multiplicity of connections between two objects is not important. The other two axioms (which are in fact equivalent) say that the path connecting two objects is not important.

The new-bang categories just contain the axiom $¡; ! = id_0$ which represents garbage-collection of isolated nodes.

We want to use the axioms to reduce any connector to a suitable normal form. We start by defining a *sorted form* that forces a standard order on connector layers.

Definition 12 (*Sorted form*). A connector $\alpha \in \mathbf{CC}(\nabla, \Delta, !, ¡)$ is in sorted form iff:

$$\alpha \equiv \alpha_¡ ; \alpha_\gamma; \alpha_\Delta ; \beta_\nabla; \beta_\gamma; \beta_!$$

where α_σ and β_σ are layers of σs and $\equiv$ is syntactic identity.

Proposition 13. *Any connector* $\alpha \in \mathbf{CC}(\nabla, \Delta, !, ¡)$ *can be transformed in sorted form using the axioms.*

Proof. The proof is by induction on the connector structure. The base case is trivial. For the inductive case, we have to prove that given a connector $\alpha\colon n \to m$ in sorted form, we can transform in sorted form any connector of the form $id_{n_1} \otimes \sigma \otimes id_{n_2}; \alpha$. We proceed by case analysis, according to the kind of σ.

Case i: We are already in sorted form.

Case γ: The connector can be moved to the layer α_γ by functoriality.

Case Δ: Either there is a sorted form or Δ has some permutations of layer α_γ on the left and we can apply naturality and functoriality to reach a sorted form.

Case ∇: We have to consider different cases according to what is attached to the two elements of the right interface of ∇.

If there are just ! and ∇ connectors then the connector is already in sorted form. If there is a Δ then we can apply the axiom $\nabla; \Delta = id$ if it is attached to both the elements of the interface and $\nabla \otimes id; id \otimes \Delta = \Delta; \nabla$ otherwise.

In the case of permutations the connector is already in sorted form if only !, identities and other permutations follow. If the permutation is followed by ∇ or Δ then we have to apply some axioms. If it is followed by ∇ then we can use naturality to move the permutations from layer α_γ to β_γ and get back to the previous cases. Otherwise we can suppose that the permutation is attached to just one interface of each connector (if not, it could be simplified using commutativity). Thus, we have a connector of the form $id \otimes \nabla; \gamma \otimes id$ and after some more permutations there is a Δ connector. Using naturality of permutations and functoriality we can always resort either to a sorted form (if there was no linear connection between ∇ and Δ) or to a redex for the axiom $\nabla \otimes id; id \otimes \Delta = \Delta; \nabla$. After having applied this axiom (one or more times) we can apply again naturality of permutations and functoriality to reach the sorted form.

Case !: The connector can be moved to the rightmost layer by functoriality. □

We now want to define for connectors a *normal form* which is strictly related to tick-tables. We first need an auxiliary definition.

Definition 14 (*Central point*). A *central point* is any element of interface shared by layers α_Δ and β_∇.

Definition 15 (*Normal form*). A connector $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ is in normal form iff:

(1) it is in sorted form;
(2) hiding connectors have central points as interface;
(3) each central point is linearly connected to at least an external interface.

Theorem 16. *Any connector* $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ *can be transformed into normal form using the axioms.*

Proof. Thanks to Proposition 13 any connector $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ can be transformed in sorted form. Thus the first condition can be satisfied.

The second condition may not hold only if there is a i adjacent to a Δ, or a ! adjacent to a ∇, or either a i or a ! adjacent to a permutation. We can apply $\mathsf{i} \otimes id; \Delta = id$ in the first case, the dual axiom in the second and the naturality of permutations in the last. Note that these transformations only delete connectors, thus the resulting connector is still in sorted form.

As far as the third condition is concerned, note that the only possibility is to have subconnectors of the form $\mathsf{i}; !$, which can be deleted by applying $\mathsf{i}; ! = id_0$. □

The next theorems state the correspondence between normal forms and synch-tables.

Theorem 17. *For each synch-table* T, *we can build a connector* $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ *in normal form such that* $T = T(\alpha)$. *Moreover, the construction is unique up to the axioms of symmetric monoidal categories and of associativity and commutativity of synch connectors.*

Proof. Given a synch-table T let us consider its base $b(T)$ (which completely characterizes the table). We build a connector α in the following way:

- we create a central point P_b for each element $b \in b(T)$;
- we build a tree Δ^n (resp. ∇^n) on the left (resp. right) of each central point P_b, where n is the number of elements in b that are in the left (resp. right) interface;

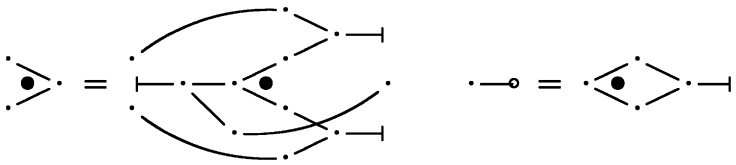

Fig. 9. Comex and zero as derived operators.

- we add permutations to connect the trees above of each central point P_b to the elements of the interfaces that correspond to elements of b.

We have to prove that $T(\alpha) = T$. Note that α has one connected component for each element in the base. Thus, for each of them we have a concrete semantics where we have a tick on all the elements of the interfaces in its domain and untick on the other ones. This is a correct assignment, thus the corresponding cell is marked as required. We now need to prove that these are the only elements in the base. Suppose by contradiction that there is another marked cell whose domain d is minimal. Let us choose a variable in d. Since each variable is linearly connected to at least a central point P, let us consider the domain m that corresponds to that central point. Since all the variables in m are connected, then they must all be in d. Thus $m \subseteq d$, which is absurd.

As far as uniqueness is concerned, note that the above construction is deterministic up to the allowed axioms. □

Theorem 18. *There is a bijective correspondence between synch-tables and connectors in* $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ *up to the axioms.*

Proof. In the proof of Theorem 17 we show how to construct the connector associated to a given synch-table and that the construction is deterministic up to the axioms.

The construction defines a function from synch-tables to equivalence classes of connectors. This function is injective since axioms are correct w.r.t. the denotational semantics. We have to prove that it is also surjective. For any connector α (that we can assume in normal form, thanks to Theorem 16), the construction maps $T(\alpha)$ to a connector α' with the same central points. Thus α and α' are equivalent, and the class of α is in the image as required. □

4.2. Adding the mutual exclusion connector

As we have already seen, connectors in $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ allow to specify only a small class of tick-tables. In particular, we can express synchronization constraints but not mutual exclusion ones. This is proved by the fact that the class $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i})$ has limited expressiveness (in particular any synch-table is uniquely determined by its base). For instance, it is not expressive enough to model all CommUnity connectors. In order to solve that problem we add the mutual exclusion connector $\overline{\nabla} : 1 \to 2$.

Following the analogy with Section 4.1, one may think that also the dual connector $\overline{\Delta}$ must be explicitly introduced, but this is not required since the complex term $id \otimes (\mathsf{i}; \nabla) \otimes id; id \otimes \overline{\nabla} \otimes id_2; id_2 \otimes \gamma \otimes id; \Delta \otimes id \otimes \Delta; ! \otimes id \otimes !$ (see Fig. 9) exhibits the same behavior as $\overline{\Delta}$. Similarly both inaction connectors $\mathbf{0}$ and $\overline{\mathbf{0}}$ can be derived as auxiliary connectors. In fact we have for instance $T(\mathbf{0}) = T(\overline{\nabla}; \Delta; !)$ (see Fig. 9).

One may start considering just the axiomatization of choice and inaction connectors, separately w.r.t. synch and hiding connectors. Thus, one individuates a gs-monoidal structure $(\overline{\nabla}, \mathbf{0})$, a cogs-monoidal structure $(\overline{\Delta}, \overline{\mathbf{0}})$ and a new-bang structure $(\mathbf{0}, \overline{\mathbf{0}})$. Unfortunately no simple axiomatization can be found for $(\overline{\nabla}, \overline{\Delta})$, since they form neither a match-share category because $T(\overline{\Delta}; \overline{\nabla}) \neq T(\overline{\nabla} \otimes id; id \otimes \overline{\Delta})$ nor an r-monoidal [6] category because $T(\overline{\Delta}; \overline{\nabla}) \neq T(\overline{\nabla} \otimes \overline{\nabla}; id \otimes \gamma \otimes id; \overline{\Delta} \otimes \overline{\Delta})$.

Thus, we resort to a complex axiomatization that deals with all the four classes of connectors at the same time. The axioms are textually written in Fig. 10. For simplicity, dual axioms are omitted. Axioms 1–7 are quite simple and their graphical representations are in Fig. 11. The other ones, which are more complex, are in Fig. 12 and commented below. The last one, which is actually an axiom scheme, is drawn only for $n = 3$. Axioms 8 and 9 deal with commutation of ∇ and $\overline{\nabla}$. The main trick here is that we get again the input value by merging using Δ copies of the two outputs of $\overline{\nabla}$. Axiom 11 deals with commutation of $\overline{\Delta}$ and $\overline{\nabla}$, but w.r.t. the conceptually similar axiom 3, we have to force mutual exclusion on all the paths. Axiom 11 shows that mutual exclusion on three actions can be enforced by imposing mutual

$$\nabla;\Delta = id \tag{1}$$
$$\nabla;\Delta = \mathbf{0};\bar{\mathbf{0}} \tag{2}$$
$$\Delta;\nabla = \nabla_2;\Delta\otimes\Delta \tag{3}$$
$$\Delta;\mathbf{0} = \mathbf{0}\otimes\mathbf{0} \tag{4}$$
$$\nabla;id\otimes\mathbf{0} = \mathbf{0};\bar{\mathbf{0}} \tag{5}$$
$$\Delta;\mathbf{0} = \mathbf{0}\otimes\mathbf{0} \tag{6}$$
$$\nabla;!_2 = ! \tag{7}$$
$$\nabla;\nabla\otimes id = \nabla;\nabla\otimes\nabla;id\otimes\Delta\otimes id;id\otimes\gamma \tag{8}$$
$$\nabla;\nabla\otimes id = \nabla;\nabla\otimes\nabla;id\otimes\Delta\otimes id;id\otimes\gamma \tag{9}$$
$$\Delta;\nabla = \nabla_2;\nabla\otimes\nabla\otimes\nabla\otimes\nabla;id\otimes\Delta\otimes(\Delta;!)\otimes\Delta\otimes id;\gamma\otimes\gamma;id\otimes(\Delta;!)\otimes id \tag{10}$$
$$\mathfrak{i};\nabla;\nabla\otimes id = \mathfrak{i}_3;\nabla\otimes\nabla\otimes\nabla;id\otimes\gamma\otimes\gamma\otimes id;\Delta\otimes\Delta\otimes\Delta \tag{11}$$
$$id_2 = \nabla\otimes\nabla;id\otimes\Delta\otimes id;id\otimes\nabla\otimes id;\Delta\otimes\Delta \tag{12}$$
$$id_2 = \nabla\otimes(\mathfrak{i};\nabla)\otimes\nabla;id\otimes\gamma\otimes\gamma\otimes id;\Delta\otimes\Delta\otimes\Delta;id\otimes\nabla\otimes id;\Delta\otimes\Delta \tag{13}$$
$$!_n = id_n\otimes\mathfrak{i};id_n\otimes\nabla^n;\Delta_n;!_n \tag{14}$$

Fig. 10. Axioms for mutual exclusion, textually.

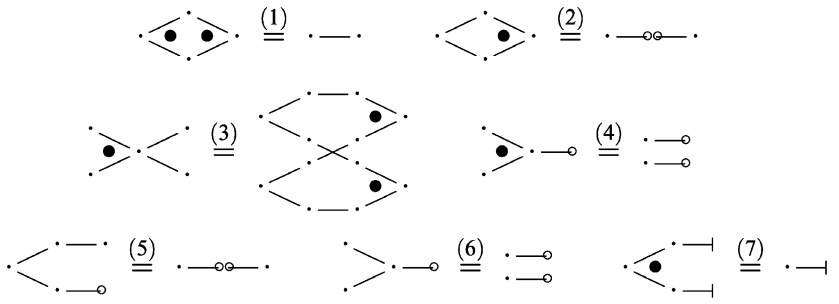

Fig. 11. Simple axioms for mutual exclusion, graphically.

exclusion separately on each pair of actions. Axiom 12 shows that given two independent actions we can freely add an action for their synchronized execution and in that case axiom 13 says that we can also force mutual exclusion on the two paths corresponding to the asynchronous execution of the two starting actions. Finally, axiom 14 means that if all the elements of the interfaces of a connector are adjacent to a node in the interface of a connector $\Delta;!$ (or of the dual form), then for each denotation we can obtain a concrete correct behavior by performing an untick on each internal node, thus there is no real constraint on the behavior of the elements of the interfaces, which can be considered disconnected and closed by hiding connectors.

We present here some useful equivalence lemmas.

Lemma 19. $\nabla_n;\Delta^n\otimes\Delta^n = \Delta^n;\nabla$.

Proof. The proof is by induction on n, and is based on the dual of axiom 3. □

Lemma 20. *For each connector $\alpha : m \to n$ let $\alpha^c : n \to m$ be its dual connector. Then $\alpha\otimes id_n;\Delta_n;!_n = id_m\otimes\alpha^c;\Delta_m;!_m$.*

Proof. The proof is by induction on the number of non-identity basic connectors in (some representation of) α. □

Also for $\mathbf{CC}(\nabla,\Delta,!,\mathfrak{i},\nabla)$ we can define a sorted form and a normal form.

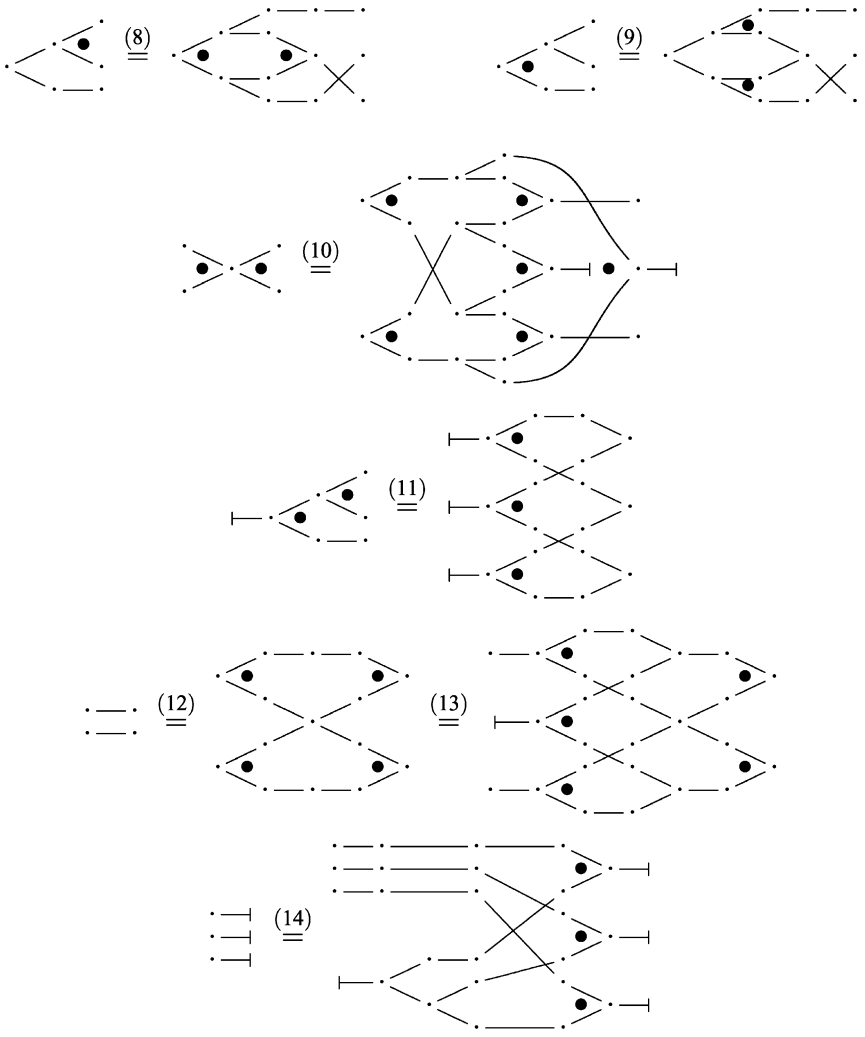

Fig. 12. Complex axioms for mutual exclusion, graphically.

Definition 21 (*Sorted form*). A connector $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathfrak{i}, \overline{\nabla})$ is in sorted form iff:

$$\alpha \equiv \alpha_{\mathfrak{i}} ; \alpha_{\overline{\mathbf{0}}}; \alpha_{\overline{\nabla}} ; \alpha_{\gamma}; \alpha_{\Delta} ; \beta_{\nabla}; \beta_{\gamma}; \beta_{\Delta} ; \beta_{\mathbf{0}}; \beta_{!}$$

where α_σ and β_σ are layers of σs and $\equiv$ is syntactic identity.

Note that the definition of central point (Definition 14) can be applied also to this new sorted form. Central points can be linearly connected to both *free variables* (i.e., external interfaces) and *hidden variables* (i.e., interfaces of hiding connectors).

Proposition 22. *Any connector* $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathfrak{i}, \overline{\nabla})$ *can be transformed in sorted form using the axioms.*

Proof. See Appendix A.

Definition 23 (*Normal form*). A connector $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathfrak{i}, \overline{\nabla})$ is in normal form iff:

(1) α has the form $\alpha_{\mathfrak{i}} ; \alpha_{\mathbf{0}}; \alpha_{\overline{\nabla}} ; \alpha_{\gamma}; \alpha_{\Delta} ; \beta_{\nabla}; \beta_{\gamma}; \beta_{\Delta} ; \beta_{\overline{\mathbf{0}}}; \beta_{!}$ (note that $\mathbf{0}$ and $\overline{\mathbf{0}}$ are swapped w.r.t. the sorted form) where α_σ and β_σ are layers of σs;

(2) hiding connectors are adjacent to either roots of mex trees or central points;
(3) there exists at most one path between a fixed central point and a fixed variable;
(4) no two central points are linearly connected to exactly the same set of variables;
(5) each central point is linearly connected to at least one free variable;
(6) each hidden variable is linearly connected to at most two central points;
(7) no two hidden variables are linearly connected to the same set of central points;
(8) each pair of central points associated with disjoint sets of free variables is linearly connected to a hidden variable;
(9) hidden variables are on the left of central points, unless they are adjacent to them.

Theorem 24. *Any connector* $\alpha \in \mathbf{CC}(\nabla, \Delta, !, \mathsf{i}, \forall)$ *can be transformed into normal form using the axioms.*

Proof. See Appendix A.

Again, there is a precise correspondence between normal forms and tick-tables.

Theorem 25. *For any* tick*-table* T *with* $T[\vec{0}, \vec{0}] = \swarrow$, *we can build a connector* α *in normal form such that* $T(\alpha) = T$. *Moreover, the construction is unique up to the axioms of symmetric monoidal categories and of associativity and commutativity of synch and choice connectors.*

Proof. See Appendix A.

The theorem below establishes a bijective correspondence between denotations and standard implementations of connectors.

Theorem 26. *There is a bijective correspondence between* tick*-tables* T *with* $T[\vec{0}, \vec{0}] = \swarrow$ *and connectors in* $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i}, \forall)$ *up to the axioms.*

Proof. The proof is analogous to the one of Theorem 18, using Theorems 25 and 24 instead of Theorems 17 and 16, respectively. □

5. Modeling architectural connectors

The results in Section 4.2 can be used to extend the research in [4], where a mapping from CommUnity to the Tile Model was presented, and where the main result was that the translation of a CommUnity diagram is tile bisimilar to the translation of its colimit.

CommUnity [16] is a parallel program design language in the style of Unity [9] but based on action sharing. It was initially proposed in [17] to show how programs fit into Goguen's categorical approach to General Systems Theory [20]. Since then, it has evolved into an architectural description language, capitalizing on the fact that it takes to an extreme the separation between "computation" and "coordination" concerns.

The mapping of CommUnity into the Tile Model presented in [4] triggered the definition of the algebra of connectors analyzed in the previous sections. While referring to [16] for a detailed description of CommUnity and to [4] for the details of the mapping, we recall here the main facts about synchronization in CommUnity and show how these can be modeled using our algebra of connectors. CommUnity components, called *programs*, have a fixed set of (guarded) actions each, and at each step one of them (chosen among the enabled ones) is executed. Interaction between different programs, like action synchronization, is expressed at the software architecture level via diagrams made of spans of morphisms, that is, pairs of functions mapping actions of two programs into actions of a particular gluing component (with no computational behavior) called cable. Groups of actions mapped from different programs to the same action in the cable must be synchronized, in the sense that occurrences of any action in the first group must happen when and only when occurrences of an action in the second group happen.

The encoding of CommUnity into the Tile Model separates different aspects of programs (like guards, channels, actions) by exploiting a novel standard decomposition of CommUnity diagrams. The part related to the way in which

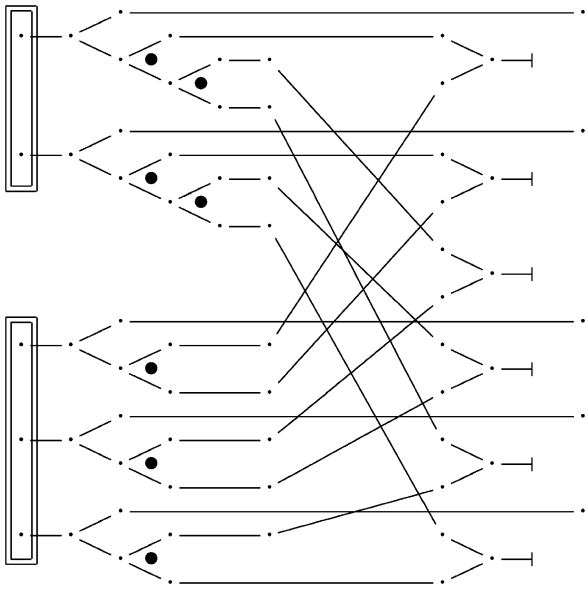

Fig. 13. Synchronizing actions.

actions in different programs are orchestrated is dealt with using the algebra of connectors we have presented in the previous sections.

In the Tile Model each action is represented by an object in the interface, and a tick is performed when an action occurs, and an untick otherwise. The encoding of a complex diagram is defined in a compositional way by encoding the involved spans one after the other (the results in [4] guarantee that the order in which the spans of the diagram are encoded is not important, because the resulting connectors are equivalent). We now show how the synchronization constraint associated with a span is imposed by a suitable connector.

In order to synchronize a tuple t_1 of n mutually exclusive actions with a tuple t_2 of m mutually exclusive actions we have to duplicate each of them. Then, using ∇ connectors, we create n links to each object in t_2 and m links to each object in t_1. Then, using Δ connectors, we merge each action in the first group with each action in the second group and we close the resulting objects using ! connectors. See Fig. 13 for an example (groups of two and three actions, respectively). It can be readily seen that if the upper two actions are performed by the same program (and thus are mutually exclusive) and if the lower three actions are performed by the same program (and thus are mutually exclusive), then at each execution step exactly two actions are synchronized.

Using the correspondence between observational semantics and connectors up to the axioms, we can state that the (synchronization part of the) translation of a CommUnity diagram is equal up to the axioms to the translation of its colimit. More generally, colimit computation in the categorical approach is now strongly related to normalization using suitable axioms in the algebraic approach.

Example 27. We show here an example of the translation from CommUnity into the Tile Model and of the algebraic computation of the colimit. Since CommUnity uses a more abstract view than the Tile Model, simple programs are mapped into quite large tile configurations. Thus, to keep the presentation manageable we concentrate on simple programs. Also, w.r.t. [4], instead of having at least one action executed by each program in each transition, with a default hidden action available, we drop hidden actions and allow connectors to stay idle. Also, we drop the part of configurations related to variables, since dealing with state is outside the scope of the present paper. Thus guard managers, which are the managers of actions (the name comes from guards that tells if an action is enabled, which are the main information managed by guard managers), may include conditions referring to a non represented state.

Let us consider a simple program with two actions with guards q and r. Its translation is composed by two guard managers $gm[q]\colon 0 \to 1$, $gm[r]\colon 0 \to 1$ and a glue $\mathsf{i};\nabla$. The three components are connected through two spans of morphisms merging the two actions of the glue with the two actions of the guard managers. Since we will no more use

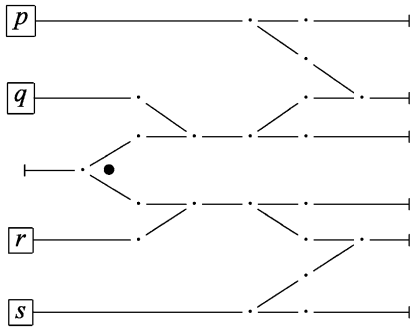

Fig. 14. The translation of the diagram.

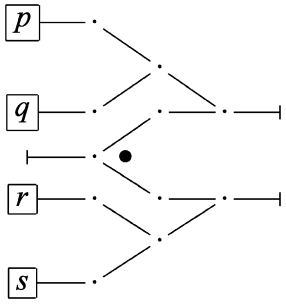

Fig. 15. The translation, simplified.

the actions of the guard managers (external morphisms refer only to the glue), we will close them immediately instead of at the end of the translation to simplify the presentation. Thus the obtained component can be written (using some axioms) as

$$Q = gm[q] \otimes (\mathsf{i}; \triangledown) \otimes gm[r]; \Delta \otimes \Delta.$$

In a similar way, components with only one action can be represented by just their guard manager, since they have the form $gm[p] \otimes \mathsf{i}; \Delta$, which reduces to $P = gm[p]$.

We can now consider a diagram with the two programs above and another program $S = gm[s]$, and morphisms connecting the first action of Q to P and the second one to S. Its translation is represented in Fig. 14.

The colimit is a program with two actions with guards $p \wedge q$ and $r \wedge s$, respectively, whose translation is in Fig. 16. We now show how the same result can be computed algebraically. In addition to the axioms for connectors we need also an axiom schema for guard managers, which has the form $gm[p] \otimes gm[q]; \Delta = gm[p \wedge q]$ and must be instantiated for each pair of predicates p and q. This axiom essentially says that synchronizing two actions corresponds to have a unique action with the conjunction of guards. Starting from Fig. 14 and by using the axioms for connectors (actually, in this example, axioms for choice and inaction connectors are not necessary) we get to the system in Fig. 15, which is in normal form (as far as the connector part is concerned).

Then we can reduce to the diagram in Fig. 16, as expected, using the axioms for guard managers.

6. Modeling coordination connectors

Reo [1] is an exogenous coordination language with channel-based connectors. In fact, Reo connectors are structured communication media that components can instantiate, compose, connect to and, of course, where I/O operations can be performed. In particular, the main feature is that *channels* are not necessarily two-end wires, but they can have any number of *source ends* (able to accept input) and *sink ends* (producing output), typed according to the direction of the flow of data. There can be synchronous and asynchronous channels, and they can be composed by conjoining their ends to form *nodes*. Components can write on input nodes (containing only source ends, where the written data

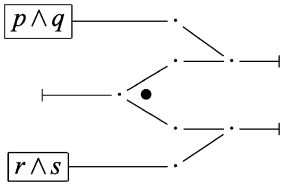

Fig. 16. The translation of the colimit.

is replicated to all ends) and take input from output nodes (containing only sink ends, where the input is selected nondeterministically from the data item offered by each end, if any is available).

Reo describes a basic set of channel types whose instances can be used to form arbitrarily large and complex connectors. Synchronous channel types include: Sync with exactly one source and one sink end that are synchronized by communicating the data item from its source to its sink atomically; SyncDrain that reads the data simultaneously from its two source ends (all incoming data items are then lost); SyncSpout that acts as an unbounded source of data items on its two sink ends (outgoing data items are not related to each other); LossySync has a source and a sink end and it synchronizes the source with the sink but not vice versa (components can read from a LossySync channel only if a writer is ready, but writers can produce data items even if no reader is present). Asynchronous channel types include: AsyncDrain with only two source ends, and AsyncSpout which has only two sink ends, both guaranteeing that two operations on their two ends never succeed simultaneously; channels with bounded and unbounded buffers; channels with delays; different kinds of filters for data items.

As noticed, e.g. in [10], input nodes act essentially as synch, while output nodes as mex. Those nodes conjoining both source and sink ends, called *mixed nodes*, behave like self-contained "pumping stations" that combine the behavior of an input and an output node. Indeed most semantic models for Reo connectors introduce for simplicity two additional primitive connectors (they are not Reo channels) called either merge and fork (in [30]) or merge and replicator (in [10]). Mixed nodes can then be trivially expressed in terms of the two additional connectors, with the advantage that composition can now be expressed according to the *plugging principle*: (i) connector instances are composed by plugging a sink end of one connector to the source end of another connector and (ii) combined nodes become hidden, so that each node is a sink/source for at most one connector.

The modeling of Reo connectors turned out as an interesting exercise for the application of our framework, as it suggested some straightforward enrichments of our basic algebra of connectors. In the modeling we leave aside "active" channels like buffered, delayed and filtered ones, which lie outside our scope.

First of all connector composition in Reo is based on name sharing instead of sequential and parallel operators. However the plugging principle guarantees that each Reo node has one input/output capability. The idea is then to type the interfaces of our connectors according to the direction of the flow of data, and then to model explicitly each node with the connector of the shape $\mathsf{i}; \nabla$. More precisely, the objects of our horizontal category are now strings of plus ($+$ for source) and minus ($-$ for sink) instead of natural numbers, and each connector $\alpha\colon n \to m$ can now exist in many (2^{n+m}) typed variants $\alpha(\tau_i, \tau_o)$ with $\tau_i \in \{+,-\}^n$ and $\tau_o \in \{+,-\}^m$, depending on the plus / minus typing of its left and right interfaces. For example, by letting λ denote the empty string, we may have $\mathsf{i}(\lambda,+)\colon \lambda \to +$ and $\nabla(+,+-)\colon + \to +-$. We can then encode Reo nodes as $\mathsf{i}(\lambda,+);\nabla(+,+-)\colon \lambda \to +-$. Note that the typing of internal nodes is of course irrelevant, so that for example

$$(\mathsf{i};\nabla)(\lambda,+-) = \mathsf{i}(\lambda,+);\nabla(+,+-) = \mathsf{i}(\lambda,-);\nabla(-,+-).$$

This extension is effortless for the application of our techniques, because they can be applied to multisorted signatures in exactly the same way, even if we presented the untyped case (where objects are natural numbers) for simplicity. Even the denotational semantics based on tick-tables is still valid (we need just to record the typing of the input/output interfaces).

If we assume that the content of a transmitted data item is irrelevant, then we are essentially done, because it is immediate to express all stateless channels of Reo as suitable connectors in our algebra by interpreting tick as the presence of some data and untick as the absence of data. In fact we can encode most common channel types as illustrated in Fig. 17. Without giving the full details of the encoding here, any complex Reo connector can then be represented as a connector $\alpha(\lambda,\tau)$, whose right interface contains all the pluggable input and output nodes. Connectors can be joined

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{Sync} & = & (\Delta;!)(+-,\lambda) & \mathsf{SyncDrain} & = & (\Delta;!)(++,\lambda)\\
\mathsf{LossySync} & = & (\nabla\otimes id;id\otimes\Delta;!\otimes!)(+-,\lambda) & \mathsf{SyncSpout} & = & (\Delta;!)(--,\lambda)\\
\mathsf{Replicator} & = & (\Delta^3;!)(+--,\lambda) & \mathsf{AsyncDrain} & = & (\blacktriangle;!)(++,\lambda)\\
\mathsf{Merge} & = & (\blacktriangle\otimes id;\Delta;!)(++-,\lambda) & \mathsf{AsyncSpout} & = & (\blacktriangle;!)(--,\lambda)
\end{array}$$

Fig. 17. Encoding of Reo channel types.

$$\gamma \xrightarrow[y\otimes x]{x\otimes y} \gamma \text{ with } x,y\in \mathbf{Act}\cup\{untick\} \qquad \nabla \xrightarrow[a\otimes a]{a} \nabla \qquad \nabla \xrightarrow[\mathsf{untick}\otimes\mathsf{untick}]{\mathsf{untick}} \nabla \qquad \mathbf{0} \xrightarrow[id_0]{\mathsf{untick}} \mathbf{0}$$

$$! \xrightarrow[id_0]{a} ! \qquad ! \xrightarrow[id_0]{\mathsf{untick}} ! \qquad \nabla \xrightarrow[\mathsf{untick}\otimes\mathsf{untick}]{\mathsf{untick}} \nabla \qquad \nabla \xrightarrow[a\otimes\mathsf{untick}]{a} \nabla \qquad \nabla \xrightarrow[\mathsf{untick}\otimes a]{a} \nabla$$

Fig. 18. Tiles for data-sensitive, ordinary connectors.

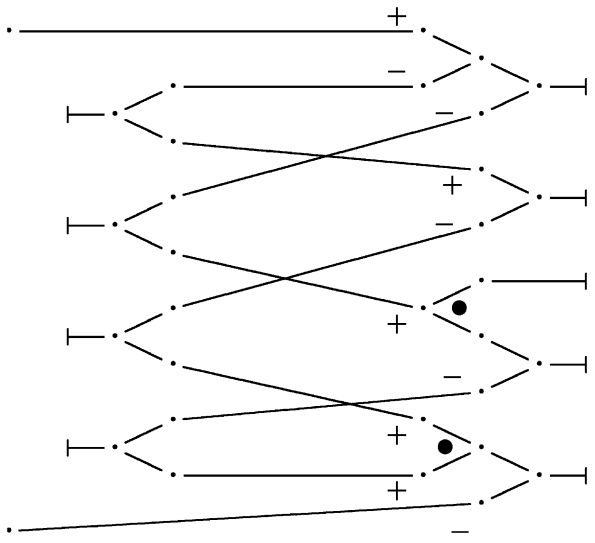

Fig. 19. Translation of a Reo connector.

by putting them in parallel and synchronizing the conjoined ends (permutations are used to arrange the order imposed in the interface, if needed).

On the other hand, if we assume that the content of a transmitted data item must be taken into account we need to replace the ordinary tick action by any admissible action, say in the alphabet **Act**. The corresponding tiles for ordinary connectors should be changed according to Fig. 18, where $a \in \mathbf{Act}$. Also the dimension of tick-tables should be expanded to represent all the possible combinations of data items. However, all the axioms remain valid and the reduction to normal form is still possible. The only problem is that SyncDrain and SyncSpout can hardly be encoded, because they should allow the simultaneous consumption/production of *possibly different* data items, while we can only express the simultaneous consumption/production of *the same* data item (unless also allowing asynchronous consumption/production of data item).

The negative results about SyncDrain and SyncSpout suggest that other kinds of mex and synch connectors could be considered, which allow for the nondeterministic choice of one data item when two different ones are available. The investigation of the axioms that rule such additional connectors is left to future work.

Example 28. In this example we show how the theory developed so far can be applied to derive interesting properties about Reo connectors. In particular, we show that two nodes connected by both a Sync channel and a LossySync channel (in the same direction) are equivalent to two nodes connected only by the Sync channel. The translation of the above described connector is obtained as follows: a Replicator, a Sync, a LossySync and a Merge are composed in parallel, then corresponding nodes (which always have opposite polarities) are merged using i; ∇ connectors, together with permutations when required. The result is in Fig. 19, where connectors are on the right side and node mergers are on the left side. By using the axioms for synch and hiding connectors we can reduce the expression to

$$\nabla \otimes id;\ \overline{\nabla} \otimes id \otimes id;\ ! \otimes \Delta \otimes id;\ \Delta;\ !$$

After that we can use axiom 8 and naturality of symmetry to individuate a subconnector of the shape $\nabla;\ \Delta$, which can be substituted by inaction connectors. Thus, we reduce to

$$\overline{\nabla} \otimes id;\ id \otimes \mathbf{0} \otimes id;\ id \otimes \overline{\mathbf{0}} \otimes id;\ \Delta \otimes id;\ \Delta;\ !$$

which can be written by monoidality as

$$(\overline{\nabla};\ id \otimes \mathbf{0};\ id \otimes \overline{\mathbf{0}};\ \Delta) \otimes id;\ \Delta;\ ! = \Delta;\ !$$

Since $\Delta;\ !$ is exactly the translation of a Sync channel (the typing of the interface is preserved by axioms), this concludes our example.

7. Conclusion and future work

We have presented different classes of connectors and we have shown how they can be analyzed from different points of view: their concrete structures can be described by graphs, their operational and observational semantics are given using tiles and tile bisimilarity, while the representation based on tick-tables provides a denotational semantics. We have proved that there is a bijective correspondence among connectors up to suitable axioms, classes of bisimilar connectors and denotations, thus proving the coherence of all views. This allows to extend the result of [4] proving that there is a correspondence between colimit computation in a categorical framework and normalization up to the axioms in an algebraic framework. Our work leaves an open problem: we argue that the axiom schema 14 (see Figs. 10 and 12) is needed for the completeness of the axiomatization, but we do not know whether a different finite axiomatization of $\mathbf{CC}(\nabla, \Delta, !, \mathsf{i}, \overline{\nabla})$ exists or not.

Though we have focused on stateless connectors, state information can be accommodated quite well within the tile framework. For example, the encoding of CommUnity [4] exploits a global (synchronized) state, separated from the algebra of connectors presented here. For Reo, where the state can be distributed within FIFO channels, we can use tiles with different initial and final configurations to represent the state in a natural (and distributed) way. However, defining normal forms for the resulting equivalence classes of state-full connectors looks like a difficult task.

As future work we plan to study the complexity of our reduction to normal form and to generalize our connectors, in the line of Reo, to a setting where we have a richer set **Act** of actions ruled by a synchronization algebra. Another interesting extension is the study of probabilistic connectors and the corresponding axiomatization.

Acknowledgments

We warmly thank José Luiz Fiadeiro and Antónia Lopes for their help in understanding the principles of CommUnity and David Costa for pointing out analogies between our research and the one based on coloring of Reo connectors in [10].

Appendix A. Proofs

Proof of Lemma 6. Let us consider two connectors $\alpha\colon n \to h$ and $\beta\colon h \to m$. A cell $T(\alpha;\beta)[i, j]$ is marked iff there exists a concrete semantics with a tick on each element of interface in $d_{T(\alpha;\beta)}([i, j])$ and an untick on all the

other elements of interface. In this case we can choose one of these concrete semantics, which induces two denotational semantics for α and β such that the effect of α coincides with the trigger of β. Thus, there is a k such that $T(\alpha)[i,k] = \blacktriangleleft$ and $T(\beta)[k,j] = \blacktriangleleft$. Thus $\bigvee_k (T(\alpha)[i,k] \wedge T(\beta)[k,j]) = \blacktriangleleft$, as required. Vice versa if the cell $T(\alpha;\beta)[i,j]$ is not marked, there is no such concrete semantics. For each i, j, k we must have either $T(\alpha)[i,k] \neq \blacktriangleleft$ or $T(\beta)[k,j] \neq \blacktriangleleft$ and so $\bigvee_k (T(\alpha)[i,k] \wedge T(\beta)[k,j]) \neq \blacktriangleleft$.

As far as parallel composition is concerned, let us consider two connectors $\alpha\colon n \to h$ and $\beta\colon l \to m$. Refining the marked cells in $T(\beta)$ by $T(\alpha)$ and the unmarked cells by the empty table means that for each pair of cells in $T(\beta)$ and $T(\alpha)$, respectively, the cell in $T(\alpha \otimes \beta)$ with as domain the disjoint union of the domains is marked iff both the starting cells were marked. This is the correct denotational semantics for the composed connector, since this corresponds to take the union of two concrete semantics for α and β, which is a concrete semantics for $\alpha \otimes \beta$.

For dual connectors the proof is trivial, by structural induction on the connector. □

Proof of Proposition 9. The first property is easily proved by induction on the structure of the connector.

As far as the other properties are concerned, notice that in a network of synch and hiding connectors, the only constraint is that connected nodes must have the same value. We can divide the nodes in equivalence classes, with two nodes being in the same class iff they are connected. The domain of each marked cell must correspond to the union of a set of equivalence classes. In particular each equivalence class is either completely contained in a domain, or it is disjoint from it.

Let us consider for instance the intersection of two domains (the proof is similar for other operations): each equivalence class is either contained in both the domains and thus in the intersection, or it is disjoint from at least one domain and thus from the intersection. Thus the intersection is the union of some set of equivalence classes. Since different equivalence classes are independent this is an allowed configuration and it corresponds to the domain of a marked cell. □

Proof of Proposition 22. This is the most complex proof in the paper, and it extends the proof of Proposition 13 to deal with all the connectors in $\mathbf{CC}(\nabla, \Delta, !, \mathrm{i}, \triangledown)$. We have to show that basic connectors that are in the wrong layer can be made to commute with adjacent layers until they reach the correct layer. In general, during the operation, other basic connectors may be created, and in this case we have to check that the procedure indeed terminates.

We start by presenting some commutation properties that we will use later:

(1) Thanks to naturality γ can traverse a layer from the side where connectors have arity 1 to the side where they have arity 2. Note that this is enough to move any γ connector to either layer α_γ or β_γ.
(2) Δ connectors can be commuted with ∇ connectors using axiom $\Delta; \nabla = \nabla \otimes \nabla; id \otimes \gamma \otimes id; \Delta \otimes \Delta$. This operation creates some γ connectors which can be moved to the correct layer by what is already proved.
(3) Dually, Δ can be commuted with $\triangledown$.
(4) Δ connectors can be commuted with $\triangledown$ connectors using axiom 10 in Fig. 10. This creates γ, ∇ and ! connectors. When applying this property we must check that we are able to move the created connectors to their correct layers without cycling.
(5) ∇ can be commuted with $\triangledown$ using axiom 8 in Fig. 10 ($\nabla; \triangledown \otimes id = \triangledown; \nabla \otimes \nabla; id \otimes \Delta \otimes id; id \otimes \gamma$). This will create γ connectors and Δ connectors. In this case too we have to check that there is no risk to cycle.
(6) Dually, Δ can be commuted with Δ. This will create γ connectors and $\triangledown$ connectors. Here a check for avoiding cycles is required too.
(7) We can commute any combination of ∇, γ and Δ thanks to Theorem 16.

We will now prove the thesis by induction on the construction of the connector (see Lemma 2). The base case is trivial. For the inductive case we have to prove that given any connector $\alpha\colon m \to n$ in sorted form, we can transform any connector $id_{n_1} \otimes \sigma \otimes id_{n_2}; \alpha$ in sorted form. We consider different cases, according to the kind of σ. Note that we can always move !, i, $\mathbf{0}$ and $\overline{\mathbf{0}}$ connectors to layers $\beta_!$, α_i, $\beta_{\mathbf{0}}$ and $\alpha_{\overline{\mathbf{0}}}$ as required. Thus we can just focus on the other connectors. We can now consider the different cases:

Case $\triangledown$: Trivial.

Case γ: Property 1 is enough to move γ through the layer $\alpha_\triangledown$ if needed.

Case Δ: The Δ connectors can be commuted with the $\triangledown$ connectors using property 3 and the γ connectors using property 1 (moving γ towards the left).

Case ∇ and Δ: We prove these two cases in one single step since commuting ∇ connectors also produces Δ connectors and vice versa. The ∇ connectors need to be commuted with: (i-a) $\forall$, (ii-a) γ, and (iii-a) Δ. The Δ connectors need to be commuted with: (i-b) $\forall$, (ii-b) γ, (iii-b) Δ, and (iv-b) ∇.

- Parts (i-a) and (i-b) are by induction on the maximum number of $\forall$ connectors in all the paths of layer $\alpha_{\forall}$. For ∇ connectors, the commutation can be done thanks to property 5, but each time that we apply the axiom involved, we create one Δ and one γ connector. The γ connectors can be moved to layer α_{γ} using property 1. As far as Δ connectors are concerned, the step (i-b) can be done using property 4, which creates additional γ, ∇ and ! connectors. As usual ! connectors and symmetries can be moved to the wanted layers. As far as the ∇ connector is concerned, all the ∇ connectors created have to commute with one $\forall$ connector less, and this can be done thanks to the inductive hypothesis.
- Steps (ii-a) and (ii-b) can both be done thanks to property 1.
- The ∇ connectors can commute with layer α_{Δ} using property 7 and thus reaching the layer β_{∇} as required, what proves part (iii-a).
- As far as Δ connectors are concerned, we can use property 6 to commute them with layer α_{Δ}. This will produce $\forall$ connectors that need to be moved back to layer $\alpha_{\forall}$. This can be done using properties 1 and 3. These new $\forall$ connectors may have to be traversed again by the ∇ connectors that created them, and in this case a different strategy must be used. Note that when applying axiom 8 ($\nabla; \forall \otimes id = \forall; \nabla \otimes \nabla; id \otimes \Delta \otimes id; id \otimes \gamma$) the Δ connector is created attached to an element of interface which is not the same to which the $\forall$ connector in the left hand side was attached to, and this means that we risk to cycle only if there are two different paths starting from the two elements of the right interface of the same ∇ connector, which arrive to the same Δ connector. If they join on a Δ connector then we can individuate a connector of the form $\nabla; \forall \otimes \forall; id \otimes \Delta \otimes id$ which is equal up to the axioms to $\forall; \nabla \otimes id; id \otimes \gamma$. Thus we can use this equality to commute ∇ and $\forall$ without cycling. If they join on a $\forall$ connector instead we can individuate a connector $\forall; \Delta$ which is equal up to the axioms to $\mathbf{0}; \overline{\mathbf{0}}$. Thus both paths are removed and we do not cycle. This concludes part (iii-b).
- To finish we have to commute the Δ connectors with the layer β_{∇} (iv-b). This can be done using property 2. □

Proof of Theorem 24. Thanks to Proposition 22 we can suppose that the connector is already in sorted form. We have to satisfy all the conditions in Definition 23.

For condition 1 just note that $\mathbf{0}$ (resp. $\overline{\mathbf{0}}$) connectors absorb each connector they are attached to, unless this is a $\forall$ (resp. Δ) connector (from the side with the interface of arity 2) and in that case they are absorbed by it.

Condition 2 may not be satisfied only if there is a hiding connector adjacent to a leaf of a synch connector or to a permutation, but in the first case we can use axiom $\nabla; ! \otimes id = id$ (or the dual one) to delete it, while in the second case we can use naturality to delete the γ itself.

As far as condition 3 is concerned, note that if a central point is linearly connected using two different paths to the same variable, then we can individuate either a $\forall; \Delta$ or a $\nabla; \Delta$ connector. Thus we can apply the axiom $\forall; \Delta = \mathbf{0}; \overline{\mathbf{0}}$ (or the dual one) to remove the paths and then propagate the inaction connectors.

In order to satisfy condition 4 we must merge any two central points linearly connected to the same variables. We can use Lemma 19 on the left of the central point and the dual property on the right. We can then delete the $\forall; \Delta$ in the centre using $\forall; \Delta = id$. Note that the result is equal to the starting connector, but with one central point less as required.

Suppose condition 5 is not satisfied. If the central point has a $\mathfrak{i}$ connector as left (trivial) tree then we can apply the axiom schema and delete the central point. Otherwise we can use Lemma 20 to reach the previous situation. Note that the created ! connectors can be simplified with the ∇ connectors they are attached to.

To satisfy condition 6 we can apply axiom 11 in Fig. 10.

Suppose condition 7 is not satisfied. Then we can individuate a connector of the form $\mathfrak{i}_2; \forall \otimes \forall; \Delta_2$ which can be transformed using the axioms into $\mathfrak{i}_2; \Delta; \forall = \mathfrak{i}; \forall$.

Suppose that condition 8 is not satisfied. Then we can use axiom 13 in Fig. 10 to add the required hidden variable, together with a new central point for the synchronized execution of the two actions. If this one existed already then we can first delete it using axiom 12 in Fig. 10.

Finally suppose that condition 9 is not satisfied, that is, we have a hidden variable on the right of a central point, which is not adjacent to it. By what is already proved it must be adjacent to a mutual exclusion tree, which is linearly

connected to exactly two central points. Thus we can individuate a connector of the form $\nabla \otimes \nabla; id \otimes (\Delta; !) \otimes id$ which can be transformed into $id \otimes (\mathrm{i}; \forall) \otimes id; \Delta \otimes \Delta$ using the definition of Δ and then simplifying the resulting connector. □

Proof of Theorem 25. Given a tick-table T, the connector α is realized in the following way:

- any input (resp. output) variable that has value 0 in all the concrete semantics is connected to a $\mathbf{0}$ (resp. $\overline{\mathbf{0}}$) connector;
- for other input (resp. output) variables, we count the number of checked cells that have that variable in the domain and we build a tree of $\forall^n$ (resp. Δ^n);
- for each pair of central points with disjoint sets of free variables, we create a hidden variable associated to them on the left;
- for each checked cell in the table, we create a central point in the connector, with two outgoing trees Δ^n to the left and ∇^m to the right. Here n counts the input variables in the domain of the cell, plus the hidden variables associated to the central point while m is the number of output variables in the domain;
- we connect leaves of synchronization trees with leaves of mutual exclusion trees using permutations, connecting each central point to the associated variables.

It remains to be proved that $T(\alpha) = T$. If a variable has value 0 in all the concrete semantics, then it is adjacent to an inaction connector and its value is forced to zero. We will no more consider these variables. We call a node active in a concrete semantics iff a tick action is performed on it. We will prove that: (i) in each concrete semantics at most a central point can be active, (ii) that the choice of the central point determines the variables in the domain, and (iii) that the corresponding cell in the tick-table must be marked.

For part (i), note that each pair of central points has a common variable. Thus if two central points are active there are two active leaves in the corresponding mutual exclusion tree, which is absurd. For part (ii), note that the variables in the domain are exactly the ones which are linearly connected to the central point. These variables must be in the domain since one leaf of their mutual exclusion tree is connected to the active central point, and no other variable can be in the domain otherwise mutual exclusion fails. Part (iii) follows since by construction there is a central point for each marked cell, and this central point is linearly connected exactly to the variables in the domain.

As far as uniqueness is concerned, note that the above construction is deterministic up to the allowed axioms. □

References

[1] F. Arbab, Reo: a channel-based coordination model for component composition, Math. Struct. Comput. Sci. 14 (3) (2004) 1–38.
[2] F. Arbab, J. Rutten, A coinductive calculus of component connectors, in: Proc. of WADT'02, Lecture Notes in Computer Science, Vol. 2755, Springer, Berlin, 2003, pp. 35–56.
[3] J.A. Bergstra, C.A. Middelburg, G. Stefanescu, Network algebra for asynchronous dataflow, Internat. J. Comput. Math. 65 (1997) 57–88.
[4] R. Bruni, J.L. Fiadeiro, I. Lanese, A. Lopes, U. Montanari, New insights on architectural connectors, in: Proc. of IFIP TCS 2004, Kluwer, Dordrecht, 2004, pp. 367–379.
[5] R. Bruni, F. Gadducci, U. Montanari, Normal forms for partitions and relations, in: Proc. of WADT'98, Lecture Notes in Computer Science, Vol. 1589, Springer, Berlin, 1999, pp. 31–47.
[6] R. Bruni, F. Gadducci, U. Montanari, Normal forms for algebras of connections, Theoret. Comput. Sci. 286 (2) (2002) 247–292. Full version of [5].
[7] R. Bruni, I. Lanese, U. Montanari, Complete axioms for stateless connectors, in: Proc. of CALCO'05, Lecture Notes in Computer Science, Vol. 3629, Springer, Berlin, 2005, pp. 98–113.
[8] V.E. Cazanescu, G. Stefanescu, Towards a new algebraic foundation of flowchart scheme theory, Fund. Inform. 13 (1990) 171–210.
[9] K. Chandy, J. Misra, Parallel Program Design, Addison-Wesley, Reading, MA, 1988.
[10] D. Clarke, D. Costa, F. Arbab, Connector colouring I: synchronization and context dependency, in: Proc. of FOCLASA'05, Electr. Notes Theor. Comput. Sci. 154 (1) (2006) 101–119.
[11] A. Corradini, F. Gadducci, An algebraic presentation of term graphs, via gs-monoidal categories, Appl. Categorical Struct. 7 (1999) 299–331.
[12] A. Corradini, U. Montanari, An algebraic semantics for structured transition systems and its application to logic programs, Theoret. Comput. Sci. 103 (1992) 51–106.
[13] P. Degano, U. Montanari, A model for distributed systems based on graph rewriting, J. ACM 34 (2) (1987) 411–449.
[14] H. Ehrig, M. Pfender, H.J. Schneider, Graph grammars: an algebraic approach, in: Proc. IEEE Conf. on Automata and Switching Theory, 1973, pp. 167–180.
[15] J.L. Fiadeiro, Categories for Software Engineering, Springer, Berlin, 2004.
[16] J.L. Fiadeiro, A. Lopes, M. Wermelinger, A mathematical semantics for architectural connectors, Generic Programming, Lecture Notes in Computer Science, Vol. 2793, Springer, Berlin, 2003, pp. 190–234.

[17] J.L. Fiadeiro, T. Maibaum, Categorical semantics of parallel program design, Sci. Comput. Programming 28 (1997) 111–138.
[18] F. Gadducci, U. Montanari, The tile model, in: Proof, Language and Interaction: Essays in Honour of Robin Milner, MIT Press, Cambridge, 2000, pp. 133–166.
[19] F. Gadducci, U. Montanari, Comparing logics for rewriting: rewriting logic action calculi and tile logic, Theoret. Comput. Sci. 285 (2) (2002) 319–358.
[20] J. A. Goguen, Categorical foundations for general systems theory, in: Advances in Cybernetics and Systems Research, Transcripta Books, 1973, pp. 121–130.
[21] C.A.R. Hoare, CSP—Communicating Sequential Processes, International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[22] P. Katis, N. Sabadini, R.F.C. Walters, Bicategories of processes, J. Pure and Appl. Algebra 115 (1997) 141–178.
[23] Y. Lafont, Interaction combinators, Inform. and Comput. 137 (1) (1997) 69–101.
[24] K.G. Larsen, L. Xinxin, Compositionality through an operational semantics of contexts, in: Proc. of ICALP'90, Lecture Notes in Computer Science, Vol. 443, Springer, Berlin, 1990, pp. 526–539.
[25] S. MacLane, Categories for the Working Mathematician, Springer, Berlin, 1971.
[26] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, Theoret. Comput. Sci. 96 (1992) 73–155.
[27] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1989.
[28] R. Milner, Turing, computation and communication, Turing anniversary lecture, October 1997. ⟨http://www.fairdene.com/picalculus/milner-infomatics.pdf⟩.
[29] R. Milner, Bigraphical reactive systems, in: Proc. of CONCUR 2001, Lecture Notes in Computer Science, Vol. 2154, Springer, Berlin, 2001, pp. 16–35.
[30] M.R. Mousavi, M. Sirjani, F. Arbab, Formal semantics and analysis of component connectors in Reo, in: Proc. of FOCLASA'05, Electr. Notes Theor. Comput. Sci. 154 (1) (2006) 83–99.
[31] A. Rensink, Bisimilarity of open terms, Inform. and Comput. 156 (1/2) (2000) 345–385.
[32] G. Stefanescu, Network Algebra, Discrete Mathematics and Theoretical Computer Science, Springer, Berlin, 2000.