Theoretical Computer Science 664 (2017) 144–163

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

# On the geometry and algebra of networks with state

N. Sabadini, F. Schiavio *, R.F.C. Walters

*Dipartimento di Scienza e Alta Tecnologia, Università dell' Insubria, via Valleggio, 11, Como, Italy*

ARTICLE INFO

*Article history:*
Received 10 March 2015
Received in revised form 30 November 2015
Accepted 22 January 2016
Available online 25 January 2016

*Keywords:*
Networks
Petri nets
Monoidal categories
Spans of graphs

ABSTRACT

In [1] an algebra of automata with interfaces, **Span**(**Graph**), was introduced with main operation being communicating-parallel composition – a system is represented by an expression in this algebra. 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**). Note that **Span**(**Graph**) yields purely compositional descriptions of systems.

In this paper we give an alternative globally (non-compositional) view of the same systems. In order to do this we make mathematically precise the network geometry in terms of monoidal graphs, and assignment of state in terms of morphisms of monoidal graphs. We call such globally described systems *networks with state*. To relate networks with state and **Span**(**Graph**) systems we use the algebra of cospans of monoidal graphs.

As an illustration we give a new representation of a (non-compositionally described) class of Petri nets in the compositional setting of **Span**(**Graph**). We also present a tool for calculating with networks with state.

Both algebras, of spans and of cospans, are symmetric monoidal categories with commutative separable algebra structures on the objects.

We include a short section, following [2], in which we show that a simple modification of the algebra allows the description of networks in which the tangling of connectors may be represented, yielding a connection with the theory of knots.



## 1. Introduction

In 1997 Katis, Sabadini and Walters introduced a parallel algebra of automata, **Span**(**Graph**), in [1] and at the same conference [3] showed how P/T nets could be embedded in this algebra. In 2000 [4] the same authors introduced a related sequential algebra of automata, namely **Cospan**(**Graph**) and showed how the two algebras combined could model hierarchical systems with evolving geometry. Many of the remarks we make in this paper apply both to **Span**(**Graph**) and **Cospan**(**Graph**) though we concentrate in this paper on the parallel algebra.

A system described in **Span**(**Graph**) (which we will call a **Span**(**Graph**) system) has two aspects: (i) an algebraic expression and (ii) its evaluation in **Span**(**Graph**). In [1] we showed how an algebraic expression yields an informal network geometry, and its evaluation an automaton of states and transitions of the system.

It is intrinsic to **Span**(**Graph**) systems that their geometry and state space are given compositionally. The aim of this paper is to give an alternative description of **Span**(**Graph**) systems, both of their network geometry and of their state space, a description which is global and non-compositional.

* Corresponding author.
*E-mail addresses:* nicoletta.sabadini@uninsubria.it (N. Sabadini), filippo.schiavio@gmail.com (F. Schiavio).

http://dx.doi.org/10.1016/j.tcs.2016.01.028
0304-3975/

**Remark 1.** All the systems we will consider are described at level of their control structure. Hence will be assumed finite, that is have a finite set of states and transitions. Moreover they will have only a finite number of components.

### 1.1. The global geometry of **Span**(**Graph**) systems

The algebra **Span**(**Graph**) is a symmetric monoidal category with extra structure and the informal pictorial representations of expressions in the algebra given in [1] followed the string diagrams [5] for expressions in monoidal categories. The diagrams used were not however formally justified, and shared the defect of string diagrams of being *progressive*, that is that composition is done from left to right (see Caveat 3.2 of [5]). This means that the diagrams are very close to the algebraic expressions they represent. The natural geometry of systems does not have this simple form – consider, for example, the geometry of a Petri net or of a circuit diagram. We give here instead a precise mathematical formulation of the global (non-compositional) geometry of **Span**(**Graph**) systems in terms of monoidal graphs which we believe correspond very well to natural system geometry.

The one possible objection to monoidal graphs as network geometries is that components have two sides (instead of $n$ sides); this arises from the fact that we need to relate the global geometry to the algebra **Span**(**Graph**) which has nullary, unary, and binary operations.

### 1.2. The global state space of **Span**(**Graph**) systems

The second contribution of this paper is to show how not only the geometry but also the state space of a **Span**(**Graph**) system may be given globally and non-compositionally. We define what we call *networks with state* to be a morphism of monoidal graph from the network geometry to the large monoidal graph of Spans of graphs. This amounts to the assignment of state and transitions to each component and each connector of the network in a consistent way. The global space of states and transitions is then given by a limit.

### 1.3. Relating global with compositional

In order to prove that systems described globally as networks with state are the same as compositional **Span**(**Graph**) systems we need of course to describe networks with state compositionally. This involves introducing open networks (without states) as certain cospans of monoidal graphs and their algebra, as well as open networks with state.

Once we have made the connection we are able to give, as an illustration, a new representation of a (non-compositionally defined) class of Petri nets, C/E nets, as **Span**(**Graph**) systems, a representation more faithful than that of [3] to the geometry of Petri nets.

The separation of the two aspects of **Span**(**Graph**) systems into their geometry and their state space also permits us to give an efficient tool for calculating random behaviours of **Span**(**Graph**) systems.

### 1.4. Earlier work

The geometry of monoidal categories began with Penrose [6], and Joyal and Street [7] and is surveyed in [5] (though that papers does not consider the structure here discussed). The two main algebras of this paper, Spans and Cospans, were introduced by J. Benabou [8]. The algebra of **Span**(**Graph**), symmetric monoidal categories in which each object has a commutative separable algebra structure compatible with the tensor product, what have been called elsewhere [9] WSCC-categories (WSCC=well-supported compact closed) has been studied in detail by Sabadini and Walters with collaborators Katis and Rosebrugh, especially in [1,10,4,11–13], beginning with the work on relations with Carboni [14] in 1987.

The work has numerous antecedents in computer science – we mention just S. Eilenberg [15], S.L. Bloom, Z. Esik [16], Gh. Stefanescu [17]. The algebra has connections with quantum field theory [18], and as we will describe in this paper with the theory of knots.

There are many other models of networks consisting of components and connectors (see the introduction to [19] and references therein, though it fails to mention [1] which is earlier than most of the other models and influenced some) but they lack the connections with algebra, geometry and physics.

Other authors, like Baez, Fong, and collaborators, used an approach similar to our in order to describe an algebra of networks applied to control theory and circuit diagrams (see for example [20,21] and [22]).

### 1.5. Outline of the paper

Since to read the present paper it is crucial to have a thorough understanding of how systems are modelled in [1] we begin in the Section 2 with an informal review of the **Span**(**Graph**) model as well as of the related sequential algebra **Cospan**(**Graph**) [4]. The elements of **Span**(**Graph**) are *spans of graphs*, and the main operation is communicating-parallel composition of automata; the elements of **Cospan**(**Graph**) are *cospans of graphs* and the main operation is sequential composition of automata.

Notice that in speaking of the category of cospans we should consider cospans only up to an isomorphism of the central graph of the cospan. In practice we will always consider representative cospans, and any equation we state will be true only up to isomorphism. The same proviso should be applied to our discussion later of spans, and systems.

In Section 3 we introduce the abstract notion of monoidal graph, a notion already explicit for example in [7], which is the notion we use to represent closed networks. We describe a way of picturing monoidal graphs as nets of components with *ordered* sets of (left-hand and right-hand) ports which are joined by wires. We then introduce certain cospans of monoidal graphs which we use to represent open networks. We describe explicitly the algebra of open networks, which as mentioned in 1.1 is a symmetric monoidal category whose objects have commutative separable algebra structures. Finally we give the theorem that the algebra of open networks is free and hence any open system may be written as an expression in the algebra in terms of constants and components.

In the Section 4 we define network with state. The technical definition requires morphisms of monoidal graphs, but may be described intuitively as follows: an open *network with state* is an open network each wire of which has an associated graph, (and hence each port of a component has an associated graph), and each component of which has a span of graphs from the product of graphs of the left-hand ports to the product of graphs of the right-hand ports. We describe the global states space of an open net with state as a certain limit of the state space of the components, a definition which agrees with the usual engineering view of a circuit. The equivalence of networks with state and **Span**(**Graph**) systems is shown here, we illustrate with examples.

In the same section we show how to model C/E nets by particular networks with state, giving some simple examples including a mutual exclusion Petri net. We would like to make clear some perhaps unconventional aspects of our modelling. Firstly we consider concurrent execution of the C/E net since we believe this is the natural semantics (if two independent events have conditions satisfied we see no reason why both should not occur together). Interleaving seems to us an unnatural semantics which however we could model by introducing a non-deterministic scheduler component. The second unusual aspect is that we put a spatial ordering on incoming arcs to a transition (and also on outgoing arcs). This in no way changes the modelling possibilities or behaviours of C/E nets, but does have the advantage of connecting net theory with other parts of mathematics and physics: we have included Section 6 on tangles to emphasize this. We believe that the introduction of commutative monoidal categories (rather than symmetric monoidal categories) in [23] was a mistaken direction. In an example we indicate another problem we see with Petri nets, namely that sequential processes are represented in terms of parallel components leading to unnecessary state explosion.

In Section 5 we describe a tool, written by Filippo Schiavio, for executing random behaviours in nets with state.

Section 6 reviews material taken from an earlier mathematical paper [2] which shows that a closely related algebra allows the description of networks in which the tangling of connections may be represented. The only new material in this section is an example of the correspondence between geometric and algebraic proofs of the equality of tangled networks. We include this material to show the connection of the network-with-state model to geometry and physics, indicating its mathematical maturity.

Finally, the Section 7 contains a summary and some indications of further research questions.

## 2. Review of the Span and Cospan algebras of automata

We give an informal review of the Span and Cospan algebras of automata (we suggest that for a better comprehension of this paper and for more details it is advisable to see the original papers). The algebra **Cospan**(**Graph**) was introduced in [4] as a sequential algebra for automata, with essentially the Kleene operations of sequential composition, choice and iteration. What is missing in Kleene is the operation of communicating parallel composition, and joint actions, fundamental to describe any mechanism. Consider for example how to gearwheels interact during their movement by simultaneous change of their state

The algebra **Span**(**Graph**) was introduced in [1] as a "parallel" algebra for automata. 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 must 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.

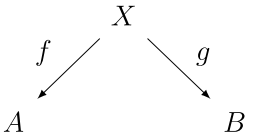
In both cases the algebra involved is a well-known one introduced by Jean Benabou in 1967 [8] and developed in 1987 by Carboni, Walters [14] as an algebra of relations, which provides a natural mathematical framework for describing nets of automata, both graphically with circuit-like diagrams, and as terms in a suitable algebra.

We describe first the abstract algebra of Benabou.

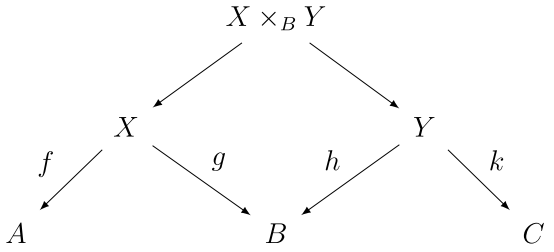
### 2.1. The algebra of spans

**Definition 1.** Give a category **C** with finite limits we define a new category **Span**(**C**) as follows: 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:

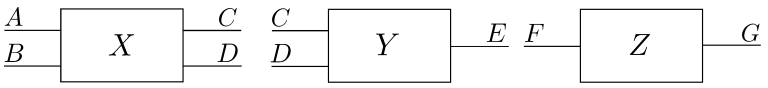



Composition of spans $(f : X \to A, g : X \to B)$ and $(k : Y \to B, h : Y \to C)$ is by pullback (restricted product):

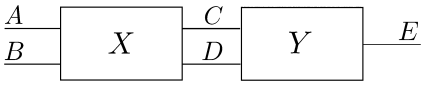



The identity span of object $A$ is $(1_A, 1_A)$. The category **Span**(**C**) is actually symmetric monoidal with the tensor of two spans $(f : X \to A, g : X \to B)$ and $(h : Y \to C, k : Y \to D)$ being $(f \times h : X \times Y \to A \times C, g \times k : X \times Y \to B \times D)$.

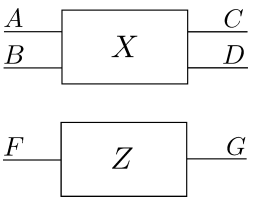
In [1] we introduced an informal geometric description (in the style of monoidal category string diagrams) for the operations in **Span**(**C**). For example, we represented spans $(X \to A \times B, X \to C \times D)$, $(Y \to C \times D, Y \to E)$ and $(Z \to F, X \to G)$ by pictures of *components with ports*:




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



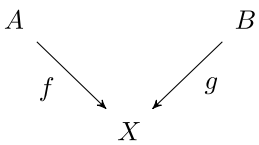

while the tensor of the first and third is pictured as:



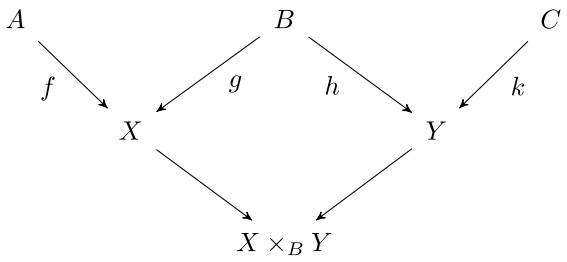

In addition there are constants of the algebra which are pictured as operation on wires which enable the depiction of fanning out of wires and feedback, and hence of general circuit diagrams. In the following, when we will consider **Span**(**Graph**) as an algebra of automata where parallel with or without communication are the main operations, we will use the expression *parallel feedback*.

### 2.2. The algebra of cospans

**Definition 2.** 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 as follows: 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 $(f : A \to X, g : B \to X)$ 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. 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*.

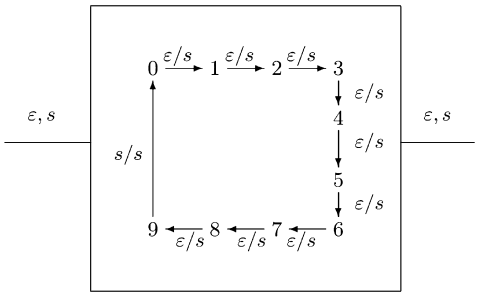
### 2.3. **Span**(**Graph**), *a parallel algebra of automata*

**Span**(**Graph**) is the category of spans in the category of (finite) directed graphs. 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* (or communication ports, using name more used in circuit theory). 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 $\delta_0$ associates to a state and to a transition of the automaton $X$ the corresponding state and transition of the left ports $A$; the morphism $\delta_1$ 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 $\delta_0$ and $\delta_1$ 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.

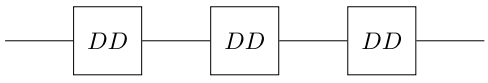
In the case that the left and right ports have one state, the operations of composition and tensor of spans have a simple description in terms of operations on automata. The tensor of two automata has states being pairs of states, one of each automata, and has as transitions pairs of transitions between 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*.

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

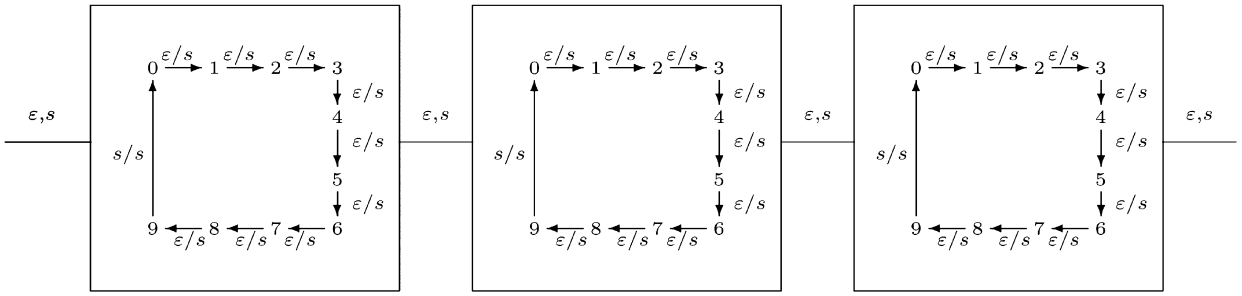



We are actually interested in a variation of this example which we shall call *DecimalDigit* or more shortly $DD$ which has the same ports but in addition to the ten transitions above also ten loops one on each state, each labelled $(\varepsilon/\varepsilon)$. The label $\varepsilon$ is to be thought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

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:




Now we could calculate the composite in **Span**(**Graph**) by first composing two $DD$'s and then composing the result with the third $DD$. This is the compositional algebraic viewpoint.

However there is a global geometrical alternative in which we build the final automata directly: a state of *DecimalCounter* is a triple of states one of each $DD$, and a transition of *DecimalCounter* is a triple of transitions with the property that the *labels on the connected ports agree*. The labels on the transitions of *DecimalCounter* are the labels on the non-connected ports.

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 $\varepsilon/\varepsilon$ 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 2.** The main point of this paper is to make mathematically explicit this relation between global/geometrical and compositional/algebraic points of view, as an example we give an embedding of the geometric C/E nets in the compositional algebra of **Span**(**Graph**).

**Remark 3.** 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 $(I + J)\backslash$**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 **Span**$((I + J)\backslash$**Graph**$)$.

**Remark 4.** In [24,25] we described a probabilistic versions of **Span**(**Graph**) and **Cospan**(**Graph**) in which communicating parallel composition involves conditional probability, arising from the fact that incompatible transition cannot occur and hence probabilities must be normalized. This yields a compositional theory of Markov chains.

### *2.4.* **Cospan**(**Graph**)*, a sequential algebra of automata*

**Cospan**(**Graph**) is the category of cospans in the category of (finite) directed graphs. 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 – usually $E$ and $F$ have only states and not transitions. The graph morphisms $\gamma_0$ and $\gamma_1$ are often inclusion morphisms of the initial and final states in $X$.

**Remark 5.** 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 **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 **Cospan**(**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 **Span**(**Graph**)\($E + F$), and also in the sequential algebra **Cospan**(**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. For some details of this see [4].

## 3. Networks

### 3.1. Closed networks

We formalize the notion of closed network by the notion of monoidal graph given in following definition:

**Definition 3.** A *monoidal graph* **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 **M** *discrete* when $M_1 = \emptyset$.

**Remark 6.** This notion is slighty different from the usual notion of hypergraph. Directed graph 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 will see shortly that the language of components and wires is more suitable when the domain and codomain functions are words rather than single letters.

We can represent a monoidal graph by a list of arcs (the elements of $M_1$) with the tail of the arc being its domain, and the head of the arc its codomain. 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:

$$A_1 : X_1 \to Y_1$$

$$B_1 : Y_1 \to Z_1$$

$$C_1 : Z_1 \to W_1$$

$$D_1 : W_1 \to X_1$$

$$A_2 : X_2 \to Y_2$$

$$B_2 : Y_2 \to Z_2$$

$$C_2 : Z_2 \to W_2$$

$$D_2 : W_2 \to X_2$$

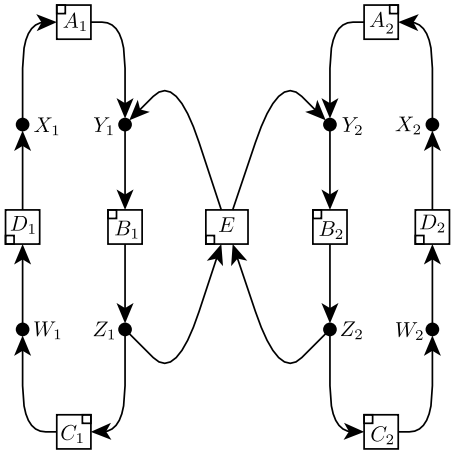
$$E : Z_1 Z_2 \to Y_1 Y_2$$

**Definition 4.** If $A : X_1 X_2 \cdots X_m \to Y_1 Y_2 \cdots Y_n$ is a component of monoidal graph **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 7.** 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 in which the main examples are Petri nets 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 8.** 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 5.** A morphism $\alpha$ from monoidal graph **M** to **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 $\alpha_0$.

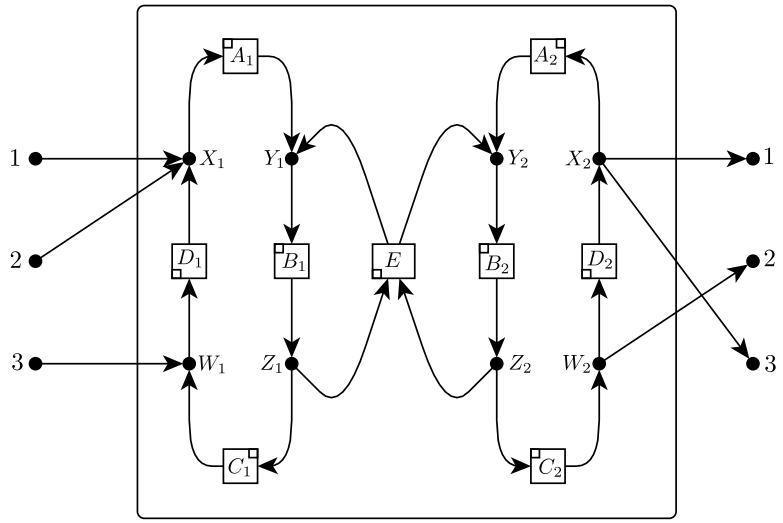
**Remark 9.** It is not difficult to verify that monoidal graphs with these morphisms form a category, denoted **MonGraph**, which is in fact a presheaf topos.

### *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. An open network is a monoidal graph together with left and right interfaces. The formal definition is as follows:

**Definition 6.** An open network consists of a monoidal graph **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 is that they form the arrows of *a symmetric monoidal category in which every object has a commutative separable algebra structure compatible with the tensor* [14,1] – what we call WSCC categories. We first give a definition of WSCC category assuming knowledge of symmetric monoidal categories.

**Definition 7.** 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 $(X, \nabla, n)$ a commutative monoid, $(X, \Delta, e)$ 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 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 $\tau$ is the twist of the symmetry):*

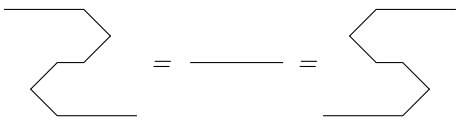
(i) $(\epsilon \otimes 1_X)(1_X \otimes \eta) = 1_X = (1_X \otimes \epsilon)(\eta \otimes 1_X)$
(ii) $\epsilon\tau = \epsilon$ *and* $\tau\eta = \eta$.

**Proof.** We prove only (i) as an illustration. To see that $(\epsilon \otimes 1_X)(1_X \otimes \eta) = 1_X$ notice that

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

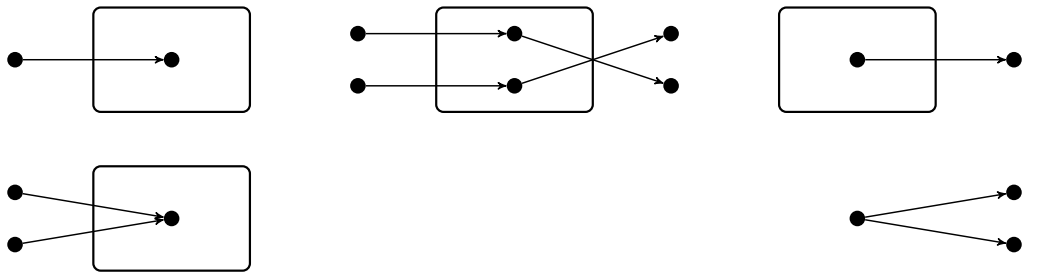
The first equality is an application of the first displayed axiom of definition 2.5 in [1]. The second equality comes from the monoid and comonoid axioms. □

**Remark 10.** 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 8.** *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 **M** and **N** whereas the wires of the composite are the quotient of the disjoint union of the wires of **M** and **N** obtained by equating the images of elements of $T$ (under $\gamma_1$) in **M** with corresponding images of the elements of $T$ (under $\gamma_0$) in **N**. The open network tensor of **M** and **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.

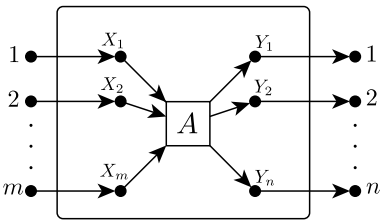


*3.4. The algebra of open networks is free*

**Theorem 1.** *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 [12] 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). Since the proof of the theorem we are describing here (even if it is more general of the one in [12]) requires a straightforward modification of the normal form in which a sum of edges is replaced by a sum of components of **M**. So we leave the proof for the reader. □

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



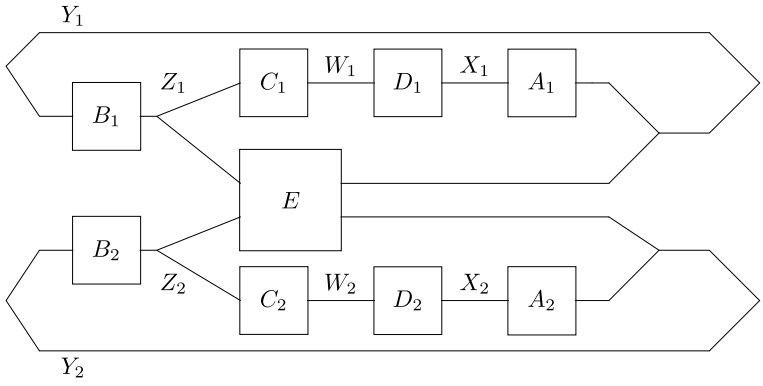

where the elements $A$, $X_1, X_2, \cdots, X_m$, $Y_1, Y_2, \cdots, Y_n$ are now *labels* – that is, if in the original monoidal graph $X_1 = X_2$ then in the cospan the corresponding wires are distinct but have the same label $X_1 = X_2$ in **M**.

**Remark 11.** 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 12.** A theorem similar to Theorem 1, for ordinary graphs only, was described by Gadducci, Heckel and Llabres [26] 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. This will be done globally, but also have a compositional description.

### *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 $\Delta$. Let $|$**Span**(**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

**Definition 9.** An open network with state is an arrow in the monoidal category

**Csp**(**MonGraph**/$|$**Span**(**Graph**)$|$).

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

Note that **Csp**(**MonGraph**/$|$**Span**(**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 $A : X_1 X_2 \cdots X_m \to Y_1 Y_2 \cdots Y_n$ a span of graphs between the products of the graphs labelling the wires.

**Definition 10.** The process of forming the global state space of a network with state is the functor preserving the WSCC structure

*globalstate* : **Csp**(**MonGraph**/$|$**Span**(**Graph**)$|$) $\longrightarrow$ **Span**(**Graph**),

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

**Theorem 2.** *The functor globalstate can be obtained by calculating a global* limit *in* **Graph**.

**Proof.** The details of the limit are as follows: the diagram in **Graph** has as objects all the labelling graphs of the wires, as well as all the labelling graphs of the spans; the arrows arise from the labelling spans – one arrow from the labelling graph of the span to each labelling graph of the associated wires. The proof is described in [13], an article on the compositionally of the calculation of limits and colimits. □

**Remark 13.** The meaning of this is that a state (state transition) of an open network with state is a tuple of states (state transitions), one for each component, such that the states (state transitions) agree in the related wire graphs; the *traditional engineering way of looking at a parallel network*.

**Corollary 1.** *The systems described by networks with state are the same as* **Span**(**Graph**) *systems.*

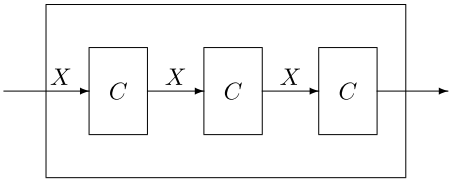
**Proof.** Consider the functor

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

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. □

**Remark 14.** If a network with state is given geometrically the global state space is calculated by a limit. However, if a network with state is given compositionally then there are two methods of calculating the global states and state transitions of a system: (i) evaluating the expression in the codomain, or (ii) by evaluating the expression in the domain to obtain the geometry of the system and then taking a limit. For calculating random behaviours (ii) is much more efficient and we take advantage of this efficiency in the tool described in Section 5.

**Example 2.** We describe *DecimalCounter*, the decimal counter with three digits given above. Consider the monoidal graph **M** with one component $C$ and one wire $X$ with $C : X \to X$. Then arrows in **Csp**(**MonGraph**/**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 **Csp**(**MonGraph**/**M**) which is an open monoidal graph with graphical representation




Now by the freeness of **Csp**(**MonGraph**/**M**) a monoidal graph morphism from **M** to |**Span**(**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 $\varepsilon, s$ 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 $CCC : X \to X$:

### *4.2. Petri nets in an algebra of automata*

In the same volume, Proceedings AMAST '97, in which Katis, Sabadini and Walters introduced the **Span**(**Graph**) algebra of automata, they also showed [3] how to embed P/T nets into this compositional setting.

The advantages of an algebra are clear: an algebra permits new conceptually significant components to be constructed, it permits encapsulation, and further it permits recursive definitions (we present an interesting example of recursion in an algebra which includes also sequential operations in [4]).

There was a possible infelicity in this embedding: both places and transitions were represented as components. We give here a new embedding in which conditions are represented by components, and events by wires. For simplicity we consider the case of C/E nets. We will make particular reference to the monoidal graph of Section 3.1 regarded as a C/E net for a simple mutual exclusion protocol.

**Beware:** In view of the multitude of variations in the literature of the notions of Petri net, of names for the elements of Petri nets, and of behaviours of Petri nets, we need to be clear about which usages we intend in this article.

We will use names associated with C/E nets. What are sometimes called places we will call *conditions*; what are sometimes called transitions of a net we will call *events*. We will speak of the *preconditions* and *postconditions* of an event. The set of events for which a condition is a precondition we call the *postset* of a condition. The set of events for which a condition is a postcondition we call the *preset* of a condition. We call a subset of conditions a *marking* of the net.

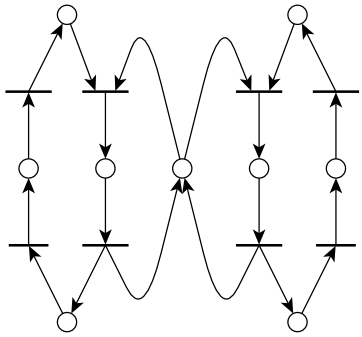
The behaviour we intend for a C/E net is as follows: an event may *occur* in a marking only when its preconditions are all satisfied (that is, are in the marking), and its postconditions are all unsatisfied (not in the marking). After the occurrence the preconditions are all unsatisfied, and the postconditions are all satisfied. Two different events can occur *concurrently* if and only if there is no element in common to the pre- and post-conditions of one event with the pre- and post-conditions

of the other. We notice that two concurrent events are not necessarily firing simultaneously in a marking. That is, two concurrent events can be asynchronous. In order to describe this behaviour in our formalism we add a loop labelled $\epsilon$ in every state. A *firing* of the net in a marking is a subset of the concurrent events in the marking. A *behaviour* of a net is a sequence of firings beginning with an initial marking.

It is clear from our geometric picture that the notion of monoidal graph is very close to that of Petri net if we think of the components as conditions, and the wires as events and the ports as connections between the conditions and the events. However there is a difference: the input ports of a component have an order on them, as do the output ports. The ordering is fundamental when relating the work to other parts of mathematics, physics and engineering: for example, circuit components have an ordering on the ports.

In the influential paper [23] Meseguer and Montanari made a different connection: they identified places with wires and transitions with components, and considered the unusual and to us unnatural notion of *commutative* monoidal category, where there is no ordering on the ports.

Notice that, under the correspondence which we will describe in detail below, the monoidal graph introduced in Section 2.1 is (apart from the fact that the ports of the components are ordered) that corresponding to the following Petri net of a simple mutual exclusion protocol:



**Definition 11.** For each condition of a Petri net $\mathcal{N}$ choose an order on the preset of the condition, and also on postset of the condition. Form from $\mathcal{N}$ the monoidal graph denoted $\Phi(\mathcal{N})$ in which the components are the conditions, and the wires are the events; if $A$ is a condition with ordered preset $X_1, X_2, \cdots, X_m$ and ordered postset $Y_1, Y_2, \cdots, Y_n$ we associate to condition $A$ the component $A : X_1X_2 \cdots X_m \to Y_1Y_2 \cdots Y_n$.

**Definition 12.** Given a Petri net $\mathcal{N}$ we consider closed network with states $\Phi(\mathcal{N})$ as follows. Each condition $A$ becomes a component with two states 0 and 1. Each event $X$ becomes a wire whose graph has one state, and which has two transitions, an $\varepsilon$ transition and a transition which represents the occurrence of the event $X$, conventionally named $f_X$, but in examples often labelled in such a way to make the meaning of the occurrence clear.

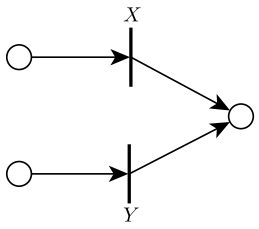
Finally we describe the transitions of the components. Each state of a component has transitions from the state to itself labelled $\varepsilon$ on each of the ports (that is, projecting to $\varepsilon$ on each of the input and output wires). Further if a condition $A$ is a precondition of the event $X$, then in the graph associated to $A$ there is an arc from 1 to 0 labelled $f_X$ on the port $X$ and $\varepsilon$ on all other ports. Lastly, if a condition $A$ is a postcondition of the event $X$, then in graph associated to $A$ there is an arc from 0 to 1 labelled $f_X$ on the port $X$ and $\varepsilon$ on all other ports.

Notice that the global state space of the closed network has $2^k$ elements where $k$ is the number of conditions in the Petri net; a state corresponds exactly to a *marking* of the C/E net.

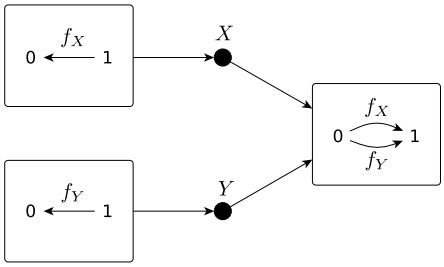
We give some pictorial examples which should clarify these definitions, and make clear also that a global transition of the network with state corresponds exactly to a firing of the C/E Petri net. The examples may strike one at first as a complicated way to describe C/E nets, but the advantage is that the behaviour is programmable by automata. This allows descriptions quite different from the simple control structures of Petri nets. For examples, see [1,10,4] and Example 6 of this paper.

**Important Note:** In the following examples we omit all mention of $\varepsilon$ transitions on the wires. Strictly speaking each wire should have an $\varepsilon$ transition, and each transition of each component should have as many labels as there are ports – the purely $\varepsilon$ transitions should have label $\varepsilon$ on all ports, the other transitions should have all ports except one labelled $\varepsilon$. We have indicated only the non-$\varepsilon$ labels.

**Example 3.** Consider the following Petri net with any initial marking:

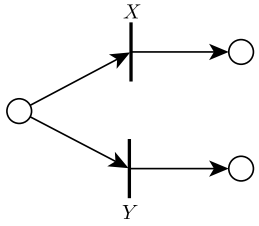



The network with states obtained from the Petri net is:

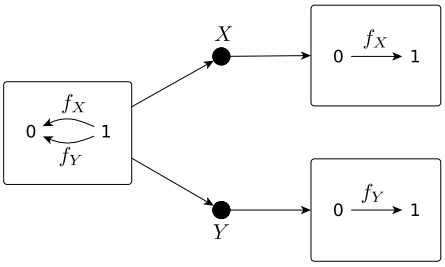



The satisfaction of a condition in a marking of the Petri net corresponds to state 1 in the corresponding component. If the common postcondition is unsatisfied in a marking then either of the events may occur (but at most one) provided the precondition of the event is satisfied.

**Example 4.** Consider the following Petri net with any initial marking:



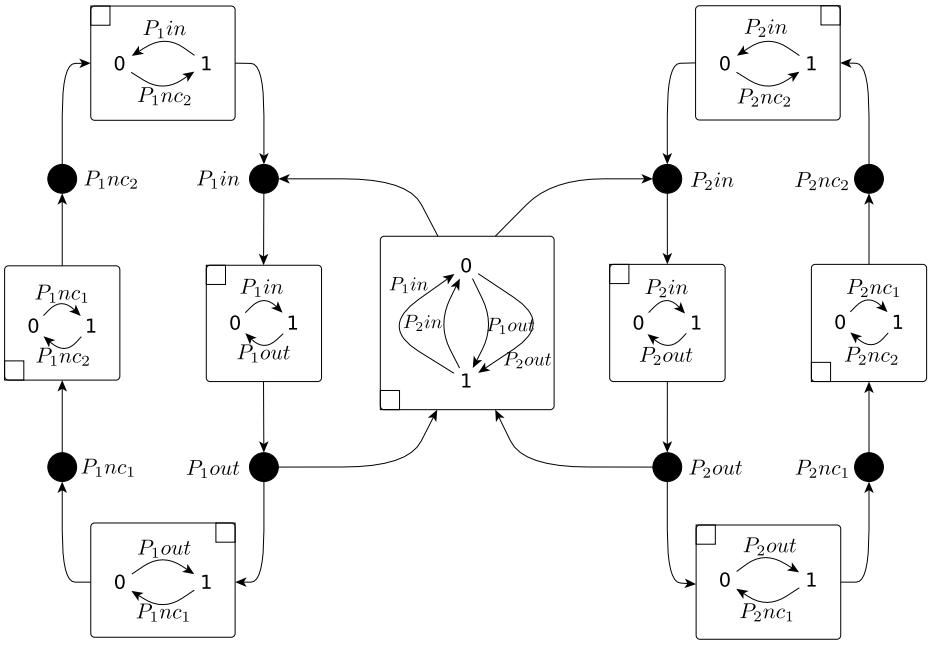

The network with state obtained from the Petri net is:




If a postcondition of an event is unsatisfied and the precondition is satisfied then that event may occur (but at most one event).

**Example 5.** *The mutual exclusion protocol* Notice that in the following example we have not used the name $f_X$ for the non-$\varepsilon$ transition of event $X$; we have used suggestive names. For example, $P_1nc_1$ means the occurrence of a non-critical event of process 1; $P_1in$ means the occurrence of the event of process 1 entering the critical region.

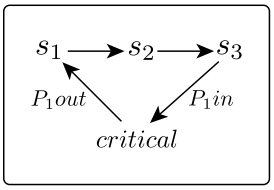
The following network with state corresponding to a Petri net may have any initial marking but the intended initial marking is that each process has one token in a non-critical condition, and the middle mutex component has a token.




Remember that a global state transition consists of a state transition in each component such that the labelling on common wires is the same. In each global state there is a state transition which is $\varepsilon$ on each wire. Other global state transitions involve at least one non-$\varepsilon$ label on wires; that is, involve the occurrence of one or more corresponding events. It is straightforward to see that these global state transitions correspond exactly to firings of the C/E net.

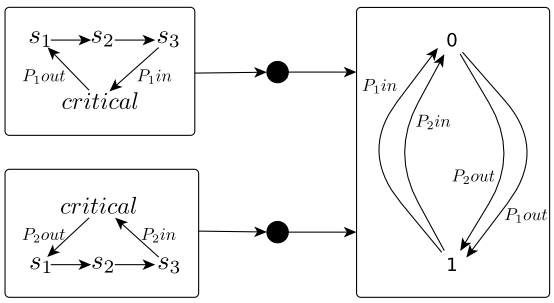
**Example 6.** The example Petri net of Remark 15 models a mutual exclusion protocol for two processes $P_1$, $P_2$ each of which can be in three non-critical states or one critical state. The protocol involves a mutex component which can be free or busy.

A natural way to describe each process is to present an automaton. In the case of $P_1$ it might be:




Note that non critical actions are $\varepsilon$-labelled because they are not important in mutual exclusion context and can be interpreted as silent actions.

Using these automata the protocol can be formalized by a simpler network with state:



Notice that the state space of the Petri net has $2^9$ elements, whereas the state space of the network with state just introduced has $2^5$ elements. This is due to the fact that the Petri net permits markings which are not intended in the modelled system.

In the Petri net there are $2^4$ possible marking of each process and 2 markings of the mutex component yielding a total of $2^4 \times 2 \times 2^4 = 2^9$ possible marking. The intention however is that each process has 4 possible states, and the mutex component 2 yielding $4 \times 2 \times 4 = 2^5$ states for the system. The point is that a sequential process represented as an $n$ state automaton is necessarily represented as a Petri net with $n$ components and hence an exponential (in $n$) number of possible markings. The advantage of the more general notion of network with state is that each entity of the system can be encapsulated as a single component, and sequential processes may be represented sequentially.

**Remark 15.** Recently Montanari and collaborators [27] have studied compositional theories of Petri Nets using the *Tile Model*, and ideas of Sobocinski [28,29] and [3,1]. There is some similarity between the work of [28] and the work presented here, namely that symmetric monoidal categories with structured objects are used. However instead of the separation between geometric gluing using cospans and communicating actions using spans Sobocinski has different operations which we do not regard as optimal. In particular it shares the problem of Petri nets of representing sequential processes inevitably with parallel components. To be more clear, in our model the distributed system is given by sequential components connected throught interfaces.

The problem arise from the fact that Petri nets are not intended to model sequential components.

Note that to model a sequential process with $n$ states in a Petri net we have to use $n$ conditions, with one event between each pairs of adjacient conditions (exactly what we done in the mutual exclusion example). That is, the process with $n$ states becomes a Petri net with $2^n$ possible markings. In order to represent the original sequential behaviour with $n$ states the Petri net must be equipped with an initial marking where only a single condition has a token. The *net* with a different initial marking can have a totally different behaviour (even highly concurrent).

## 5. A tool for networks with state

The second author has developed a program for calculating the network from an expression in **Cospan**(**MonGraph**/|**Span**(**Graph**)|) and producing random executions. We want to emphasize that the generation of random behaviours is scalable in the presence of many of components, thanks to the fact that we avoid the compositional evaluation of the expression in Span(graph). Instead a behaviour is built by taking at each step a tuple of transitions admissible starting from the current state.

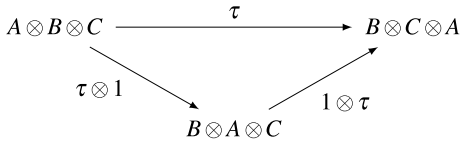
The application accepts as input the definition of components (labelled automata), networks of components, described as algebraic expressions, and commands to generate output diagrams representing the structure and behaviour of given expressions. The program is entirely developed in Java by using two external applications for input/output management, respectively Antlr3 and Graphviz. The first application is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions. The second is a graph generation tool, used for drawing networks and state/transition diagrams. The program together with examples, including the mutual exclusion protocol described in this paper, are available at [30].

## 6. Tangled networks

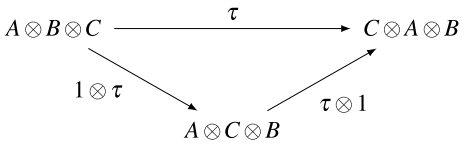
This section is a brief review of some results in [2] together with some new material relating algebraic and geometric proofs. We include it to illustrate the fact that the algebra we have described above is not ad hoc, but fits neatly into the general theory of the relation between monoidal categories and geometry which has had such an explosive development in recent years in mathematics and physics. We mention only four items [31,32,18,5] from the vast literature.

**Definition 13.** A *braided strict monoidal category* ([33]) is a category $\mathbf{C}$ with a functor, called tensor, $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ and a "unit" object $I$ together with a natural family of isomorphisms $\tau_{A,B} : A \otimes B \to B \otimes A$ called twist satisfying

1) $\otimes$ is associative and unitary on objects and arrows,
2) the following diagrams commute for objects $A, B, C$:



and




We will denote the n-fold tensor product of an object $A$ by $A^n$.

**Definition 14.** A *commutative Frobenius algebra* in a braided monoidal category consists of an object $G$ and four arrows $\nabla : G \otimes G \to G$, $\Delta : G \to G \otimes G$, $n : I \to G$ and $e : G \to I$ making $(G, \nabla, n)$ a monoid, $(G, \Delta, e)$ a comonoid and satisfying the equations

$$(1_G \otimes \nabla)(\Delta \otimes 1_G) = \Delta\nabla = (\nabla \otimes 1_G)(1_G \otimes \Delta) : G \otimes G \to G \otimes G$$
$$\nabla\tau = \nabla : G \otimes G \to G$$
$$\tau\Delta = \Delta : G \to G \otimes G$$

where $\tau$ is the braiding.

**Definition 15.** Given a monoidal graph **M** the *free braided strict monoidal category in which the objects of* **M** *are equipped with commutative Frobenius algebra structures* is called $\mathbf{TCircD_M}$. Its arrows are called *tangled circuit diagrams*.

### 6.1. Equality of tangled circuits

We give an example of a proof of the equality of two tangled circuits, an example given in [2] but without the equational proof provided here.

**Proposition 2.** *If $R : X \times Y \to I$ is an arrow in the monoidal graph* **M** *then*

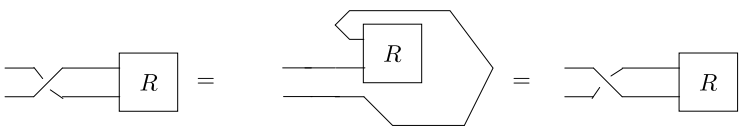
$$R\tau^{-1}_{Y,X} = \epsilon_X(1_X \otimes R \otimes 1_X)(\eta_X \otimes 1_Y \otimes 1_X) = R\tau_{X,Y}.$$

We give two proofs, first an algebraic proof and then a geometric version of the same proof.

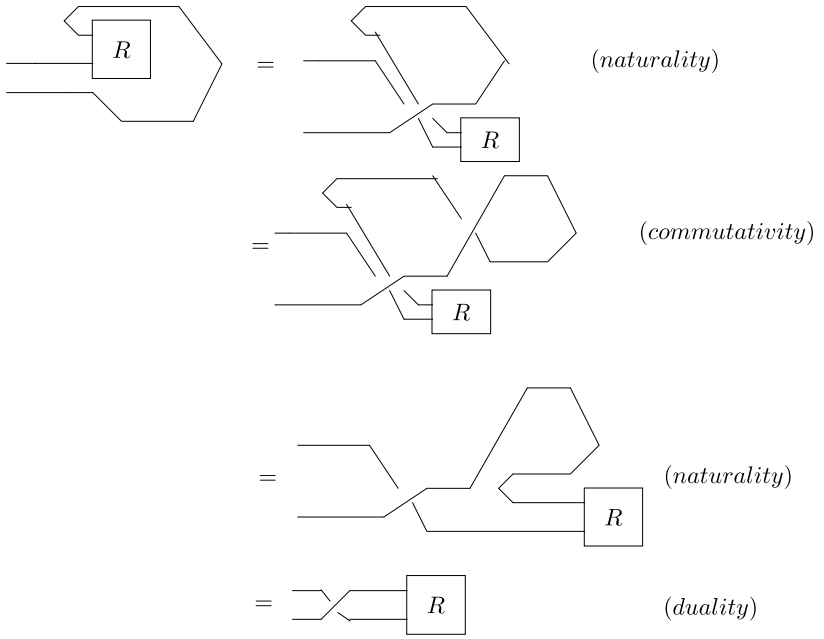
**Proof.** (Algebraic) Consider the middle expression. It clearly suffices to prove that it is equal to the first expression.

$$\begin{aligned}
&\epsilon_X(1_X \otimes R \otimes 1_X)(\eta_X \otimes 1_Y \otimes 1_X) \\
&= \epsilon_X(1_X \otimes 1_X \otimes R)(1_X \otimes \tau^{-1}_{X\otimes Y,X})(\eta_X \otimes 1_Y \otimes 1_X) && \text{(naturality of twist)} \\
&= \epsilon_X(1_X \otimes 1_X \otimes R)(1_X \otimes \tau^{-1}_{X,X} \otimes 1_Y)(1_X \otimes 1_X \otimes \tau^{-1}_{Y,X})(\eta_X \otimes 1_Y \otimes 1_X) && \text{(property of twist)} \\
&= R(\epsilon_X \otimes 1_X \otimes 1_Y)(1_X \otimes \tau^{-1}_{X,X} \otimes 1_Y)(\eta_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{Y,X}) && \text{(functoriality)} \\
&= R(\epsilon_X\tau^{-1}_{X,X} \otimes 1_X \otimes 1_Y)(1_X \otimes \tau^{-1}_{X,X} \otimes 1_Y)(\eta_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{Y,X}) && \text{(commutativity)} \\
&= R(\epsilon_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{X,X} \otimes 1_X \otimes 1_Y)(1_X \otimes \tau^{-1}_{X,X} \otimes 1_Y)(\eta_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{Y,X}) && \text{(functoriality)} \\
&= R(\epsilon_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{X\otimes X,X} \otimes 1_Y)(\eta_X \otimes 1_X \otimes 1_Y)(\tau^{-1}_{Y,X}) && \text{(property of twist)} \\
&= R(\epsilon_X \otimes 1_X \otimes 1_Y)(1_X \otimes \eta_X \otimes 1_Y)\tau^{-1}_{Y,X} && \text{(naturality)} \\
&= R\tau^{-1}_{Y,X} && \text{(duality)} \quad \square
\end{aligned}$$

**Proof.** (Geometric) First a picture of the equations:



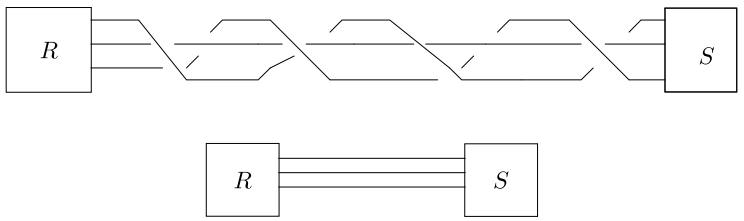
It is clearly sufficient to prove the first equation. Consider:



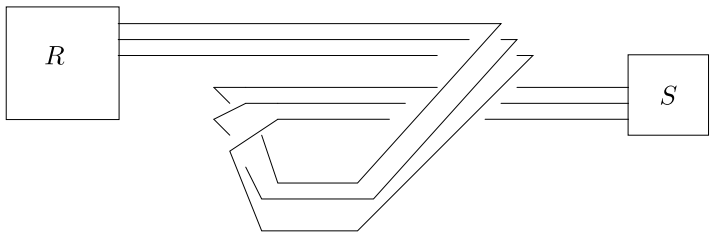

Notice that the geometric proof skips some steps since the functoriality and the properties of twist are geometrically obvious. □

### 6.2. Dirac's belt trick

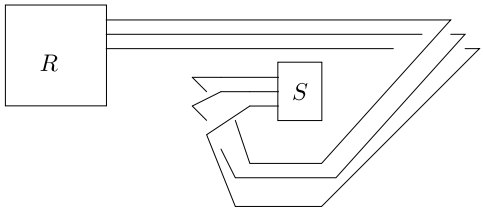
We claim that the following two circuits are equal in **TCircD**, that is that a rotation through $4\pi$ of a component $I \to X^3$ is equal to the identity. We suspect, but are unable to prove, that a rotation through $2\pi$ is not the identity.



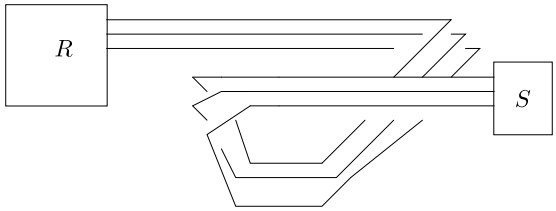

We give a sketch of a proof only. We may prove that the first (twisted) circuit is equal to



Naturality gives equality to




and then to




which is equal to the untwisted circuit.

**Remark 16.** For further connections with braids and knots, including the compositional calculation of knot groups and colourings, see [2].

## 7. Conclusions and further research

We believe we have presented a very natural and mathematically appropriate model of networks with state, which can be described both globally and compositionally, giving as an example C/E nets, in terms of the algebras of spans of graphs, and cospans of monoidal graphs.

We did not present an algebra of sequential processes with state, which naturally would be represented in terms of cospans of graphs. The paper [10] describes systems constructed by sequential (cospans of graphs) and parallel (spans of graphs) operations but does not include the network algebra (cospans of monoidal graphs). An aim would be to have a model with all three aspects. The paper [10] also includes the beginnings of recursive definitions, and we are currently researching this direction.

## References

[1] P. Katis, N. Sabadini, R. Walters, Span(graph): a categorical algebra of transition systems, 1349 (1997) 307–321, http://dx.doi.org/10.1007/BFb0000479.
[2] R. Rosebrugh, N. Sabadini, R. Walters, Tangled circuits, 26 (27) (2012) 743–767.
[3] P. Katis, N. Sabadini, R. Walters, 1997, Representing p/t nets in span(graph).
[4] P. Katis, N. Sabadini, R. Walters, A formalization of the IWIM model, 1906 (2000) 267–283, http://dx.doi.org/10.1007/3-540-45263-X_17.
[5] P. Selinger, A survey of graphical languages for monoidal categories, 813 (2011) 289–355, http://dx.doi.org/10.1007/978-3-642-12821-9_4.
[6] R. Penrose, Applications of negative dimensional tensors, Combin. Math. Appl. (1971), www.sciencedirect.com/science/article/pii/0890540190900138.
[7] A. Joyal, R. Street, The geometry of tensor calculus, i, Adv. Math. 88 (1) (1991) 55–112, http://www.sciencedirect.com/science/article/pii/000187089190003P.
[8] J. Bènabou, Introduction to bicategories, 47 (1967) 1–77, http://dx.doi.org/10.1007/BFb0074299.
[9] R. Rosebrugh, N. Sabadini, R. Walters, Tangled circuits, 26 (27) (2012) 743–767.
[10] P. Katis, N. Sabadini, R. Walters, On the algebra of feedback and systems with boundary, Rend. Circ. Mat. Palermo (2) (2000) 123–156.
[11] P. Katis, Nicoletta Sabadini, Robert F.C. Walters, Feedback, trace and fixed-point semantics, RAIRO Theor. Inform. Appl. 36 (2) (2002) 181–194, http://dx.doi.org/10.1051/ita:2002009.
[12] R. Rosebrugh, N. Sabadini, R. Walters, A formalization of the iwim model, 15 (6) (2005) 164–177.
[13] R. Rosebrugh, N. Sabadini, R. Walters, Calculating colimits compositionally, in: P. Degano, R. De Nicola, J. Meseguer (Eds.), Concurrency, Graphs and Models, in: Lecture Notes in Comput. Sci., vol. 5065, Springer, Berlin, Heidelberg, 2008, pp. 581–592.
[14] A. Carboni, R. Walters, Cartesian bicategories i, J. Pure Appl. Algebra 49 (1) (1987) 11–32, http://dx.doi.org/10.1016/0022-4049(87)90121-6.
[15] S. Eilenberg, Automata, Languages and Machines, Vol. A, Academic Press, New York, 1972.
[16] S.L. Bloom, Z. Ésik, Iteration Theories: The Equational Logic of Iterative Processes, Springer-Verlag New York, Inc., New York, NY, USA, 1993.
[17] G. Stefănescu, Network Algebra, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.
[18] J. Kock, Frobenius Algebras and 2D Topological Quantum Field Theories, Cambridge University Press, 2004.
[19] S. Bliudze, J. Sifakis, The algebra of connectors: structuring interaction in bip, http://doi.acm.org/10.1145/1289927.1289935, 2007, 11–20.
[20] B. Fong, Decorated cospans, arXiv:1502.00872.
[21] J.C. Baez, J. Erbele, Categories in control, arXiv:1405.6881.
[22] J.C. Baez, B. Fong, A compositional framework for passive linear networks, arXiv:1504.05625.
[23] J. Meseguer, U. Montanari, Petri nets are monoids, Inform. and Comput. 88 (2) (1990) 105–155, http://dx.doi.org/10.1016/0890-5401(90)90013-8.
[24] L.d.F. Albasini, N. Sabadini, R.F.C. Walters, The compositional construction of Markov processes, arXiv:0901.2434.
[25] L.d.F. Albasini, N. Sabadini, R.F.C. Walters, The compositional construction of Markov processes II, arXiv:1005.0949.

[26] F. Gadducci, R. Heckel, M. Llabrés, A bi-categorical axiomatisation of concurrent graph rewriting, in: Conference on Category Theory and Computer Science, {CTCS} '99, in: Electron. Notes Theor. Comput. Sci., vol. 29, 1999, pp. 80–100.
[27] R. Bruni, H. Melgratti, U. Montanari, P. Sobocinski, Connector algebras for C/E and P/T nets' interactions, arXiv:1307.0204.
[28] P. Sobociński, Representations of petri net interactions, 6269 (2010) 554–568, http://dx.doi.org/10.1007/978-3-642-15375-4_38.
[29] J. Lantair, P. Sobociński Wicca, Lts generation tool for wire calculus, 6859 (2011) 407–412, http://dx.doi.org/10.1007/978-3-642-22944-2_31.
[30] F. Schiavio, SpanTools program, http://sourceforge.net/projects/spantool/, 2012.
[31] P.J. Freyd, D.N. Yetter, Braided compact closed categories with applications to low dimensional topology, Adv. Math. 77 (2) (1989) 156–182, http://dx.doi.org/10.1016/0001-8708(89)90018-2.
[32] D. Yetter, Functorial Knot Theory: Categories of Tangles, Coherence, Categorical Deformations, and Topological Invariants, Ser. Knots Everything, World Scientific, 2001, https://books.google.it/books?id=_E4DcaneMX4C.
[33] A. Joyal, R. Street, Braided monoidal categories, Adv. Math. 102 (1) (1993) 20–78.