Cospan/Span(Graph): an Algebra for Open, Reconfigurable Automata Networks

Alessandro Gianola[1], Stefano Kasangian[2], and Nicoletta Sabadini[3]

1 **Università degli Studi di Milano, Via Saldini 50, Milano, Italy**
`alessandro.gianola93@gmail.com`
2 **Università degli Studi di Milano, Via Saldini 50, Milano, Italy**
`stefano.kasangian@unimi.it`
3 **Università dell'Insubria, Via Valleggio 11, Como, Italy**
`nicoletta.sabadini@uninsubria.it`

Abstract

Span(**Graph**) was introduced by Katis, Sabadini and Walters as a categorical algebra of automata with interfaces, with main operation being communicating-parallel composition. Additional operations provide also a calculus of connectors or wires among components. A system so described has two aspects: an informal network geometry arising from the algebraic expression, and a space of states and transition given by its evaluation in **Span**(**Graph**). So, **Span**(**Graph**) yields purely compositional, hierarchical descriptions of networks with a fixed topology. The dual algebra **Cospan(Graph)** allows to describe also the sequential behaviour of systems. Both algebras, of spans and of cospans, are symmetrical monoidal categories with commutative separable algebra structures on the objects. Hence, the combined algebra **CospanSpan(Graph)** can be interpreted as a general algebra for reconfigurable/hierarchical networks, generalizing the usual Kleene's algebra for classical automata. We present some examples of systems described in this setting.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Categories, Automata, Composition, Networks

Digital Object Identifier 10.4230/LIPIcs.CALCO.2017.2

Category Invited Talk

1 Introduction

In the last decades, starting with the pioneering approach of C.A. Petri, many formalisms have been proposed in the effort of describing "concurrent" systems, a notion, at least in the opinion of Sabadini and Walters, always very obscure. Some of these formalisms (for example Milner's CCS and in general Process Algebras) insisted correctly on the *compositional* point of view, typical of classical Regular Expressions, by extending the latter with new operators. Unfortunately, the natural correspondence with classical automata, i.e. state/transition systems, was totally lost. This was, in the opinion of Sabadini and Walters, a mistake, since finite automata provide in a natural way the control structure of discrete dynamical (state/transition) systems. Slowly (and - alas - not yet certainly), it has been realized that the semantic object under consideration was *not* a "concurrent" system, but more clearly a distributed network of interacting components/agents/automata. Sabadini-Walters proposed this point of view from the beginning, focusing on the development of an algebra of networks of automata, that should generalize in a natural way the classical algebra of Kleene for

sequential automata. What is missing in Kleene's algebra for automata is the operation of ***communicating parallel***, fundamental to describe any mechanism. Consider for example how two gearwheels interact during their movement by simultaneuos change of their state. We argue that this more fundamental kind of interaction - that it is not input/output message passing, by its nature sequential, but on the contrary a simultaneous change of state - underlies any communication among machines or physical devices. But there is a second issue that should be taken into account in order to provide a compositional description for networks: a network is by default an *open* entity, and it is not possible to build an open system starting with closed ones. Hence, open systems should be taken as basic building blocks, and this forced us to introduce a new notion, *automata with communication ports or interfaces*. But what is the mathematical nature of automata with interfaces? And what is the correct algebra for composing them?

The algebra **Span**(**Graph**) was introduced in [8] as a "parallel" algebra for *automata with interfaces*, I.E. the two fundamental operations are parallel composition with or without communication. The automata are *open* (that is, have interfaces or ports) as is necessary to achieve compositionality. Communication is through synchronization on common actions of the control of different components (not message passing). Communicating automata evolve simultaneously, not in interleaving. In order to have an algebra with at most binary operations the interfaces should be grouped into two sets, what we call respectively the left and right combined ports. There is no implication that left ports are input ports, or that right ports are output ports.

The algebra **Cospan**(**Graph**) was introduced in [9] as a sequential algebra for automata with interfaces, with essentially the Kleene operations of sequential composition, choice and iteration.

In both cases the algebra involved is a well-known one introduced by Jean Benabou in 1967 [3] and developed in 1987 by Carboni, Walters [4] as an algebra of relations, which provides a natural mathematical framework for describing nets of automata, both graphically with circuit-like diagrams, and as terms in a suitable algebra.

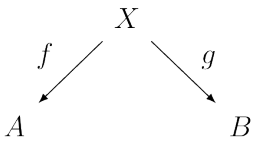
The present paper is essentially a review of previous works of the last twenty years, but it has also some novelties (we suggest that for a better comprehension of this paper and for details, the original papers are available).

The paper is organized as follows: in Section 2, we introduce the algebras of **Span**(**C**) and **Cospan**(**C**), first in the case of an abstract category **C** and then specializing to the case of **Graph**, providing also a series of examples; in Section 3, we discuss closed and open networks relating them to monoidal graphs; in conclusion, in Section 4, we describe open networks with state, showing that it is a reformulation of **Span**(**Graph**).

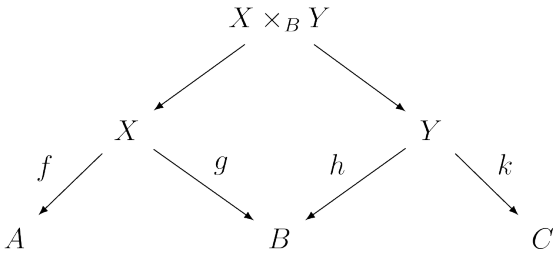
2 Review of the Span and Cospan algebras of automata

2.1 The algebra of spans

▶ **Definition 1.** Given a category **C** with finite limits, we define a new category **Span**(**C**) describing its objects and arrows. Objects of **Span**(**C**) are the same objects of **C**; arrows of **Span**(**C**) from A to B are spans, that is pairs of arrows $(f : X \to A, g : X \to B)$ of **C** with common domain, often written:

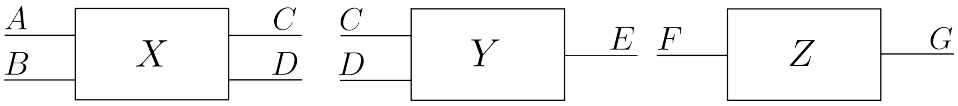


Composition of spans $(f : X \to A, g : X \to B)$ and $(h : Y \to B, k : Y \to C)$ is by pullback (restricted product):

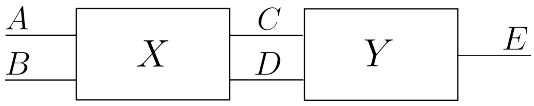


A span $(f : X \to A, g : X \to B)$ will also be written $X : A \to B$, and the composition of span will be indicated with the notation $X;Y : A \longrightarrow C$. The identity span of object A is $(1_A, 1_A)$. The category $\mathbf{Span}(\mathbf{C})$ is actually symmetric monoidal with the tensor of two spans $(f : X \to A, g : X \to B)$ and $(h : Y \to C, k : Y \to D)$ being $(f \times h : X \times Y \to A \times C, g \times k : X \times Y \to B \times D)$.

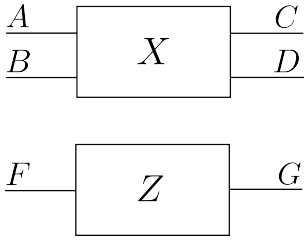
In [7] an informal geometric description (in the style of monoidal category string diagrams) was introduced for the operations in $\mathbf{Span}(\mathbf{C})$. For example, spans $(X \to A \times B, X \to C \times D)$, $(Y \to C \times D, Y \to E)$ and $(Z \to F, X \to G)$ are represented by pictures of *components with ports*:



Then the composite of the first two spans is pictured as:



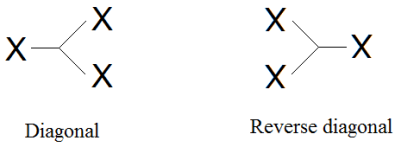
while the tensor of the first and third is pictured as:



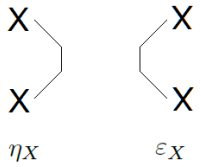
In addition there are constants of the algebra which are pictured as operation on wires which enable the depiction of fanning out of wires and feedback, and hence of general circuit diagrams. We give some examples of constants in $\mathbf{Span}(\mathbf{C})$ which are described also in [7].

- The identity span $1_X : X \longrightarrow X$ has head X and two legs $1_X, 1_X$. It is denoted by a plain wire.

- The span with head X and legs $1_X : X \longrightarrow X$, $\Delta_X : X \longrightarrow X \times X$ is called the diagonal of X and is denoted also $\Delta : X \longrightarrow X \times X$. The span with head X and legs $\Delta_X : X \longrightarrow X \times X$,$1_X : X \longrightarrow X$ is called the reverse diagonal, and is denoted $\nabla : X \times X \longrightarrow X$



- The span with head $X \times Y$ and legs $1_{X\times Y} : X \times Y \longrightarrow X \times Y$, $p_X : X \times Y \longrightarrow X$ is denoted p_X, is called a projection and is pictured by the termination of the wire Y. There is also a similarly defined reverse projection denoted p_X^*.
- Consider the terminal object, denoted I: in case of $\mathbf{C} = \mathbf{Graph}$, the terminal graph has one vertex and one edge, by necessity a loop. The span with head X and legs $! : X \longrightarrow I$, $\Delta_X : X \longrightarrow X \times X$ is called η_X. The span with head X and legs $\Delta_X : X \longrightarrow X \times X$, $! : X \longrightarrow I$ is called ε_X The two spans are pictured in the following figure:



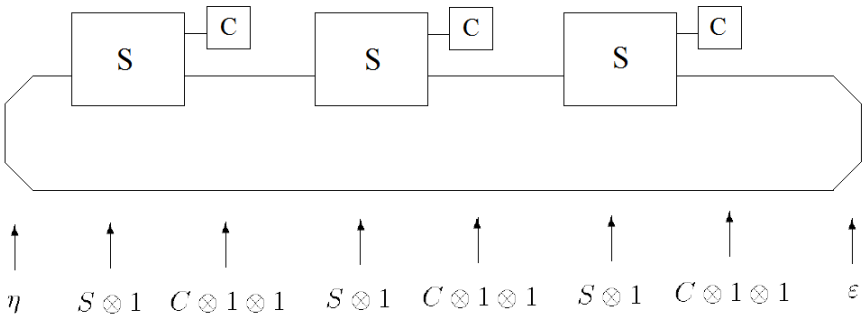
Technically, η and ε are the unit and counit of the self-dual compact-closed structure on $\mathbf{Span}(\mathbf{C})$. They permit a feedback operation on distributed systems.

The correspondence between constants and operations, and the geometric representations given above, result in the fact that expressions in the algebra have corresponding circuit or system diagrams. This is clarified in the following example.

▶ **Example 2.** Given spans $S : X \longrightarrow X \times X$ and $C : X \longrightarrow I$, the expression

$$\eta_X ; (S \otimes 1_X) ; (C \otimes 1_X \otimes 1_X) ; (S \otimes 1_X) ; (C \otimes 1_X \otimes 1_X) ; (S \otimes 1_X) ; (C \otimes 1_X \otimes 1_X) ; \varepsilon_X$$

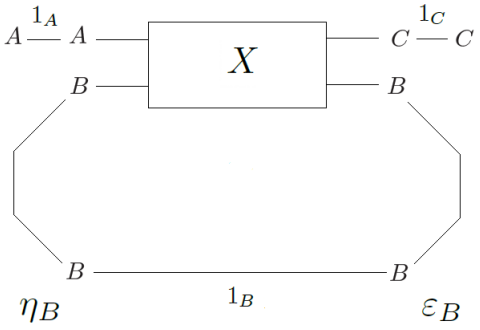
has system diagram (which graphically forms a feedback):



In the following, when we will consider **Span**(**Graph**) as an algebra of automata where parallel composition with or without communication are the main operations, we will use the expression *abstract parallel feedback* referring to the following definition (given by using the constants of **Span**(**C**)):

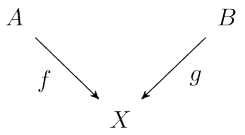
▶ **Definition 3.** Given a span $X : A \times B \longrightarrow C \times B$, we call *abstract parallel feedback* with respect to B, denoted by $AbPfb_B(X)$, the span denoted by the following algebraic expression:

$$(1_A \otimes \eta_B); (X \otimes 1_B); (1_C \otimes \varepsilon_B)$$

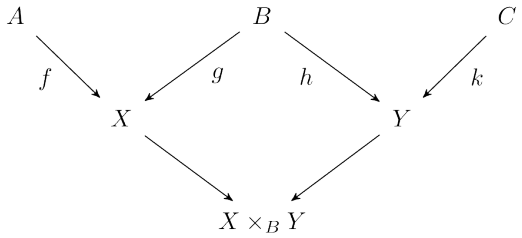


2.2 The algebra of cospans

▶ **Definition 4.** There is a dual construction **Cospan**(**C**) for categories **C** with finite colimits. In fact, **Cospan**(**C**) = **Span**($\mathbf{C}^{\mathbf{op}}$), but it is better seen describing explicitly its objects and arrows. Objects of **Cospan**(**C**) are the same as objects of **C**; arrows of **Cospan**(**C**) from A to B are cospans, that is, pairs of arrows $(f : A \to X, g : B \to X)$ of **C** with common codomain, also written as:

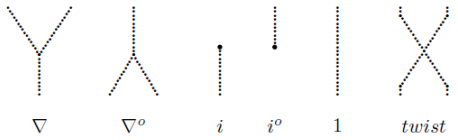


Composition (which is also called *restricted sum*) of $(f : A \to X, g : B \to X)$ and $(h : B \to Y, k : C \to Y)$ is by pushout (glued sum):



The identity cospan of object A is $(1_A, 1_A)$.

Again there are constants of the algebra which are pictured as operation on wires which enable the depiction of joining of wires and sequential feedback. Graphically, we can present these operations as the picture below:



As before, when we will consider **Cospan**(**Graph**) as an algebra of automata where sequential compositions are the main operations, we will use the expression *sequential feedback*.

2.3 Span(Graph), a parallel algebra of automata

Surprisingly, **Span**(**C**), when **C** is the category of (finite) directed graphs (**Span**(**Graph**)), provides a very natural mathematical framework for describing the composition of *automata with interfaces* (or communication ports, as in circuit theory). Consider a span of graphs $(\delta_0 : X \to A, \delta_1 : X \to B)$. The graph X may be considered as the graph of states and transitions of an automata with interfaces, and it is called the *head* of the span. The graph A is the graph of states and transitions of the combined left ports and B is the graph of states and transitions of the combined right ports. The graph morphism δ_0 associates to a state and to a transition of the automaton X the corresponding state and transition of the left ports A; the morphism δ_1 does the same for the right ports.

For all the examples of this paper the left and right ports have only one state so that we tend to ignore that; then δ_0 and δ_1 are a double labelling of the transitions of the automaton X by transitions on the left ports and transitions on the right ports. More intuitively each transition of the component has an effect on all its interfaces, maybe the null effect ε.

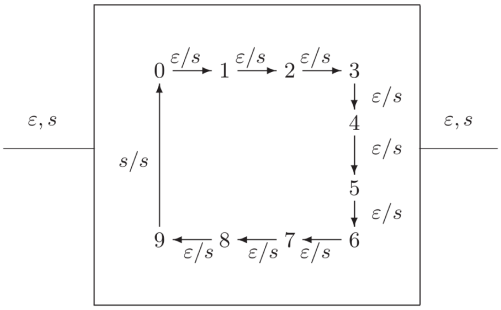
In the case that the left and right ports have one state, the operations of composition and tensor of spans have a simple description in terms of operations on automata. The tensor of two automata has states being pairs of states, one of each automata, and has as transitions pairs of transitions between the corresponding pairs of states. The composition of automata has similarly states being pairs of states, and transitions being pairs of transitions but *only those pairs of transitions whose labels on the connected ports are the same.* In the following, we will call the span composition *parallel composition with communication* and the tensor *parallel composition without communication.*

We give the definition of *parallel feedback* also in the concrete case of **Span**(**Graph**):

▶ **Definition 5.** Given a span of graphs $X : A \times B \longrightarrow C \times B$, we call *parallel feedback* with respect to B, denoted by $Pfb_B(X)$, the span of graphs obtained in the following way: the head of $Pfb_B(X)$ is the graph whose vertices are the vertices of X and whose arcs are the arcs x of X such that $(pr_B \circ \delta_2)(x) = (pr_B \circ \delta_1)(x)$; the interfaces of $Pfb_B(X)$ are the functions $pr_A \circ \delta_1$ and $pr_C \circ \delta_2$.

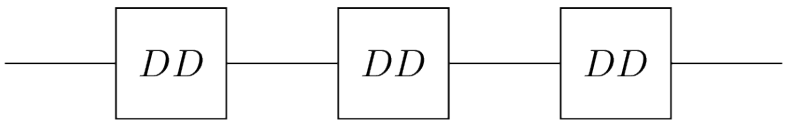
The diagrammatic representation of $Pfb_B(X)$ involves joining the right interface B to the left interface B.

▶ **Example 6** (Decimal Counter)**.** In the following example the left and right ports are both graphs with one state and two transitions ε and s which are displayed on the ports. The automaton has ten states and ten transition (in a circle through the states). The morphisms δ_0 and δ_1 are indicated by doubly-labelling the transitions of the automaton (so, for example, $\delta_0(0 \to 1) = \varepsilon$ and $\delta_1(0 \to 1) = s$).

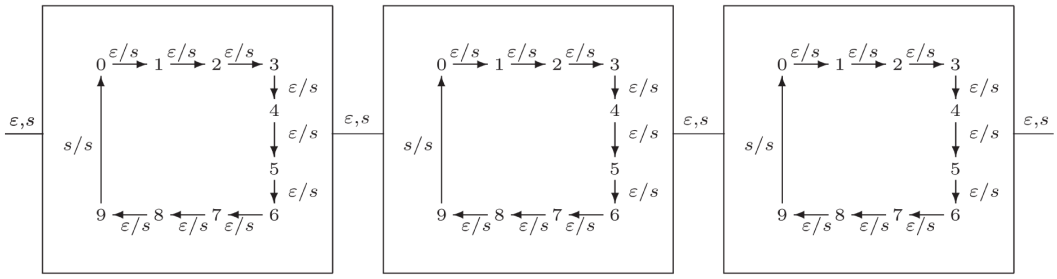


We are actually interested in a variation of this example which we shall call *DecimalDigit* or more shortly *DD* which has the same ports but in addition to the ten transitions above also ten loops one on each state, each labelled $(\varepsilon/\varepsilon)$.

Using composition in **Span**(**Graph**) we can form a new automaton *DecimalCounter* as the span composition of (for simplicity only) three *DD*'s:



or in more detail (though omitting the loops) as:



We notice that a state of *DecimalCounter* is a triple of states one of each DD, and a transition of *DecimalCounter* is a triple of simultaneous transitions with the property that the *labels on the connected ports agree*.

For example, starting in state $(0, 1, 8)$ the following is a sequence of transitions of *DecimalCounter*:

$$(0, 1, 8) \xrightarrow{(\varepsilon/s)} (0, 1, 9) \xrightarrow{(\varepsilon/\varepsilon)} (0, 1, 9) \xrightarrow{(\varepsilon/s)} (0, 2, 0)$$

(remember the ε/ε loops on each state of DD) so a transition s on the right-most port increments the counter. In fact, starting from state $(0, 0, 0)$, after 123 transitions labelled s on the right, the state of the network is $(1, 2, 3)$.

Notice that the transition

$$(9, 9, 9) \xrightarrow{(s/s)} (0, 0, 0)$$

occurs in one step.

▶ Remark. If one wants a parallel algebra of automata which also includes initial and final states of the automata, a slight variation of the above is possible: instead of the category of graphs take the category $(I + J)\backslash\mathbf{Graph}$ whose objects are graph morphisms $(I + J) \to X$, that is consist of two graph morphisms $I \to X$ and $J \to X$, to be thought of as the initial states and final states of X. Then the parallel algebra is $\mathbf{Span}((I + J)\backslash\mathbf{Graph})$.

▶ Remark. In [1], [2] a probabilistic version of $\mathbf{Span}(\mathbf{Graph})$ and $\mathbf{Cospan}(\mathbf{Graph})$ is described, and in that context communicating parallel composition involves conditional probability, arising from the fact that incompatible transition cannot occur and hence probabilities must be normalized. This provides a generalization of Rabin and Segala-Lynch probabilistic automata and this yields a compositional theory of Markov chains.

▶ Remark. In [5] *timed actions* with different duration have been considered. Composition is obtained by considering a linear (w.r.t. transitions) number of extra "internal" states. The intended meaning is that a component that interacts with a "faster" one could be still doing an action (hence being in an internal state) when the other one has completed the transition. We give a simple example:

▶ Example 7 (Two actions which synchronize with an arbitrary duration). Consider two automata $\mathcal{G}_1$, $\mathcal{G}_2$ and two "non-atomic" actions, one of $\mathcal{G}_1$, one of $\mathcal{G}_2$. The action of $\mathcal{G}_1$ is $\{a/b : 0 \to 1, \varepsilon/\varepsilon : 1 \to 1, d/e : 1 \to 2\}$; the action of $\mathcal{G}_2$ is $\{b/c : 0 \to 1, \varepsilon/\varepsilon : 1 \to 1, e/f : 1 \to 2\}$. In the restricted product $\mathcal{G}_1 \cdot \mathcal{G}_2$ these actions synchronize but with arbitrary duration. A typical behaviour is $(0, 0) - a/c \to (1, 1) - \varepsilon/\varepsilon \to (1, 1) - \varepsilon/\varepsilon \to ... - \varepsilon/\varepsilon \to (1, 1) - d/f \to (2, 2)$.

Note. Simple modifications of the algebra allow the description of networks in which the tangling of connectors may be represented, yielding a connection with the theory of knots [11].

2.4 Cospan(Graph), a sequential algebra of automata

Analogously, when $\mathbf{C}$ is the the category of (finite) directed graphs, $\mathbf{Cospan}(\mathbf{Graph})$ provides a sequential calculus for automata that generalizes Kleene's one. Consider a cospan of graphs $(\gamma_0 : E \to X, \gamma_1 : F \to X)$. The graph X may be considered as the graph of states and transitions of an (unlabelled) automaton. The graph E is the graph of *initial* states and transitions and F is the graph of *final* states and transitions. In all the examples considered E and F have only states and not transitions. The graph morphisms γ_0 and γ_1 are often inclusion morphisms of the initial and final states in X.

▶ Remark. If one wants a sequential algebra of automata which have labelled transitions, a slight variation of the above is possible: instead of the category of graphs take the category $\mathbf{Graph}/(A \times B)$ whose objects are graph morphisms $X \to A \times B$, that is consist of two graph morphisms $X \to A$ and $X \to B$, to be thought of as the left and right labellings of X. Then the sequential algebra is $\mathbf{Cospan}(\mathbf{Graph}/(A \times B))$.

2.5 Cospans and spans of graphs

The two algebras we have described may be combined in a natural way. Consider four graph morphisms $(\delta_0 : X \to A, \delta_1 : X \to B, \gamma_0 : E \to X, \gamma_1 : F \to X)$. From these we may obtain an arrow in the parallel algebra $\mathbf{Span}(\mathbf{Graph})\backslash(E + F)$, and also in the sequential algebra $\mathbf{Cospan}(\mathbf{Graph}/(A \times B))$, and hence we may apply both sequential and parallel operations to such automata, obtaining hierarchical nets of automata with evolving geometry. There is a distributive law of parallel composition over sequential, that will be used in the following example. For some details of this see [9].

▶ **Example 8** (Distributed Sort Algorithm)**.** In [9], it has been described in full details an example of reconfigurable network, that is a Distributed Sort Algorithm: an atomic sort A receives a stream of items to be sorted; if the atomic sort gets full, a new network gets activeted in which a divert component D, two atomic sorts A and the merge component M act in parallel. The whole system is the solution of a recursive equation

$$S = A + D \cdot (A \times S) \cdot M$$

which is expanded using the *distributive laws* (between products and the restricted sums) in the following way

$$S = A + D \cdot (A \times S) \cdot M$$

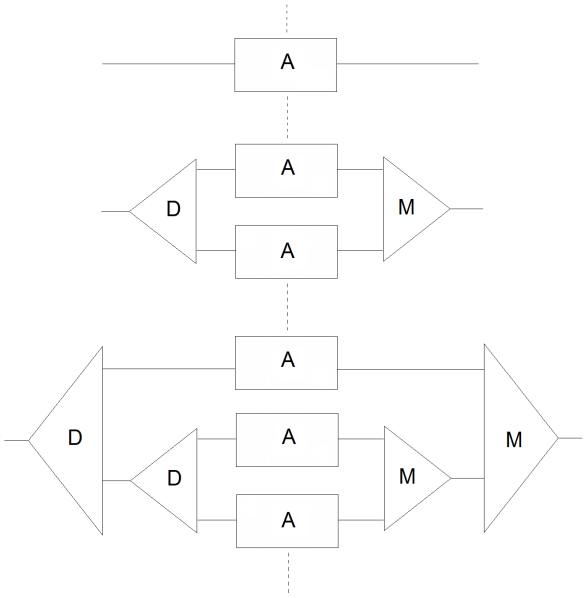
$$= A + D \cdot (A \times (A + D \cdot (A \times S) \cdot M)) \cdot M$$

$$= A + D \cdot (A \times A) \cdot M + D(\cdot(A \times D \cdot (A \times S) \cdot M)) \cdot M$$

$$= .....$$

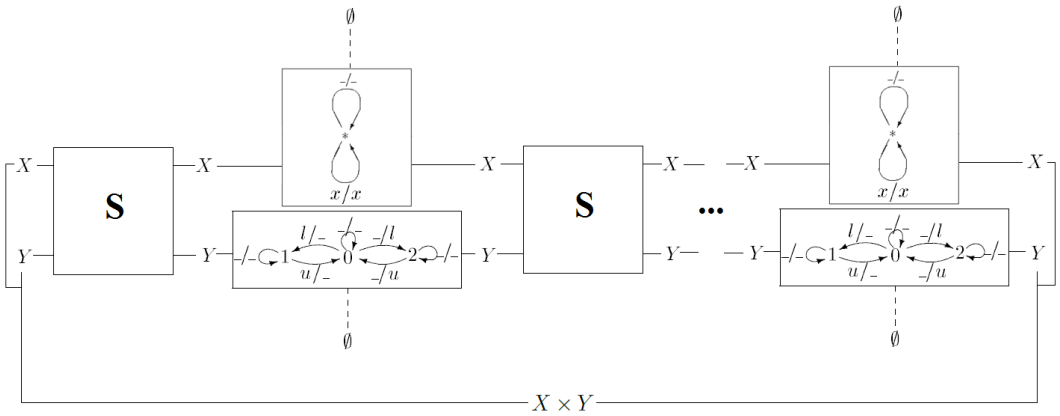
$$= A + D \cdot (A \times A) \cdot M + D(\cdot(A \times D \cdot (A \times A) \cdot M)) \cdot M + ...$$

In this equation the variables are Cospan/Span automata. It gives rise to a network that can be graphically represented as follows:



Now, we are going to see a concrete example of parallel and sequential feedbacks in **Cospan/Span(Graph)**.

▶ **Example 9** (Sofia's Birthday Party)**.** This example [9] is a generalization of the usual Dining Philosophers Problem: instead of a circle of n philosophers around a table with as many forks, we consider a circle of n seats around a table separated by forks. There are $k \leq n$ children: the protocol of each child is the same of a philosopher, but, in addition, if a child is not holding a fork and has an empty seat on the right, he can change seats. In the following picture we give a global representation of the automaton, in which the component S can either be a child (on a seat) or an empty seat:



3 Networks

3.1 Closed networks

In [12] the notion of closed network is formalized by the concept of monoidal graph given in the following definition:

▶ **Definition 10.** A *monoidal graph* $\mathbf{M}$ consists of two finite sets M_0 (wires) and M_1 (components) and two functions $domain : M_1 \to M_0^*$ and $codomain : M_1 \to M_0^*$ where M_0^* is the free monoid on M_0. We call the monoidal graph $\mathbf{M}$ *discrete* when $M_1 = \emptyset$.

▶ Remark. Directed graphs are examples of monoidal graphs in which the domain and codomain functions land in M_0 rather than M_0^*. For these it is usual to speak of elements of M_1 as edges or arcs, and elements of M_0 as vertices. We observe that language of components and wires is more suitable when the domain and codomain functions are words rather than single letters.

An example with $M_1{=}\{A_1,B_1,C_1,D_1,A_2,B_2,C_2,D_2,E\}$ and $M_0{=}\{X_1,Y_1,Z_1,W_1,X_2,Y_2,Z_2,W_2\}$ is:

$A_1 : X_1 \to Y_1$ $\quad$ $B_1 : Y_1 \to Z_1$

$C_1 : Z_1 \to W_1$ $\quad$ $D_1 : W_1 \to X_1$

$A_2 : X_2 \to Y_2$ $\quad$ $B_2 : Y_2 \to Z_2$

$C_2 : Z_2 \to W_2$ $\quad$ $D_2 : W_2 \to X_2$

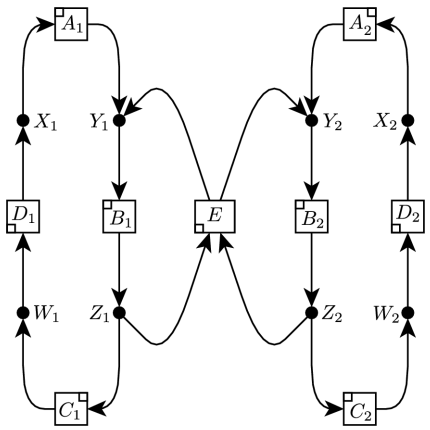
$E : Z_1 Z_2 \to Y_1 Y_2$

▶ **Definition 11.** If $A : X_1 X_2 \cdots X_m \to Y_1 Y_2 \cdots Y_n$ is a component of monoidal graph $\mathbf{M}$, then we call the elements of the set $\{1, 2, \cdots, m\}$ the *left-hand ports* of A, and the elements of $\{1, 2, \cdots, n\}$ the *right-hand ports* of A. Notice that either the set of left-hand ports or of right-hand ports may be empty.

▶ Remark. We insist that, as in **Span**(**Graph**), the left ports and right ports do not have an interpretation as input and output ports respectively. However, in the context of this paper we shall also use the term *input port* for a left-hand port, and *output port* for a right-hand port. For this reason we also indicate input ports by arrows on wires entering the component, and output ports by arrows on wires exiting the component.

The notion of ports of a component gives rise to a different *geometric* representation of monoidal graph which explains why we regard such a graph as a closed network. The elements of M_1 are represented as components with ports, the elements of M_0 are represented as connectors or wires, and the domain and codomain functions describe how the ports are joined to the connectors.

The order on the ports is indicated by a small square in the component nearest to the first left-hand port (in the example below, see in particular the component $E : Z_1 Z_2 \to Y_1 Y_2$). The right-hand ports are taken in the same order as the left-hand ports.



Note. After this section we shall usually omit the indicator of the order of the ports, where the intended order is clear.

▶ Remark. It might be objected that the simple monoidal graph with two wires X, Y and one component $A : X \to Y$ should not be considered as a *closed* network. However, while the component A has left and right ports, the monoidal graph itself does not have specified left and right ports. We will see when we define open networks in the next section that this graph may have external ports specified in many different ways.

We will need later the definition of morphism of monoidal graph.

▶ **Definition 12.** A morphism α from monoidal graph $\mathbf{M}$ to $\mathbf{N}$ consists of two functions $\alpha_1 : M_1 \to N_1$ and $\alpha_0 : M_0 \to N_0$ such that $domain_N \circ \alpha_1 = (\alpha_0)^* \circ domain_M$ and $codomain_N \circ \alpha_1 = (\alpha_0)^* \circ codomain_M$, where $(\alpha_0)^*$ is the monoid homomorphism between free monoids induced by the function α_0.

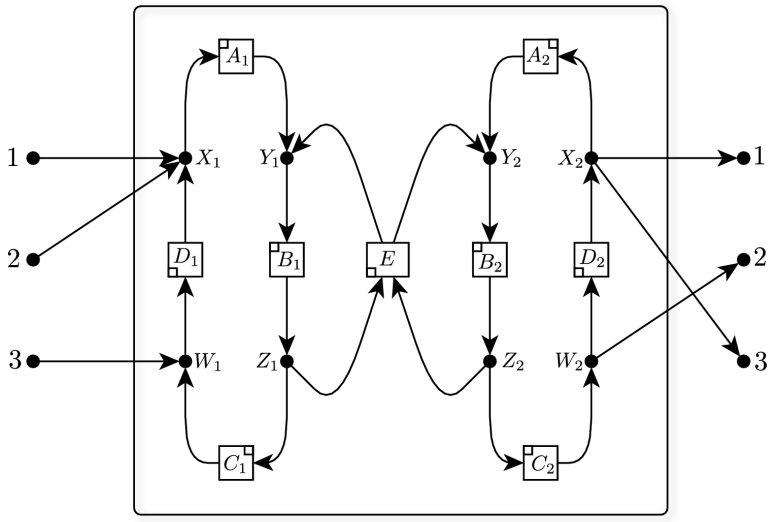
▶ Remark. It is not difficult to verify that monoidal graphs with these morphisms form a category, denoted **MonGraph**, which is in fact a presheaf topos.

3.2 Open networks

In order to show that the globally described notion of network may be also described compositionally, we need a notion of *open* network, which is introduced in [12]. An open network is a monoidal graph together with left and right interfaces. The formal definition is as follows:

▶ **Definition 13.** An open network consists of a monoidal graph $\mathbf{M}$, two sets $S = \{1, 2, \cdots, m\}$ and $T = \{1, 2, \cdots, n\}$ and two functions $\gamma_0 : S \to M_0$, $\gamma_1 : T \to M_0$. We denote the open network as $\mathbf{M} : S \to T$. If both sets S and T are empty, the only content of the network is the monoidal graph, that is, the network is closed.

We sketch an example in which the sets $S = T = \{1, 2, 3\}$. Notice it has the same form as a single component.



An open network has a simple standard categorical description. It is a *cospan of monoidal graphs* between discrete monoidal graphs.

3.3 The algebra of open networks

The algebraic structure that cospans (open networks) admit can be described in the following way: they form the arrows of *a symmetric monoidal category in which every object has a commutative separable algebra structure compatible with the tensor* [4], [8] - what is called Well-Supported-Compact-Closed categories. We first give a definition of WSCC category assuming knowledge of symmetric monoidal categories.

▶ **Definition 14.** A commutative separable algebra in a symmetric monoidal category consists of an object X and four arrows $\nabla : X \otimes X \to X$, $\Delta : X \to X \otimes X$, $n : I \to X$ and $e : X \to I$ making (X, ∇, n) a commutative monoid, (X, Δ, e) a cocommutative comonoid and satisfying the equations

$$(1_X \otimes \nabla)(\Delta \otimes 1_X) = \Delta\nabla = (\nabla \otimes 1_X)(1_X \otimes \Delta) : X \otimes X \to X \otimes X$$
$$\nabla\Delta = 1_X.$$

There are two important derived operations of a commutative separable algebra X, namely $\epsilon : X \otimes X \to I = e\nabla$ and $\eta : I \to X \otimes X = \Delta n$.

▶ Proposition 1. Given an object X with a commutative separable algebra structure the arrows $\eta : I \to X \otimes X$ and $\epsilon : X \otimes X \to I$ satisfy the equations (where τ is the twist of the symmetry):

(i) $(\epsilon \otimes 1_X)(1_X \otimes \eta) = 1_X = (1_X \otimes \epsilon)(\eta \otimes 1_X)$

(ii) $\epsilon\tau = \epsilon$ and $\tau\eta = \eta$.

Proof. We prove only (i), as done in [12]. To see that $(\epsilon \otimes 1_X)(1_X \otimes \eta) = 1_X$ notice that

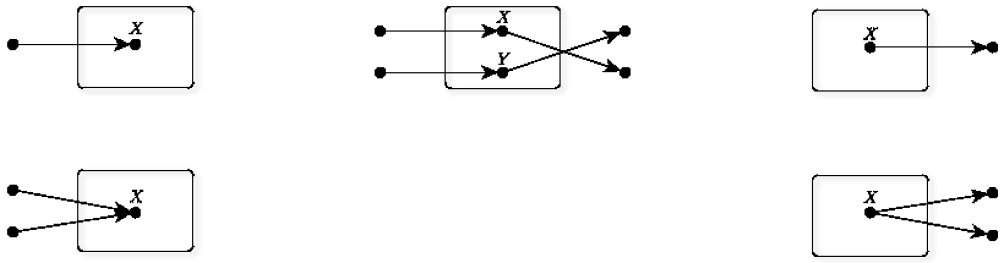
$$(e \otimes 1_X)(\nabla \otimes 1_X)(1_X \otimes \Delta)(1_X \otimes n) = (e \otimes 1_X)\Delta\nabla(1_X \otimes n) = 1_X \cdot 1_X = 1_X;$$

The first equality is an application of the first displayed axiom of definition 2.5 in [8]. The second equality comes from the monoid and comonoid axioms. ◀

▶ Remark. Axiom (i) says that X is a *self-dual object*. The reader can translate these into equations between *string* diagrams, which are representations of expressions. An example is the string diagram for (i):

▶ **Definition 15** (The operations of the algebra of open networks)**.** The open network composition of $\mathbf{M} : S \to T$ with $\mathbf{N} : T \to R$ is composition of cospans of monoidal graphs; that is, the components of the composite are the disjoint union of the components of $\mathbf{M}$ and $\mathbf{N}$ whereas the wires of the composite are the quotient of the disjoint union of the wires of $\mathbf{M}$ and $\mathbf{N}$ obtained by equating the images of elements of T (under γ_1) in $\mathbf{M}$ with the corresponding images of the elements of T (under γ_0) in $\mathbf{N}$. The open network tensor of $\mathbf{M}$ and $\mathbf{N}$ is formed by taking disjoint unions of both wires and components.

Similarly, the constants of the algebra of networks can be given by using the constants of the algebra of the monoidal graph. We don't give a precise definitions but we present them by picturing.

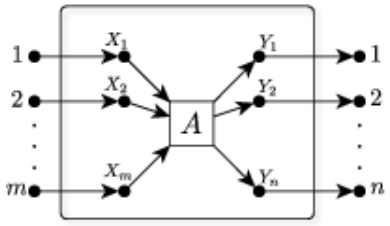


3.4 The algebra of open networks is free

▶ **Theorem 16.** *Given a monoidal graph* **M***, the free symmetric strict monoidal category constructed from* **M** *adjoining commutative separable algebra structures to the objects of* **M** *is the full subcategory of* **Cospan**(**MonGraph**/**M**) *whose objects are discrete monoidal graphs over (labelled in)* **M***.*

Proof. In [10] the special case for graphs rather than monoidal graphs is proved by demonstrating (in Proposition 3.3 of that paper) a normal form for arrows in the free category (with structure). The proof of the theorem that we are describing here (even if it is more general of the one in [10]) requires just a straightforward modification of the normal form in which a sum of edges is replaced by a sum of components of **M**. ◀

We denote this full subcategory as **Csp**(**MonGraph**/**M**). The components of **M** lie in **Csp**(**MonGraph**/**M**) as cospans as follows: if $A : X_1X_2\cdots X_m \to Y_1Y_2\cdots Y_n$ is a component of **M**, then the corresponding cospan is



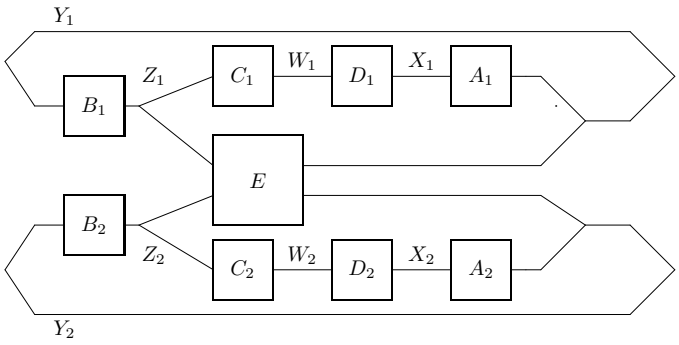
where the elements A, $X_1, X_2, \cdots, X_m$, $Y_1, Y_2, \cdots, Y_n$ are now *labels* - that is, if in the original monoidal graph $X_1 = X_2$ then in the cospan the corresponding wires are distinct but have the same label $X_1 = X_2$ in **M**.

▶ Remark. The first practical use of this theorem is that from the components of **M** we can generate any open (or closed) monoidal graph containing these components by evaluating an expression in the algebra **Csp**(**MonGraph**/**M**) starting from single components.

As an example we describe the monoidal graph **M** of section 3.1 as an expression in **Csp**(**MonGraph**/**M**) in terms of single components. The following expression evaluates to **M**:

$$(\epsilon_{Y_1}\otimes\epsilon_{Y_2})(1_{Y_1}\otimes((\nabla_{Y_1}\otimes\nabla_{Y_2})(A_1D_1C_1\otimes E\otimes A_2D_2C_2)(\Delta_{Z_1}B_1\otimes\Delta_{Z_2}B_2))\otimes 1_{Y_2})(\eta_{Y_1}\otimes\eta_{Y_2})$$

as can be seen by examining the string diagram for this expression:



▶ Remark. A theorem similar to Theorem 1, for ordinary graphs only, was described by Gadducci, Heckel and Llabres [6] for application to graph rewriting.

4 Networks with state

The aim of this section is to attach automata to each of the components and wires of a monoidal graph; that is, to add states and state transitions to each part of the network, as done in [12].

4.1 Adding state to networks

A cospan of monoidal graphs with these extra labelling automata is what we use to model *open networks with state*.

Again this has a standard categorical description. Consider the monoidal category **Span**(**Graph**): composition is pullback, tensor is product. **Span**(**Graph**) is a WSCC category, but now $\Delta : X \to X \times X$ is the span with left-leg the identity and right-leg the diagonal of the product, and $\nabla : X \times X \to X$ is the reverse of the span Δ, as defined in Section 2.1. Let $|\mathbf{Span}(\mathbf{Graph})|$ be the (large) monoidal graph whose wires are the objects of **Span**(**Graph**) (that is, graphs) and whose components are spans of graph morphisms between products of graphs. Then we have the following definition:

▶ **Definition 17.** An open network with state is an arrow in the monoidal category

$$\mathbf{Csp}(\mathbf{MonGraph}/|\mathbf{Span}(\mathbf{Graph})|).$$

Hence, a closed network with state is a morphism of monoidal graphs from **MonGraph** to $|\mathbf{Span}(\mathbf{Graph})|$.

Note that $\mathbf{Csp}(\mathbf{MonGraph}/|\mathbf{Span}(\mathbf{Graph})|)$ is also a WSCC category.

The definition simply means that associated with each wire of an open network there is a graph, and to each component $A : X_1X_2 \cdots X_m \to Y_1Y_2 \cdots Y_n$ a span of graphs between the products of the graphs labelling the wires.

▶ **Definition 18.** The process of forming the global state space of a network with state is the functor preserving the WSCC structure

$$\mathit{globalstate} : \mathbf{Csp}(\mathbf{MonGraph}/|\mathbf{Span}(\mathbf{Graph})|) \longrightarrow \mathbf{Span}(\mathbf{Graph}),$$

which is induced, using the freeness of the domain category, from the inclusion of the components of $|\mathbf{Span}(\mathbf{Graph})|$ in **Span**(**Graph**).

▶ **Theorem 19.** *The functor globalstate can be obtained by calculating a global* limit *in* **Graph**.

A sketch of the proof of the previous Theorem can be found in [12].

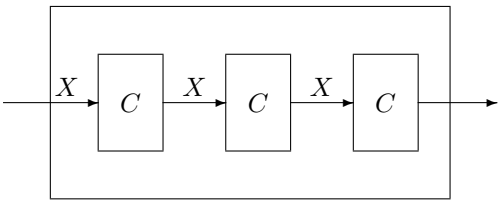
▶ **Corollary 20.** *The systems described by networks with state are the same as* **Span**(**Graph**) *systems.*

Proof. Consider the functor

$$globalstate : \mathbf{Csp}(\mathbf{MonGraph}/|\mathbf{Span}(\mathbf{Graph})|) \longrightarrow \mathbf{Span}(\mathbf{Graph}).$$

An arrow in the domain may be thought of as either (i) algebraically, *an expression in* **Span**(**Graph**) *(the compositional view of a system)* or (ii) geometrically, *a network (cospan of monoidal graphs) labelled in graphs.* The functor *globalstate* may be obtained by calculating either (i) by evaluating the expression in **Span**(**Graph**) or (ii) by calculating a global limit. ◀

▶ **Example 21.** We describe *DecimalCounter*, the decimal counter with three digits given above, in this context. Consider the monoidal graph $\mathbf{M}$ with one component C and one wire X with $C : X \to X$. Then, arrows in $\mathbf{Csp}(\mathbf{MonGraph}/\mathbf{M})$ are open monoidal graphs built out of the open monoidal graph corresponding to this component. The example we wish to consider is the composite $CCC : X \to X$ in $\mathbf{Csp}(\mathbf{MonGraph}/\mathbf{M})$ which is an open monoidal graph with the following graphical representation



Now, by the freeness of $\mathbf{Csp}(\mathbf{MonGraph}/\mathbf{M})$, a monoidal graph morphism from $\mathbf{M}$ to $|\mathbf{Span}(\mathbf{Graph})|$ induces a structure preserving functor

$$\mathbf{Csp}(\mathbf{MonGraph}/\mathbf{M}) \to \mathbf{Csp}(\mathbf{MonGraph}/|\mathbf{Span}(\mathbf{Graph})|)$$

which takes the open monoidal graph $CCC : X \to X$ to a network with state. To describe the decimal counter we choose the following morphism of monoidal graphs $\mathbf{M} \to |\mathbf{Span}(\mathbf{Graph})|$: X goes to the graph with one vertex and edges ε, s and C goes to the span of graphs DD.

The decimal counter now consists of three such automata joined in parallel according to the pattern of the open monoidal graph $CCC : X \to X$.

Acknowledgements. To Bob.

References

1. L. d. F. Albasini, N. Sabadini, and R. F. C. Walters. The compositional construction of Markov processes. *ArXiv e-prints*, January 2009. `arXiv:0901.2434`.
2. L. d. F. Albasini, N. Sabadini, and R. F. C. Walters. The compositional construction of Markov processes II. *ArXiv e-prints*, May 2010. `arXiv:1005.0949`.
3. J. Bènabou. Introduction to Bicategories. *Reports of the Midwest Category Seminar*, 47:1–77, 1967. `doi:10.1007/BFb0074299`.
4. A. Carboni and R.F.C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49(1):11 – 32, 1987.
5. A. Cherubini, N. Sabadini, and R. F. C. Walters. Timing in the cospan-span model. *Electr. Notes Theor. Comput. Sci.*, 104:81–97, 2004.
6. F. Gadducci, R. Heckel, and M. Llabrés. A bi-categorical axiomatisation of concurrent graph rewriting. *Electronic Notes in Theoretical Computer Science*, 29:80 – 100, 1999. CTCS '99, Conference on Category Theory and Computer Science.
7. P. Katis, N. Sabadini, and R. F. C. Walters. Span(graph): A categorial algebra of transition systems. In *Proceedings of AMAST '97*, pages 307–321, 1997.
8. P. Katis, N. Sabadini, and R.F.C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115:141–178, 1997.
9. P. Katis, N. Sabadini, and R.F.C. Walters. A formalization of the IWIM model. In *Proceedings of Coordination Languages and Models 2000*, volume 1906 of *LNCS*, pages 267–283. Springer, 2000.
10. R. Rosebrugh, N. Sabadini, and R.F.C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and Applications of Categories*, 15(6):164–177, 2005.
11. R. Rosebrugh, N. Sabadini, and R.F.C. Walters. Tangled circuits. *Theory and Applications of Categories*, 26(27):743–767, 2012.
12. N. Sabadini, F. Schiavio, and R.F.C. Walters. On the geometry and algebra of networks with state. *Theor. Comput. Sci.*, 64:144–163, 2017.