Span(Graph): a Canonical Feedback Algebra of Open Transition Systems [⋆]

Elena Di Lavore[1], Alessandro Gianola[2], Mario Román[1], Nicoletta Sabadini[3], and Paweł Sobociński[1]

[1] Tallinn University of Technology, Ehitajate tee 5, 12616 Tallinn, Estonia
[2] Free University of Bozen-Bolzano, Piazza Domenicani, 3, 39100 Bolzano BZ, Italy
[3] Università degli Studi dell'Insubria, Via Ravasi, 2, 21100 Varese VA, Italy

Abstract. We show that $\mathbf{Span}(\mathbf{Graph})_*$, an algebra for *open transition systems* introduced by Katis, Sabadini and Walters, satisfies a universal property. By itself, this is a justification of the canonicity of this model of concurrency. However, the universal property is itself of interest, being a formal demonstration of the relationship between feedback and state. Indeed, *feedback* categories, also originally proposed by Katis, Sabadini and Walters, are a weakening of traced monoidal categories, with various applications in computer science. A *state bootstrapping* technique, which has appeared in several different contexts, yields *free* such categories.
We show that $\mathbf{Span}(\mathbf{Graph})_*$ arises in this way, being the free feedback category over $\mathbf{Span}(\mathbf{Set})$. Given that the latter can be seen as an algebra of predicates, the algebra of open transition systems thus arises – roughly speaking – as the result of bootstrapping state to that algebra.
Finally, we generalize feedback categories endowing state spaces with extra structure: this extends the framework from mere transition systems to automata with initial and final states.

⋆ Di Lavore, Román and Sobociński were supported by the European Union through the ESF funded Estonian IT Academy research measure (2014-2020.4.05.19-0001). This work was also supported by the Estonian Research Council grant PRG1210.

Table of Contents

1 Introduction

Software engineers need models. In fact, models developed in the early years of computer science have been extremely influential on the emergence of software engineering as a discipline. Prominent examples include flowcharts and state machines, and a part of the reason for their impact and longevity is the fact that they are underpinned by relevant and well-understood mathematical theories.

However, while *concurrent* software has been intensively studied since the early 60s, the theoretical research landscape remains quite fragmented. Indeed, Abramsky [1] argues that the reason for the proliferation of models, their sometimes overly locally-optimised techniques, and the difficulty of understanding and relating their expressivity, is the fact that we still do not have a satisfactory understanding of the underlying mathematical principles of concurrency.

A way to identify such principles and arrive at more canonical models is to look for logical or mathematical justifications. An example is the recent discovery and work on of Curry-Howard style connections between calculi for concurrency and fragments of linear logic, which guided the development of session types [15]. Another possible route is to search for models that satisfy some *universal property*.

The latter approach is the remit of this paper: we focus on the $\mathbf{Span}(\mathbf{Graph})_*$ model of concurrency, introduced by Katis, Sabadini and Walters [32] as an algebra of *open transition systems*, and show that it satisfies a universal property: it is the free feedback category over the category of spans of functions.

The free construction is in itself interesting and can be described as a kind of "state-bootstrapping". We thus position our main result within the theoretical context of feedback categories, their relationship with state, and the more restrictive—yet better known—notion of traced monoidal categories. Our exploration of this wider context is justified, given the panoply of related, yet partial, accounts in the literature.

The relationship between feedback and state is well-known by engineers. In fact, a remarkable fact from electronic circuit design is how data-storing components can be built out of a combination of *stateless components* and *feedback*. A famous example is the (set-reset) "NOR latch": a circuit with two stable configurations that *stores* one bit.

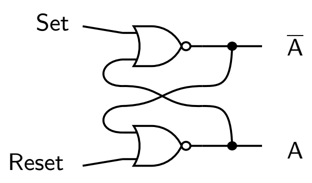


Fig. 1: NOR latch.

The NOR latch is controlled by two inputs, Set and Reset. Activating the first sets the output value to $\mathsf{A} = 1$; activating the second makes the output value return to $\mathsf{A} = 0$. This change is permanent: even when both Set and Reset are deactivated, the feedback loop maintains the last value the circuit was set to[4]—to wit,

[4] In its original description: *"the relay is designed to produce a large and* ***permanent*** *change in the current flowing in an electrical circuit by means of a small electrical stimulus received from the outside"* ([17], emphasis added).

a bit of data has been conjured out of thin air. The results of this paper allow one to see the latch as an instance of a more abstract phenomenon.

Indeed, there is a natural weakening of the notion of traced monoidal categories called *feedback categories* [35]. The construction of the *free* feedback category coincides with a "state-bootstrapping" construction, $\mathsf{St}(\bullet)$, that appears in several different contexts in the literature [7,28,31]. We recall this construction and its mathematical status (Theorem 3.11), which can be summed up by the following intuition:

$$\text{Theory of Processes} + \text{Feedback} = \text{Theory of Stateful Processes.}$$

The **Span**(**Graph**) model of concurrency, introduced in [32], is an algebra of *communicating state machines*, or — equivalently — *open transition systems*.

Let us first explain some terminology. A span $X \to Y$ in a category $\mathbf{C}$ is a pair of morphisms $l \colon A \to X$ and $r \colon A \to Y$ with a common domain (Definition 4.1). When $\mathbf{C}$ has enough structure, spans form a category. This is the case for the category of graphs **Graph**, where objects are graphs and morphisms are, intuitively, pairs of functions that respect the graph structure (Definition 4.6). Summarizing the above, the morphisms of **Span**(**Graph**) are given by pairs of graph homomorphisms, $l \colon G \to X$ and $r \colon G \to Y$, with a common domain G. We think of a span of graphs as a transition system, the graph G, with boundary interfaces X and Y.

Open transition systems interact by synchronization along a common boundary, producing a *simultaneous change of state*. This corresponds to a composition of spans, realized by taking a pullback in **Graph** (see Definition 4.7). The dual algebra of **Cospan**(**Graph**) was introduced in [34] (see Definition 4.17).

Informally, a morphism $X \to Y$ of **Span**(**Graph**) is a state machine with states and transitions, i.e. a finite graph given by the 'head' of the span. The transition system is equipped with left and right interfaces or *communication ports*, X and Y, and every transition is labeled by the effect it produces in *all* its interfaces. Let us focus on some concrete examples.

Let $\mathbb{B} = \{0, 1\}$. We abuse notation by considering $\mathbb{B}$ as a single-vertex graph with two edges, corresponding to the *signals* 0 and 1. Indeed, as we shall see in examples below, it is useful to think of single-vertex graphs as alphabets of signals available on interfaces.

In Figure 2, we depict two open transition systems as arrows of **Span**(**Graph**). The first represents a NOR gate $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$. To give an arrow of this type in **Span**(**Graph**) is to give a span of graph homomorphisms

$$\mathbb{B} \times \mathbb{B} \xleftarrow{l} N \xrightarrow{r} \mathbb{B}.$$

The graphical rendering (Figure 2, left) is a compact representation of the components of this span: the *unlabeled* graph in the bubble is N, and the labels witness the action of two homomorphisms, respectively $l \colon N \to \mathbb{B}\times\mathbb{B}$ and $r \colon N \to \mathbb{B}$. Transitions represent the valid input/output configurations of the NOR gate. For example, the edge with label $(\binom{0}{0}, 1)$, witnesses a transition whose behaviour on the left boundary is $\binom{0}{0}$ and on the right boundary 1. Note that, since the graph N has a single vertex, gates are *stateless* components.

The second component is a span $L = \{\mathsf{Set}, \mathsf{Reset}, \mathsf{Idle}\} \to \{\mathsf{A}, \overline{\mathsf{A}}\} = R$ that models a set-reset latch. The diagram below right (Figure 2), again, is a convenient illustration of the span $L \leftarrow D \to R$. Latches store one bit of information, they are *stateful components*; consequently, their transition graph has two states.

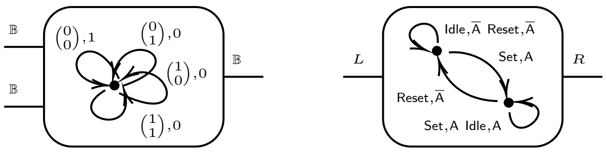

$\mathbb{B}$
$\mathbb{B}$
$\binom{0}{0},1$
$\binom{0}{1},0$
$\binom{1}{0},0$
$\binom{1}{1},0$
$\mathbb{B}$
L
$\mathsf{Idle},\overline{\mathsf{A}}$ $\mathsf{Reset},\overline{\mathsf{A}}$
Set,A
$\mathsf{Reset},\overline{\mathsf{A}}$
Set,A Idle,A
R

Fig. 2: A NOR gate and set-reset latch, in **Span**(**Graph**).

In both transition systems of Figure 2 the *interfaces* are stateless: indeed, they are determined by a mere set – the self-loops of a single-vertex graph. This is a restriction that occurs rather frequently: in fact, *transition systems with interfaces* are the arrows of the full subcategory of **Span**(**Graph**) on objects that are single-vertex graphs, which we denote by **Span**(**Graph**)$_*$. The objects of **Span**(**Graph**)$_*$ represent interfaces, and a morphism $X \to Y$ encodes a transition system with left interface X and right interface Y. Analogously, the relevant subcategory of **Cospan**(**Graph**) is **Cospan**(**Graph**)$_*$, the full subcategory on sets, or graphs with an empty set of edges.

Definition. **Span**(**Graph**)$_*$ *is the full subcategory of* **Span**(**Graph**) *with objects the single-vertex graphs.*

The problem with **Span**(**Graph**)$_*$ is that it is mysterious from the categorical point of view; the morphisms are graphs, but the boundaries are sets. *Decorated* and *structured* spans and cospans [19,3] are frameworks that capture such phenomena, which occur frequently when composing network structures. Nevertheless, they do not answer the question of *why* they arise naturally.

As stated previously, the main contribution of this paper is the characterization of **Span**(**Graph**)$_*$ in terms of a universal property: it is the free feedback category over the category of spans of functions. We now state this more formally.

Theorem. *The free feedback category over* **Span**(**Set**) *is isomorphic to the full subcategory of* **Span**(**Graph**) *given by single-vertex graphs,* **Span**(**Graph**)$_*$. *That is, there is an isomorphism of categories*

$$\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*.$$

Universal constructions, such as the "state-bootstrapping" $\mathsf{St}(\bullet)$ construction that yields free categories with feedback, characterize the object of interest up to equivalence, making it *the canonical object* satisfying some properties. Recall that Abramsky's concern [1] is that the lack of consensus about the intrinsic primitives of concurrency risks making the results about any particular model of concurrency too dependent on the specific syntax employed. Characterising a model as satisfying a universal property side-steps this concern.

Given that **Span**(**Set**), the category of spans of functions, can be considered an *algebra of predicates* [4,10], the high level intuition that summarizes our main contribution (Theorem 4.12) can be stated as:

$$\text{Algebra of Predicates} + \text{Feedback} = \text{Algebra of Transition Systems}.$$

We similarly prove (in Section 4.4) that the free feedback category over **Cospan**(**Set**) is isomorphic to $\mathbf{Cospan}(\mathbf{Graph})_*$, the full subcategory on discrete graphs of **Cospan**(**Graph**).

Finally, Section 5 shows how the same framework of feedback categories can be extended from transition systems to categories with a structured state space (Theorem 5.6), such as categories of automata. As examples, we recover Mealy deterministic finite automata (Proposition 5.10) and we introduce span automata (Definition 5.11).

1.1 Related Work

This article is an extended version of "A Canonical Algebra of Open Transition Systems" [38], presented at the International Conference on Formal Aspects of Component Software (FACS) 2021. With respect to the conference version, we significantly generalised the framework of feedback categories: Section 5 is completely new material. At the same time, Sections 3 and 4 extend the original manuscript adding new proofs (to propositions 4.9 and 4.10, lemma 4.11, and theorem 4.12) and giving a more complete account of the algebra of spans (Sections 4.1 and 4.2). In an effort to make the paper more self-contained, we also include a new preliminary Section 2, which summarises the necessary concepts from category theory.

Span/**Cospan**(**Graph**) has been used for the modeling of concurrent systems [9,20,21,22,32,34,48,51,52]. Similar approaches to compositional modeling of networks have used *decorated* and *structured cospans* [19,3]. However, these models have not previously been characterized in terms of a universal property.

In [35], the $\mathsf{St}(\bullet)$ construction (under a different name) is exhibited as the free *feedback category*. Feedback categories have been arguably under-appreciated but, at the same time, the $\mathsf{St}(\bullet)$ construction has made multiple appearances as a "state bootstrapping" technique across the literature. The $\mathsf{St}(\bullet)$ construction is used to describe a string diagrammatic syntax for *concurrency theory* in [7]; a variant of it had been previously applied in the setting of *cartesian bicategories* in [31]; and it was again rediscovered to describe a *memoryful geometry of interaction* in [28]. However, a coherent account of both feedback categories and their relation with these stateful extensions has not previously appeared. This motivates our extensive preliminaries in Sections 3.1 and 3.2.

1.2 Synopsis

Section 2 consists of background material on symmetric monoidal categories and equivalences between them. Section 3 contains preliminary discussions on traced monoidal categories and categories with feedback; it explicitly describes $\mathsf{St}(\bullet)$, the free feedback category. It collects mainly expository material. Section 4 exhibits a universal property for the $\mathbf{Span}(\mathbf{Graph})_*$ and $\mathbf{Cospan}(\mathbf{Graph})_*$ models of concurrency and Section 4.5 highlights a specific application. Section 5 extends the framework of feedback categories to capture categories of automata.

2 Preliminaries: Symmetric Monoidal Categories

2.1 Theories of Processes

Resources and processes. We start by setting up an abstract framework for what it means to describe a theory of processes. A theory of processes contains two kinds of components: some *resource types*, which we name $A, B, C, \dots$; and some *processes*, which we name $f, g, h, \dots$.

Each process f has an associated input resource type (say, A); and an associated output resource type (say, B). Executing the process f will require some inputs of type A and will produce some outputs of type B. We write this situation as $f\colon A \to B$.

Throughout the paper, we make use of *string diagrams*: a formal diagrammatic syntax for theories of processes [29,39]. In a diagram, every ocurrence of a resource type is represented by a laballed wire; every process is represented by a box, with input wires representing its input type on the left, and output wires representing its output type on the right (Figure 3).

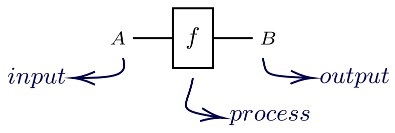
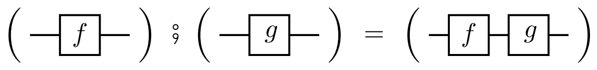
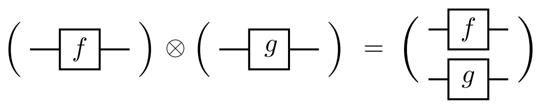


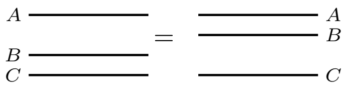
Fig. 3: String diagram for a process $f\colon A \to B$.

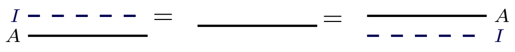
Operations in a theory of processes. Theories of processes allow two operations on processes: sequential composition (⨟) and parallel composition ($\otimes$). The former is depicted as horizontal concatenation of diagrams, the latter as vertical juxtaposition.





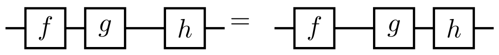
Joining resources. In a theory of processes, resources can be joined. Given a resource type A and a resource type B, we can construct the *joint resource type* $A \otimes B$, which puts together resources of type A and type B. Resource joining may be implemented in diverse ways, depending on the theory of processes. However, it must satisfy some basic axioms:

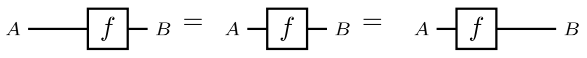
- joining three process resource types together can be done in two ways; these should coincide,
$$A \otimes (B \otimes C) = (A \otimes B) \otimes C, \tag{1}$$


- there must exist a resource type representing the absence of resources, which we call the *unit resource type* I; it must be neutral with respect to process joining
$$A \otimes I = A = A \otimes I. \tag{2}$$


Sequential composition. In a theory of processes, we can compose processes in two different ways. The first is sequential composition: given two processes such that the output type of the first coincides with the input type of the second, say $f \colon A \to B$ and $g \colon B \to C$, their *sequential composition* is the process $(f \mathbin{⨾} g) \colon A \to C$ that results from executing f and using its output to execute g.

Composing may mean different things in different process theories, but it must always satisfy the following axioms:

- sequencing together three processes $f \colon A \to B$, $g \colon B \to C$ and $h \colon C \to D$ can be done in two different ways, these should coincide,
$$(f \mathbin{⨾} g) \mathbin{⨾} h = f \mathbin{⨾} (g \mathbin{⨾} h); \tag{3}$$


- there must exist a process representing "doing nothing" with a resource A that we write as id_A – the *identity* transformation – which must be neutral with respect to sequential composition,
$$\mathrm{id}_A \mathbin{⨾} f = f = f \mathbin{⨾} \mathrm{id}_B. \tag{4}$$


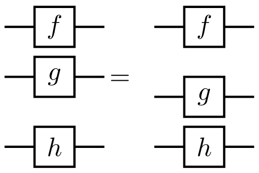
We say that a process $f \colon A \to B$ is *reversible* if it has a reverse counterpart, $f^{-1} \colon B \to A$, such that executing one after the other is the same as having done nothing, $f \mathbin{⨾} f^{-1} = \mathrm{id}_A$ and $f^{-1} \mathbin{⨾} f = \mathrm{id}_B$. This is usually called an *isomorphism.* In this situation, we say that A and B are *isomorphic*, and we write that as $A \cong B$.

Parallel composition. The second way of composing two processes is to do so in parallel. Given any two processes $f\colon A \to B$ and $f'\colon A' \to B'$, their parallel composition is a process $(f \otimes f')\colon A \otimes A' \to B \otimes B'$ that results from jointly executing both processes over the joint input resource type, so as to produce the joint output resource type.

The implementation of parallel composition will usually be related to the implementation of resource joining in the same theory. It must satisfy the following axioms:

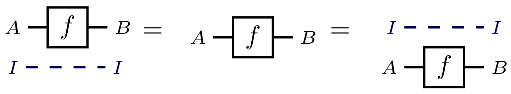
- composing three processes in parallel can be done in two ways; these should coincide,

$$f \otimes (g \otimes h) = (f \otimes g) \otimes h; \tag{5}$$



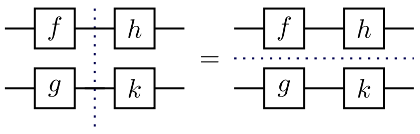
- doing nothing with no resources should be the unit for parallel composition; the identity transformation on the unit resource type I must satisfy

$$f \otimes \mathrm{id}_I = f = \mathrm{id}_I \otimes f; \tag{6}$$



- executing two processes in parallel and then other two processes in parallel must yield the same result as executing in parallel the sequential compositions of both pairs,

$$(f \otimes g) ⨟ (h \otimes k) = (f ⨟ h) \otimes (g ⨟ k). \tag{7}$$

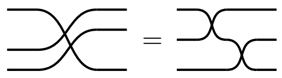


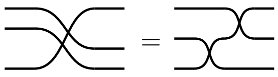
Swapping. Finally, we want to be able to route resources to each specific process. Any theory of processes, given any two resource types A and B, must contain a process $\sigma_{A,B}\colon A \otimes B \to B \otimes A$. This process is called the *swap*, which only permutes the order in which resources are organized. It must satisfy the following axioms.

- Swapping twice is the same as swapping once with a joint type,

$$\sigma_{A,B\otimes C} = (\sigma_{A,B} ⨟ \mathrm{id}_C) ⨟ (\mathrm{id}_B \otimes \sigma_{A,C}); \tag{8}$$

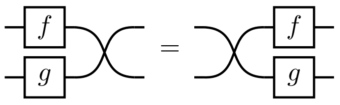
$$\sigma_{A\otimes B,C} = (\mathrm{id}_A \mathbin{⨟} \sigma_{B,C}) \mathbin{⨟} (\sigma_{A,C} \otimes \mathrm{id}_B). \tag{9}$$





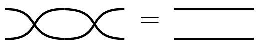
- Swapping two process inputs is the same as swapping the executing place and swapping the output.

$$(f \otimes g) \mathbin{⨟} \sigma_{B,B'} = \sigma_{A,A'} \mathbin{⨟} (g \otimes f). \tag{10}$$



- Swapping and swapping again is the same as doing nothing.

$$\sigma_{A,B} \mathbin{⨟} \sigma_{B,A} = \mathrm{id}_{A\otimes B}. \tag{11}$$



Symmetric monoidal categories. The algebraic structures that capture this notion of process theory are “symmetric monoidal categories” [39]. The resource types are usually called *objects*, while the processes are usually called *morphisms*. Reversible processes are called *isomorphisms*.

Definition 2.1. *A symmetric monoidal category [39] is a tuple*

$$\mathbf{C} = (\mathbf{C}_{\mathrm{obj}}, \mathbf{C}_{\mathrm{mor}}, (⨟), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \sigma),$$

specifying a set of objects, or resource types, $\mathbf{C}_{\mathrm{obj}}$; a set of morphisms, or processes, $\mathbf{C}_{\mathrm{mor}}$; a composition operation; a family of identity morphisms; a tensor operation on objects and morphisms; a unit object and a family of swapping morphisms; satisfying all of the axioms of this section (1-11), possibly up to reversible coherence isomorphisms of the form,

$$\begin{aligned} \alpha_{A,B,C} &: (A \otimes B) \otimes C \to A \otimes (B \otimes C), \\ \lambda_A &: I \otimes A \to A, \text{ and} \\ \rho_A &: A \otimes I \to A. \end{aligned}$$

Coherence isomorphisms must commute with all suitably typed processes and must satisfy all possible formal equations between them. We usually denote by $\mathbf{C}(A, B)$ the set of morphisms from A to B.

Note that we do allow the axioms to be satisfied *up to a reversible coherence isomorphism.* For an example, consider the theory of pure functions between sets joined by the cartesian product. It is not true that, given three sets A, B and C, the following two sets are *equal*, $A \times (B \times C) \cong (A \times B) \times C$; they are merely in a one-to-one correspondence. A symmetric monoidal category is *strict* only if these reversible transformations are identities. It was proven by MacLane (his Coherence Theorem, Theorem 2.8 [39]) that the axioms (1-11) are valid for both strict and non-strict monoidal categories.

Example 2.2. The paradigmatic theory of processes uses mathematical sets as types and functions as processes. We can check that the following functions, with the cartesian product, satisfy the axioms (1-11), thus forming a symmetric monoidal category.

$$\mathbf{Set} = (\mathsf{Sets}, \mathsf{Functions}, (\circ), \mathrm{id}, \times, \langle \bullet, \bullet \rangle, 1, (a,(b,c)) \mapsto ((a,b),c),$$
$$(a,*) \mapsto a, (*,a) \mapsto a, (a,b) \mapsto (b,a)).$$

Example 2.3. The theory of linear transformations uses dimensions (natural numbers) as types and matrices over the real numbers as processes. We can check that matrices, with the direct sum, satisfy the axioms (1-11), thus forming a symmetric monoidal category.

$$\mathbf{Mat} = (\mathbb{N}, \mathsf{Matrices}, (\cdot), (+), \oplus, 0, \mathbf{I}, \mathbf{I}, \mathbf{I}, \mathbf{I}, \mathbf{S}),$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{S}$ is the permutation matrix,

$$\mathbf{I}_n = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots_{n} & \vdots \\ 0 & \dots & 1 \end{pmatrix}; \qquad \mathbf{S}_{n,m} = \begin{pmatrix} 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots_{n} & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 \\ 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots_{m} & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix}.$$

Example 2.4. It can happen that two theories of processes share the same elements, but differ on how they are combined. The theory of choice in finite sets uses again functions, but instead of the cartesian product, it uses the disjoint union. We can check that the following functions satisfy again the axioms (1-11).

$$\mathbf{FinSet} = (\mathsf{FinSets}, \mathsf{Functions}, (\circ), \mathrm{id}, (+), [\bullet, \bullet], 0, (a|(b|c)) \mapsto ((a|b)|c),$$
$$(a|\emptyset) \mapsto a, (\emptyset|a) \mapsto a, (a|b) \mapsto (b|a)).$$

When designing software, the advantage of an algebraic structure such as monoidal categories is reusability: we can encapsulate the operations of our theory of processes into a separate module, and we can abstractly work with them without knowing the particulars of the theory of processes at hand. The axioms (1-11) are straightforward to check for most theories of processes – even if we will not take the time to do so in this text – but they are a powerful abstraction: once the axioms are satisfied, we can start reasoning with string diagrams.

2.2 Monoidal Equivalence

In this final preliminary section, we recall what it means to have a transformation between monoidal categories (symmetric strong monoidal functor, Definition 2.5), what it means to have two equivalent monoidal categories (monoidal equivalence, Definition 2.7) and the statement of the Coherence Theorem: every monoidal category is equivalent to a strict one (Theorem 2.8).

Monoidal functors. Every time we consider an algebraic structure, it is natural to also consider what is a good notion of transformation between two such algebraic structures. A transformation of algebraic structures should preserve the key ingredients of the algebraic construction. In the case of symmetric monoidal categories, these transformations are called *monoidal functors*, and they preserve the operation of composition.

Definition 2.5. *A* symmetric strong monoidal functor *between two symmetric monoidal categories with coherence isomorphisms*

$$\mathbf{C} = (\mathbf{C}_{\mathrm{obj}}, \mathbf{C}_{\mathrm{mor}}, (\fatsemi), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}}, \sigma^{\mathbf{C}}), \text{ and}$$
$$\mathbf{D} = (\mathbf{D}_{\mathrm{obj}}, \mathbf{D}_{\mathrm{mor}}, (\fatsemi), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \alpha^{\mathbf{D}}, \lambda^{\mathbf{D}}, \rho^{\mathbf{D}}, \sigma^{\mathbf{D}})$$

is a tuple $\mathbf{F} = (F_{\mathrm{obj}}, F_{\mathrm{mor}}, \phi, \varphi)$, consisting of

- *a function that assigns objects of the first category to objects of the second category, $F_{\mathrm{obj}} \colon \mathbf{C}_{\mathrm{obj}} \to \mathbf{D}_{\mathrm{obj}}$,*
- *and a function that assigns morphisms of the first category to morphisms of the second category, $F_{\mathrm{mor}} \colon \mathbf{C}_{\mathrm{mor}} \to \mathbf{D}_{\mathrm{mor}}$.*
- *a coherence isomorphism $\phi_{A,B} \colon FA \otimes FB \to F(A \otimes B)$,*
- *and a coherence isomorphism $\varphi \colon J \to FI$.*

Traditionally, functions both on objects, F_{obj} and morphisms, F_{mor} are denoted by F. The functor must be such that every morphism $f \colon A \to B$ is assigned a morphism $F(f) \colon FA \to FB$, whose source and target are the images of the original source and target. Moreover, it must satisfy the following axioms,

- *compositions must be preserved, $F(f \fatsemi g) = F(f) \fatsemi F(g)$,*
- *identities must be preserved, $F(\mathrm{id}_A) = \mathrm{id}_{FA}$,*
- *tensoring must be transported by the natural transformations, meaning that*

$$F(f \otimes g) = \mu \fatsemi (F(f) \otimes F(g)) \fatsemi \mu^{-1},$$

- *associators, unitors and swaps must be transported by the natural transformations, meaning that*

$$\begin{aligned} F(\alpha^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\mu^{-1} \otimes \mathrm{id}) \fatsemi \alpha^{\mathbf{D}} \fatsemi (\mathrm{id} \otimes \mu) \fatsemi \mu, \\ F(\lambda^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\varphi^{-1} \otimes \mathrm{id}) \fatsemi \lambda^{\mathbf{D}}, \\ F(\rho^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\mathrm{id} \otimes \varphi^{-1}) \fatsemi \rho^{\mathbf{D}}, \\ F(\sigma^{\mathbf{C}}) &= \mu^{-1} \fatsemi \sigma^{\mathbf{D}} \fatsemi \mu. \end{aligned}$$

Example 2.6. For instance, there is a strong monoidal functor translating from the theory of choice in finite sets, $\mathbf{FinSet}_+$ (Example 2.4), to the theory of linear transformations $\mathbf{Mat}$ (Example 2.3) that sends the finite sets $A = \{a_0, \dots, a_{n-1}\}$ and $B = \{b_0, \dots, b_{m-1}\}$ to their cardinalities, n and m; and each function $f \colon A \to B$ to the matrix $F_{ij} \colon n \to m$ that contains a 1 on the entry F_{ij} when $f(a_i) = b_j$, and contains a 0 otherwise.

Definition 2.7. *A* monoidal equivalence *of categories is a symmetric strong monoidal functor* $F \colon \mathbf{C} \to \mathbf{D}$ *that is*

1. essentially surjective *on objects, meaning that for each* $X \in \mathbf{D}_{\text{obj}}$*, there exists* $A \in \mathbf{C}_{\text{obj}}$ *such that* $F(A) \cong X$*;*
2. essentially injective *on objects, meaning that* $F(A) \cong F(B)$ *implies* $A \cong B$*; it can be proven that every monoidal functor is essentially injective, so this condition, though conceptually important, is superfluous;*
3. surjective on morphisms*, or* full*, meaning that for each* $g \colon FA \to FB$ *there exists some* $f \colon A \to B$ *such that* $F(f) = g$*;*
4. injective on morphisms*, or* faithful*, meaning that given any two morphisms* $f \colon A \to B$ *and* $g \colon A \to B$ *such that* $F(f) = F(g)$*, it holds that* $f = g$*.*

In this situation, we say that $\mathbf{C}$ *and* $\mathbf{D}$ *are* equivalent*, and we write that as* $\mathbf{C} \cong \mathbf{D}$*. Moreover, when the monoidal functor is injective and surjective on objects, we say that* $\mathbf{C}$ *and* $\mathbf{D}$ *are* isomorphic*.*

Theorem 2.8 (Coherence theorem, [39, Theorem 2.1, Chapter VII]). *Every monoidal category is monoidally equivalent to a strict monoidal category.*

Let us comment further on how we use the coherence theorem. Each time we have a morphism $f \colon A \to B$ in a monoidal category, we have a corresponding morphism $A \to B$ in its strictification. This morphism can be lifted to the original category to uniquely produce, say, a morphism $(\lambda_A \mathbin{⨾} f \mathbin{⨾} \lambda_B{}^{-1}) \colon I \otimes A \to I \otimes B$. Each time the source and the target are clearly determined, we simply write f again for this new morphism.

The reason to avoid this explicit notation on our definitions and proofs is that it would quickly become verbose and distractive. Equations seem conceptually easier to understand when written assuming the coherence theorem – and they become even clearer when drawn as string diagrams, which implicitly hide these bureaucratic isomorphisms. In fact, in the work of Katis, Sabadini and Walters [35], strictness is assumed from the start for the sake of readibility, even though—as argued above—it is not a necessary assumption.

Theorem 2.8 and Section 2.1 can be summarized by the slogan:

> *"Any theory of processes satisfying the axioms of symmetric monoidal categories (1-11) can be reasoned about using string diagrams".*

3 Feedback Categories

In this section we recall feedback categories, originally introduced in [35], and contrast them with the stronger notion of *traced monoidal categories* in Section 3.2. We discuss the relationship between feedback and delay in Section 3.3. Next, we recall the construction of the *free* feedback category in Section 3.4, and give examples in Section 3.5.

3.1 Feedback Categories

Feedback categories [35] were motivated by examples such as *Elgot automata* [18], *iteration theories* [6] and *continuous dynamical systems* [33]. These categories feature a *feedback operator*, $\mathsf{fbk}(\bullet)$, which takes a morphism $S \otimes A \to S \otimes B$ and *"feeds back"* one of its outputs to one of its inputs of the same type, yielding a morphism $A \to B$ (Figure 4, left). When using string diagrams, we depict the action of the feedback operator as a loop with a double arrowtip (Figure 4, right): string diagrams must be acyclic, and so the feedback operator cannot be confused with a normal wire.

$$\frac{f \colon S \otimes A \to S \otimes B}{\mathsf{fbk}_S(f) \colon A \to B}$$

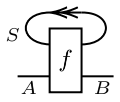


Fig. 4: Type and graphical notation for the operator $\mathsf{fbk}_S(\bullet)$.

Capturing a reasonable notion of feedback requires the operator to interact coherently with the flow imposed by the structure of a symmetric monoidal category. This interaction is expressed by a few straightforward axioms, which we list below.

Definition 3.1. *A* feedback category [35] *is a symmetric monoidal category* $\mathbf{C}$ *endowed with an operator* $\mathsf{fbk}_S \colon \mathbf{C}(S \otimes A, S \otimes B) \to \mathbf{C}(A, B)$*, which satisfies the following axioms (A1-A5, see also Figure 5).*

(A1). Tightening. *Feedback must be natural in* $A, B \in \mathbf{C}$*, its input and output. This is to say that for every morphism* $f \colon S \otimes A \to S \otimes B$ *and every pair of morphisms* $u \colon A' \to A$ *and* $v \colon B \to B'$*,*

$$u \mathbin{⨾} \mathsf{fbk}_S(f) \mathbin{⨾} v = \mathsf{fbk}_S((\mathrm{id} \otimes u) \mathbin{⨾} f \mathbin{⨾} (\mathrm{id} \otimes v)).$$

(A2). Vanishing. *Feedback on the empty tensor product, the unit, does nothing. That is to say that, for every* $f \colon A \to B$*,*

$$\mathsf{fbk}_I(f) = f.$$

(A3). Joining. *Feedback on a monoidal pair is the same as two consecutive applications of feedback. That is to say that, for every morphism* $f \colon S \otimes T \otimes A \to S \otimes T \otimes B$*,*

$$\mathsf{fbk}_T(\mathsf{fbk}_S(f)) = \mathsf{fbk}_{S \otimes T}(f).$$

(A4). Strength. Feedback has the same result if it is taken in parallel with another morphism. That is to say that, for every morphism $f\colon S \otimes A \to S \otimes B$ and every morphism $g\colon A' \to B'$,

$$\mathsf{fbk}_S(f) \otimes g = \mathsf{fbk}_S(f \otimes g).$$

(A5). Sliding. Feedback is invariant to applying an isomorphism "just before" or "just after" the feedback. In other words, feedback is dinatural over the isomorphisms of the category. That is to say that for every $f\colon T \otimes A \to S \otimes B$ and every isomorphism $h\colon S \to T$,

$$\mathsf{fbk}_T(f \fatsemi (h \otimes \mathrm{id})) = \mathsf{fbk}_S((h \otimes \mathrm{id}) \fatsemi f).$$

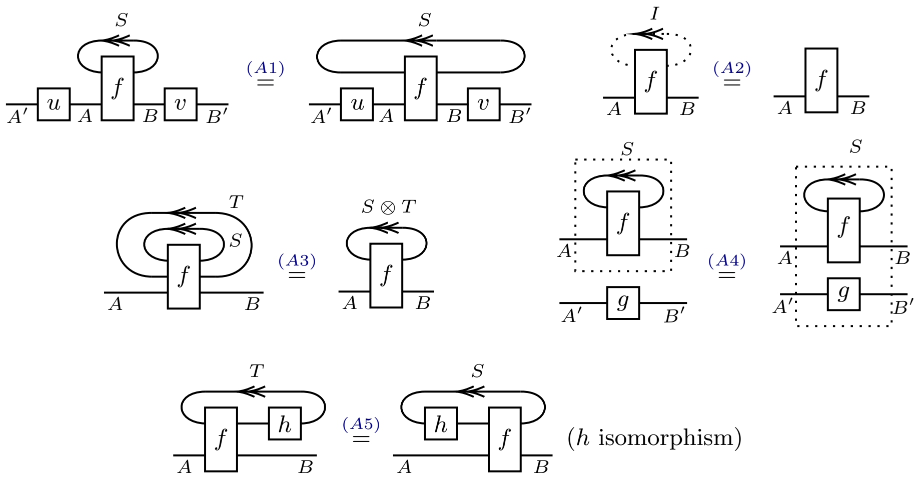


Fig. 5: Diagrammatic depiction of the axioms of feedback.

The natural notion of homomorphism between feedback categories is that of a symmetric monoidal functor that moreover preserves the feedback structure. These are called *feedback functors.*

Definition 3.2. *A feedback functor $F\colon \mathbf{C} \to \mathbf{D}$ between two feedback categories $(\mathbf{C}, \mathsf{fbk}^{\mathbf{C}})$ and $(\mathbf{D}, \mathsf{fbk}^{\mathbf{D}})$ is a strong symmetric monoidal functor such that feedback is transported, that is,*

$$F(\mathsf{fbk}^{\mathbf{C}}_S(f)) = \mathsf{fbk}^{\mathbf{D}}_{F(S)}(\mu \fatsemi Ff \fatsemi \mu^{-1}),$$

where $\mu_{A,B}\colon F(A) \otimes F(B) \to F(A \otimes B)$ is the isomorphism of the strong monoidal functor F. We write Feedback *for the category of (small) feedback categories and feedback functors. There is a forgetful functor $\mathcal{U}\colon$* Feedback $\to$ SymMon.

Remark 3.3. Thanks to the coherence theorem (Theorem 2.8), we can present the axioms of a feedback category as in Definition 3.1, omitting associators and unitors. In fact, to be explicit, the statement of the vanishing axiom is

$$\mathsf{fbk}_I(\lambda_A \mathbin{⨟} f \mathbin{⨟} \lambda_B^{-1}) = f$$

because the feedback operator, fbk_I, needs to be applied to a morphism $I \otimes A \to I \otimes B$, and the only morphism whose strictification has type $A \to B$ is $(\lambda_A \mathbin{⨟} f \mathbin{⨟} \lambda_B^{-1}) \colon I \otimes A \to I \otimes B$ (see Theorem 2.8). Similarly, the joining axiom really states that

$$\mathsf{fbk}_S(\mathsf{fbk}_T(f)) = \mathsf{fbk}_{S\otimes T}(\alpha_{S,T,A} \mathbin{⨟} f \mathbin{⨟} \alpha^{-1}_{S,T,B}).$$

Remark 3.4. Our feedback operator takes a morphism $S \otimes A \to S \otimes B$ with the first component S of the tensor in both the domain and the codomain being the object "fed back". Given that S appears in the first position in both the domain and the codomain, we refer to this as *aligned feedback*.

An alternative definition is possible, and appears in the exposition of traces by Ponto and Shulman [44]. We call this *twisted feedback*: here $\mathsf{fbk}(\bullet)$ is an operator that takes a morphism $S \otimes A \to B \otimes S$—note the position of S in the codomain—and yields a morphism $A \to B$.

$$\frac{f \colon S \otimes A \to B \otimes S}{\mathsf{fbk}_S(f) \colon A \to B}$$

The advantage of using *twisted feedback* is that sequential composition of processes with feedback does not require symmetry of the underlying monoidal category (see [31], where the authors consider a category with twisted feedback). However, parallel composition *does* require symmetry. Given that we study the monoidal category of feedback processes, and aligned feedback diagrams are more readable, we use only aligned feedback in this paper.

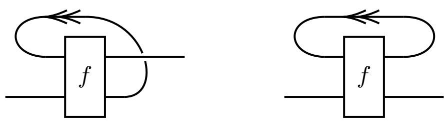


Fig. 6: Twisted vs. aligned feedback

3.2 Traced Monoidal Categories

Feedback categories are a weakening of traced monoidal categories, which have found several applications in computer science. Indeed, since their conception [29] as an abstraction of the *trace* of a matrix in linear algebra, they were used in linear logic and geometry of interaction [1,23,24], programming language semantics [26], semantics of recursion [2] and fixed point operators [27,5].

Between feedback categories and traced monoidal categories there is an intermediate notion called *right traced category* [49]. Here, the sliding axiom applies not only to isomorphisms but rather to arbitrary morphisms. This strengthening is already unsuitable for our purposes (see Remark 3.12). However, the difference in the sliding axiom is not dramatic: we will generalize the notion of feedback category to allow the choice of morphisms that can be "slid" through the feedback loop (Section 5). For example, it is possible to require the sliding axiom for all the morphisms, as in the case of right traced categories, or just isomorphisms, as in the case of feedback categories. The more serious conceptual difference between feedback categories and traced monoidal categories is the "yanking axiom" of traced monoidal categories (in Figure 7). The yanking axiom is incontestably elegant from the geometrical point of view: strings are "pulled", and feedback (the loop with two arrowtips) disappears.

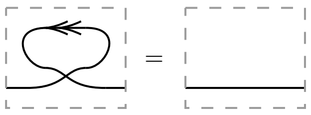


Fig. 7: The yanking axiom.

Strengthening the sliding axiom and adding the yanking axiom yields the definition of traced monoidal category.

Definition 3.5. *A traced monoidal category [29,49] is a feedback category that additionally satisfies the* yanking axiom $\mathsf{fbk}(\sigma) = \mathrm{id}$ *and the* sliding axiom, $\mathsf{fbk}_T(f \mathbin{⨾} (h \otimes \mathrm{id})) = \mathsf{fbk}_S((h \otimes \mathrm{id}) \mathbin{⨾} f)$*, for an arbitrary morphism* $h\colon S \to T$*. We commonly denote by* $\mathsf{tr}(\bullet)$ *the feedback operator of a traced monoidal category.*

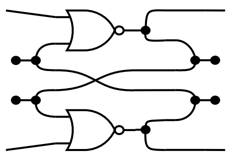


Fig. 8: Diagram for the NOR latch, modeled with a trace in **Span**(**Graph**).

There is scope for questioning the validity of the yanking axiom in many applications that feature feedback. If feedback can disappear without leaving any imprint, that must mean that it is *instantaneous*: its output necessarily mirrors its input.[5] Importantly for our purposes, this implies that a feedback satisfying the yanking equation is "memoryless", or "stateless".

[5] In other words, traces are used to talk about processes in *equilibrium*, processes that have reached a *fixed point*. A theorem by Hasegawa [27] and Hyland [5] corroborates this interpretation: a trace in a cartesian category corresponds to a *fixpoint operator*.

In engineering and computer science, instantaneous feedback is actually a rare concept; a more common notion is that of *guarded feedback*. Consider *signal flow graphs* [50,40]: their categorical interpretation in [8] models feedback not by the usual trace, but by a trace "guarded by a register", that *delays the signal* and violates the yanking axiom (see Remark 7.8 *op.cit.*).

Example 3.6. Let us return to our running example of the NOR latch from Figure 1. We have seen how to model NOR gates in **Span**(**Graph**) in Figure 2, and the algebra of **Span**(**Graph**) does include a trace. However, imitating the real-world behavior of the NOR latch with *just* a trace is unsatisfactory: the trace of **Span**(**Graph**) is built out of stateless components, and tracing stateless components yields a stateless component (see Figure 8, later detailed in Section 4.2).

3.3 Delay and Feedback

As we have discussed previously, the major conceptual difference between feedback categories and traced monoidal categories is the rejection of the yanking axiom. Indeed, a non-trivial delay is what sets apart feedback categories from traced monoidal categories.

We can isolate the delay component in a feedback category. Consider the process that only "feeds back" the input to itself and then just outputs that "fed back" input. The process interpretation of monoidal categories (Section 2.1) allows us to understand this process as delaying its input and returning it as output [16]. This process, $\partial_A := \mathsf{fbk}_A(\sigma_{A,A})$, is called the *delay endomorphism* and is illustrated in Figure 9.

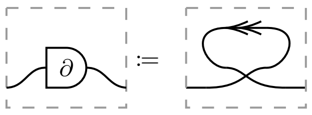


Fig. 9: Definition of *delay*.

If a category has enough structure, feedback can be understood as the combination of *trace* and *delay* in a formal sense. *Compact closed categories* are traced monoidal categories where every object A has a dual $A^\star$ and the trace is constructed from two pieces $\varepsilon \colon A \otimes A^\star \to I$ and $\eta \colon I \to A^\star \otimes A$. While not every traced monoidal category is compact closed, they all embed fully faithfully into a compact closed category.[6] In a compact closed category, a feedback operator is necessarily a trace "guarded" by a *delay*.

Proposition 3.7 (Feedback from delay [7]). *Let* $\mathbf{C}$ *be a compact closed category with* $\mathsf{fbk}^{\mathbf{C}}$ *a feedback operator that takes a morphism* $S \otimes A \to S \otimes B$ *to a morphism* $A \to B$*, satisfying the axioms of feedback (in Figure 5) but possibly*

[6] This is the **Int** construction from Joyal, Street and Verity [29].

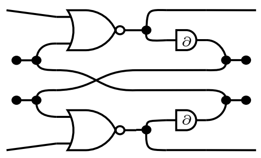


Fig. 10: NOR latch with feedback.

failing to satisfy the yanking axiom (Figure 7) of traced monoidal categories. Then, the feedback operator is necessarily of the form

$$\mathsf{fbk}_S^{\mathbf{C}}(f) \coloneqq (\eta \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes f) ⨟ (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}) ⨟ (\varepsilon \otimes \mathrm{id})$$

where $\partial_A \colon A \to A$ is a family of endomorphisms satisfying

- $\partial_A \otimes \partial_B = \partial_{A\otimes B}$ *and* $\partial_I = \mathrm{id}$, *and*
- $\partial_A ⨟ h = h ⨟ \partial_B$ *for each isomorphism* $h \colon A \cong B$.

In fact, any family of morphisms ∂_A satisfying these properties determines uniquely a feedback operator that has ∂_A as its delay endomorphisms.

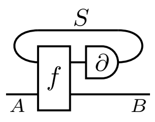


Fig. 11: Feedback from delay.

Proof. Given a family ∂_S satisfying the two properties, we can define a feedback structure, shown in Figure 11, to be $\mathsf{fbk}_S^{\mathbf{C}}(f) \coloneqq (\eta \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes f) ⨟ (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}) ⨟ (\varepsilon \otimes \mathrm{id})$ and check that it satisfies all the axioms of feedback (Figure 5). Note here that, as expected, the yanking equation is satisfied precisely when delay endomorphisms are identities, $\partial_A = \mathrm{id}_A$.

Let us now show that any feedback operator in a compact closed category is of this form (Figure 12). Indeed,

$$\begin{aligned}\mathsf{fbk}_S^{\mathbf{C}}(f) &= \mathsf{fbk}_S^{\mathbf{C}}((\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id}) ⨟ (\sigma \otimes \sigma \otimes f) ⨟ (\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id}))\\ &= (\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id}) ⨟ (\mathsf{fbk}_S^{\mathbf{C}}(\sigma) \otimes \sigma \otimes f) ⨟ (\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id})\\ &= (\eta \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes f) ⨟ (\mathrm{id} \otimes \mathsf{fbk}_S^{\mathbf{C}}(\sigma) \otimes \mathrm{id}) ⨟ (\varepsilon \otimes \mathrm{id}).\end{aligned}$$

Here we have used the fact that the trace is constructed by two separate pieces: ε and η; and then the fact that the feedback operator, like trace, can be applied "locally" (see the axioms in Figure 5). □

Example 3.8. Consider again the NOR latch of Figure 1. The algebra of the category **Span**(**Graph**) does include a feedback operator that is *not* a trace – the difference is an additional *stateful* delay component. As we shall see, this notion of feedback is canonical. We shall also see that the delay enables us to capture the real-world behavior of the NOR latch (Figure 10).

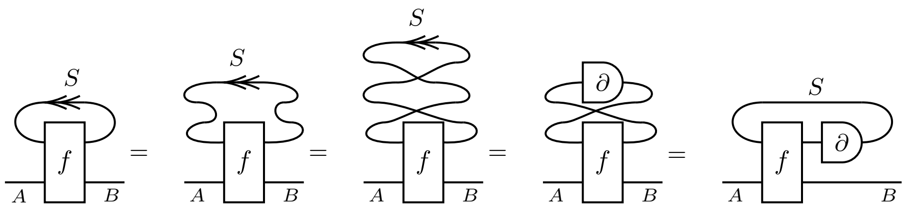


Fig. 12: Feedback in a compact closed category.

The emergence of state from feedback is witnessed by the St(•) construction, which we recall below.

3.4 St(•), the Free Feedback Category

Here we show how to obtain the free feedback category on a symmetric monoidal category. The St(•) construction is a general way of endowing a system with state. It appears multiple times in the literature in slightly different forms: it is used to arrive at a stateful resource calculus in [7]; a variant is used for geometry of interaction in [28]; it coincides with the free feedback category presented in [35]; and yet another, slightly different formulation was given in [31].

Definition 3.9 (Category of stateful processes, [35]). *Let $(\mathbf{C}, \otimes, I)$ be a symmetric monoidal category. We write* St$(\mathbf{C})$ *for the category with the objects of* $\mathbf{C}$ *but where morphisms $A \to B$ are pairs $(S \mid f)$, consisting of a* state space *$S \in \mathbf{C}$ and a morphism $f \colon S \otimes A \to S \otimes B$. We consider morphisms up to isomorphism classes of their state space, and thus*

$$(S \mid f) = (T \mid (h^{-1} \otimes \mathrm{id}) ⨾ f ⨾ (h \otimes \mathrm{id})), \qquad \text{for any isomorphism } h \colon S \cong T.$$

When depicting a stateful process (Figure 13), we mark the state strings.

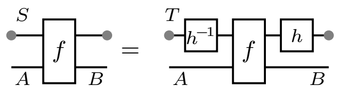


Fig. 13: Equivalence of stateful processes. We depict stateful processes by marking the space state.

We define the *identity stateful process* on $A \in \mathbf{C}$ as $(I \mid \mathrm{id}_{I \otimes A})$. *Sequential composition* of the two stateful processes $(S \mid f) \colon A \to B$ and $(T \mid g) \colon B \to C$ is defined by $(S \mid f) ⨾ (T \mid g) = (S \otimes T \mid (\sigma \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes f) ⨾ (\sigma \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes g))$, see Figure 14, left. *Parallel composition* of the two stateful processes $(S \mid f) \colon A \to B$ and $(S' \mid f') \colon A' \to B'$ is defined by $(S \mid f) \otimes (S' \mid f') = (S \otimes S' \mid (\mathrm{id} \otimes \sigma \otimes \mathrm{id}) ⨾ (f \otimes f') ⨾ (\mathrm{id} \otimes \sigma \otimes \mathrm{id}))$, see Figure 14, right. In both cases, the state spaces of the components are tensored together.

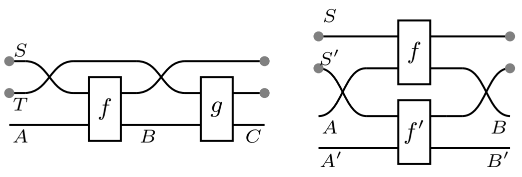


Fig. 14: Sequential and parallel composition of stateful processes.

This defines a symmetric monoidal category. Moreover, the operator

$$\mathsf{store}_T(S \mid f) \coloneqq (S \otimes T \mid f), \text{ for } f\colon S \otimes T \otimes A \to S \otimes T \otimes B,$$

which "stores" some information into the state, makes it a feedback category, see Figure 15.

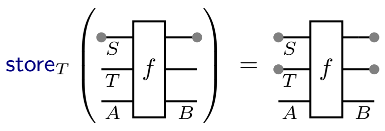


Fig. 15: The $\mathsf{store}(\bullet)$ operation, diagrammatically.

Proposition 3.10. *Sequential composition of stateful processes is associative. That is, for every* $f\colon S \otimes A \to S \otimes B$*, every* $g\colon T \otimes B \to T \otimes C$ *and every* $h\colon R \otimes C \to R \otimes D$*,*

$$((S \mid f) \fatsemi (T \mid g)) \fatsemi (R \mid h) = (S \mid f) \fatsemi ((T \mid g) \fatsemi (R \mid h)).$$

Proof. We can see both morphisms are equal by applying transformations of string diagrams: i.e. the axioms of symmetric monoidal categories (Figure 16).

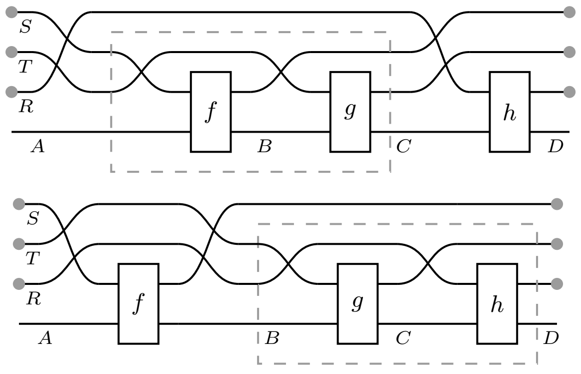


Fig. 16: Associativity of sequential composition.

The state spaces are isomorphic thanks to the associator $\alpha\colon (S \otimes T) \otimes R \to S \otimes (T \otimes R)$. □

Unitality and monoidality of stateful processes follow a similar reasoning. These properties yield the following result.

Theorem 3.11 ([35], Proposition 2.6). *The category* $\mathsf{St}(\mathbf{C})$*, endowed with the* $\mathsf{store}(\bullet)$ *operator, is the free feedback category over a symmetric monoidal category* $\mathbf{C}$*.*

Remark 3.12. Stateful processes are defined *up to isomorphism of the state space.* This is captured by axiom (A5) of feedback categories and, as mentioned in Section 3.2, relaxing it to allow sliding of arbitrary morphisms, would yield a notion of equality of stateful processes that would be too strong for our purposes: it would equate automata with a different number of states and boundary behavior (Example 4.14). Considering stronger notions of equivalence of processes is possible and leads to interesting models of computation [16]. Expanding this line of research is outside the scope of the present manuscript.

Remark 3.13 (Coherence and sliding). There are cases where we do need to be careful about the correct use of associators and unitors. For instance, we could be tempted to conclude that coherence implies that, for any $f\colon ((S\otimes T)\otimes R)\otimes A \to ((S\otimes T)\otimes R)\otimes B$, the following equation holds $((S\otimes T)\otimes R \mid f) = (S\otimes(T\otimes R) \mid f)$ without needing to invoke the equivalence relation of stateful processes. This would allow us to construct the category $\mathsf{St}(\bullet)$ of stateful processes without having to quotient them by the equivalence relation. However, this equality is only enabled by the fact that $\alpha_{S,T,R}$ is an isomorphism: we have

$$((S \otimes T) \otimes R \mid f) = (S \otimes (T \otimes R) \mid \alpha_{S,R,T} ⨟ f ⨟ \alpha^{-1}_{S,R,T}),$$

even if we write the equation omitting the coherence maps. This is also what will allow us to notate stateful processes diagramatically. We will mark the wires forming the state space; the order in which they are tensored does not matter thanks again to the equivalence relation that we are imposing.

3.5 Examples

All *traced monoidal categories* are feedback categories, since the axioms of feedback are a strict weakening of the axioms of trace. A more interesting source of examples is the $\mathsf{St}(\bullet)$ construction we just defined. We present some examples of state constructions below.

Example 3.14 (Mealy transition systems). A *Mealy deterministic transition system* with boundaries A and B, and state space S was defined [41, §2.1] to be just a function $f\colon S \times A \to S \times B$. It is not difficult to see that, up to isomorphism of the state space, they are morphisms of $\mathsf{St}(\mathbf{Set})$. They compose following Definition 3.9, and form a feedback category $\mathbf{Mealy} \coloneqq \mathsf{St}(\mathbf{Set})$.

Definition 3.15. *A* Mealy transition system *from A to B is a tuple* $\mathbf{M} = (S, t, o)$*, where S is a set called the* state space*, $t \colon S \times A \to S$ is a function called the* transition function*, and $o \colon S \times A \to B$ is a function called the* output function*.*

Two Mealy transition systems are equal whenever their transition functions are equal up to isomorphism of the state space. That is, two deterministic transition systems $\mathbf{M} = (S, t_M, o_M)$ *and* $\mathbf{N} = (T, t_N, o_N)$ *are considered equal whenever there exists an isomorphism $h \colon S \cong T$ between their state spaces such that*

$$h(t_M(s, a)) = t_N(h(s), a) \quad \textit{and} \quad o_M(s, a) = o_N(s, a).$$

Whenever $t(s_0, a) = s_1$ and $o(s_0, a) = b$, we write $s_0 \overset{a/b}{\to} s_1$. We may also write a transition and output in a single function, $f(s_0, a) = (t(s_0, a), o(s_0, a)) = (s_1, b)$.

The feedback of **Mealy** transition systems transforms input/output pairs into states. Figure 17 is an example: a transition system with a single state becomes a transition system with two states, $\{s_1, s_0\}$. We compute this feedback by transforming each transition $(s_i, i/s_o)$ into a transition $(i/)$ from s_i to s_o.

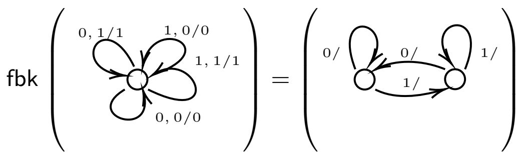


Fig. 17: Feedback of a Mealy transition system. Every transition has a label i/o indicating inputs (i) and outputs (o).

Example 3.16 (Elgot automata). Similarly, when we consider **Set** with the monoidal structure given by the disjoint union, we recover *Elgot automata* [18], which are given by a transition function $S + A \to S + B$. These transition systems motivate the work of Katis, Sabadini and Walters in [31,35].

Definition 3.17. *An* Elgot transition system *with initial states in A and final states in B is a tuple* $\mathbf{E} = (S, p, d)$ *where S is a set called the* state space*, $p \colon A \to S + B$ is a function called* initial step *and $d \colon S \to S + B$ is a function called* iterative step*.*

An Elgot transition system is interpreted as follows. We start by providing an initial state A. We then compute the initial step $p(a)$ which can result either in an internal state $p(a) = s \in S$ or in a final state $p(a) = b \in B$. In the later case, we are done and we return $b \in B$; in the former case, we repeatedly apply the iterative step: $d(p(a)), d(d(p(a))), \ldots$ until we reach a final state.

Example 3.18 (Linear dynamical systems). A *linear dynamical system* with inputs in $\mathbb{R}^n$, outputs in $\mathbb{R}^m$ and state space $\mathbb{R}^k$ is given by a number k, representing the dimension of the state space, and a matrix over the real numbers [30]

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathbf{Mat}(k+m, k+n).$$

Two linear dynamical systems,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \text{ and } \begin{pmatrix} A' & B' \\ C' & D \end{pmatrix},$$

are considered equivalent if there is an invertible matrix $H \in \mathbf{Mat}(k,k)$ such that $A' = H^{-1}AH$, $B' = BH$, and $C' = H^{-1}C$.

Linear dynamical systems are morphisms of a feedback category which coincides with $\mathsf{St}(\mathbf{Mat})$, the free feedback category over the category of matrices **Mat** as defined in Example 2.3. The feedback operator is defined by

$$\mathsf{fbk}_l \left(k, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix} \right) = \left(k+l, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix} \right),$$

where $\left(\begin{smallmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{smallmatrix}\right) \in \mathbf{Mat}(k+l+m, k+l+n)$.

4 Span(Graph): an Algebra of Transition Systems

Span(**Graph**) [32] is an algebra of "open transition systems". It has applications in *concurrency theory* and *verification* [31,32,34,36,22], and has been recently applied to biological systems [20,21]. Just as ordinary Petri nets have an underlying (firing) semantics in terms of transition systems, **Span**(**Graph**) is used as a semantic universe for a variant of open Petri nets, see [52,9].

An *open transition system* is a morphism of **Span**(**Graph**): a transition graph endowed with two *boundaries* or *communication ports*. Each transition has an effect on each boundary, and this data is used for synchronization. This conceptual picture actually describes a subcategory, **Span**(**Graph**)$_*$, where boundaries are mere sets: the alphabets of synchronization signals. We shall recall the details of **Span**(**Graph**)$_*$ and prove that it is universal, our main result:

Span(**Graph**)$_*$ is the free feedback category over **Span**(**Set**).

4.1 The Algebra of Spans

Definition 4.1. *A* span [4,10] *from A to B, both objects of a category* **C***, is a pair of morphisms with a common domain,*

$$A \xleftarrow{f} E \xrightarrow{g} B.$$

The object E is the "head" of the span, and the morphisms $f\colon E \to A$ and $g\colon E \to B$ are the left and right "legs", respectively.

When the category **C** has pullbacks, we can sequentially compose two spans $A \leftarrow E \rightarrow B$ and $B \leftarrow F \rightarrow C$ obtaining $A \leftarrow E \times_B F \rightarrow C$. Here, $E \times_B F$ is the pullback of E and F along B: for instance, in **Set**, $E \times_B F$ is the subset of $E \times F$ given by pairs that have the same image in B.

Remark 4.2 (Notation for spans). We denote a span $A \overset{f}{\leftarrow} X \overset{g}{\rightarrow} B$ in **C** as

$$\{f(x); g(x)\}_{x \in X} \in \mathbf{Span}(A, B),$$

where $x\colon U \to X$, for some object U of **C**, can be thought of as some generalized element that we compose with the two legs: e.g. in the category of sets, when $U = 1$, elements of a set X can be seen as functions $x\colon 1 \to X$. Sometimes, these generalized elements will come with conditions that must be listed with the morphism set. For instance, in Figure 18, a composition of spans has a pullback as its head, so any generalized element of its head is now a pair of morphisms $x\colon U \to X$ and $y\colon U \to Y$ satisfying the extra condition $g(x) = h(y)$:

$$\{f(x); g(x)\}_x \fatsemi \{h(y); k(y)\}_y = \{f(x); k(y)\}_{x,y}^{g(x)=h(y)}.$$

$$\begin{tikzcd}
 & & X \times_B Y \arrow[dl, "\pi_X"'] \arrow[dr, "\pi_Y"] & & \\
 & X \arrow[dl, "f"'] \arrow[dr, "g"] & & Y \arrow[dl, "h"'] \arrow[dr, "k"] & \\
A & & B & & C
\end{tikzcd}$$

Fig. 18: Composition of spans.

In other words, we are saying that the set of generalized elements of the head of the span is $\{x, y \mid g(x) = h(y)\}$. The advantage of this notation is that we can reason in any category with finite limits as we do in the category of sets: using *elements*. Whenever two sets of generalized elements of the head of a span are isomorphic, the Yoneda lemma [39] provides an isomorphism between the heads. That isomorphism makes the two spans equivalent when it commutes with the two legs.

Definition 4.3. *Let* **C** *be a category with pullbacks.* **Span(C)** *is the category that has the same objects as* **C** *and isomorphism classes of spans between them as morphisms. That is, two spans are considered* equal *if there is an isomorphism between their heads that commutes with both legs. Dually, if* **C** *is a category with pushouts,* **Cospan(C)** *is the category* **Span**$(\mathbf{C}^{op})$.

Span(**C**) is a symmetric monoidal category when **C** has products. The parallel composition of $\{f_1(x); g_1(x)\}_{x \in X} \in$ **Span**(A_1, B_1) and $\{f_2(y); g_2(y)\}_{y \in Y} \in$ **Span**(A_2, B_2) is given by the componentwise product

$$\{(f_1(x), f_2(y)); (g_1(x), g_2(y))\}_{x \in X, y \in Y} \in \mathbf{Span}(A_1 \times A_2, B_1 \times B_2).$$

An example is again **Span**(**Set**).

Remark 4.4 (Variable change). We will be considering spans "up to isomorphism of their head". This means that, given any isomorphism $\phi \colon X \to Y$, the following two spans are considered equal

$$\{f(\phi(x)); g(\phi(x))\}_{x \in X} = \{f(y); g(y)\}_{y \in Y}.$$

Moreover, if two spans are equal, then such a variable change does necessarily exist.

Example 4.5. Let us now detail some useful constants of the algebra of **Span**(**C**), which we will use to construct the NOR latch circuit from Figure 10.

The Frobenius algebra [10] ($\prec$, $\succ$, $\bullet\!-$, $-\!\bullet$) is used for the "wiring". The following spans are constructed out of diagonals $A \to A \times A$ and units $A \to 1$.

$$(\prec)_A = \{a; (a, a)\}_{a \in A} \in \mathbf{Span}(A, A \times A) \qquad (-\!\bullet)_A = \{a; *\}_a \in \mathbf{Span}(A, 1)$$
$$(\succ)_A = \{(a, a); a\}_{a \in A} \in \mathbf{Span}(A \times A, A) \qquad (\bullet\!-)_A = \{*; a\}_a \in \mathbf{Span}(1, A)$$

These induce a compact closed structure (and thus a trace), as follows:

$$(\bullet\!\!-\!\!\subset)_A = \{*; (a,a)\}_{a\in A} \in \mathbf{Span}(1, A\times A)$$
$$(\supset\!\!-\!\!\bullet)_A = \{(a,a); *\}_{a\in A} \in \mathbf{Span}(A\times A, 1).$$

Finally, we have a braiding making the category symmetric,

$$(\times) = \{(a,b); (b,a)\}_{a\in A, b\in B} \in \mathbf{Span}(A\times B, B\times A).$$

In general, any function $f\colon A\to B$ can be lifted to a span $\{a; f(a)\}_{a\in A} \in \mathbf{Span}(A,B)$ covariantly, and to a span $\{f(a); a\}_{a\in A} \in \mathbf{Span}(B,A)$, contravariantly.

4.2 The Algebra of Open Transition Systems

Definition 4.6. *The category* **Graph** *has graphs $G = (s,t\colon E \rightrightarrows V)$ as objects, i.e. pairs of morphisms from* edges *to* vertices *returning the* source *and* target *of each edge. A morphism of graphs $(e,v)\colon G\to G'$ is given by functions $e\colon E\to E'$ and $v\colon V\to V'$ such that $e\,⨟\,s' = s\,⨟\,v$ and $e\,⨟\,t' = t\,⨟\,v$ (see Figure 19)*[7]*.*

$$\begin{tikzcd}
E \arrow[r, "e"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & E' \arrow[d, bend right, "s'"'] \arrow[d, bend left, "t'"] \\
V \arrow[r, "v"] & V'
\end{tikzcd}$$

Fig. 19: Morphism of graphs.

We now focus on $\mathbf{Span}(\mathbf{Graph})_*$, those spans of graphs that have single vertex graphs $(A \rightrightarrows 1)$ as the boundaries.

Definition 4.7. *An* open transition system *is a morphism of $\mathbf{Span}(\mathbf{Graph})_*$: a span of sets $\{f(e); g(e)\}_{e\in E} \in \mathbf{Span}(A,B)$ where the head is the set of edges of a graph $s,t\colon E \rightrightarrows V$, i.e. the transitions (see Figure 20). Two open transition systems are considered* equal *if there is an isomorphism between their graphs that commutes with the legs. Open transition systems whose graph $E \rightrightarrows 1$ has a single vertex are called* stateless*.*

$$\begin{tikzcd}
A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & B \arrow[d, bend right] \arrow[d, bend left] \\
1 & V \arrow[l] \arrow[r] & 1
\end{tikzcd}$$

Fig. 20: A morphism of $\mathbf{Span}(\mathbf{Graph})_*$.

[7] Equivalently, **Graph** is the presheaf category on the diagram $(\bullet \rightrightarrows \bullet)$, i.e. the category of functors $(\bullet \rightrightarrows \bullet) \to \mathbf{Set}$ and natural transformations between them.

Sequential composition (the *communicating-parallel operation* of [32]) of two open transition systems with spans

$$\{f(e); g(e)\}_{e \in E} \in \mathbf{Span}(A, B) \text{ and } \{h(e'); k(e')\}_{e' \in E'} \in \mathbf{Span}(B, C)$$

and graphs $s, t\colon E \rightrightarrows S$ and $s', t'\colon E' \rightrightarrows S'$ yields the open transition system with the composite span

$$\{f(e); k(e')\}^{g(e)=h(e')}_{(e,e') \in E \times E'} \in \mathbf{Span}(A, C)$$

and graph $(s \times s', t \times t')\colon E \times_B E' \rightrightarrows S \times S'$. This means that the only allowed transitions are those that synchronize E and E' on the common boundary B.

Parallel composition (the *non communicating-parallel operation* of [32]) of two open transition systems with spans

$$\{f(e); g(e)\}_{e \in E} \in \mathbf{Span}(A, B) \text{ and } \{f'(e'); g'(e')\}_{e' \in E'} \in \mathbf{Span}(A', B')$$

and graphs $s, t\colon E \rightrightarrows V$ and $s', t'\colon E' \rightrightarrows V'$ yields the open transition system with span

$$\{(f(e), f'(e')); (g(e), g(e'))\}_{e, e' \in E \times E'} \in \mathbf{Span}(A \times A', B \times B')$$

and graph $(s \times s', t \times t')\colon E \times E' \rightrightarrows V \times V'$.

Remark 4.8 (Components of $\mathbf{Span}(\mathbf{Graph})_*$*).* Any span in $\mathbf{Span}(A, B)$ can be lifted to $\mathbf{Span}(\mathbf{Graph})_*(A, B)$ by making the head represent the graph $E \rightrightarrows 1$. Apart from the components lifted from $\mathbf{Span}$, which we call *stateless*, we will need to add a single stateful component to model all of $\mathbf{Span}(\mathbf{Graph})_*$: the delay in Figure 21.

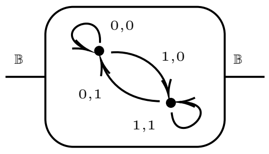


Fig. 21: Delay morphism over the set $\mathbb{B} := \{0, 1\}$.

The delay ($\text{-}\boxed{\partial}\text{-}_A$) on a given set A is given by the span $\{a_2; a_1\}_{a_1, a_2 \in A \times A} \in \mathbf{Span}(A, A)$ together with the graph $\pi_1, \pi_2\colon A \times A \to A$. This is to say that the delay receives on the left what the target of its transition (its next state) will be, while signalling on the right what the source of its transition (its current state) is. This is *not* an arbitrary choice: it is defined as the canonical delay obtained from the feedback structure in $\mathbf{Span}(\mathbf{Graph})_*$ (as in Section 3, $\partial_A = \mathsf{fbk}(\sigma_{A,A})$).

$$(\text{-}\boxed{\partial}\text{-})_A = \mathsf{fbk}(\{(a_1, a_2); (a_2, a_1)\}_{a_1, a_2}).$$

We can use this delay to correctly model a stateful NOR latch from the function $\texttt{NOR}\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ (as we saw in Figure 2).

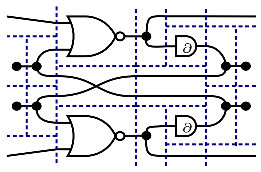


Fig. 22: Decomposing the circuit.

The NOR latch circuit of Figure 10 is the composition of two NOR gates where the outputs of each gate have been copied and fed back as input to the other gate. The algebraic expression, in **Span**(**Graph**)$_*$, of this circuit is obtained by decomposing it into its components, as in Figure 22.

$$(\mathrm{id} \otimes \bullet\!\!\prec \otimes \bullet\!\!\prec \otimes \mathrm{id}) \fatsemi (\texttt{NOR} \otimes \sigma \otimes \texttt{NOR}) \fatsemi (\prec \otimes \mathrm{id} \otimes \prec) \\ \fatsemi (\mathrm{id} \otimes \partial \otimes \mathrm{id} \otimes \partial \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \succ\!\!\bullet \otimes \succ\!\!\bullet \otimes \mathrm{id})$$

The graph obtained from computing this expression, together with its transitions, is shown in Figure 23. This time, our model is indeed stateful. It has four states: two states representing a correctly stored signal, $\overline{\mathsf{A}} = (1,0)$ and $\mathsf{A} = (0,1)$; and two states representing transitory configurations $\mathsf{T}_1 = (0,0)$ and $\mathsf{T}_2 = (1,1)$.

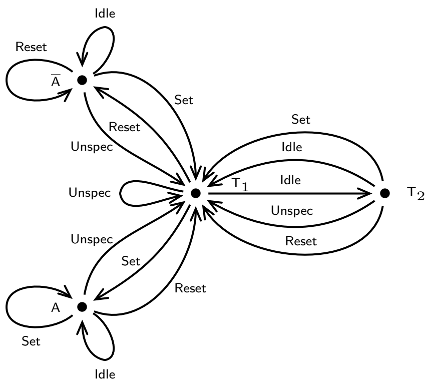


Fig. 23: Span of graphs representing the NOR latch

The *left boundary* can receive a *set* signal, $\mathsf{Set} = \binom{1}{0}$; a *reset* signal, $\mathsf{Reset} = \binom{0}{1}$; none of the two, $\mathsf{Idle} = \binom{0}{0}$; or both of them at the same time, $\mathsf{Unspec} = \binom{1}{1}$, which is known to cause unspecified behavior in a NOR latch. The signal on the *right boundary*, on the other hand, is always equal to the state the transition goes to and does not provide any additional information: we omit it from Figure 23.

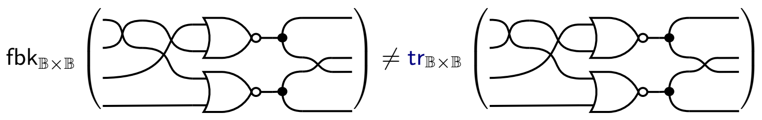

$\mathsf{fbk}_{\mathbb{B}\times\mathbb{B}}$
$\neq$ $\mathsf{tr}_{\mathbb{B}\times\mathbb{B}}$

Fig. 24: Applying $\mathsf{fbk}(\bullet)$ over the circuit gives the NOR latch.

Activating the signal Set makes the latch reach the state A in (at most) two transition steps. Activating Reset does the same for $\overline{\mathsf{A}}$. After any of these two cases, deactivating all signals, Idle, keeps the last state.

Moreover, the (real-world) NOR latch has some unspecified behavior that gets also reflected in the graph: activating both Set and Reset at the same time, what we call Unspec, causes the circuit to enter an unstable state where it bounces between the states T_1 and T_2 after an Idle signal. Our modeling has reflected this "unspecified behavior" as expected.

Feedback and trace. In terms of feedback, the circuit of Figure 23 is equivalently obtained as the result of taking feedback over the stateless morphism in Figure 24.

But $\mathbf{Span}(\mathbf{Graph})_*$ is also canonically traced: it is actually compact closed. What changes in the modeling if we would have used the trace instead? As we argued for Figure 8, we obtain a stateless transition system. The valid transitions are

$$\{(\mathsf{Unspec}, \mathsf{T}_1), (\mathsf{Idle}, \mathsf{A}), (\mathsf{Idle}, \overline{\mathsf{A}}), (\mathsf{Set}, \mathsf{A}), (\mathsf{Reset}, \overline{\mathsf{A}})\}.$$

They encode important information: they are the *equilibrium* states of the circuit. However, unlike the previous graph, this one would not get us the correct allowed transitions: under this modeling, our circuit could freely bounce between $(\mathsf{Idle}, \mathsf{A})$ and $(\mathsf{Idle}, \overline{\mathsf{A}})$, which is not the expected behavior of a NOR latch.

The fundamental piece making our modeling succeed the previous time was feedback with *delay*. Next we show that this feedback is canonical.

4.3 Span(Graph) as a Feedback Category

This section presents our main theorem. We introduce a mapping that associates to each stateful span of sets a corresponding span of graphs. This mapping is well-defined and lifts to a functor $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$. Finally, we prove that it is an isomorphism $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*$.

First of all, we need to be able to explicitly compute the composition of stateful spans, following the composition of stateful morphisms in Definition 3.9. This is Proposition 4.9. Then, we will characterize isomorphisms in the category of spans in Proposition 4.10.

Proposition 4.9. *Let* $\mathbf{C}$ *be a category with finite limits. Consider two stateful spans in the category* $\mathsf{St}(\mathbf{Span}(\mathbf{C}))$,

$$\{(\sigma(x), f(x)); (\sigma'(x), g(x))\}_{x\in X} \in \mathbf{Span}(S \times A, S \times B),$$

$$\{(\tau(y), h(y)); (\tau'(y), k(y))\}_{y \in Y} \in \mathbf{Span}(T \times B, T \times C).$$

their composition is then given by the span

$$\{(\sigma(x), \tau(y), f(x)); (\sigma'(x), \tau'(y), k(y))\}^{g(x)=h(y)}_{(x,y)\in X\times Y} \in \mathbf{Span}(S\times T\times A, S\times T\times B),$$

where the head is $X \times_B Y$, the pullback of g and h.

Proof. Using the notation for spans, we apply the definition of sequential composition in a category of stateful processes (Definition 3.9).

$$\begin{aligned}
&(\{(s,t);(t,s)\}_{s,t} \otimes \{a;a\}_a) \fatsemi (\{t;t\}_t \otimes \{(\sigma(x), f(x)); (\sigma'(x), g(x))\}_x) \\
&\qquad \fatsemi (\{(t,s);(s,t)\}_{s,t} \otimes \{b;b\}_b) \fatsemi (\{s;s\}_s \otimes \{(\tau(y), h(y)); (\tau'(y), k(y))\}_y) \\
= \quad & \textit{(Computing tensors)} \\
&\{(s,t,a);(t,s,a)\}_{s,t,a} \fatsemi \{(t,\sigma(x),f(x));(t,\sigma'(x),g(x))\}_{t,x} \\
&\qquad \fatsemi \{(t,s,b);(s,t,b)\}_{s,t,b} \fatsemi \{(s,\tau(y),h(y));(s,\tau'(y),k(y))\}_{s,y} \\
= \quad & \textit{(Computing compositions)} \\
&\{(\sigma(x),t,f(x));(t,\sigma'(x),g(x))\}_{t,x} \fatsemi \{(\tau(y),s,h(y));(s,\tau'(y),k(y))\}_{s,y} \\
= \quad & \textit{(Computing compositions)} \\
&\{(\sigma(x),\tau(y),f(x));(\sigma'(x),\tau'(y),k(y))\}^{g(x)=h(y)}_{x,y}.
\end{aligned}$$

This last formula corresponds indeed to the pullback we stated. □

Proposition 4.10. *Let* **C** *be a category with all finite limits. An isomorphism $A \cong B$ in its category of spans,* **Span**(**C**)*, is always of the form*

$$\{a; \phi(a)\}_{a\in A} \in \mathbf{Span}(\mathbf{C})(A,B),$$

where the left leg is an identity and the right leg $\phi \colon A \to B$ is an isomorphism in the base category **C**.

Proof. Let $\{f(x); g(x)\}_{x\in X} \in \mathbf{Span}(A,B)$ and $\{h(y); k(y)\}_{y\in Y} \in \mathbf{Span}(B,A)$ be mutual inverses. This means that

$$\{f(x);g(x)\}_{x\in X} \fatsemi \{h(y);k(y)\}_{y\in Y} = \{f(x);k(y)\}^{g(x)=h(y)}_{x,y} = \{a;a\}_{a\in A},$$

$$\{h(y);k(y)\}_{y\in Y} \fatsemi \{f(x);g(x)\}_{x\in X} = \{h(y);g(x)\}^{k(y)=f(x)}_{x,y} = \{b;b\}_{b\in B}.$$

In turn, this implies the existence of variable changes $(\alpha_X, \alpha_Y) \colon A \to X \times_B Y$ and $(\beta_Y, \beta_X) \colon B \to Y \times_A X$ such that they are the inverses of (f,k) and (g,h) respectively.

We can thus write the spans as having the identity on the left leg.

$$\{f(x);g(x)\}_{x\in X} = \{f(\alpha_X(a)); g(\alpha_X(a))\}_{a\in A} = \{a; g(\alpha_X(a))\}_{a\in A}.$$

$$\{h(y);k(y)\}_{y\in Y} = \{h(\beta_Y(b)); k(\beta_Y(b))\}_{b\in B} = \{b; k(\beta_Y(b))\}_{b\in B}.$$

Finally, composing them again, we get that $(g \fatsemi \alpha_X)$ and $(k \fatsemi \beta_Y)$ must be mutual inverses, thus isomorphisms. □

We are now ready to prove the main result. The following Lemma 4.11 proves that we can translate stateful spans to spans of graphs. The main Theorem 4.12 follows from it.

Lemma 4.11. *Let* $\mathbf{C}$ *be a category with all finite limits. The following assignment of stateful processes over* $\mathbf{Span}(\mathbf{C})$ *to morphisms of* $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))$ *is well-defined.*

$$K\left(S \;\middle|\; \begin{tikzcd} & E \arrow[dl, "{(s,f)}"'] \arrow[dr, "{(t,g)}"] & \\ S\times A & & S\times B \end{tikzcd} \right) := \left(\begin{tikzcd} A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & B \arrow[d, bend right] \arrow[d, bend left] \\ 1 & S \arrow[l] \arrow[r] & 1 \end{tikzcd} \right)$$

Proof. We first check that two *isomorphic* spans are sent to *isomorphic* spans of graphs. Let

$$\{(s(e), f(e)); (t(e), g(e))\}_{e\in E} \in \mathbf{Span}(S\times A, S\times B) \text{ and}$$
$$\{(s'(e'), f'(e')); (t'(e'), g'(e'))\}_{e'\in E'} \in \mathbf{Span}(S\times A, S\times B)$$

be two spans that are isomorphic with the variable change $\phi\colon E \cong E'$. Then, (ϕ, id) is an isomorphism of spans of graphs, also making the relevant diagram commute (Figure 25).

$$\begin{tikzcd} & E \arrow[dl, "{(s,f)}"'] \arrow[dr, "{(t,g)}"] \arrow[dd, "\phi"] & \\ S\times A & & S\times B \\ & E' \arrow[ul, "{(s',f')}"] \arrow[ur, "{(t',g')}"'] & \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "f"'] \arrow[r, "\phi"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] \arrow[rrr, bend left=40, "g"] & E' \arrow[r, "g'"] \arrow[d, bend right, "s'"'] \arrow[d, bend left, "t'"] \arrow[ll, bend right=40, "f'"'] & B \arrow[d, bend right] \arrow[d, bend left] \\ 1 & S \arrow[l] \arrow[r, "\mathrm{id}"] \arrow[rr, bend right=40] & S \arrow[r] \arrow[ll, bend left=40] & 1 \end{tikzcd}$$

Fig. 25: Isomorphic spans result in isomorphic spans of graphs.

We show now that the assignment preserves the equivalence relation of stateful processes. Isomorphisms in a category of spans are precisely spans whose two legs are isomorphisms (by Proposition 4.10, or the more general result of [43]). This means that an isomorphism in $\mathbf{Span}(\mathbf{Set})$ can be always rewritten as $\{s; \phi(s)\}_{s\in S} \in \mathbf{Span}(S, T)$, where the left leg is an identity and the right leg is $\phi\colon S \to T$, some isomorphism. Its inverse can be written analogously as $T \leftarrow S \to S$. In order to prove that the quotient relation induced by the feedback is preserved, we need to check that equivalent spans of sets are sent to isomorphic spans of graphs. If two spans are equivalent with the variable change $\phi\colon S \cong T$, then the corresponding graphs are isomorphic with the isomorphism of graphs (id, ϕ), see Figure 26.

$$\begin{tikzcd}
& \cdot \arrow[dl, "\phi\times\mathrm{id}"'] \arrow[dr, dash] & & E \arrow[dl, "{(s,a)}"'] \arrow[dr, "{(t,b)}"] & & \cdot \arrow[dl, dash] \arrow[dr, "\phi\times\mathrm{id}"] & \\
T\times A & & S\times A & & S\times B & & T\times B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, shift right] \arrow[d, shift left] & E \arrow[l, "a"'] \arrow[r, "\mathrm{id}"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] \arrow[rrr, bend left, "b"] & E \arrow[r, "b"] \arrow[ll, bend right, "a"'] \arrow[d, shift right, "s\mathbin{;}\phi"'] \arrow[d, shift left, "t\mathbin{;}\phi"] & B \arrow[d, shift right] \arrow[d, shift left] \\
1 & S \arrow[l] \arrow[r, "h"] \arrow[rr, bend right] & T \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

Fig. 26: Equivalent spans result in isomorphic spans of graphs.

□

Theorem 4.12. *There exists an isomorphism of categories* $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*$. *That is, the free feedback category over* $\mathbf{Span}(\mathbf{Set})$ *is isomorphic to the full subcategory of* $\mathbf{Span}(\mathbf{Graph})$ *given by single-vertex graphs.*

Proof. We prove that there is a fully faithful functor $K\colon \mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$ defined on objects as $K(A) = (A \rightrightarrows 1)$ and defined on morphisms as in Lemma 4.11.

We now show that K is functorial, preserving composition and identities. The identity morphism on A in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ has state space 1, so it is a span $1 \times A \leftarrow A \to 1 \times A$ and it is sent to the identity span on the graph $A \rightrightarrows 1$.

Composition is also preserved. Let us consider two stateful spans

$$\{(s(e), a(e)); (s'(e), b(e))\}_{e\in E} \in \mathbf{Span}(S\times A, S\times B) \text{ and}$$
$$\{(t(f), b'(f)); (t'(f), c(f))\}_{f\in F} \in \mathbf{Span}(S\times B, S\times C)$$

By Proposition 4.9, their composition is given by the span

$$\{(s(e), t(f), a(e)); (s'(e), t'(f), c(f))\}^{b(e)=b'(f)}_{(e,f)\in E\times F} \in \mathbf{Span}(S\times T\times A, S\times T\times C)$$

where the head $E \times_B F$ is the pullback of b and b'.

$$\begin{tikzcd}
& & E\times_B F \arrow[dl, "\pi_E"'] \arrow[d, shift right] \arrow[d, shift left] \arrow[dr, "\pi_F"] & & \\
& E \arrow[dl, "a"'] \arrow[d, shift right] \arrow[d, shift left] \arrow[dr, "b"] & S\times T \arrow[dl, "\pi_S"'] \arrow[dr, "\pi_T"] & F \arrow[dl, "b'"'] \arrow[d, shift right] \arrow[d, shift left] \arrow[dr, "c"] & \\
A \arrow[d, shift right] \arrow[d, shift left] & S \arrow[dl] \arrow[dr] & B \arrow[d, shift right] \arrow[d, shift left] & T \arrow[dl] \arrow[dr] & C \arrow[d, shift right] \arrow[d, shift left] \\
1 & & 1 & & 1
\end{tikzcd}$$

Fig. 27: Pullback of graphs.

We have composed two stateful spans and we want to show that the graph corresponding to their composition is the pullback of the graphs corresponding to them. Computing a pullback of graphs can be done separately on edges and vertices, as graphs form a presheaf category (see Figure 27). Note how the resulting graph is precisely the graph corresponding, under the assignment K, to the stateful span computed above.

We have shown that K is a functor. The final step is to show that it is fully-faithful. We can see that it is full: every span of single-vertex graphs given by $\{f(e); g(e)\}_{e \in E} \in \mathbf{Span}(A, B)$ and $s, t \colon E \rightrightarrows S$ is the image of some span, namely

$$\{s(e), f(e); t(e), g(e)\}_{e \in E} \in \mathbf{Span}(S \times A, S \times B).$$

Let us check it is also faithful. Suppose that two morphisms in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$, $S \times A \leftarrow E \rightarrow S \times B$ and $S' \times A \leftarrow E' \rightarrow S' \times B$, are sent to equivalent spans of graphs, i.e. there exist $h \colon E \cong E'$ and $k \colon S' \cong S$ making the diagrams in Figure 28 commute.

$$\begin{tikzcd}
A \arrow[d, shift right] \arrow[d, shift left] & E \arrow[l, "a"'] \arrow[r, "h"] \arrow[rr, bend left, "b"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & E' \arrow[r, "b'"] \arrow[ll, bend right, "a'"'] \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] & B \arrow[d, shift right] \arrow[d, shift left] \\
1 & S \arrow[l] \arrow[r, "k"] \arrow[rr, bend right] & S' \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

Fig. 28: Equivalent spans of graphs.

The isomorphism k makes the following spans equivalent as stateful processes.

$$S \times A \leftarrow E \rightarrow S \times B$$
$$S' \times A \leftarrow E \rightarrow S' \times B$$

Moreover, the isomorphism h makes the following spans equivalent as spans, showing faithfullness of K.

$$S' \times A \leftarrow E \rightarrow S' \times B$$
$$S' \times A \leftarrow E' \rightarrow S' \times B$$

We have shown that there exists a fully-faithful functor from the free feedback category over $\mathbf{Span}(\mathbf{Set})$ to the category $\mathbf{Span}(\mathbf{Graph})$ of spans of graphs. The functor restricts to an equivalence between $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ and the full subcategory of $\mathbf{Span}(\mathbf{Graph})$ on single-vertex graphs. It is moreover bijective on objects, giving an isomorphism of categories. $\square$

Example 4.13. The characterization $\mathbf{Span}(\mathbf{Graph})_* \cong \mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ that we prove in Theorem 4.12 lifts the inclusion $\mathbf{Set} \rightarrow \mathbf{Span}(\mathbf{Set})$ to a feedback preserving functor $\mathbf{Mealy} \rightarrow \mathbf{Span}(\mathbf{Graph})_*$. This inclusion embeds a deterministic transition system into the graph that determines it.

Example 4.14. Following from Remark 3.12, we present an example of spans of graphs that would be equated if we assumed the sliding axiom (A5) of feedback categories for arbitrary morphisms rather than just isomorphisms. Consider the spans of sets $\alpha\colon W \to V$ and $h\colon V \to W \times \mathbb{B}$ as in Figure 29, where $V = \{v_1, v_2\}$ and $W = \{w\}$. Depending on where the feedback operation is applied, we obtain two different spans of graphs, $g = \mathsf{fbk}_V(h \,⨟\, (\alpha \otimes \mathrm{id}))$ and $f = \mathsf{fbk}_W((\alpha \otimes \mathrm{id}) \,⨟\, h)$: the first one contains an additional transition. If we were to impose that the sliding axiom holds for non-isomorphisms, we could erase this additional transition, and obtain that $f = g$ by sliding α through the feedback loop.

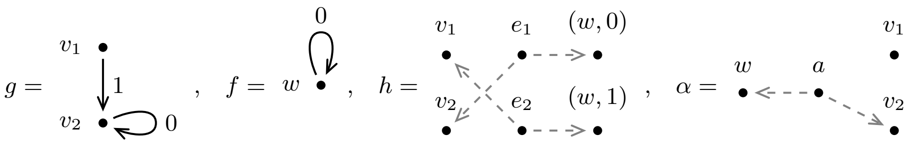


Fig. 29: Spans of graphs that would be equated by a stronger notion of equivalence: $g = \mathsf{fbk}(h \,⨟\, (\alpha \otimes \mathrm{id})) \sim \mathsf{fbk}((\alpha \otimes \mathrm{id}) \,⨟\, h) = f$.

4.4 Cospan(Graph) as a Feedback Category

Theorem 4.12 can be generalized to any category $\mathbf{C}$ with finite limits, where we can define graphs and spans of them.

A graph in a category $\mathbf{C}$ is given by two objects, E and V, and two morphisms in $\mathbf{C}$, the source and the target $s, t\colon E \to V$. A morphism of graphs $\alpha\colon G \to G'$ in $\mathbf{C}$ is a pair of morphisms, $\alpha_E\colon E \to E'$ and $\alpha_V\colon V \to V'$, in $\mathbf{C}$ that commute with the sources and the targets. In categorical terms, these can be reformulated as functors and natural transformations.

Definition 4.15. *Let* $\mathbf{C}$ *be a category with finite limits. A graph in* $\mathbf{C}$ *is a functor from the diagram* $(\bullet \rightrightarrows \bullet)$ *to* $\mathbf{C}$. *A morphism of graphs in* $\mathbf{C}$ *is a natural transformation between the corresponding functors. Graphs in* $\mathbf{C}$ *form a category* $\mathbf{Graph}(\mathbf{C})$.

Categories of functors into $\mathbf{C}$ have all the limits that $\mathbf{C}$ has [39]. We can then form the category $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))$ and take its full subcategory on objects of the form $A \rightrightarrows 1$, i.e. $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$, to obtain:

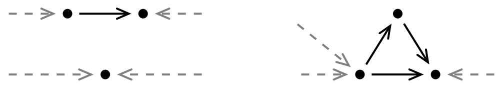
Theorem 4.16. *There exists an isomorphism of categories* $\mathsf{St}(\mathbf{Span}(\mathbf{C})) \cong \mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$. *That is, the free feedback category over* $\mathbf{Span}(\mathbf{C})$ *is equivalent to the full subcategory on* $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))$ *given by single-vertex graphs.*

$\mathbf{Cospan}(\mathbf{Graph})_*$ is the dual algebra to $\mathbf{Span}(\mathbf{Graph})_*$. Its morphisms represent graphs with discrete boundaries: while, in $\mathbf{Span}(\mathbf{Graph})_*$, each transition in the graph is assigned a boundary behavior, a morphism in $\mathbf{Cospan}(\mathbf{Graph})_*$

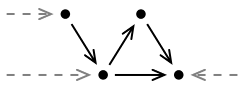
is a graph where some vertices are marked as left boundary or right boundary vertices. This allows graphs to be composed by identifying these boundary vertices.

Definition 4.17. *A* graph with discrete boundaries $g\colon X \to Y$ *is given by a graph* $G = (s, t\colon E \rightrightarrows V)$ *and two functions,* $l\colon X \to V$ *and* $r\colon Y \to V$*, marking the boundary vertices.*

Example 4.18. We represent the legs of a cospan as dashed arrows pointing to some vertices of the apex graph.



The composition of the above cospans of graphs is given by



,

where the vertices in the common boundary have been identified.

Cospan(**Graph**)$_*$ can be also characterized as a free feedback category. We know that **Cospan**(**Set**) $\cong$ **Span**(**Set**^{op}), we note that **Graph**(**Set**^{op}) $\cong$ **Graph**^{op}(**Set**) (which has the effect of flipping edges and vertices), and we can use Theorem 4.16 because **Set** has all finite colimits. The explicit assignment is similar to the one shown in Lemma 4.11.

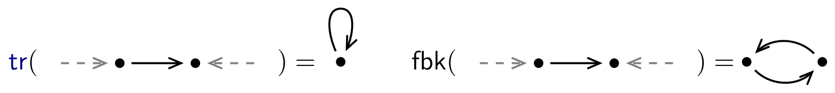
$$K\left(S \;\middle|\; \begin{tikzcd} & S & \\ E+A \arrow[ur, "{[t|a]}"] & & E+B \arrow[ul, "{[s|b]}"'] \end{tikzcd}\right) := \left(\begin{tikzcd} A \arrow[r, "a"] & S & B \arrow[l, "b"'] \\ 0 \arrow[u, bend left] \arrow[u, bend right] \arrow[r] & E \arrow[u, bend left, "t"] \arrow[u, bend right, "s"'] & 0 \arrow[l] \arrow[u, bend left] \arrow[u, bend right] \end{tikzcd}\right)$$

Corollary 4.19. *There is an isomorphism*

$$\mathsf{St}(\mathbf{Cospan}(\mathbf{Set})) \cong \mathbf{Cospan}(\mathbf{Graph})_*.$$

Cospan(**Graph**) is also compact closed and, in particular, traced. As in the case of **Span**(**Graph**), the feedback structure given by the universal property is different from the trace. In the case of **Cospan**(**Graph**), tracing has the effect of identifying the output and input vertices of the graph; while feedback adds an additional edge from the output to the input vertices.

Example 4.20. Tracing the cospan of a one-edge graph identifies the two vertices making it into a self-loop. On the other hand, taking feedback of the same cospan has the effect of adding another edge from the right boundary to the left one.



4.5 Syntactical Presentation of Cospan(FinGraph)

The observation in Proposition 3.7 has an important consequence in the case of finite sets. We write **FinGraph** for **Graph**(**FinSet**). **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [37], meaning that any morphism written out of the operations of a special commutative Frobenius monoid and the structure of a symmetric monoidal category is precisely a cospan of finite sets. Figure 30 represents the generators and the axioms of the generic special commutative Frobenius monoid.

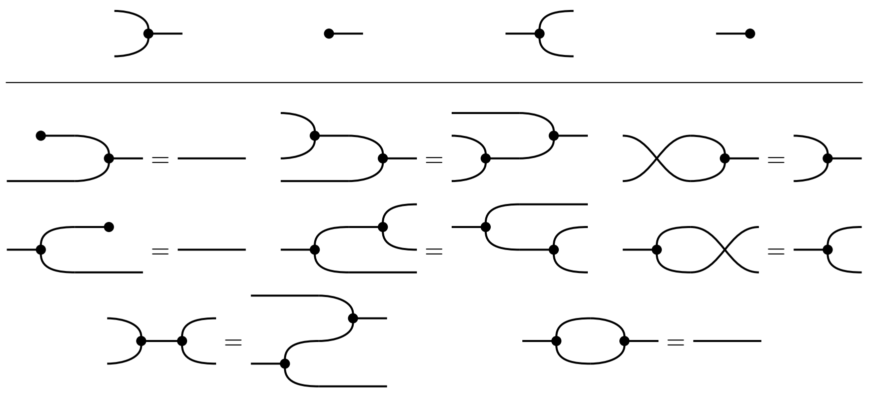


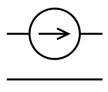
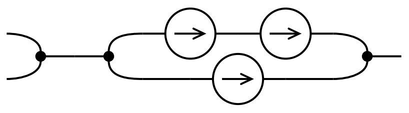
Fig. 30: Generators and axioms of the generic special commutative Frobenius monoid.

But we also know that **Cospan**(**FinSet**), with an added generator to its PROP structure [7] is St(**Cospan**(**FinSet**)), or, equivalently, **Cospan**(**FinGraph**). This means that any morphism written out of the operations of a special commutative Frobenius algebra plus a freely added generator of type $(\text{-}\boxed{\partial}\text{-}) \colon 1 \to 1$ is a morphism in **Cospan**(**FinGraph**)$_*$. This way, we recover one of the main results of [47] as a direct corollary of our characterization.

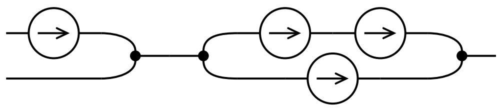
Proposition 4.21 (Proposition 3.2 of [47]). **Cospan**(**FinGraph**)$_*$ *is the generic special commutative Frobenius monoid with an added generator.*

Proof. It is known that the category **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [37]. Adding a free generator $(\text{-}\boxed{\partial}\text{-}) \colon 1 \to 1$ to its PROP structure corresponds to adding a family $(\text{-}\boxed{\partial}\text{-})_n \colon n \to n$ with the conditions on Proposition 3.7. Now, Proposition 3.7 implies that adding such a generator to **Cospan**(**FinSet**) results in St(**Cospan**(**FinSet**)). Finally, we use Theorem 4.12 to conclude that St(**Cospan**(**FinSet**)) $\cong$ **Cospan**(**FinGraph**)$_*$. □

Example 4.22. The delay generator $\text{-}\boxed{\partial}\text{-} \colon 1 \to 1$ in **Cospan**(**FinGraph**)$_*$ can be interpreted as a single edge. Thus, we draw it as $\text{-}\textcircled{\rightarrow}\text{-} : 1 \to 1$. The cospans of graphs in Example 4.18 are represented by the string diagrams

 and  .

Their composition is

 .

5 Structured state spaces

This section extends the framework of feedback categories from mere transition systems to automata with initial and final states. In order to achieve this, we generalize the feedback construction to deal with a richer structure. The key ingredient in the generalization of feedback categories is a close examination of the sliding axiom: deciding which processes can be "slid" determines the notion of equality we want to apply.

5.1 Structured Feedback Categories

In order to capture automata, the state space S needs to be equipped with "extra structure". This is achieved by letting the state space live in a different category $\mathbf{S}$ from the base category $\mathbf{C}$ and by having a way of "forgetting" the extra structure it carries through a monoidal functor $\mathsf{R}\colon \mathbf{S} \to \mathbf{C}$.

A structured feedback operator on $\mathbf{C}$, then, takes a morphism acting on a structured state space (Figure 31).

$$\frac{f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B}{\mathsf{fbk}_S(f)\colon A \to B}$$

Fig. 31: Type of the operator $\mathsf{fbk}_S(\bullet)$.

Definition 5.1 (Structured feedback category). *A structured feedback category is a symmetric monoidal category $(\mathbf{C}, \otimes, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}})$ together with a symmetric monoidal category, $(\mathbf{S}, \boxtimes, J, \alpha^{\mathbf{S}}, \lambda^{\mathbf{S}}, \rho^{\mathbf{S}})$, representing structured state spaces, and a symmetric monoidal functor $(\mathsf{R}, \varepsilon, \mu)\colon \mathbf{S} \to \mathbf{C}$ endowed with an operator $\mathsf{fbk}_S\colon \mathbf{C}(\mathsf{R}S \otimes A, \mathsf{R}S \otimes B) \to \mathbf{C}(A, B)$, which satisfies the following axioms (B1-B5).*

(B1). Tightening. *For every $S \in \mathbf{S}$, every morphism $f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$ and every pair of morphisms $u\colon A' \to A$ and $v\colon B \to B'$,*

$$u \mathbin{⨾} \mathsf{fbk}_S(f) \mathbin{⨾} v = \mathsf{fbk}_S((\mathrm{id} \otimes u) \mathbin{⨾} f \mathbin{⨾} (\mathrm{id} \otimes v)).$$

(B2). Vanishing. *For every $f\colon A \to B$,*

$$\mathsf{fbk}_J((\varepsilon^{-1} \otimes \mathrm{id}) \mathbin{⨾} f \mathbin{⨾} (\varepsilon \otimes \mathrm{id})) = f.$$

(B3). Joining. *For every $S, T \in \mathbf{S}$ and every morphism $f\colon \mathsf{R}S \otimes \mathsf{R}T \otimes A \to \mathsf{R}S \otimes \mathsf{R}T \otimes B$,*

$$\mathsf{fbk}_T(\mathsf{fbk}_S(f)) = \mathsf{fbk}_{S \otimes T}((\mu^{-1} \otimes \mathrm{id}) \mathbin{⨾} f \mathbin{⨾} (\mu \otimes \mathrm{id})).$$

(B4). Strength. *For every $S \in \mathbf{S}$, every morphism $f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$ and every morphism $g\colon A' \to B'$,*

$$\mathsf{fbk}_S(f) \otimes g = \mathsf{fbk}_S(f \otimes g).$$

(B5). Sliding. For every $S, T \in \mathbf{S}$, every $f\colon \mathsf{R}T \otimes A \to \mathsf{R}S \otimes B$ and every $h\colon S \to T$ in $\mathbf{S}$,

$$\mathsf{fbk}_T(f \fatsemi (\mathsf{R}h \otimes \mathrm{id})) = \mathsf{fbk}_S((\mathsf{R}h \otimes \mathrm{id}) \fatsemi f).$$

Remark 5.2. The sliding axiom encodes the fact that applying a transformation to the state space $h\colon S \to T$ just before computing the next state should be essentially the same as applying the same transformation to the state space just before retrieving the current state. In the particular case where all transformations are asked to be reversible (i.e. isomorphisms)[8], this sliding axiom (B5) particularizes to the sliding axiom of plain feedback categories (A5).

Definition 5.3 (Structured feedback functor). *A structured feedback functor $(F, G)\colon \mathbf{C} \to \mathbf{C}'$ between two structured feedback categories $(\mathbf{C}, \mathbf{S}, \mathsf{R}, \mathsf{fbk})$ and $(\mathbf{C}', \mathbf{S}', \mathsf{R}', \mathsf{fbk}')$ is a pair of symmetric monoidal functors, (F, ε, μ) and $(G, \varepsilon^G, \mu^G)$, with types $F\colon \mathbf{C} \to \mathbf{C}'$ and $G\colon \mathbf{S} \to \mathbf{S}'$ such that $R \fatsemi F = G \fatsemi R'$ and*

$$F(\mathsf{fbk}_S(f)) = \mathsf{fbk}'_{GS}(\mu \fatsemi Ff \fatsemi \mu^{-1}).$$

We write $\mathsf{SFeedback}$ *for the category of (small) structured feedback categories and structured feedback functors. There is a forgetful functor $\mathcal{U}\colon \mathsf{SFeedback} \to$* SymMon.

5.2 Structured St(•) Construction

In the same way that the free feedback category was realized by stateful processes, the free structured feedback category is realized by stateful processes with a structured state space $S \in \mathbf{S}$. A functor $\mathsf{R}\colon \mathbf{S} \to \mathbf{C}$ forgets the extra structure of this space.

Following the analogy, stateful processes with structured state space are pairs $(S \mid f)$ consisting of a structured state space $S \in \mathbf{S}$ and a morphism $f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$. We shall consider morphisms up to sliding of their state space, as in Figure 32.

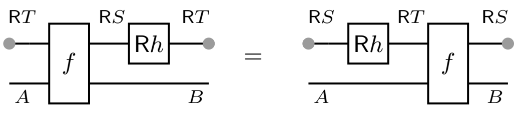


Fig. 32: Equivalence of structured stateful processes. We depict structured stateful processes by marking the state space.

Definition 5.4 (Category of structured stateful processes). *Consider a pair of symmetric monoidal categories $(\mathbf{C}, \otimes, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}})$ and $(\mathbf{S}, \boxtimes, J, \alpha^{\mathbf{S}}, \lambda^{\mathbf{S}}, \rho^{\mathbf{S}})$ and a symmetric monoidal functor $(\mathsf{R}, \varepsilon, \mu)\colon \mathbf{S} \to \mathbf{C}$. We write $\mathsf{St}(\mathbf{C}, \mathsf{R})$ for the*

[8] Here, $\mathbf{S}$ is the subcategory of isomorphisms of $\mathbf{C}$ and R is the inclusion functor.

category with the objects of $\mathbf{C}$ *but where morphisms* $A \to B$ *are pairs* $(S \mid f)$ *consisting of a state space* $S \in \mathbf{S}$ *and a morphism* $f \colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$*. We consider morphisms up to equivalence of their state spaces, where the equivalence relation is generated by*

$$(S \mid (\mathsf{R}h \otimes \mathrm{id}) \mathbin{⨟} f) \sim_{\mathbf{S}} (T \mid f \mathbin{⨟} (\mathsf{R}h \otimes \mathrm{id})) \text{ for any } h \colon S \to T.$$

Identities, composition, monoidal product and the feedback operator $\mathsf{store}(\bullet)$ *are defined in analogous ways as for stateful processes (Definition 3.9). When depicting a structured stateful process (Figure 32) we mark the state strings.*

Remark 5.5. In other words, structured stateful processes are elements of the following coproduct quotiented by the smallest equivalence relation $(\sim_{\mathbf{S}})$ satisfying sliding: $((\mathsf{R}h \otimes \mathrm{id}) \mathbin{⨟} f) \sim_{\mathbf{S}} (f \mathbin{⨟} (\mathsf{R}h \otimes \mathrm{id}))$.

$$\mathsf{St}(\mathbf{C}, \mathsf{R})(X, Y) := \left(\sum_{S \in \mathbf{S}} \mathbf{C}(\mathsf{R}S \otimes A, \mathsf{R}S \otimes B) \right) / (\sim_{\mathbf{S}}).$$

This quotient is a particular form of colimit called a *coend* [39].

Repeating the proof from Katis, Sabadini and Walters [32] in this generalized setting, we can show that structured stateful processes form the free structured feedback category.

Theorem 5.6. *The category* $\mathsf{St}(\mathbf{C}, \mathsf{R})$*, endowed with the* $\mathsf{store}(\bullet)$ *operator, is the free structured feedback category over a symmetric monoidal category* $\mathbf{C}$ *with a symmetric monoidal functor* $\mathsf{R} \colon \mathbf{S} \to \mathbf{C}$*.*

5.3 Categories of Automata

Let us present classical automata as an example of the construction of structured feedback. Automata have a structured state space where a particular state is considered the "initial state", and a subset of states are considered "final". We can canonically recover a suitable category of automata as the free feedback category over these structured spaces.

Definition 5.7. *An* automaton state space (S, i_S, f_S) *is a finite set* S *together with an* initial state $i_S \in S$ *and a subset of final states* $f_S \colon S \to 2$*. The product of two automaton state spaces,* (S, i_S, f_S) *and* (T, i_T, f_T)*, is the state space* $(S \times T, (i_S, i_T), f_S \wedge f_T)$*, where the final states are pairs of final states,* $(f_S \wedge f_T)(s,t) = f_S(s) \wedge f_T(t)$*. A morphism of automaton state spaces* $\alpha \colon (S, i_S, f_S) \to (T, i_T, f_T)$ *is a function* $\alpha \colon S \to T$ *such that* $i_S \mathbin{⨟} \alpha = i_T$ *and* $f_S = \alpha \mathbin{⨟} f_T$*. Automaton state spaces form a symmetric monoidal category,* $\mathbf{AutSt}$*.*

Remark 5.8. As a consequence, an isomorphism of automata state spaces

$$\alpha \colon (S, i_S, f_S) \cong (T, i_T, f_T)$$

is an isomorphism $\alpha \colon S \cong T$ such that $i_S \mathbin{⨟} \alpha = i_T$ and $f_S = \alpha \mathbin{⨟} f_T$. These form a subcategory $\mathbf{Iso}(\mathbf{AutSt})$ with forgetful functors $\mathsf{U}_{\mathsf{Iso}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{FinSet}$ and $\mathsf{U}_{\mathsf{Aut}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{Span}(\mathbf{FinSet})$.

Definition 5.9. A Mealy deterministic finite automaton (S, A, B, i_S, f_S, t_S) *is given by a finite set of states* S*, a finite alphabet of input symbols* A *and a finite alphabet of output symbols* B*, an initial state* $i_S \in S$*, a set of final states* $f_S \colon S \to 2$*, and a transition function* $t_S \colon S \times A \to S \times B$*. The product of two deterministic finite automata,*

$$(S, A, B, i_S, f_S, t_S) \text{ and } (S', A', B', i_{S'}, f_{S'}, t_{S'}),$$

is the automaton $(S \times T, A, C, (i_S, i_{S'}), (f_S \wedge f_{S'}), (t_S \times t_{S'}))$*, where the transition function computes a pair of independent transitions,*

$$(t_S \times t_{S'})(s, s', a, a') = (t_S(s, a), t_{S'}(s', a')).$$

The sequential synchronization of two deterministic finite automata,

$$(S, A, B, i_S, f_S, t_S) \text{ and } (T, B, C, i_T, f_T, t_T),$$

is the automaton $(S \times T, A, C, (i_S, i_T), (f_S \wedge f_T), (t_S \wedge t_T))$*, where the transition function* $(t_S \wedge t_T)$ *uses the output of the first transition as the input of the second*

$$(t_S \wedge t_T)(s, t, a) = (s', t', c) \text{ where } t_S(s, a) = (s', b) \text{ and } t_T(t, b) = (t', c).$$

We consider Mealy deterministic automata up to isomorphism of their state space. Mealy deterministic finite automata form a symmetric monoidal category, **MealyAut***, with sequential composition and product as defined above.*

This construction, together with the results of Section 5.2, leads to the following result.

Proposition 5.10. *The category of Mealy deterministic finite automata is the free structured feedback category over the isomorphisms of automaton state spaces* $\mathsf{U}_{\mathsf{Aut}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{FinSet}$*, that is,*

$$\mathbf{MealyAut} \cong \mathsf{St}(\mathbf{FinSet}, \mathsf{U}_{\mathsf{Aut}}).$$

5.4 Automata in Span(Graph)

Our final construction is the canonical category of automata over the algebra of predicates given by spans. These automata are analogous to the previous transition systems in $\mathbf{Span}(\mathbf{Graph})_*$, but their state space contains an initial state and a set of final states. A similar definition has appeared previously in the literature for the modeling of Petri nets [45].

Definition 5.11. *A span automaton with left labels* A *and right labels* B

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X)$$

is given by a finite set of states S_X*, a finite set of transitions* E_X *with source and target* $s_X, t_X \colon E_X \to S_X$ *and with left* $l_X \colon E_X \to A$ *and right* $r_X \colon E_X \to B$

labels, an initial state $i_X \in S_X$, and a subset of final states $f_X \colon S_X \to 2$. The product of two span automata

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X) \textit{ and}$$
$$\mathbf{Y} = (E_Y, S_Y, A', B', s_Y, t_Y, l_X, r_X, i_Y, f_Y),$$

is given by the component-wise product

$$\mathbf{X} \otimes \mathbf{Y} \coloneqq (E_X \times E_Y, S_X \times S_Y, A \times A', B \times B', \\ s_X \times s_Y, t_X \times t_Y, l_X \times l_Y, r_X \times r_Y, i_X \times i_Y, f_X \wedge f_Y).$$

The composition of two span automata

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X) \textit{ and}$$
$$\mathbf{Y} = (E_Y, S_Y, B, C, s_Y, t_Y, l_X, r_X, i_Y, f_Y),$$

is given by a pullback

$$\mathbf{X} \mathbin{⨾} \mathbf{Y} \coloneqq (E_X \times_B E_Y, S_X \times S_Y, A, C, s_X \times s_Y, t_X \times t_Y, l_X, r_Y, i_X \times i_Y, f_X \wedge f_Y),$$

where $E_X \times_B E_Y$ is the pullback of $E_X \overset{r_X}{\to} B \overset{l_Y}{\leftarrow} E_Y$. We consider span automata up to isomorphism of their state space. Span automata form a symmetric monoidal category **SpanAut** *with sequential composition and the monoidal product defined above.*

This construction, together with the results of Section 5.2, leads to the following result.

Proposition 5.12. *The category of span automata is the free structured feedback category over the category of spans with the inclusion functor from automaton state spaces, $\mathsf{U}_{\mathsf{AutSpan}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{Span}(\mathbf{FinSet})$, that is,*

$$\mathbf{SpanAut} \cong \mathsf{St}(\mathbf{Span}(\mathbf{FinSet}), \mathsf{U}_{\mathsf{AutSpan}}).$$

Example 5.13. By the universal property of the $\mathsf{St}(\bullet)$ construction, each Mealy automaton in **MealyAut** functorially induces a span automaton in **SpanAut** whose graph is the graph of the Mealy automaton.

Consider the following Mealy automaton $\mathbf{X}$ in Figure 33, left. It has a state space $S_X = \{0, 1, 2\}$, left labels $A = \{a, b\}$ and trivial right labels $B = \mathbf{1}$, with initial state $i_X = 0$ and a unique final state $f_X(2) = \textbf{true}$, while $f_X(0) = f_X(1) = \textbf{false}$. Its transition function is given by $t_X(0, a) = 1$, $t_X(0, b) = 2$, $t_X(1, a) = 2$, $t_X(1, b) = 1$, $t_X(2, a) = 1$ and $t_X(2, b) = 1$.

The corresponding span automaton $\mathbf{sX}$ (Figure 33, right) is not only the transition system, but also the markings for initial and final state. Explicitly, its set of edges – or transitions – is given by tuples

$$E_{sX} = \{(0, a, 1), (0, b, 0), (1, a, 1), (1, b, 2), (2, a, 1), (2, b, 0)\},$$

its state space is again $S_{sX} = \{0, 1, 2\}$, its left and right boundaries are again $A = \{a, b\}$ and $B = \mathbf{1}$, with initial state $i_X = 0$ and a unique final state $f_X(2) = \textbf{true}$, while $f_X(0) = f_X(1) = \textbf{false}$.

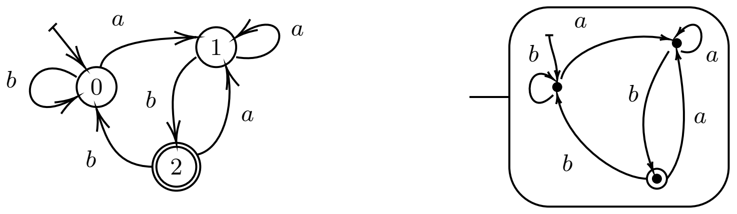


Fig. 33: Mealy automaton and associated span automaton.

6 Conclusions and Further Work

6.1 Discussion

We started this manuscript pointing out the fragmented landscape of models for concurrent software. We have now formally proven that any theory of predicates with a notion of feedback that accepts reasonable axioms (A1-A5) must already contain a simulation of **Span**(**Graph**)$_*$: the algebra of open transition systems of Katis, Sabadini and Walters.

From an applied point of view, our results inform a minimal software architecture for a library that constructs and analyzes these modular transition systems, that are canonical models of concurrency. For instance, an object-oriented programmer may

- define a specific class for "theory of resources", providing methods for the fundamental operations of composing, tensoring, identities and swapping, and providing subclasses for "resources" and "processes", if necessary;
- implement an abstract method that computes the $\mathtt{St}(\bullet)$ construction: this method will take a theory of processes and instantiate the free feedback category as a new theory of processes;
- work uniformly with multiple theories of processes and the theories of automata they generate, providing auxiliary methods (for reachability, connectedness, joining, . . .) that work across different theories of transition systems;
- thus, thanks to our results, obtain at the end of the day a graph – in **Span**(**Graph**) – that can be understood as a transition system and analyzed with the same library.

A functional programmer may want to follow the architecture of the public Haskell implementation of the ideas of this paper [46], where our running example (Figure 1) is showcased.

The problem we solve is one of abstraction. When faced with the task of implementing models for transition systems, we could be tempted to simply implement each one of them separately. Our result shows that a much more succinct choice is possible. Reducing the lines of code is important: transition systems are usually formally analyzed to prove correctness; the less lines there are to analize in the core implementation, the easiest it will be to be sure of its correctness.

Open transition systems allow us to construct transition systems from a few building blocks; compositionality eases both implementation and verification. Specifically, this could open an avenue to study the problem of compositional verification, where the automatic analysis of a global system must be modularly reduced to the analysis of its components.

6.2 Conclusion

We have characterized $\mathbf{Span}(\mathbf{Graph})_*$, an algebra of open transition systems, as the free feedback category over the category of spans of functions. To do so, we have used the $\mathsf{St}(\bullet)$ construction, characterized as the free feedback category in [35]. We have given this characterization more generally, for any category $\mathbf{C}$ with finite limits: the category $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$ of spans of graphs in $\mathbf{C}$ is the free feedback category over the category of spans in $\mathbf{C}$. Finally, we have defined a generalization of feedback categories to capture automata with initial and final states.

Further work will look at timed [11] and probabilistic [13,14] versions of the Cospan/Span model to connect it with recent work on modeling probabilistic programs with feedback categories [16]. We also plan to investigate the relationship between generalized feedback categories (Section 5) to approaches based on guarded recursion [25] and coalgebras [12,42].

References

1. Samson Abramsky. What are the fundamental structures of concurrency? We still don't know! *CoRR*, abs/1401.4973, 2014. URL: http://arxiv.org/abs/1401.4973, arXiv:1401.4973.
2. Jirí Adámek, Stefan Milius, and Jiri Velebil. Elgot algebras. *Log. Methods Comput. Sci.*, 2(5), 2006. doi:10.2168/LMCS-2(5:4)2006.
3. John C. Baez and Kenny Courser. Structured cospans. *CoRR*, abs/1911.04630, 2019.
4. Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
5. Nick Benton and Martin Hyland. Traced premonoidal categories. *RAIRO Theor. Informatics Appl.*, 37(4):273–299, 2003. doi:10.1051/ita:2003020.
6. Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes.* EATCS Monographs on Theoretical Computer Science. Springer, 1993. doi:10.1007/978-3-642-78034-9.
7. Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. doi:10.1145/3290338.
8. Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear Relations on Streams. *Information and Computation*, 252:2–29, 2017. doi:10.1016/j.ic.2016.03.002.
9. Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for P/T nets interactions. In *Concurrency Theory (CONCUR '11)*, volume 6901 of *LNCS*, pages 312–326. Springer, 2011. doi:10.1007/978-3-642-23217-6_21.

10. Aurelio Carboni and Robert F. C. Walters. Cartesian Bicategories I. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987.
11. Alessandra Cherubini, Nicoletta Sabadini, and Robert F. C. Walters. Timing in the Cospan/Span model. *Electronic Notes in Theoretical Computer Science*, 104:81–97, 2004.
12. Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and reasoning with guarded recursion for coinductive types. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2015. doi:10.1007/978-3-662-46678-0_26.
13. Luisa de Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of Markov processes. *Applied Categorical Structures*, 19(1):425–437, 2011.
14. Luisa de Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of Markov processes II. *RAIRO-Theoretical Informatics and applications*, 45(1):117–142, 2011.
15. Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009. doi:10.1007/978-3-642-14458-5_1.
16. Elena Di Lavore, Giovanni de Felice, and Mario Román. Monoidal streams for dataflow programming. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533365.
17. William Henry Eccles and Frank Wilfred Jordan. Improvements in ionic relays. *British patent number: GB 148582*, 1918.
18. Calvin C. Elgot. Monadic computation and iterative algebraic theories. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 175–230. Elsevier, 1975.
19. Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.
20. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, Filippo Schiavio, and Simone Tini. CospanSpan(Graph): a compositional description of the heart system. *Fundam. Informaticae*, 171(1-4):221–237, 2020.
21. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, and Simone Tini. Compositional modeling of biological systems in CospanSpan(Graph). In *Proc. of ICTCS 2020.* CEUR-WS, To appear.
22. Alessandro Gianola, Stefano Kasangian, and Nicoletta Sabadini. Cospan/Span(Graph): an Algebra for Open, Reconfigurable Automata Networks. In Filippo Bonchi and Barbara König, editors, *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, volume 72 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CALCO.2017.2.
23. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
24. Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.

25. Sergey Goncharov and Lutz Schröder. Guarded traced categories. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2018. doi:10.1007/978-3-319-89366-2_17.
26. Masahito Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 1997. doi:10.1007/3-540-62688-3_37.
27. Masahito Hasegawa. The uniformity principle on traced monoidal categories. In Richard Blute and Peter Selinger, editors, *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002*, volume 69 of *Electronic Notes in Theoretical Computer Science*, pages 137–155. Elsevier, 2002. doi:10.1016/S1571-0661(04)80563-2.
28. Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014. doi:10.1145/2603088.2603124.
29. André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447 – 468, 04 1996. doi:10.1017/S0305004100074338.
30. Rudolf Emil Kalman, Peter L. Falb, and Michael A. Arbib. *Topics in mathematical system theory*, volume 1. McGraw-Hill New York, 1969.
31. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.
32. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Span(Graph): A Categorial Algebra of Transition Systems. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings*, volume 1349 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 1997. doi:10.1007/BFb0000479.
33. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. On the algebra of feedback and systems with boundary. In *Rendiconti del Seminario Matematico di Palermo*, 1999.
34. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A formalization of the IWIM model. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models, 4th International Conference, COORDINATION 2000, Limassol, Cyprus, September 11-13, 2000, Proceedings*, volume 1906 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2000. doi:10.1007/3-540-45263-X_17.
35. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *RAIRO-Theor. Informatics Appl.*, 36(2):181–194, 2002. doi:10.1051/ita:2002009.
36. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A Process Algebra for the Span(Graph) Model of Concurrency. *arXiv preprint arXiv:0904.3964*, 2009.

37. Stephen Lack. Composing PROPs. *Theory and Applications of Categories*, 13(9):147–163, 2004.
38. Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. A Canonical Algebra of Open Transition Systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2021. doi:10.1007/978-3-030-90636-8_4.
39. Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.
40. S. J. Mason. Feedback Theory - Some properties of signal flow graphs. *Proceedings of the Institute of Radio Engineers*, 41(9):1144–1156, 1953. doi:10.1109/JRPROC.1953.274449.
41. George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.
42. Stefan Milius and Tadeusz Litak. Guard your daggers and traces: Properties of guarded (co-) recursion. *Fundamenta Informaticae*, 150(3-4):407–449, 2017.
43. Duško Pavlović. Maps I: relative to a factorisation system. *Journal of Pure and Applied Algebra*, 99(1):9–34, 1995.
44. Kate Ponto and Michael Shulman. Traces in symmetric monoidal categories. *Expositiones Mathematicae*, 32(3):248–273, 2014. URL: http://dx.doi.org/10.1016/J.EXMATH.2013.12.003, doi:10.1016/j.exmath.2013.12.003.
45. Julian Rathke, Pawel Sobocinski, and Owen Stephens. Compositional Reachability in Petri Nets. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2014. doi:10.1007/978-3-319-11439-2_18.
46. Mario Román. Span graph via the state construction, 2022. URL: https://github.com/mroman42/feedback-span-graph.
47. Robert Rosebrugh, Nicoletta Sabadini, and Robert F. C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15(6):164–177, 2005.
48. Nicoletta Sabadini, Filippo Schiavio, and Robert F. C. Walters. On the geometry and algebra of networks with state. *Theor. Comput. Sci.*, 664:144–163, 2017.
49. Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. doi:10.1007/978-3-642-12821-9_4.
50. Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines*. Bell Telephone Laboratories, 1942.
51. Paweł Sobociński. A non-interleaving process calculus for multi-party synchronisation. In *2nd Interaction and Concurrency Experience: Structured Interactions, (ICE 2009)*, volume 12 of *EPTCS*, 2009. URL: http://users.ecs.soton.ac.uk/ps/papers/ice09.pdf, doi:10.4204/eptcs.12.6.
52. Paweł Sobociński. Representations of Petri net interactions. In *Concurrency Theory, 21th International Conference, (CONCUR 2010)*, volume 6269 of *Lecture Notes in Computer Science*, pages 554–568. Springer, 2010. doi:10.1007/978-3-642-15375-4_38.