Monoidal Streams for Dataflow Programming

Elena Di Lavore
Giovanni de Felice
Mario Román

Abstract

We introduce monoidal streams: a generalization of causal stream functions to monoidal categories. In the same way that streams provide semantics to dataflow programming with pure functions, monoidal streams provide semantics to dataflow programming with theories of processes represented by a symmetric monoidal category. At the same time, monoidal streams form a feedback monoidal category, which can be used to interpret signal flow graphs. As an example, we study a stochastic dataflow language.

Keywords: Monoidal stream, Stream, Monoidal category, Dataflow programming, Feedback, Signal flow graph, Coalgebra, Stochastic process.

1 Introduction

Dataflow languages. Dataflow (or *stream-based*) programming languages, such as Lucid [35, 77], follow a paradigm in which every declaration represents an infinite list of values: a *stream* [8, 75]. The following program in a Lucid-like language (Figure 1) computes the Fibonacci sequence, thanks to a Fby ("followed by") operator.

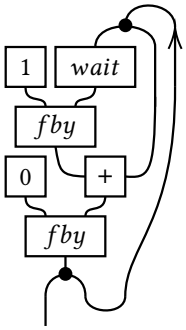
$$fib = 0 \text{ Fby } (fib + (1 \text{ Fby } \text{Wait}(fib)))$$

Figure 1. *The Fibonacci sequence is 0 followed by the Fibonacci sequence plus the Fibonacci sequence preceded by a 1.*

The control structure of dataflow programs is inspired by *signal flow graphs* [8, 57, 69]. Signal flow graphs are diagrammatic specifications of processes with feedback loops, widely used in control system engineering. In a dataflow program, feedback loops represent how the current value of a stream may depend on its previous values. For instance, the previous program (Figure 1) corresponds to the signal flow graph in Figure 2.

Monoidal categories. Any theory of processes that *compose sequentially and in parallel*, satisfying reasonable axioms, forms a *monoidal category*. Examples include functions [49], probabilistic channels [19, 30], partial maps [21], database queries [14], linear resource theories [23] and quantum processes [2]. Signal flow graphs are the graphical syntax for *feedback monoidal categories* [15, 16, 26, 33, 46]: they are the *string diagrams* for any of these theories, extended with *feedback*.

Yet, semantics of dataflow languages have been mostly restricted to theories of pure functions [8, 24, 25, 58, 76]:



fib = **fbk**(copy;
$\partial(1 \times \text{wait}) \times \text{id}$;
$\partial(\text{fby}) \times \text{id}$;
$\partial(+)$;
$0 \times \text{id}$;
fby;
copy)

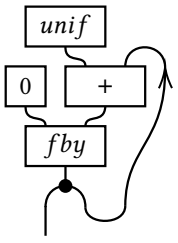
Figure 2. Fibonacci: signal flow graph and morphism.

what are called *cartesian* monoidal categories. We claim that this restriction is actually inessential; dataflow programs may take semantics in non-cartesian monoidal categories, exactly as their signal flow graphs do.

The present work provides this missing semantics: we construct *monoidal streams* over a symmetric monoidal category, which form a *feedback monoidal category*. Monoidal streams model the values of a *monoidal dataflow language*, in the same way that streams model the values of a classical dataflow language. This opens the door to stochastic, effectful, or quantum dataflow languages. In particular, we give semantics and string diagrams for a *stochastic dataflow programming language*, where the following code can be run.

$$walk = 0 \text{ Fby } (\text{Uniform}(-1, 1) + walk)$$

Figure 3. *A stochastic dataflow program. A random walk is 0 followed by the random walk plus a stochastic stream of steps to the left (-1) or to the right (1), sampled uniformly.*



walk = **fbk**(
$\partial(\text{unif}) \otimes \text{id}$;
$0 \otimes \partial(+)$;
fby;
copy)

Figure 4. Random walk: signal flow graph and morphism.

1.1 Contributions

Our main novel contribution is the definition of a feedback monoidal category of *monoidal streams* over a symmetric monoidal category (Stream, Definition 5.1 and theorem 5.10).

Monoidal streams form a final coalgebra; for sufficiently well-behaved monoidal categories (Definition 4.9), we give an explicit construction of this coalgebra (Definition 4.7).

In cartesian categories, the *causal functions* of Uustalu and Vene [75] (see also [72]) are a particular case of our monoidal streams (Theorems 6.1 and 6.3). In the category of stochastic functions, our construction captures the notion of *controlled stochastic process* [27, 66] (Theorem 7.2).

In order to arrive to this definition, we unify the previous literature: we characterize the cartesian "intensional stateful sequences" of Katsumata and Sprunger with a final coalgebra (Theorem 2.5), and then "extensional stateful sequences" in terms of the "feedback monoidal categories" of Katis, Sabadini and Walters [46] (Theorem 3.8). We justify observational equivalence with a refined fixpoint equation that employs *coends* (Theorem 4.11). We strictly generalize "stateful sequences" from the cartesian to the monoidal case.

Finally, we extend a type theory of symmetric monoidal categories with a feedback operator (Section 8) and we use it as a stochastic dataflow programming language.

1.2 Related work

Coalgebraic streams. Uustalu and Vene [75] provide elegant *comonadic* semantics for a (cartesian) Lucid-like programming language. We shall prove that their exact technique cannot be possibly extended to arbitrary monoidal categories (Theorem 6.1). However, we recover their semantics as a particular case of our monoidal streams (Theorem 6.3).

Stateful morphism sequences. Sprunger and Katsumata constructed the category of *stateful sequences* in the cartesian case [73]. Our work is based on an unpublished work by Román [65] that first exteneded this definition to the symmetric monoidal case, using *coends* to justify *extensional equality*. Shortly after, Carette, de Visme and Perdrix [18] rederived this construction and applied it to the case of completely positive maps between Hilbert spaces, using (a priori) a slightly different notion of equality. We synthetise some of this previous work, we justify it for the first time using coalgebra and we particularize it to some cases of interest.

Feedback. Feedback monoidal categories are a weakening of *traced monoidal categories*. The construction of the free such categories is originally due to Katis, Sabadini and Walters [46]. Feedback monoidal categories and their free construction have been repurposed and rediscovered multiple times in the literature [13, 32, 38, 44]. Di Lavore et al. [26] summarize these uses and introduce *delayed feedback*.

General and dependent streams. Our work concerns *synchronous* streams: those where, at each point in time $t = 0, 1, \dots$, the stream process takes exactly one input and produces exactly one output. This condition is important in certain contexts like, for instance, real-time embedded systems; but it is not always present. The study of asynchronous stream transformers and their universal properties is considerably different [1], and we refer the reader to the recent work of Garner [31] for a discussion on *non-synchronous* streams. Finally, when we are concerned with *dependent streams* indexed by time steps, a possible approach, when our base category is a topos, is to use the *topos of trees* [9].

Categorical dataflow programming. Category theory is a common tool of choice for dataflow programming [32, 56, 67]. In particular, profunctors and coends are used by Hildebrandt, Panangaden and Winskel [37] to generalise a model of non-deterministic dataflow, which has been the main focus [51, 54, 60] outside cartesian categories.

1.3 Synopsis

This manuscript contains three main definitions in terms of universal properties (*intensional, extensional and observational streams*, Definitions 2.2, 3.7 and 4.1); and three explicit constructions for them (*intensional, extensional and observational sequences*, Definitions 2.4, 2.8 and 4.7). Each definition is refined into the next one: each construction is a quotienting of the previous one.

Sections 1.4 and 2.3 contain expository material on coalgebra and dinaturality. Section 2 presents intensional monoidal streams. Section 3 introduces extensional monoidal streams in terms of feedback monoidal categories. Section 4 introduces the definitive *observational* equivalence and defines *monoidal streams*. Section 5 constructs the feedback monoidal category of monoidal streams. Sections 6 and 7 present two examples: cartesian and stochastic streams. Section 8 introduces a type theory for feedback monoidal categories.

1.4 Prelude: Coalgebra

In this preparatory section, we introduce some background material on coalgebra [3, 41, 67]. Coalgebra is the category-theoretic study of stateful systems and infinite data-structures, such as streams. These structures arise as *final coalgebras*: universal solutions to certain functor equations.

Let us fix an endofunctor $F\colon \mathsf{C} \to \mathsf{C}$ through the section.

Definition 1.1. A *coalgebra* (Y, β) is an object $Y \in \mathsf{C}$, together with a morphism $\beta\colon Y \to FY$. A *coalgebra morphism* $g\colon (Y,\beta) \to (Y',\beta')$ is a morphism $g\colon Y \to Y'$ such that $g;\beta' = \beta;Fg$.

Coalgebras for an endofunctor form a category with coalgebra morphisms between them. A *final coalgebra* is a final object in this category. As such, final coalgebras are unique up to isomorphism when they exist.

Definition 1.2. A *final coalgebra* is a coalgebra (Z,γ) such that for any other coalgebra (Y,β) there exists a unique coalgebra morphism $g\colon (Y,\beta) \to (Z,\gamma)$.

Our interest in final coalgebras derives from the fact that they are canonical fixpoints of an endofunctor. Specifically,

Lambek's theorem (Theorem 1.4) states that whenever the final coalgebra exists, it is a fixpoint.

Definition 1.3. A *fixpoint* is a coalgebra (Y, β) such that $\beta\colon Y \to FY$ is an isomorphism. A *fixpoint morphism* is a coalgebra morphism between fixpoints: fixpoints and fixpoint morphisms form a full subcategory of the category of coalgebras. A *final fixpoint* is a final object in this category.

Theorem 1.4 (Lambek, [50])**.** *Final coalgebras are fixpoints. As a consequence, when they exist, they are final fixpoints.*

The last question before continuing is how to explicitly construct a final coalgebra. This is answered by Adamek's theorem (Theorem 1.5). The reader may be familiar with Kleene's theorem for constructing fixpoints [74]: the least fixpoint of a monotone function $f\colon X \to X$ in a directed-complete partial order $(X, \leqslant)$ is the supremum of the chain $\bot \leqslant f(\bot) \leqslant f(f(\bot)) \leqslant \dots$, where $\bot$ is the least element of the partial order, whenever this supremum is preserved by f. This same result can be categorified into a fixpoint theorem for constructing final coalgebras: the directed-complete poset becomes a category with ω-chain limits; the monotone function becomes an endofunctor; and the least element becomes the final object.

Theorem 1.5 (Adamek, [4])**.** *Let* D *be a category with a final object* 1 *and* ω*-shaped limits. Let* $F\colon \mathsf{D} \to \mathsf{D}$ *be an endofunctor. We write* $L = \lim_n F^n 1$ *for the limit of the following* ω*-chain, which is called the* terminal sequence *of* F*.*

$$1 \xleftarrow{!} F1 \xleftarrow{F!} FF1 \xleftarrow{FF!} FFF1 \xleftarrow{FFF!} \dots$$

Assume that F *preserves this limit, meaning that the canonical morphism* $FL \to L$ *is an isomorphism. Then,* L *is the final* F*-coalgebra.*

2 Intensional Monoidal Streams

This section introduces a preliminary definition of *monoidal stream* in terms of a fixpoint equation (in Figure 5). In later sections, we refine both this definition and its characterization into the definitive notion of *monoidal stream.*

Let $(\mathsf{C}, \otimes, I)$ be a fixed symmetric monoidal category.

2.1 The fixpoint equation

Classically, type-variant streams have a neat coinductive definition [41, 67] that says:

> *"A stream of type* $\mathbb{A} = (A_0, A_1, \dots)$ *is an element of* A_0 *together with a stream of type* $\mathbb{A}^+ = (A_1, A_2, \dots)$*".*

Formally, streams are the final fixpoint of the equation

$$\mathbf{S}(A_0, A_1, \dots) \cong A_0 \times \mathbf{S}(A_1, A_2, \dots);$$

and this fixpoint is computed to be $\mathbf{S}(\mathbb{A}) = \prod_{n\in\mathbb{N}}^{\infty} A_n$.

In the same vein, we want to introduce not only streams but *stream processes* over a fixed theory of processes.

> *"A stream process from* $\mathbb{X} = (X_0, X_1, \dots)$ *to* $\mathbb{Y} = (Y_0, Y_1, \dots)$ *is a process from* X_0 *to* Y_0 *communicating along a channel* M *with a stream process from* $\mathbb{X}^+ = (X_1, X_2, \dots)$ *to* $\mathbb{Y}^+ = (Y_1, Y_2, \dots)$*."*

Streams are recovered as stream processes on an empty input, so we take this more general slogan as our preliminary definition of *monoidal stream* (in Definition 2.2). Formally, they are the final fixpoint of the equation in Figure 5.

$$\mathbf{T}(\mathbb{X}, \mathbb{Y}) \cong \sum_{M\in\mathsf{C}} \hom(X_0, M \otimes Y_0) \times \mathbf{T}(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Figure 5. Fixpoint equation for intensional streams.

Remark 2.1 (Notation). Let $\mathbb{X} \in [\mathbb{N}, \mathsf{C}]$ be a sequence of objects $(X_0, X_1, \dots)$. We write $\mathbb{X}^+$ for its *tail* $(X_1, X_2, \dots)$. Given $M \in \mathsf{C}$, we write $M \cdot \mathbb{X}$ for the sequence $(M \otimes X_0, X_1, X_2, \dots)$; As a consequence, we write $M \cdot \mathbb{X}^+$ for $(M \otimes X_1, X_2, X_3, \dots)$.

Definition 2.2. The set of *intensional monoidal streams* $\mathbf{T}\colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathrm{Set}$, depending on inputs and outputs, is the final fixpoint of the equation in Figure 5.

Remark 2.3 (Initial fixpoint). There exists an obvious fixpoint for the equation in Figure 5: the constant empty set. This solution is the *initial* fixpoint, a minimal solution. The *final* fixpoint will be realized by the set of *intensional sequences.*

2.2 Intensional sequences

We now construct the set of intensional streams explicitly (Theorem 2.5). For this purpose, we generalize the "stateful morphism sequences" of Katsumata and Sprunger [73] from cartesian to arbitrary symmetric monoidal categories (Definition 2.4). We derive a novel characterization of these "sequences" as the desired final fixpoint (Theorem 2.5).

In the work of Katsumata and Sprunger, a stateful sequence is a sequence of morphisms $f_n\colon M_{n-1} \times X_n \to M_n \times Y_n$ in a cartesian monoidal category. These morphisms represent a process at each point in time $n = 0, 1, 2, \dots$. At each step n, the process takes an input X_n and, together with the stored memory M_{n-1}, produces some output Y_n and writes to a new memory M_n. The memory is initially empty, with $M_{-1} = \mathbf{1}$ being the final object by convention. We extend this definition to any symmetric monoidal category.

Definition 2.4. Let $\mathbb{X}$ and $\mathbb{Y}$ be two sequences of objects representing inputs and outputs, respectively. An *intensional sequence* is a sequence of objects $(M_0, M_1, \dots)$ together with a sequence of morphisms

$$(f_n\colon M_{n-1} \otimes X_n \to M_n \otimes Y_n)_{n\in\mathbb{N}},$$

where, by convention, $M_{-1} = I$ is the unit of the monoidal category. In other words, the set of intensional sequences is

$$\mathbf{Int}(\mathbb{X}, \mathbb{Y}) := \sum_{M\in[\mathbb{N},\mathsf{C}]} \prod_{n=0}^{\infty} \hom(M_{n-1} \otimes X_n, M_n \otimes Y_n).$$

We now prove that intensional sequences are the final fixpoint of the equation in Figure 5. The following Theorem 2.5 serves two purposes: it gives an explicit final solution to this fixpoint equation and it gives a novel universal property to intensional sequences.

Theorem 2.5. *Intensional sequences are the explicit construction of intensional streams,* $\mathbf{T} \cong \mathbf{Int}$. *In other words, they are a fixpoint of the equation in Figure 5, and they are the final such one.*

Proof sketch. It is known that categories of functors over sets, such as $[[\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}], \mathrm{Set}]$, have all limits. Adamek's theorem (Theorem 1.5) states that, if the following limit is a fixpoint, it is indeed the final one.

$$\lim_{n\in\mathbb{N}} \sum_{M_0,\dots,M_n} \prod_{t=0}^{n} \hom(M_{t-1} \otimes X_t, M_t \otimes Y_t) \tag{1}$$

Connected limits commute with coproducts and the limit of the nth-product is the infinite product. Thus, Equation (1) is isomorphic to $\mathbf{Int}(\mathbb{X}, \mathbb{Y})$. It only remains to show that $\mathbf{Int}(\mathbb{X}, \mathbb{Y})$ is a fixpoint, which means it should be isomorphic to the following expression.

$$\sum_{M_0} \hom(X_0, M_0 \otimes Y_0) \times \sum_{M\in[\mathbb{N},\mathrm{C}]} \prod_{n=1}^{\infty} \hom(M_{n-1} \otimes X_n, M_n \otimes Y_n). \tag{2}$$

Cartesian products distribute over coproducts, so Equation (2) is again isomorphic to $\mathbf{Int}(\mathbb{X}, \mathbb{Y})$. □

2.3 Interlude: Dinaturality

During the rest of this text, we deal with two different definitions of what it means for two processes to be equal: *extensional* and *observational equivalence*, apart from pure *intensional equality*. Fortunately, when working with functors of the form $P\colon \mathrm{C}^{op} \times \mathrm{C} \to \mathrm{Set}$, the so-called *endoprofunctors*, we already have a canonical notion of equivalence.

Endoprofunctors $P\colon \mathrm{C}^{op} \times \mathrm{C} \to \mathrm{Set}$ can be thought as indexing families of processes $P(M, N)$ by the types of an input channel M and an output channel N. A process $p \in P(M, N)$ writes to a channel of type N and then reads from a channel of type M.

Now, assume we also have a transformation $r\colon N \to M$ translating from output to input types. Then, we can *plug the output to the input*: the process p writes with type N, then r translates from N to M, and then p uses this same output as its input M. This composite process can be given two sligthly different descriptions; the process could

- translate *after writing*, $P(M, r)(p) \in P(M, M)$, or
- translate *before reading*, $P(r, N)(p) \in P(N, N)$.

These two processes have different types. However, with the output plugged to the input, it does not really matter when to apply the translation. These two descriptions represent the same process: they are *dinaturally equivalent*.

Definition 2.6 (Dinatural equivalence)**.** For any functor $P\colon \mathrm{C}^{op} \times \mathrm{C} \to \mathrm{Set}$, consider the set

$$S_P = \sum_{M\in\mathrm{C}} P(M, M).$$

Dinatural equivalence, $(\sim)$, on the set S_P is the smallest equivalence relation satisfying $P(M, r)(p) \sim P(r, N)(p)$ for each $p \in P(M, N)$ and each $r \in \hom(N, M)$.

Coproducts quotiented by dinatural equivalence construct a particular form of colimit called a *coend*. Under the process interpretation of profunctors, taking a coend means *plugging an output to an input* of the same type.

Definition 2.7 (Coend, [53, 55])**.** Let $P\colon \mathrm{C}^{op} \times \mathrm{C} \to \mathrm{Set}$ be a functor. Its *coend* is the coproduct of $P(M, M)$ indexed by $M \in \mathrm{C}$, quotiented by dinatural equivalence.

$$\int^{M\in\mathrm{C}} P(M, M) = \left(\sum_{M\in\mathrm{C}} P(M, M) \Big/ \sim \right).$$

That is, the coend is the colimit of the diagram with a *cospan* $P(M, M) \leftarrow P(M, N) \to P(N, N)$ for each $f\colon N \to M$.

2.4 Towards extensional memory channels

Let us go back to intensional monoidal streams. Consider a family of processes $f_n\colon M_{n-1} \otimes X_n \to Y_n \otimes N_n$ reading from memories of type M_n but writing to memories of type N_n. Assume we also have processes $r_n\colon N_n \to M_n$ translating from output to input memory. Then, we can consider the process that does f_n, translates from memory N_n to memory M_n and then does f_{n+1}. This process is described by two different intensional sequences,

- $(f_n; (r_n \otimes \mathrm{id})\colon M_{n-1} \otimes X_n \to M_n \otimes Y_n)_{n\in\mathbb{N}}$, and
- $((r_{n-1} \otimes \mathrm{id}); f_n\colon N_{n-1} \otimes X_n \to N_n \otimes Y_n)_{n\in\mathbb{N}}$.

These two intensional sequences have different types for the memory channels. However, in some sense, they represent *the same process description*. If we do not care about what exactly it is that we save to memory, we should consider two such processes to be equal (as in Figure 6, where "the same process" can keep two different values in memory). Indeed, dinaturality in the memory channels M_n is the smallest equivalence relation $(\sim)$ satisfying

$$(f_n; (r_n \otimes \mathrm{id}))_{n\in\mathbb{N}} \sim ((r_{n-1} \otimes \mathrm{id}); f_n)_{n\in\mathbb{N}}.$$

This is precisely the quotienting that we perform in order to define *extensional sequences*.

Definition 2.8. *Extensional equivalence* of intensional sequences, $(\sim)$, is dinatural equivalence in the memory channels M_n. An *extensional sequence* from $\mathbb{X}$ to $\mathbb{Y}$ is an equivalence class

$$\langle f_n\colon M_{n-1} \otimes X \to M_n \otimes Y \rangle_{n\in\mathbb{N}}$$

of intensional sequences under extensional equivalence.

In other words, the set of extensional sequences is the set of intensional sequences substituting the coproduct by a coend,

$$\mathbf{Ext}(\mathbb{X},\mathbb{Y}) = \int^{M\in[\mathbb{N},\mathsf{C}]} \prod_{i=0}^{\infty} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

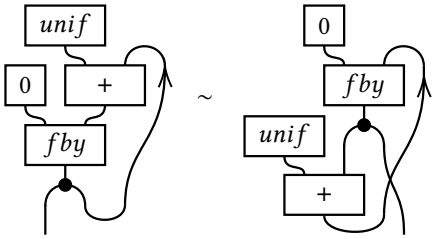


Figure 6. Extensionally equivalent walks keeping different memories: the *current* position vs. the *next* position.

3 Extensional Monoidal Streams

In this section, we introduce *extensional monoidal streams* in terms of a universal property: *extensional streams are the morphisms of the free delayed-feedback monoidal category* (Theorem 3.8).

Feedback monoidal categories come from the work of Katis, Sabadini and Walters [46]. They are instrumental to our goal of describing and composing signal flow graphs: they axiomatize a graphical calculus that extends the well-known string diagrams for monoidal categories with *feedback loops* [26, 46]. Constructing the *free feedback monoidal category* (Definition 3.4) will lead to the main result of this section: extensional sequences are the explicit construction of extensional streams (Theorem 3.8).

We finish the section by exploring how extensional equivalence may not be enough to capture true observational equality of processes (Example 3.10).

3.1 Feedback monoidal categories

Feedback monoidal categories are symmetric monoidal categories with a "feedback" operation that connects outputs to inputs. They have a natural axiomatization (Definition 3.1) that has been rediscovered independently multiple times, with only slight variations [10, 13, 45, 46]. It is weaker than that of *traced monoidal categories* [26] while still satisfying a normalization property (Theorem 3.5). We present a novel definition that generalizes the previous ones by allowing the feedback operator to be guarded by a monoidal endofunctor.

Definition 3.1. A *feedback monoidal category* is a symmetric monoidal category $(\mathsf{C},\otimes,I)$ endowed with a monoidal endofunctor $\mathbf{F}\colon \mathsf{C}\to\mathsf{C}$ and an operation

$$\mathbf{fbk}_S\colon \hom(\mathbf{F}(S)\otimes X, S\otimes Y)\to\hom(X,Y)$$

for all S, X and Y objects of C; this operation needs to satisfy the following axioms.

(A1). Tightening: $u;\mathbf{fbk}_S(f);v = \mathbf{fbk}_S((\mathrm{id}_{FS}\otimes u);f;(\mathrm{id}_S\otimes v))$.
(A2). Vanishing: $\mathbf{fbk}_I(f) = f$.
(A3). Joining: $\mathbf{fbk}_T(\mathbf{fbk}_S(f)) = \mathbf{fbk}_{S\otimes T}(f)$
(A4). Strength: $\mathbf{fbk}_S(f)\otimes g = \mathbf{fbk}_S(f\otimes g)$.
(A5). Sliding: $\mathbf{fbk}_S((\mathbf{F}h\otimes \mathrm{id}_X)\,;f) = \mathbf{fbk}_T(f\,;(h\otimes \mathrm{id}_Y))$.

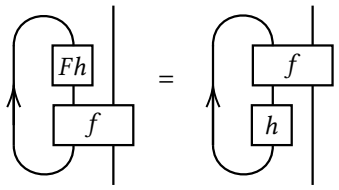


Figure 7. The sliding axiom (A5).

A *feedback functor* is a symmetric monoidal functor that preserves the feedback structure (Appendix, Definition A.9).

Remark 3.2 (Wait or trace). In a feedback monoidal category $(\mathsf{C},\mathbf{fbk})$, we construct the morphism $\mathrm{wait}_X\colon X\to FX$ as a feedback loop over the symmetry, $\mathrm{wait}_X = \mathbf{fbk}(\sigma_{X,X})$. A *traced monoidal category* [43] is a feedback monoidal category guarded by the identity functor such that $\mathrm{wait}_X = \mathrm{id}_X$.

The "state construction", $\mathrm{St}(\bullet)$, realizes the *free* feedback monoidal category. As it happens with feedback monoidal categories, this construction has appeared in the literature in slightly different forms. It has been used for describing a "memoryful geometry of interaction" [38], "stateful resource calculi" [13], and "processes with feedback" [44, 46].

The idea in all of these cases is the same: we allow the morphisms of a monoidal category to depend on a "state space" S, possibly guarded by a functor. Adding a state space is equivalent to freely adding feedback [26].

Definition 3.3. A *stateful morphism* is a pair (S,f) consisting of a "state space" $S\in\mathsf{C}$ and a morphism $f\colon \mathbf{F}S\otimes X\to S\otimes Y$. We say that two stateful morphisms are *sliding equivalent* if they are related by the smallest equivalence relation satisfying $(S,(\mathbf{F}r\otimes\mathrm{id})\,;h)\sim(T,h\,;(r\otimes\mathrm{id}))$ for each $h\colon X\otimes\mathbf{F}T\to S\otimes Y$ and each $r\colon S\to T$.

In other words, sliding equivalence is *dinaturality in* S.

Definition 3.4 ($\mathrm{St}(\bullet)$ construction, [26, 46])**.** We write $\mathrm{St}_{\mathbf{F}}(\mathsf{C})$ for the symmetric monoidal category that has the same objects as C and whose morphisms from X to Y are stateful morphisms $f\colon \mathbf{F}S\otimes X\to S\otimes Y$ *up to sliding*.

$$\hom_{\mathrm{St}_{\mathbf{F}}(\mathsf{C})}(X,Y) := \int^{S\in\mathsf{C}} \hom_{\mathsf{C}}(\mathbf{F}S\otimes X, S\otimes Y).$$

Theorem 3.5 (see [46])**.** $\mathrm{St}_{\mathbf{F}}(\mathsf{C})$ *is the free feedback monoidal category over* $(\mathsf{C},\mathbf{F})$.

3.2 Extensional monoidal streams

Monoidal streams should be, in some sense, the minimal way of adding feedback to a theory of processes. The output

of this feedback, however, should be delayed by one unit of time: the category $[\mathbb{N}, \mathbf{C}]$ is naturally equipped with a *delay* endofunctor that shifts a sequence by one. Extensional monoidal streams form the free delayed-feedback category.

Definition 3.6 (Delay functor)**.** Let $\partial\colon [\mathbb{N}, \mathbf{C}] \to [\mathbb{N}, \mathbf{C}]$ be the endofunctor defined on objects $\mathbb{X} = (X_0, X_1, \dots)$, as $\partial(\mathbb{X}) = (I, X_0, X_1, \dots)$; and on morphisms $\mathbb{f} = (f_0, f_1, \dots)$ as $\partial(\mathbb{f}) = (\mathrm{id}_I, f_0, f_1, \dots)$.

Definition 3.7. The set of *extensional monoidal streams*, depending on inputs and outputs, $\mathbf{R}\colon [\mathbb{N}, \mathbf{C}]^{op} \times [\mathbb{N}, \mathbf{C}] \to$ Set, is the hom-set of the free feedback monoidal category over $([\mathbb{N}, \mathbf{C}], \partial)$.

We characterize now extensional streams in terms of extensional sequences and the $\mathrm{St}(\bullet)$-construction.

Theorem 3.8. *Extensional sequences are the explicit construction of extensional streams,* $\mathbf{R} \cong \mathbf{Ext}$.

Proof. Note that $\mathbf{Ext}(\mathbb{X}, \mathbb{Y}) = \mathrm{St}_\partial([\mathbb{N}, \mathbf{C}])(\mathbb{X}, \mathbb{Y})$. That is, the extensional sequences we defined using dinaturality coincide with the morphisms of $\mathrm{St}_\partial([\mathbb{N}, \mathbf{C}])$, the free feedback monoidal category over $([\mathbb{N}, \mathbf{C}], \partial)$ in Definition 3.4. □

As a consequence, the calculus of signal flow graphs given by the syntax of feedback monoidal categories is sound and complete for extensional equivalence over $[\mathbb{N}, \mathbf{C}]$.

3.3 Towards observational processes

Extensional sequences were an improvement over intensional sequences because they allowed us to equate process descriptions that were *essentially the same*. However, we could still have two processes that are "observationally the same" without them being described in the same way.

Remark 3.9 (Followed by). As we saw in the Introduction, *"followed by"* is a crucial operation in dataflow programming. Any sequence can be decomposed as $\mathbb{X} \cong X_0 \cdot \partial(\mathbb{X}^+)$.[1] We call *"followed by"* to the coherence map in $[\mathbb{N}, \mathbf{C}]$ that witnesses this decomposition.

$$\mathrm{fby}_{\mathbb{X}}\colon X_0 \cdot \partial(\mathbb{X}^+) \to \mathbb{X}$$

In the case of constant sequences $\mathbb{X} = (X, X, \dots)$, we have that $\mathbb{X}^+ = \mathbb{X}$; which means that "followed by" has type $\mathrm{fby}_{\mathbb{X}}\colon X \cdot \partial\mathbb{X} \to \mathbb{X}$.

Example 3.10. Consider the extensional stateful sequence, in any cartesian monoidal category, that saves the first input to memory without ever producing an output. *Observationally*, this is no different from simply discarding the first input, $(!)_X\colon X \to 1$. However, in principle, we cannot show that these are *extensionally equal*, that is, $\mathbf{fbk}(\mathrm{fby}_{\mathbb{X}}) \neq (!)_X$.

More generally, discarding the result of any stochastic or deterministic signal flow graph is, *observationally*, the same as doing nothing (Figure 8, consequence of Theorem 7.2).

[1] This can also be seen as the isomorphism making "sequences" a final coalgebra. That is, the first slogan we saw in Section 2.1.

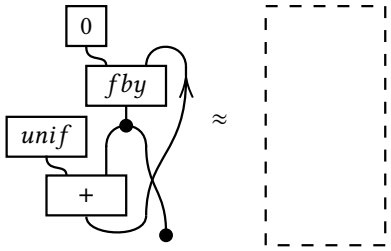


Figure 8. *Observationally*, a silent process does nothing.

4 Observational Monoidal Streams

In this section, we introduce our definitive *monoidal streams*: *observational streams* (Definition 4.1). Their explicit construction is given by observational sequences: extensional sequences quotiented by observational equivalence.

Intuitively, two processes are observationally equal if they are "equal up to stage n", for any $n \in \mathbb{N}$. We show that, in sufficiently well-behaved monoidal categories (which we call *productive*, Definition 4.9), the set of observational sequences given some inputs and outputs is the final coalgebra of a fixpoint equation (Figure 9). The name "observational equivalence" is commonly used to denote equality on the final coalgebra: Theorem 4.11 justifies our use of the term.

4.1 Observational streams

We saw in Section 2 that we can define intensional sequences as a solution to a fixpoint equation. We now consider the same equation, just substituting the coproduct for a coend.

Definition 4.1 (Observational streams)**.** The set of *observational monoidal streams*, depending on inputs and outputs, is the functor $\mathbf{Q}\colon [\mathbb{N}, \mathbf{C}]^{op} \times [\mathbb{N}, \mathbf{C}] \to$ Set given by the final fixpoint of the equation in Figure 9.

$$\mathbf{Q}(\mathbb{X}, \mathbb{Y}) \cong \int^{M \in \mathbf{C}} \hom(X_0, M \otimes Y_0) \times \mathbf{Q}(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Figure 9. Fixpoint equation for observational streams.

The explicit construction of this final fixpoint will be given by observational sequences (Theorem 4.11).

4.2 Observational sequences

We said that observational equivalence is "equality up to stage n", so our first step will be to define what it means to truncate an extensional sequence at any given $n \in \mathbb{N}$.

Definition 4.2 (n-Stage process)**.** An *n-stage process* from inputs $\mathbb{X} = (X_0, X_1, \dots)$ to outputs $\mathbb{Y} = (Y_0, Y_1, \dots)$ is an element of the set

$$\mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}) = \int^{M_0, \dots, M_n} \prod_{i=0}^{n} \hom(M_{i-1} \otimes X_i, M_i \otimes Y_i).$$

Remark 4.3. In other words, n-stage processes are n-tuples $(f_i\colon M_{i-1} \otimes X_i \to M_i \otimes Y_i)_{i=0}^n$, for some choice of M_i *up to*

dinaturality, that we write as

$$\langle f_0 | f_1 | \dots | f_n | \in \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}).$$

In this notation, *dinaturality* means that morphisms can *slide past the bars*. That is, for any $r_i : N_i \to M_i$ and any tuple, dinaturality says that

$$\begin{aligned}&\langle f_0; (r_0 \otimes \mathrm{id}) | f_1; (r_1 \otimes \mathrm{id}) | \dots | f_n; (r_n \otimes \mathrm{id}) |\\ &= \langle f_0 | (r_0 \otimes \mathrm{id}); f_1 | \dots | (r_{n-1} \otimes \mathrm{id}); f_n | .\end{aligned}$$

Note that the last r_n is removed by dinaturality.

Definition 4.4 (Truncation)**.** The *k-truncation* of an extensional sequence $\langle f_n : M_{n-1} \otimes X_n \to M_n \otimes Y_n \rangle \in \mathbf{Ext}(\mathbb{X}, \mathbb{Y})$ is $\langle f_0 | \dots | f_k | \in \mathbf{Stage}_k(\mathbb{X}, \mathbb{Y})$. Truncation is well-defined under dinatural equivalence (Remark 4.3).

For $k \leqslant n$, the *k-truncation* of an n-stage process given by $\langle f_0 | f_1 | \dots | f_n | \in \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y})$ is $\langle f_0 | \dots | f_k | \in \mathbf{Stage}_k(\mathbb{X}, \mathbb{Y})$. This induces functions $\pi_{n,k} : \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}) \to \mathbf{Stage}_k(\mathbb{X}, \mathbb{Y})$, with the property that $\pi_{n,m}; \pi_{m,k} = \pi_{n,k}$.

Definition 4.5 (Observational equivalence)**.** Two extensional stateful sequences

$$\langle f \rangle_{n \in \mathbb{N}}, \langle g \rangle_{n \in \mathbb{N}} \in \int^{M \in [\mathbb{N}, \mathbf{C}]} \prod_{i=0}^{\infty} \hom(M_{i-1} \otimes X_i, Y_i \otimes M_i)$$

are *observationally equivalent* when all their n-stage truncations are equal. That is, $\langle f_0 | \dots | f_n | = \langle g_0 | \dots | g_n |$, for each $n \in \mathbb{N}$. We write this as $f \approx g$.

Remark 4.6. Formally, this is to say that the sequences $\langle f \rangle_{n \in \mathbb{N}}$ and $\langle g \rangle_{n \in \mathbb{N}}$ have the same image on the limit

$$\lim_n \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}),$$

over the chain $\pi_{n,k} : \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}) \to \mathbf{Stage}_k(\mathbb{X}, \mathbb{Y})$.

Definition 4.7. An *observational sequence* from $\mathbb{X}$ to $\mathbb{Y}$ is an equivalence class

$$[\langle f_n : M_{n-1} \otimes X_n \to M_n \otimes Y_n \rangle_{n \in \mathbb{N}}]_{\approx}$$

of extensional sequences under observational equivalence. In other words, the set of observational sequences is

$$\mathbf{Obs}(\mathbb{X}, \mathbb{Y}) \cong \left(\int^{M \in [\mathbb{N}, \mathbf{C}]} \prod_{i=0}^{\infty} \hom(M_{i-1} \otimes X_i, M_i \otimes Y_i) \right) \Big/ \approx$$

4.3 Productive categories

The interaction between extensional and observational equivalence is of particular interest in some well-behaved categories that we call *productive categories*. In productive categories, observational sequences are the final fixpoint of an equation (Theorem 4.11), analogous to that of Section 2.

An important property of programs is *termination*: a terminating program always halts in a finite amount of time. However, some programs (such as servers, drivers) are not actually intended to terminate but to produce infinite output streams. A more appropriate notion in these cases is that of *productivity*: a program that outputs an infinite stream of data is productive if each individual component of the stream is produced in finite time. To quip, *"a productive stream is a terminating first component followed by a productive stream"*.

The first component of our streams is only defined *up to some future*. It is an equivalence class $\alpha \in \mathbf{Stage}_1(\mathbb{X}, \mathbb{Y})$, with representatives $\alpha_i : X_0 \to M_i \otimes Y_0$. But, if it does terminate, there is a process $\alpha_0 : X_0 \to M_0 \otimes Y_0$ in our theory representing the process just until Y_0 is output.

Definition 4.8 (Terminating component)**.** A 1-stage process $\alpha \in \mathbf{Stage}_1(\mathbb{X}, \mathbb{Y})$ is *terminating relative to* C if there exists $\alpha_0 : X_0 \to M_0 \otimes Y_0$ such that each one of its representatives, $\langle \alpha_i | = \alpha$, can be written as $\alpha_i = \alpha_0; (s_i \otimes \mathrm{id})$ for some $s_i : M_0 \to M_i$.

The morphisms s_i represent what is unique to each representative, and so we ask that, for any $u : M_0 \otimes A \to U \otimes B$ and $v : M_0 \otimes A \to V \otimes B$, the equality $\langle \alpha_i \otimes \mathrm{id}_A; u \otimes \mathrm{id}_{Y_0} | = \langle \alpha_j \otimes \mathrm{id}_A; v \otimes \mathrm{id}_{Y_0} |$ implies $\langle s_i \otimes \mathrm{id}_A; u | = \langle s_j \otimes \mathrm{id}_A; v |$.

Definition 4.9 (Productive category)**.** A symmetric monoidal category $(\mathbf{C}, \otimes, I)$ is *productive* when every 1-stage process is terminating relative to C.

Remark 4.10. Cartesian monoidal categories are productive (Proposition C.1). *Markov categories* [30] with *conditionals* and *ranges* are productive (Theorem A.21). Free symmetric monoidal categories and compact closed categories are always productive.

Theorem 4.11. *Observational sequences are the explicit construction of observational streams when the category is productive. More precisely, in a productive category, the final fixpoint of the equation in Figure 9 is given by the set of observational sequences,* **Obs**.

Proof sketch. The terminal sequence for this final coalgebra is given by $\mathbf{Stage}_n(\mathbb{X}, \mathbb{Y})$. In productive categories, we can prove that the limit $\lim_n \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y})$ is a fixpoint of the equation in Figure 9 (Lemma D.6). Finally, in productive categories, observational sequences coincide with this limit (Theorem D.7). □

5 The Category of Monoidal Streams

We are ready to construct Stream: the feedback monoidal category of monoidal streams. Let us recast the definitive notion of monoidal stream (Definition 4.1) coinductively.

Definition 5.1. A *monoidal stream* $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ is a triple consisting of

- $M(f) \in \mathrm{Obj}(\mathbf{C})$, the *memory*,
- $\mathrm{now}(f) \in \hom(X_0, M(f) \otimes Y_0)$, the *first action*,
- $\mathrm{later}(f) \in \mathrm{Stream}(M(f) \cdot \mathbb{X}^+, \mathbb{Y}^+)$, the *rest of the action*,

quotiented by dinaturality in M.

Explicitly, monoidal streams are quotiented by the equivalence relation $f \sim g$ generated by

- the existence of $r : M(g) \to M(f)$,

- such that $\mathrm{now}(f) = \mathrm{now}(g); r$,
- and such that $r \cdot \mathrm{later}(f) \sim \mathrm{later}(g)$.

Here, $r \cdot \mathrm{later}(f) \in \mathrm{Stream}(M(g) \cdot \mathbb{X}^+, \mathbb{Y}^+)$ is obtained by precomposition of the first action of $\mathrm{later}(f)$ with r.

Remark 5.2. This is a coinductive definition of the functor

$$\mathrm{Stream} \colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathrm{Set}.$$

In principle, arbitrary final coalgebras do not need to exist. Moreover, it is usually difficult to explicitly construct such coalgebras [4]. However, in productive categories, this coalgebra does exist and is constructed by observational sequences. From now on, we reason *coinductively* [48], a style particularly suited for all the following definitions.

5.1 The symmetric monoidal category of streams

The definitions for the operations of sequential and parallel composition are described in two steps. We first define an operation that takes into account an extra *memory channel* (Figure 10); we use this extra generality to strengthen the *coinduction hypothesis*. We then define the desired operation as a particular case of this coinductively defined one.

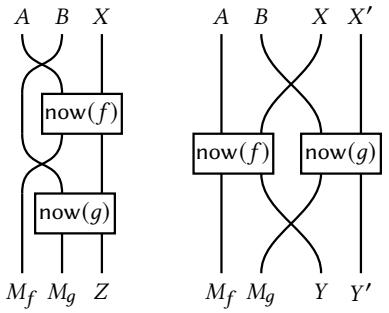


Figure 10. String diagrams for the first action of sequential and parallel composition with memories.

Definition 5.3 (Sequential composition)**.** Given two streams $f \in \mathrm{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$ and $g \in \mathrm{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$, we compute $(f^A; g^B) \in \mathrm{Stream}((A \otimes B) \cdot \mathbb{X}, \mathbb{Z})$, their *sequential composition with memories A and B*, as

- $M(f^A ; g^B) = M(f) \otimes M(g)$,
- $\mathrm{now}(f^A ; g^B) = \sigma ; (\mathrm{now}(f) \otimes \mathrm{id}) ; \sigma ; (\mathrm{now}(g) \otimes \mathrm{id})$,
- $\mathrm{later}(f^A ; g^B) = \mathrm{later}(f)^{M(f)} ; \mathrm{later}(g)^{M(g)}$.

We write $(f ; g)$ for $(f^I ; g^I) \in \mathrm{Stream}(\mathbb{X}, \mathbb{Z})$; the *sequential composition* of $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ and $g \in \mathrm{Stream}(\mathbb{Y}, \mathbb{Z})$.

Definition 5.4. The *identity* $\mathrm{id}_{\mathbb{X}} \in \mathrm{Stream}(\mathbb{X}, \mathbb{X})$ is defined by $M(\mathrm{id}_{\mathbb{X}}) = I$, $\mathrm{now}(\mathrm{id}_{\mathbb{X}}) = \mathrm{id}_{\mathbb{X}_0}$, and $\mathrm{later}(\mathrm{id}_{\mathbb{X}}) = \mathrm{id}_{\mathbb{X}^+}$.

Definition 5.5 (Parallel composition)**.** Given two streams $f \in \mathrm{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$ and $g \in \mathrm{Stream}(B \cdot \mathbb{X}', \mathbb{Y}')$, we compute $(f^A \otimes g^B) \in \mathrm{Stream}((A \otimes B) \cdot (\mathbb{X} \otimes \mathbb{X}'), \mathbb{Y} \otimes \mathbb{Y}')$, their *parallel composition with memories A and B*, as

- $M(f^A \otimes g^B) = M(f) \otimes M(g)$,
- $\mathrm{now}(f^A \otimes g^B) = \sigma; (\mathrm{now}(f) \otimes \mathrm{now}(g)); \sigma$,
- $\mathrm{later}(f^A \otimes g^B) = \mathrm{later}(f)^{M(f)} \otimes \mathrm{later}(g)^{M(g)}$.

We write $(f \otimes g)$ for $(f^I \otimes g^I) \in \mathrm{Stream}(\mathbb{X} \otimes \mathbb{X}', \mathbb{Y} \otimes \mathbb{Y}')$; we call it the *parallel composition* of $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ and $g \in \mathrm{Stream}(\mathbb{X}', \mathbb{Y}')$.

Definition 5.6 (Memoryless and constant streams)**.** Each sequence $\mathbb{f} = (f_0, f_1, \ldots)$, with $f_n \colon X_n \to Y_n$, induces a stream $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ defined by $M(f) = I$, $\mathrm{now}(f) = f_0$, and $\mathrm{later}(f) = \mathbb{f}^+$. Streams of this form are called *memoryless*, i.e. their memories are given by the monoidal unit.

Moreover, each morphism $f_0 \colon X \to Y$ induces a *constant* memoryless stream that we also call $f \in \mathrm{Stream}(X, Y)$, defined by $M(f) = I$, $\mathrm{now}(f) = f_0$, and $\mathrm{later}(f) = f$.

Theorem 5.7. *Monoidal streams over a productive symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a symmetric monoidal category* Stream *with a symmetric monoidal identity-on-objects functor from* $[\mathbb{N}, \mathsf{C}]$.

Proof. Appendix, Theorem E.5. □

5.2 Delayed feedback for streams

Monoidal streams form a *delayed feedback* monoidal category. Given some stream in $\mathrm{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, we can create a new stream in $\mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ that passes the output in $\mathbb{S}$ as a memory channel that gets used as the input in $\partial\mathbb{S}$. As a consequence, the category of monoidal streams has a graphical calculus given by that of feedback monoidal categories. This graphical calculus is complete for extensional equivalence (as we saw in Theorem 3.8).

Definition 5.8 (Delay functor)**.** The functor from Definition 3.6 can be lifted to a monoidal functor $\partial \colon \mathrm{Stream} \to \mathrm{Stream}$ that acts on objects in the same way. It acts on morphisms by sending a stream $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ to the stream given by $M(\partial f) = I$, $\mathrm{now}(\partial f) = \mathrm{id}_I$ and $\mathrm{later}(\partial f) = f$.

Definition 5.9 (Feedback operation)**.** Given any morphism of the form $f \in \mathrm{Stream}(N \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, we define $\mathbf{fbk}(f^N) \in \mathrm{Stream}(N \cdot \mathbb{X}, \mathbb{Y})$ as

- $M(\mathbf{fbk}(f^N)) = M(f) \otimes S_0$,
- $\mathrm{now}(\mathbf{fbk}(f^N)) = \mathrm{now}(f)$ and
- $\mathrm{later}(\mathbf{fbk}(f^N)) = \mathbf{fbk}(\mathrm{later}(f)^{M(f) \otimes S_0})$.

We write $\mathbf{fbk}(f) \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ for $\mathbf{fbk}(f^I)$, the feedback of $f \in \mathrm{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$

Theorem 5.10. *Monoidal streams over a symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a* ∂*-feedback monoidal category* $(\mathrm{Stream}, \mathbf{fbk})$.

Proof. Appendix, Theorem E.16. □

Corollary 5.11. *There is a "semantics" identity-on-objects feedback monoidal functor* $\mathbf{Sm} \colon \mathrm{St}_\partial[\mathbb{N}, \mathsf{C}] \to \mathrm{Stream}$ *from the free* ∂*-feedback monoidal category to the category of monoidal streams. Every extensional stateful sequence* $\langle f_n \colon M_{n-1} \otimes$

$X_n \to Y_n \otimes M_n\rangle_{n\in\mathbb{N}}$ *gives a monoidal stream* $\mathbf{Sm}(f)$*, which is defined by* $M(\mathbf{Sm}(f)) = M_0$*,*

$$\mathrm{now}(\mathbf{Sm}(f)) = f_0, \text{ and } \mathrm{later}(\mathbf{Sm}(f)) = \mathbf{Sm}(f^+),$$

and this is well-defined. Moreover, this functor is full when C *is productive; it is not generally faithful.*

Proof. We construct **Sm** from Theorems 3.5 and 5.10. Moreover, when C is productive, by Theorem 4.11, monoidal streams are extensional sequences quotiented by observational equivalence, giving the fullness of the functor. □

6 Cartesian Streams

Dataflow languages such as Lucid or Lustre [35, 77] can be thought of as using an underlying cartesian monoidal structure: we can copy and discard variables and resources without affecting the order of operations. These abilities correspond exactly to cartesianity thanks to Fox's theorem (Theorem C.5, see [29]).

6.1 Causal stream functions

In the cartesian case, there is available literature on the categorical semantics of dataflow programming languages [8, 24, 25, 58, 76]. Uustalu and Vene [75] provide elegant comonadic semantics to a Lucid-like programming language using the non-empty list comonad. In their framework, *streams* with types $\mathbb{X} = (X_0, X_1, \ldots)$ are families of elements $\mathbf{1} \to X_n$. *Causal stream functions* from $\mathbb{X} = (X_0, X_1, \ldots)$ to $\mathbb{Y} = (Y_0, Y_1, \ldots)$ are families of functions $f_n \colon X_0 \times \cdots \times X_n \to Y_n$. Equivalently, they are, respectively, the states $(\mathbb{1} \to \mathbb{X})$ and morphisms $(\mathbb{X} \to \mathbb{Y})$ of the cokleisli category of the comonad $\mathbf{List}^+ \colon [\mathbb{N}, \mathrm{Set}] \to [\mathbb{N}, \mathrm{Set}]$ defined by

$$(\mathbf{List}^+(\mathbb{X}))_n \coloneqq \prod_{i=0}^{n} X_i.$$

This comonad can be extended to other base categories, $\mathbf{List}^+ \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$ *only* as long as C is cartesian. Indeed, we can prove that the mere existence of such a comonad implies cartesianity of the base category. For this, we introduce a refined version of Fox's theorem (Theorem C.7).

Theorem 6.1. *Let* $(\mathrm{C}, \otimes, I)$ *be a symmetric monoidal category. Let* $\mathbf{List}^+ \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$ *be the functor defined by*

$$\mathbf{List}^+(X)_n \coloneqq \bigotimes_{i=0}^{n} X_i.$$

This functor is monoidal, with oplaxators $\psi_0^+ \colon \mathbf{List}^+(I) \to I$ *and* $\psi_{X,Y} \colon \mathbf{List}^+(X \otimes Y) \to \mathbf{List}^+(X) \otimes \mathbf{List}^+(Y)$ *given by symmetries, associators and unitors.*

The monoidal functor $\mathbf{List}^+ \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$ *has a monoidal comonad structure if and only if its base monoidal category* $(\mathrm{C}, \otimes, I)$ *is cartesian monoidal.*

Proof sketch. The cartesian structure can be shown to make $\mathbf{List}^+$ an opmonoidal comonad. Conversely, the opmonoidal comonad structure implies that every object should have a natural and uniform counital comagma structure. By our refined statement of Fox's theorem (Theorem C.7), this implies cartesianity. See Appendix, Theorem C.4. □

This means that we cannot directly extend Uustalu and Vene's approach to the monoidal case. However, we prove in the next section that our definition of monoidal streams particularizes to their *causal stream functions* [73, 75].

6.2 Cartesian monoidal streams

The main claim of this section is that, in a cartesian monoidal category, monoidal streams instantiate to *causal stream functions* (Theorem 6.3). Let us fix such a category, $(\mathrm{C}, \times, 1)$.

The first observation is that the universal property of the cartesian product simplifies the fixpoint equation that defines monoidal streams. This is a consequence of the following chain of isomorphisms, where we apply a Yoneda reduction to simplify the coend.

$$\begin{aligned}
&\mathrm{Stream}(\mathbb{X}, \mathbb{Y}) \cong \\
&\textstyle\int^M \mathrm{hom}(X_0, M \times Y_0) \times \mathrm{Stream}(M \cdot \mathbb{X}^+, \mathbb{Y}) \cong \\
&\textstyle\int^M \mathrm{hom}(X_0, M) \times \mathrm{hom}(X_0, Y_0) \times \mathrm{Stream}(M \cdot \mathbb{X}^+, \mathbb{Y}) \cong \\
&\mathrm{hom}(X_0, Y_0) \times \mathrm{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+).
\end{aligned}$$

Explicitly, the Yoneda reduction works as follows: the first action of a stream $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ can be uniquely split as $\mathrm{now}(f) = (f_1, f_2)$ for some $f_1 \colon X_0 \to Y_0$ and $f_2 \colon X_0 \to M(f)$. Under the *dinaturality* equivalence relation, $(\sim)$, we can always find a unique representative with $M = X_0$ and $f_2 = \mathrm{id}_{X_0}$.

The definition of monoidal streams in the cartesian case is thus simplified (Definition 6.2). From there, the explicit construction of cartesian monoidal streams is straightforward.

Definition 6.2 (Cartesian monoidal streams)**.** The set of *cartesian monoidal streams*, given inputs $\mathbb{X}$ and outputs $\mathbb{Y}$, is the terminal fixpoint of the equation

$$\mathrm{Stream}(\mathbb{X}, \mathbb{Y}) \cong \mathrm{hom}(X_0, Y_0) \times \mathrm{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

In other words, a cartesian monoidal stream $f \in \mathrm{Stream}(\mathbb{X}, \mathbb{Y})$ is a pair consisting of

- $\mathrm{fst}(f) \in \mathrm{hom}(X_0, Y_0)$, the *first action*, and
- $\mathrm{snd}(f) \in \mathrm{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+)$, the *rest of the action.*

Theorem 6.3. *In the cartesian case, the final fixpoint of the equation in Figure 9 is given by the set of causal functions,*

$$\mathrm{Stream}(\mathbb{X}, \mathbb{Y}) = \prod_{n\in\mathbb{N}}^{\infty} \mathrm{hom}(X_0 \times \cdots \times X_n, Y_n).$$

That is, the category Stream *of monoidal streams coincides with the cokleisli monoidal category of the non-empty list monoidal comonad* $\mathbf{List}^+ \colon [\mathbb{N}, \mathrm{C}] \to [\mathbb{N}, \mathrm{C}]$.

Proof. By Adamek's theorem (Theorem 1.5). □

Corollary 6.4. *Let* $(\mathrm{C}, \times, \mathbf{1})$ *be a cartesian monoidal category. The category* Stream *is cartesian monoidal.*

6.3 Example: the Fibonacci sequence

Consider $(\mathrm{Set}, \times, \mathbf{1})$, the cartesian monoidal category of small sets and functions. And let us go back to the morphism $\mathrm{fib} \in \mathrm{Stream}(\mathbf{1}, \mathbb{N})$ that we presented in the Introduction (Figure 2). By Theorem 6.3, a morphism of this type is, equivalently, a sequence of natural numbers. Using the previous definitions in Sections 5 and 6, we can explicitly compute this sequence to be $\mathrm{fib} = [0, 1, 1, 2, 3, 5, 8, \dots]$ (see the Appendix, Example G.1).

7 Stochastic Streams

Monoidal categories are well suited for reasoning about probabilistic processes. Several different categories of probabilistic channels have been proposed in the literature [6, 19, 59]. They were largely unified by Fritz [30] under the name of *Markov categories*. For simplicity, we work in the discrete stochastic setting, i.e. in the Kleisli category of the finite distribution monad, Stoch, but we will be careful to isolate the relevant structure of Markov categories that we use.

The main result of this section is that *controlled stochastic processes* [27, 66] are precisely monoidal streams over Stoch. That is, controlled stochastic processes are the canonical solution over Stoch of the fixpoint equation in Figure 9.

7.1 Stochastic processes

We start by recalling the notion of *stochastic process* from probability theory and its "controlled" version. The latter is used in the context of *stochastic control* [27, 66], where the user has access to the parameters or optimization variables of a probabilistic model.

A *discrete stochastic process* is defined as a collection of random variables $Y_1, \dots Y_n$ indexed by discrete time. At any time step n, these random variables are distributed according to some $p_n \in \mathbf{D}(Y_1 \times \cdots \times Y_n)$. Since the future cannot influence the past, the marginal of p_{n+1} over Y_{n+1} must equal p_n. When this occurs, we say that the family of distributions $(p_n)_{n\in\mathbb{N}}$ is *causal*.

More generally, there may be some additional variables $X_1, \dots, X_n$ which we have control over. In this case, a *controlled stochastic process* is defined as a collection of *controlled random variables* distributing according to $f_n \colon X_1 \times \cdots \times X_n \to \mathbf{D}(Y_1, \dots, Y_n)$. Causality ensures that the marginal of f_{n+1} over Y_{n+1} must equal f_n.

Definition 7.1. Let $\mathbb{X}$ and $\mathbb{Y}$ be sequences of sets. A *controlled stochastic process* $f \colon \mathbb{X} \to \mathbb{Y}$ is a sequence of functions

$$f_n \colon X_n \times \cdots \times X_1 \to \mathbf{D}(Y_n \times \cdots \times Y_1)$$

satisfying *causality* (the *marginalisation property*). That is, such that f_n coincides with the marginal distribution of f_{n+1} on the first n variables, making the diagram in Figure 11 commute.

$$\begin{array}{ccc} X_0 \times \cdots \times X_{n+1} & \xrightarrow{f_{n+1}} & D(Y_0 \times \cdots \times Y_{n+1}) \\ \downarrow{\scriptstyle \pi_{0,\dots,n}} & & \downarrow{\scriptstyle D\pi_{0,\dots,n}} \\ X_0 \times \cdots \times X_n & \xrightarrow{f_n} & D(Y_0 \times \cdots \times Y_n) \end{array}$$

Figure 11. Marginalisation for stochastic processes.

Controlled stochastic processes with componentwise composition, identities and tensoring, are the morphisms of a symmetric monoidal category StochProc.

Stochastic monoidal streams and stochastic processes not only are the same thing but they compose in the same way: they are *isomorphic* as categories.

Theorem 7.2. *The category of stochastic processes* StochProc *is monoidally isomorphic to the category* Stream *over* Stoch.

Proof sketch. Appendix, Theorem A.29. The proof of this result is non-trivial and relies on a crucial property concerning ranges in Stoch. The proof is moreover written in the language of Markov categories where the property of ranges can be formulated in full abstraction. □

We expect that the theorem above can be generalised to interesting categories of probabilistic channels over measurable spaces (such as the ones covered in [19, 30, 59]).

Corollary 7.3. StochProc *is a feedback monoidal category.*

7.2 Examples

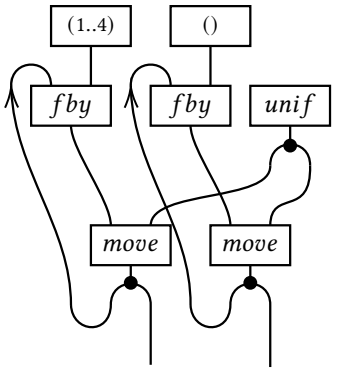
We have characterized in two equivalent ways the notion of controlled stochastic process. This yields a categorical semantics for probabilistic dataflow programming: we may use the syntax of feedback monoidal categories to specify simple stochastic programs and evaluate their semantics in StochProc.

Example 7.4 (Random Walk). Recall the morphism $\mathrm{walk} \in \mathrm{Stream}(\mathbf{1}, \mathbb{Z})$ that we depicted back in Figure 4.

Here, $\mathrm{unif} \in \mathrm{Stream}(\mathbf{1}, \{-1, 1\})$, is a uniform random generator that, at each step, outputs either 1 or (-1). The output of this uniform random generator is then added to the current position, and we declare the starting position to be 0. Our implementation of this morphism, following the definitions from Section 5 (Example G.2) is, by Theorem 7.2, a discrete stochastic process, and it produces samples like the following ones.

$$[0, 1, 0, -1, -2, -1, -2, -3, -2, -3, \dots]$$
$$[0, 1, 2, 1, 2, 1, 2, 3, 4, 5, \dots]$$
$$[0, -1, -2, -1, -2, -1, 0, -1, 0, -1, \dots]$$

Example 7.5 (Ehrenfest model). The Ehrenfest model [47, §1.4] is a simplified model of particle diffusion.



ehr =
(1...4) ⊗ (); **fbk**(σ;
fby ⊗ fby ⊗ unif;
id ⊗ id ⊗ copy;σ;
move ⊗ move;
copy)

Figure 12. Ehrenfest model: sig. flow graph and morphism.

Assume we have two urns with 4 balls, labelled from 1 to 4. Initially, the balls are all in the first urn. We randomly (and uniformly) pick an integer from 1 to 4, and the ball labelled by that number is removed from its box and placed in the other box. We iterate the procedure, with independent uniform selections each time.

Our implementation of this morphism, following the definitions from Section 5 (Example G.3) yields samples such as the following.

$$[([2,3,4],[1]),\quad ([3,4],[1,2]),\quad ([1,3,4],[2]),$$
$$([1,4],[2,3]),\quad ([1],[2,3,4]),\quad ([],[1,2,3,4]),$$
$$([2],[1,3,4]),\quad \ldots]$$

8 A dataflow programming language

In this section, we introduce the syntax for two Lucid-like dataflow programming languages and their semantics in monoidal streams. The first one is *deterministic* and it takes semantics in the feedback monoidal category of set-based monoidal streams. The second one is *stochastic* and it takes semantics in the feedback monoidal category of stochastic processes, or stochastic monoidal streams.

We do so by presenting a type theory for *feedback* monoidal categories (similar to [36, 71]). Terms of the type theory represent programs in our language.

8.1 Type theory for monoidal categories

We start by considering a *type theory for symmetric monoidal categories* over some generators forming a multigraph $\mathcal{G}$. Instead of presenting a type theory from scratch, we extend the basic type theory for symmetric monoidal categories described by Shulman [70]. Details are in the Appendix (Appendix F). Here, we only illustrate it with an example.

$$\textsc{Gen}\ \frac{f \in \mathcal{G}(A_1,\ldots,A_n;B) \qquad \Gamma_1 \vdash x_1 : A_1 \ldots \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1,\ldots,\Gamma_n) \vdash f(x_1,\ldots,x_n) : B}$$

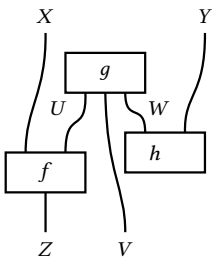
$$\textsc{Pair}\ \frac{\Gamma_1 \vdash x_1 : A_1 \ \ldots \ \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1,\ldots,\Gamma_n) \vdash [x_1,\ldots,x_n] : A_1 \otimes \ldots \otimes A_n} \qquad \textsc{Var}\ \frac{}{x : A \vdash x : A}$$

$$\textsc{Split}\ \frac{\Delta \vdash m : A_1 \otimes \cdots \otimes A_n \qquad \Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash z : C}{\mathrm{Shuf}(\Gamma,\Delta) \vdash \textsc{Split}\ m \to [x_1,\ldots,x_n]\ \textsc{in}\ z : C}$$

Figure 13. Type theory of symm. monoidal categories [70].

The type theory for symmetric monoidal categories is *linear* [34, 52, 68] in the sense that any introduced variable must be used exactly once. This is for a good reason: monoidal categories represent linear theories of processes, where *copying* and *discarding* may not be allowed in general.

Example 8.1. In a monoidal category, let $f\colon X \otimes U \to Z$, $g\colon I \to U \otimes V \otimes W$ and $h\colon V \otimes Y \to I$. The following is a string diagram together with its term in the type theory.



$\textsc{Split}\ g \to [u,v,w]\ \textsc{in}$
$\textsc{Split}\ h(w,y) \to []\ \textsc{in}$
$[f(x,u),v]$

8.2 Adding feedback

We now extend the theory with delay and feedback. We start by considering a ∂ operator on types, which extends to contexts inductively as $\partial[] = []$ and $\partial(\Gamma, x{:}A) = \partial\Gamma, (x : \partial A)$. We provide formation rules for introducing delay and feedback. These need to satisfy equalities making Delay a functor and Fbk a feedback operator.

$$\textsc{Delay}\ \frac{\Gamma \vdash x : A}{\partial\Gamma \vdash x : \partial A} \qquad \textsc{Fbk}\ \frac{\Gamma, s : \partial S \vdash x(s) : S \otimes A}{\Gamma \vdash \textsc{Fbk}\ s.\ x(s) : A}$$

As in Remark 3.2, we define $\textsc{Wait}(x) \equiv \textsc{Fbk}\ y\ \textsc{in}\ [x,y]$.

8.3 Adding generators

In both versions of the language (deterministic and stochastic), we include "copy" and "followed by" operations, representing the corresponding monoidal streams. Copying does not need to be natural (in the stochastic case, it will not be)

and it does not even need to form a comonoid.

$$\textsc{Copy}\ \frac{\Gamma \vdash x : A}{\Gamma \vdash \textsc{Copy}(x) : A \otimes A} \qquad \textsc{Fby}\ \frac{\Gamma \vdash x : A \qquad \Delta \vdash y : \partial(A)}{\textsc{Shuf}(\Gamma, \Delta) \vdash x\ \textsc{Fby}\ y : A}$$

In fact, recursive definitions make sense only when we have a *copy* operation, that allows us to rewrite the definition as a feedback that ends with a copy. That is,

$$M = x(M) \quad \text{means} \quad M = \textsc{Fbk}\ m\ \textsc{in}\ \textsc{Copy}(x(m)).$$

Moreover, in the deterministic version of our language we allow non-linearity: a variable can occur multiple times, implicitly copying it.

8.4 Examples

Example 8.2. Recall the example from the introduction (and Section 6.3).

$$\mathit{fib} = 0\ \textsc{Fby}\ (\mathit{fib} + (1\ \textsc{Fby}\ \textsc{Wait}\ \mathit{fib}))$$

Its desugaring, following the previous rules, is below.

$$\begin{aligned}\mathit{fib} = \textsc{Fbk}\ f\ \textsc{in}\ \textsc{Copy}\\ (0\ \textsc{Fby}\\ \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2))\ )\end{aligned}$$

Example 8.3 (Ehrenfest model). The Ehrenfest model described in Figure 12 has the following specification in the programming language.

$$\begin{aligned}\mathit{urns} = [(1,2,3,4),()]\ \textsc{Fby}\\ \textsc{Split}\ \mathit{urns} \to [u_1, u_2]\ \textsc{in}\\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]\end{aligned}$$

Sampling twice from the same distribution is different from copying a single sample, and Split allows us to express this difference: instead of calling the Uniform distribution twice, this program calls it once and then copies the result.

9 Conclusions

Monoidal streams are a common generalization of streams, causal functions and stochastic processes. In the same way that streams give semantics to dataflow programming [35, 77] with plain functions, monoidal streams give semantics to dataflow programming with monoidal theories of processes. Signal flow graphs are a common tool to describe control flow in dataflow programming. Signal flow graphs are also the natural string diagrams of feedback monoidal categories. Monoidal streams form a feedback monoidal category, and signal flow graphs are a formal syntax to describe and reason about monoidal streams. The second syntax we present comes from the type theory of monoidal categories, and it is inspired by the original syntax of dataflow programming. We have specifically studied stochastic dataflow programming, but the same framework allows for *linear*, *quantum* and *effectful* theories of resources.

The literature on dataflow and feedback is rich enough to provide multiple diverging definitions and approaches. What we can bring to this discussion are universal constructions. Universal constructions justify some mathematical object as *the canonical object* satisfying some properties. In our case, these exact properties are extracted from three, arguably under-appreciated, but standard category-theoretic tools: *dinaturality*, *feedback*, and *coalgebra*. *Dinaturality*, profunctors and coends, sometimes regarded as highly theoretical developments, are the natural language to describe how processes communicate and compose. *Feedback*, sometimes eclipsed by trace in the mathematical literature, keeps appearing in multiple variants across computer science. *Coalgebra* is the established tool to specify and reason about stateful systems.

9.1 Further work

Other theories. Many interesting examples of theories of processes are not monoidal but just *premonoidal categories* [42, 62]. For instance, the kleisli categories of arbitrary monads, where effects (e.g. reading and writing to a global state) do not need to commute. Premonoidal streams can be constructed by restricting dinaturality to their *centres*. Another important source of theories of processes that we have not covered is that of *linearly distributive* and **-autonomous categories* [11, 12, 22, 68].

Within monoidal categories, we would like to make monoidal streams explicit in the cases of partial maps [21] for dataflow programming with different clocks [76], nondeterministic maps [17, 51] and quantum processes [18]. A final question we do not pursue here is *expressivity*: the class of functions a monoidal stream can define.

The 2-categorical view. We describe the morphisms of a category as a final coalgebra. However, it is also straightforward to describe the 2-endofunctor that should give rise to this category as a final coalgebra itself.

Implementation of the type theory. Justifying that the output of monoidal streams is the expected one requires some computations, which we have already implemented separately in the Haskell programming language (Appendix, Appendix G). Agda has similar foundations and supports the coinductive definitions of this text (Section 5). It is possible to implement a whole interpreter for a Lucid-like stochastic programming language with a dedicated parser, but that requires some software engineering effort that we postpone for further work.

References

[1] Martín Abadi and Michael Isard. Timely Dataflow: A Model. In Susanne Graf and Mahesh Viswanathan, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1*

International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings, volume 9039 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2015. doi:10.1007/978-3-319-19195-9_9.

[2] Samson Abramsky and Bob Coecke. Categorical quantum mechanics. *Handbook of quantum logic and quantum structures*, 2:261–325, 2009. arXiv:0808.1023.

[3] Jiří Adámek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.

[4] Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974. URL: http://eudml.org/doc/16649.

[5] Jiří Adámek. On Terminal Coalgebras Derived from Initial Algebras. In Markus Roggenbach and Ana Sokolova, editors, *8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom*, volume 139 of *LIPIcs*, pages 12:1–12:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CALCO.2019.12.

[6] John C. Baez, Brendan Fong, and Blake S. Pollard. A Compositional Framework for Markov Processes. *Journal of Mathematical Physics*, 57(3):033301, March 2016. arXiv:1508.06448, doi:10.1063/1.4941578.

[7] Jon Beck. Distributive laws. In *Seminar on triples and categorical homology theory*, pages 119–140. Springer, 1969.

[8] Albert Benveniste, Paul Caspi, Paul Le Guernic, and Nicolas Halbwachs. Data-flow synchronous languages. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 1–45. Springer, 1993. doi:10.1007/3-540-58043-3_16.

[9] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 55–64. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.16.

[10] 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.

[11] Richard Blute. Linear logic, coherence, and dinaturality. *Theor. Comput. Sci.*, 115(1):3–41, 1993. doi:10.1016/0304-3975(93)90053-V.

[12] Richard F Blute, J Robin B Cockett, Robert AG Seely, and Todd H Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996.

[13] 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.

[14] Filippo Bonchi, Jens Seeber, and Paweł Sobociński. Graphical conjunctive queries. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.13.

[15] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*, pages 435–450. Springer, 2014.

[16] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Full abstraction for signal flow graphs. *ACM SIGPLAN Notices*, 50(1):515–526, 2015.

[17] Manfred Broy and Gheorghe Ştefănescu. The algebra of stream processing functions. *Theoretical Computer Science*, 258(1-2):99–129, 2001.

[18] Titouan Carette, Marc de Visme, and Simon Perdrix. Graphical language with delayed trace: Picturing quantum computing with finite memory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470553.

[19] Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019. arXiv:1709.00322, doi:10.1017/S0960129518000488.

[20] J. Robin B. Cockett, Xiuzhan Guo, and Pieter Hofstra. Range Categories I: General theory. *Theory and Applications of Categories*, 26(17):412–452, 2012.

[21] J. Robin B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1-2):223–259, 2002. doi:10.1016/S0304-3975(00)00382-0.

[22] J Robin B Cockett and Robert AG Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

[23] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Inf. Comput.*, 250:59–86, 2016. doi:10.1016/j.ic.2016.02.008.

[24] Patrick Cousot. Syntactic and semantic soundness of structural dataflow analysis. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 96–117. Springer, 2019. doi:10.1007/978-3-030-32304-2_6.

[25] Antonin Delpeuch. A complete language for faceted dataflow programs. In John Baez and Bob Coecke, editors, *Proceedings Applied Category Theory 2019, ACT 2019, University of Oxford, UK, 15-19 July 2019*, volume 323 of *EPTCS*, pages 1–14, 2019. doi:10.4204/EPTCS.323.1.

[26] 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*, pages 63–81, Cham, 2021. Springer International Publishing.

[27] Wendell Helms Fleming and Raymond W. Rishel. *Deterministic and Stochastic Optimal Control.* Number vol 1 in Applications of Mathematics. Springer-Verlag, Berlin ; New York, 1975.

[28] Brendan Fong and David I Spivak. Supplying bells and whistles in symmetric monoidal categories. *arXiv preprint arXiv:1908.02633*, 2019.

[29] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976.

[30] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. URL: http://arxiv.org/abs/1908.07021, arXiv:1908.07021.

[31] Richard Garner. Stream processors and comodels. *arXiv preprint arXiv:2106.05473*, 2021.

[32] Simon J. Gay and Rajagopal Nagarajan. Intensional and extensional semantics of dataflow programs. *Formal Aspects Comput.*, 15(4):299–318, 2003. doi:10.1007/s00165-003-0018-1.

[33] Dan R. Ghica, George Kaye, and David Sprunger. Full abstraction for digital circuits, 2022. arXiv:2201.10456.

[34] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[35] Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. *IEEE Trans. Software Eng.*, 18(9):785–793, 1992. doi:10.1109/32.159839.

[36] Masahito Hasegawa. *Models of sharing graphs: a categorical semantics of let and letrec.* PhD thesis, University of Edinburgh, UK, 1997. URL: http://hdl.handle.net/1842/15001.

[37] Thomas Hildebrandt, Prakash Panangaden, and Glynn Winskel. A relational model of non-deterministic dataflow. In *International Conference on Concurrency Theory*, pages 613–628. Springer, 1998.

[38] 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.

[39] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S. Nikhil, Will Partain, and John Peterson. Report on the Programming Language Haskell, A Non-strict, Purely Functional Language. *ACM SIGPLAN Notices*, 27(5):1, 1992. doi:10.1145/130697.130699.

[40] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. doi:10.1016/S0167-6423(99)00023-4.

[41] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

[42] Alan Jeffrey. Premonoidal categories and flow graphs. *Electron. Notes Theor. Comput. Sci.*, 10:51, 1997. doi:10.1016/S1571-0661(05)80688-7.

[43] 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.

[44] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.

[45] 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.

[46] 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.

[47] Frank P. Kelly. *Reversibility and stochastic networks*. Cambridge University Press, 2011.

[48] Dexter Kozen and Alexandra Silva. Practical coinduction. *Mathematical Structures in Computer Science*, 27(7):1132–1152, 2017. doi:10.1017/S0960129515000493.

[49] J. Lambek. Cartesian closed categories and typed λ-calculi. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, Lecture Notes in Computer Science, pages 136–175, Berlin, Heidelberg, 1986. Springer. doi:10.1007/3-540-17184-3_44.

[50] Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103(2):151–161, 1968.

[51] Edward A. Lee and Eleftherios Matsikoudis. The semantics of dataflow with firing. *From Semantics to Computer Science: Essays in Honour of Gilles Kahn*, pages 71–94, 2009.

[52] Patrick Lincoln and John C. Mitchell. Operational aspects of linear lambda calculus. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 235–246. IEEE Computer Society, 1992. doi:10.1109/LICS.1992.185536.

[53] Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. doi:10.1017/9781108778657.

[54] Nancy A. Lynch and Eugene W. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, 1989.

[55] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.

[56] Konstantinos Mamouras. Semantic foundations for deterministic dataflow and stream processing. In Peter Müller, editor, *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12075 of *Lecture Notes in Computer Science*, pages 394–427. Springer, 2020. doi:10.1007/978-3-030-44914-8_15.

[57] 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.

[58] José Nuno Oliveira. *The formal semantics of deterministic dataflow programs*. PhD thesis, University of Manchester, UK, 1984. URL: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.376586.

[59] Prakash Panangaden. The Category of Markov Kernels. *Electronic Notes in Theoretical Computer Science*, 22:171–187, January 1999. doi:10.1016/S1571-0661(05)80602-4.

[60] Prakash Panangaden and Eugene W. Stark. Computations, residuals, and the power of indeterminacy. In *International Colloquium on Automata, Languages, and Programming*, pages 439–454. Springer, 1988.

[61] Ross Paterson. A new notation for arrows. In Benjamin C. Pierce, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, pages 229–240. ACM, 2001. doi:10.1145/507635.507664.

[62] John Power. Premonoidal categories as categories with algebraic structure. *Theor. Comput. Sci.*, 278(1-2):303–321, 2002. doi:10.1016/S0304-3975(00)00340-6.

[63] John Power and Hayo Thielecke. Closed freyd- and kappa-categories. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 625–634. Springer, 1999. doi:10.1007/3-540-48523-6_59.

[64] John Power and Hiroshi Watanabe. Distributivity for a monad and a comonad. In Bart Jacobs and Jan J. M. M. Rutten, editors, *Coalgebraic Methods in Computer Science, CMCS 1999, Amsterdam, The Netherlands, March 20-21, 1999*, volume 19 of *Electronic Notes in Theoretical Computer Science*, page 102. Elsevier, 1999. doi:10.1016/S1571-0661(05)80271-3.

[65] Mario Román. Comb diagrams for discrete-time feedback. *CoRR*, abs/2003.06214, 2020. arXiv:2003.06214.

[66] Sheldon M. Ross. *Stochastic processes*, volume 2. John Wiley & Sons, 1996.

[67] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

[68] Robert A.G. Seely. *Linear logic, *-autonomous categories and cofree coalgebras*. Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.

[69] Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines*. Bell Telephone Laboratories, 1942.

[70] Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. URL: https://mikeshulman.github.io/catlog/catlog.pdf.

[71] Michael Shulman. A practical type theory for symmetric monoidal categories, 2021. arXiv:1911.00818.

[72] David Sprunger and Bart Jacobs. The differential calculus of causal functions. *CoRR*, abs/1904.10611, 2019. URL: http://arxiv.org/abs/1904.10611, arXiv:1904.10611.

[73] David Sprunger and Shin-ya Katsumata. Differentiable causal computations via delayed trace. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785670.

[74] Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical theory of domains*, volume 22 of *Cambridge tracts in*

theoretical computer science. Cambridge University Press, 1994.

[75] Tarmo Uustalu and Varmo Vene. The essence of dataflow programming. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 2–18. Springer, 2005. doi:10.1007/11575467_2.

[76] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. In Jiří Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 263–284. Elsevier, 2008. doi:10.1016/j.entcs.2008.05.029.

[77] William W Wadge, Edward A Ashcroft, et al. *Lucid, the dataflow programming language*, volume 303. Academic Press London, 1985.

A Monoidal categories

Definition A.1 ([55])**.** A **monoidal category**,

$$(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho),$$

is a category C equipped with a functor $\otimes\colon \mathsf{C} \times \mathsf{C} \to \mathsf{C}$, a unit $I \in \mathsf{C}$, and three natural isomorphisms: the associator $\alpha_{A,B,C}\colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, the left unitor $\lambda_A\colon I \otimes A \cong A$ and the right unitor $\rho_A\colon A \otimes I \cong A$; such that $\alpha_{A,I,B}; (\mathrm{id}_A \otimes \lambda_B) = \rho_A \otimes \mathrm{id}_B$ and $(\alpha_{A,B,C} \otimes \mathrm{id}); \alpha_{A,B\otimes C,D}; (\mathrm{id}_A \otimes \alpha_{B,C,D}) = \alpha_{A\otimes B,C,D}; \alpha_{A,B,C\otimes D}$. A monoidal category is *strict* if α, λ and ρ are identities.

Definition A.2 (Monoidal functor, [55])**.** Let

$$(\mathsf{C}, \otimes, I, \alpha^{\mathsf{C}}, \lambda^{\mathsf{C}}, \rho^{\mathsf{C}}) \text{ and } (\mathsf{D}, \boxtimes, J, \alpha^{\mathsf{D}}, \lambda^{\mathsf{D}}, \rho^{\mathsf{D}})$$

be monoidal categories. A *monoidal functor* (sometimes called *strong monoidal functor*) is a triple (F, ε, μ) consisting of a functor $F\colon \mathsf{C} \to \mathsf{D}$ and two natural isomorphisms $\varepsilon\colon J \cong F(I)$ and $\mu\colon F(A \otimes B) \cong F(A) \boxtimes F(B)$; such that

- the associators satisfy

$$\begin{aligned}&\alpha^{\mathsf{D}}_{FA,FB,FC}; (\mathrm{id}_{FA} \otimes \mu_{B,C}); \mu_{A,B\otimes C}\\ &= (\mu_{A,B} \otimes \mathrm{id}_{FC}); \mu_{A\otimes B,C}; F(\alpha^{\mathsf{C}}_{A,B,C}),\end{aligned}$$

- the left unitor satisfies

$$(\varepsilon \otimes \mathrm{id}_{FA}); \mu_{I,A}; F(\lambda^{\mathsf{C}}_A) = \lambda^{\mathsf{D}}_{FA}$$

- the right unitor satisfies

$$(\mathrm{id}_{FA} \otimes \varepsilon); \mu_{A,I}; F(\rho^{\mathsf{C}}_{FA}) = \rho^{\mathsf{D}}_{FA}.$$

A monoidal functor is a *monoidal equivalence* if it is moreover an equivalence of categories. Two monoidal categories are monoidally equivalent if there exists a monoidal equivalence between them.

During most of the paper, we omit all associators and unitors from monoidal categories, implicitly using the *coherence theorem* for monoidal categories (Remark A.4).

Theorem A.3 (Coherence theorem, [55])**.** *Every monoidal category is monoidally equivalent to a strict monoidal category.*

Remark A.4. 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 $(\lambda_A; f; \lambda_B^{-1})\colon I \otimes A \to I \otimes B$. Each time the source and the target are clearly determined, we simply write f again for this new morphism.

Definition A.5 (Symmetric monoidal category, [55])**.** A *symmetric monoidal category* $(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho, \sigma)$ is a monoidal category $(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho)$ equipped with a braiding $\sigma_{A,B}\colon A \otimes B \to B \otimes A$, which satisfies the hexagon equation

$$\alpha_{A,B,C}; \sigma_{A,B\otimes C}; \alpha_{B,C,A} = (\sigma_{A,B} \otimes \mathrm{id}); \alpha_{B,A,C}; (\mathrm{id} \otimes \sigma_{A,C})$$

and additionally satisifes $\sigma_{A,B}; \sigma_{B,A} = \mathrm{id}$.

Remark A.6 (Notation). We omit symmetries when this does not cause confusion. We write $\underline{a}$ for the morphism a tensored with some identities when these can be deduced from the context. For instance, let $f\colon A \to B$, let $h\colon B \to D$ and let $g\colon B \otimes D \to E$. We write $\underline{f; h}; g$ for the morphism $(f \otimes \mathrm{id}); \sigma; (\mathrm{id} \otimes h); g$, which could have been also written as $(f \otimes \mathrm{id}); (h \otimes \mathrm{id}); \sigma; g$

Definition A.7 ([55])**.** A symmetric monoidal functor between two symmetric monoidal categories $(\mathsf{C}, \sigma^{\mathsf{C}})$ and $(\mathsf{D}, \sigma^{\mathsf{D}})$ is a monoidal functor $F\colon \mathsf{C} \to \mathsf{D}$ such that $\sigma^{\mathsf{D}}; \mu = \mu; F(\sigma^{\mathsf{C}})$.

Definition A.8. A *cartesian monoidal category* is a monoidal category whose tensor is the categorical product and whose unit is a terminal object.

Definition A.9. A *feedback functor* between two feedback monoidal categories $(\mathsf{C}, \mathbf{F}^{\mathsf{C}}, \mathbf{fbk}^{\mathsf{C}})$ and $(\mathsf{D}, \mathbf{F}^{\mathsf{D}}, \mathbf{fbk}^{\mathsf{D}})$ is a symmetric monoidal functor $G\colon \mathsf{C} \to \mathsf{D}$ such that $G; \mathbf{F}^{\mathsf{D}} = \mathbf{F}^{\mathsf{C}}; G$ and

$$G(\mathbf{fbk}^{\mathsf{C}}_S(f)) = \mathbf{fbk}^{\mathsf{D}}_{GS}(\mu_{\mathbf{F}S,A}; Gf; \mu^{-1}_{S,B}),$$

for each $f\colon \mathbf{F}S \otimes X \to S \otimes Y$, where $\mu_{A,B}\colon G(A) \otimes G(B) \to G(A \otimes B)$ is the structure morphism of the monoidal functor G.

Theorem A.10 (see [46])**.** $\mathrm{St}_{\mathbf{F}}(\mathsf{C})$ *is the free category with feedback over* $(\mathsf{C}, \mathbf{F})$.

Proof sketch. Let $(\mathsf{D}, \mathbf{F}^{\mathsf{D}}, \mathbf{fbk}^{\mathsf{D}})$ be any other symmetric monoidal category with an endofunctor, and let $H\colon \mathsf{C} \to \mathsf{D}$ be such that $\mathbf{F}; H = H; \mathbf{F}^{\mathsf{D}}$. We will prove that it can be extended uniquely to a feedback functor $\tilde{H}\colon \mathrm{St}_{\mathbf{F}}(\mathsf{C}) \to \mathsf{D}$.

It can be proven that any expression involving feedback can be reduced applying the feedback axioms to an expression of the form $\mathbf{fbk}(f)$ for some $f\colon \mathbf{F}S \otimes X \to S \otimes Y$. After this, the definition of $\tilde{H}$ in this morphism is forced to be $\tilde{H}(\mathbf{fbk} f) = \mathbf{fbk}^{\mathsf{D}}(\mathsf{D})$. This reduction is uniquely up to sliding, and the morphisms of the $\mathrm{St}(\bullet)$ construction are precisely morphisms $f\colon \mathbf{F}S \otimes X \to S \otimes Y$ quotiented by sliding equivalence. This is the core of the proof in [46]. □

A.1 Markov categories

Definition A.11. The finite distribution commutative monad $\mathbf{D}\colon \mathrm{Set} \to \mathrm{Set}$ associates to each set the set of finite-support probability distributions over it.

$$\mathbf{D}(X) = \left\{ p\colon X \to [0,1] \;\middle|\; \sum_{p(x)>0}^{|\{x|p(x)>0\}|<\infty} p(x) = 1 \right\}.$$

We call **Stoch** to the symmetric monoidal kleisli category of the finite distribution monad, $\mathsf{kl}(\mathbf{D})$.

We write $f(y|x)$ for the probability $f(x)(y) \in [0,1]$. Composition, $f; g$, is defined by

$$(f; g)(z|x) = \sum_{y \in Y} g(z|y) f(y|x).$$

The cartesian product $(\times)$ in Set induces a monoidal (non-cartesian) product on $\mathsf{kl}(\mathbf{D})$. That is, $\mathsf{kl}(\mathbf{D})$ has comonoids $(\curlywedge)_X : X \to X \times X$ on every object, with $(\curlyvee)_X\colon X \to 1$ as counit. However, contrary to what happens in Set, these comultiplications are not natural: *sampling and copying the result* is different from *taking two independent samples.*

Definition A.12 (Markov category, [30, Definition 2.1])**.** A *Markov category* C is a symmetric monoidal category in which each object $X \in \mathsf{C}$ has a cocommutative comonoid structure $(X, \varepsilon = \curlyvee_X : X \to I, \delta = \curlywedge_X : X \to X \otimes X)$ with

- *uniform* comultiplications, $\curlywedge_{X\otimes Y} = (\curlywedge_X \otimes \curlywedge_Y)\underline{\sigma_{X,Y}}$;
- *uniform* counits, $\curlyvee_{X\otimes Y} = \curlyvee_X \otimes \curlyvee_Y$; and
- *natural* counits, $f; \curlyvee_Y = \curlyvee_X$ for each $f\colon X \to Y$.

Crucially, comultiplications do not need to be natural.

Remark A.13 ([30, Remark 2.4])*.* Any cartesian category is a Markov category. However, not any Markov category is cartesian, and the most interesting examples are those that fail to be cartesian, such as Stoch. The failure of comultiplication being natural makes it impossible to apply Fox's theorem (Theorem C.5).

The structure of a Markov category is very basic. In most cases, we do need extra structure to reason about probabilities: this is the role of conditionals and ranges.

Remark A.14 (Notation). In a Markov category, given any $f\colon X_0 \to Y_0$ and any $g\colon Y_0 \otimes X_0 \otimes X_1 \to Y_1$, we write $(f \triangleleft g)\colon X_0 \otimes X_1 \to Y_0 \otimes Y_1$ for the morphism defined by

$$(f \triangleleft g) = \underline{(\curlywedge_A); f; (\curlywedge_B)}; g,$$

which is the string diagram in Figure 14.

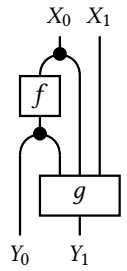


Figure 14. The morphism $(f \triangleleft g)$.

Proposition A.15. *Up to symmetries,*

$$(f \triangleleft g) \triangleleft h = f \triangleleft (g \triangleleft h).$$

We may simply write $(f \triangleleft g \triangleleft h)$ *for any of the two, omitting the symmetry.*

Proof. Using string diagrams (Figure 15). Note that $(\curlywedge)$ is coassociative and cocommutative. □

The Markov category Stoch also has *conditionals* [30], a property which we will use to prove the main result regarding stochastic processes.

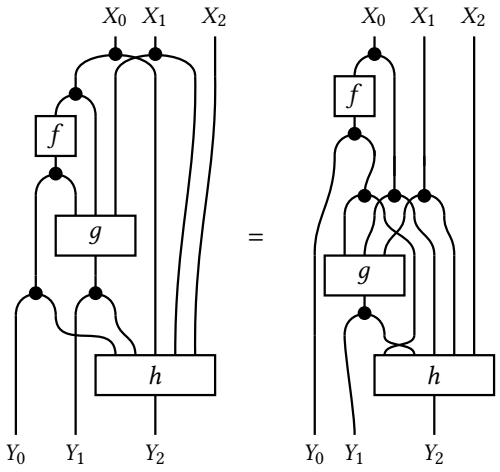


Figure 15. Associativity, up to symmetries, of the triangle operation.

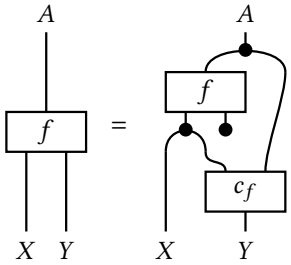


Figure 16. Condititionals in a Markov category.

Definition A.16 (Conditionals, [30, Definition 11.5])**.** Let C be a Markov category. We say that C has *conditionals* if for every morphism $f : A \to X \otimes Y$, writing $f_Y : A \to X$ for its first projection, there exists $c_f : X \otimes A \to Y$ such that $f = f_Y \triangleleft c_f$ (Figure 16).

Proposition A.17. *The Markov category* Stoch *has conditionals [30, Example 11.6].*

Proof. Let $f : A \to X \otimes Y$. If Y is empty, we are automatically done. If not, pick some $y_0 \in Y$, and define

$$c_f(y|x,a) = \begin{cases} f(x,y|a)/\sum_{x\in X} f(x,y|a) \\ \qquad \text{if } f(x,y|a) > 0 \text{ for some } x \in X, \\ (y = y_0) \quad \text{otherwise.} \end{cases}$$

It is straightforward to check that this does indeed define a distribution, and that it factors the original f as expected. □

Definition A.18 (Ranges)**.** In a Markov category, a *range* for a morphism $f : A \to B$ is a morphism $r_f : A \otimes B \to A \otimes B$ that

1. does not change its output $f \triangleleft \mathrm{id}_{A\otimes B} = f \triangleleft r_f$,
2. is *deterministic*, meaning $r_f ; \curlywedge_{A\otimes B} = \curlywedge_{A\otimes B} ; (r_f \otimes r_f)$,
3. and has the *range* property, $f \triangleleft g = f \triangleleft h$ must imply

$$(r_f \otimes \mathrm{id}) ; g = (r_f \otimes \mathrm{id}) ; h$$

for any suitably typed g and h.

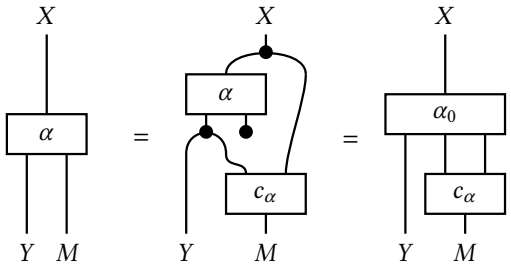


Figure 17. Productivity for Markov categories.

We say that a Markov category *has ranges* if there exists a range for each morphism of the category.

Remark A.19. There already exists a notion of categorical *range* in the literature, due to Cockett, Guo and Hofstra [20]. It arises in parallel to the notion of *support* in *restriction categories* [21]. The definition better suited for our purposes is different, even if it seems inspired by the same idea. The main difference is that we are using a *controlled range*; that is, the range of a morphism depends on the input to the original morphism. We keep the name hoping that it will not cause any confusion, as we do not deal explicitly with restriction categories in this text.

Proposition A.20. *The Markov category* Stoch *has ranges.*

Proof. Given $f : A \to B$, we know that for each $a \in A$ there exists some $b_a \in B$ such that $f(b_a|a) > 0$. We fix such $b_a \in B$, and we define $r_f : A \otimes B \to A \otimes B$ as

$$r_f(a,b) = \begin{cases} (a,b) & \text{if } f(b|a) > 0, \\ (a,b_a) & \text{if } f(b|a) = 0. \end{cases}$$

It is straightforward to check that it satisfies all the properties of ranges. □

Theorem A.21. *Any Markov category with conditionals and ranges is productive.*

Proof. Given any $\langle\alpha| \in \mathbf{Stage}_1(\mathbb{X}, \mathbb{Y})$, we can define

$$\alpha_0 = \curlywedge_A ; \underline{\alpha ; (\curlywedge_Y \otimes \downarrow_M)}.$$

This is indeed well-defined because of naturality of the discarding map $(\downarrow)_M : M \to I$ in any Markov category. Let $c_\alpha : Y \otimes X \to M$ be a conditional of α. This representative can then be factored as $\alpha = \alpha_0 ; \underline{c_\alpha}$ (Figure 17).

Now assume that for two representatives $\langle\alpha_i| = \langle\alpha_j|$ we have that $\langle\alpha_i ; u| = \langle\alpha_j ; v|$. By naturality of the discarding, $\alpha_i ; \underline{\varepsilon} = \alpha_j ; \underline{\varepsilon}$, and let r be a range of this map. Again by naturality of discarding, we have $\underline{\alpha_i} ; \underline{u} ; \underline{\downarrow} = \underline{\alpha_j} ; \underline{v} ; \underline{\downarrow}$. Let then c_i and c_j be conditionals of α_i and α_j: we have that $(\alpha_0 \triangleleft c_i) ; u ; \downarrow = (\alpha_0 \triangleleft c_j) ; v ; \downarrow$. By the properties of ranges (Figure 18), $(\alpha_0 \triangleleft r ; c_i) ; u ; \downarrow_{M(u)} = (\alpha_0 \triangleleft r ; c_j) ; v ; \downarrow_{M(v)}$, and thus, $r ; c_i ; u ; \downarrow_{M(u)} = r ; c_j ; v ; \downarrow_{M(v)}$. We pick $s_i = r ; c_i$ and we have proven that $\langle \underline{s_i} ; u| = \langle \underline{s_j} ; v|$. □

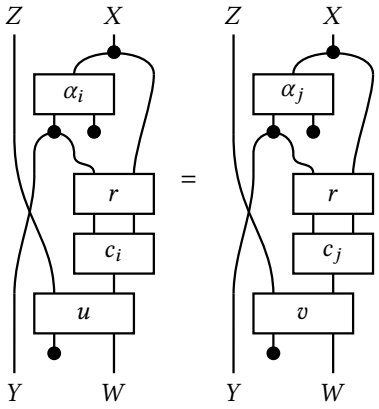


Figure 18. Applying the properties of range.

A.2 Stochastic processes

Definition A.22 (Controlled stochastic process)**.** Let $\mathbb{X} = (X_0, X_1, \dots)$ and $\mathbb{Y} = (Y_0, Y_1, \dots)$ be infinite sequences of sets. A controlled *stochastic process* $\mathbb{f} \colon \mathbb{X} \to \mathbb{Y}$ is an infinite sequence $\mathbb{f} = (f_0, f_1, \dots)$ of functions $f_n \colon X_n \times \cdots \times X_1 \to \mathbf{D}(Y_n \times \cdots \times Y_1)$ such that f_n coincides with the marginal distribution of f_{n+1} on the first n variables. In other words, $f_{n+1} \,;\, D\pi_{Y_0,\dots,Y_n} = \pi_{X_0,\dots,X_n} \,;\, f_n$.

$$\begin{array}{ccc} X_0 \times \cdots \times X_{n+1} & \xrightarrow{f_{n+1}} & D(Y_0 \times \cdots \times Y_{n+1}) \\ \downarrow{\scriptstyle \pi_{0,\dots,n}} & & \downarrow{\scriptstyle D\pi_{0,\dots,n}} \\ X_0 \times \cdots \times X_n & \xrightarrow{f_n} & D(Y_0 \times \cdots \times Y_n) \end{array}$$

Let StochProc be the category with objects infinite sequences of sets $\mathbb{X} = (X_0, X_1, \dots)$ and morphisms controlled stochastic processes $\mathbb{f} = (f_0, f_1, \dots)$ with composition and identities defined componentwise in Stoch.

Proposition A.23 (Factoring as conditionals)**.** *A stochastic process $f \colon \mathbb{X} \to \mathbb{Y}$ can be always written as*

$$f_n = c_0 \triangleleft c_1 \triangleleft \cdots \triangleleft c_n,$$

for some family of functions

$$c_n \colon Y_0 \times \cdots \times Y_{n-1} \times X_0 \times \cdots \times X_n \to Y_n,$$

called the conditionals *of the stochastic process.*

Proof. We proceed by induction, noting first that $c_0 = f_0$. In the general case, we apply conditionals to rewrite $f_{n+1} = (f_{n+1}; (\text{🗑})_{Y_{n+1}}) \triangleleft c_{n+1}$. Because of the marginalization property, we know that $f_{n+1}; (\text{🗑})_{Y_{n+1}} = f_n$. So finally, $f_{n+1} = f_n \triangleleft c_{n+1}$, which by the induction hypothesis gives the desired result. □

Proposition A.24. *If two families of conditionals give rise to the same stochastic process,*

$$c_0 \triangleleft c_1 \triangleleft \cdots \triangleleft c_n = c'_0 \triangleleft c'_1 \triangleleft \cdots \triangleleft c'_n,$$

then, they also give rise to the same n-stage processes in Stoch*,*

$$\langle c_0 \triangleleft \mathrm{id} | c_1 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle = \langle c'_0 \triangleleft \mathrm{id} | c'_1 \triangleleft \mathrm{id} | \dots | c'_n \triangleleft \mathrm{id} | \rangle .$$

Proof. We start by defining a family of morphisms r_n by induction. We take $r_0 = \mathrm{id}$ and r_{n+1} to be a *range* of $r_n; \text{🖧}; c_n$.

Let us prove now that for any $n \in \mathbb{N}$ and $i \leqslant n$,

$$r_i; \text{🖧}; c_i \triangleleft \cdots \triangleleft c_n = r_i; \text{🖧}; c'_i \triangleleft \cdots \triangleleft c'_n.$$

We proceed by induction. Observing that $c_0 = c'_0$, we prove it for $n = 0$ and also for the case $i = 0$ for any $n \in \mathbb{N}$. Assume we have it proven for n, so in particular we know that $r_i; \text{🖧}; c_i = r_i; \text{🖧}; c'_i$ for any $i \leqslant n$. Now, by induction on i, we can use the properties of ranges to show that

$$\begin{aligned} r_i; \text{🖧}; c_i \triangleleft \cdots \triangleleft c_n &= r_i; \text{🖧}; c'_i \triangleleft \cdots \triangleleft c'_n \\ (r_i; \text{🖧}; c_i \triangleleft \mathrm{id}); c_{i+1} \triangleleft \cdots \triangleleft c_n &= (r_i; \text{🖧}; c'_i \triangleleft \mathrm{id}); c'_{i+1} \triangleleft \cdots \triangleleft c'_n \\ (r_i; \text{🖧}; c_i \triangleleft r_{i+1}); c_{i+1} \triangleleft \cdots \triangleleft c_n &= (r_i; \text{🖧}; c'_i \triangleleft r_{i+1}); c'_{i+1} \triangleleft \cdots \triangleleft c'_n \\ r_{i+1}; \text{🖧}; c_{i+1} \triangleleft \cdots \triangleleft c_n &= r_{i+1}; \text{🖧}; c'_{i+1} \triangleleft \cdots \triangleleft c'_n. \end{aligned}$$

In particular, $r_n; \text{🖧}; c_n = r_n; \text{🖧}; c'_n$.

Now, we claim the following for each $n \in \mathbb{N}$ and each $i \leqslant n$,

$$\begin{aligned} &\langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle = \\ &\langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i(c_i \triangleleft \mathrm{id}) | c_{i+1} | \dots | c_n \triangleleft \mathrm{id} | \rangle . \end{aligned}$$

It is clear for $n = 0$ and for $i = 0$. In the inductive case for i,

$$\begin{aligned} \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i(c_i \triangleleft \mathrm{id}) | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle &= \\ \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i; \text{🖧}; c_i \triangleleft \mathrm{id} | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle &= \\ \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i; \text{🖧}; c_i \triangleleft r_{i+1} | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle &= \\ \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i; \text{🖧}; c_i \triangleleft \mathrm{id} | r_{i+1}(c_{i+1} \triangleleft \mathrm{id}) | \dots | c_n \triangleleft \mathrm{id} | \rangle &= \\ \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_i(c_i \triangleleft \mathrm{id}) | r_{i+1}(c_{i+1} \triangleleft \mathrm{id}) | \dots | c_n \triangleleft \mathrm{id} | \rangle . \end{aligned}$$

A particular case of this claim is then that

$$\begin{aligned} \langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | \rangle &= \\ \langle r_0(c_0 \triangleleft \mathrm{id}) | \dots | r_n(c_n \triangleleft \mathrm{id}) | \rangle &= \\ \langle r_0(c'_0 \triangleleft \mathrm{id}) | \dots | r_n(c'_n \triangleleft \mathrm{id}) | \rangle &= \\ \langle c'_0 \triangleleft \mathrm{id} | \dots | c'_n \triangleleft \mathrm{id} | \rangle . \end{aligned}$$

This can be then proven for any $n \in \mathbb{N}$. □

Corollary A.25. *Any stochastic process $f \in \mathrm{StochProc}(\mathbb{X}, \mathbb{Y})$ with a family of conditionals c_n gives rise to the observational sequence*

$$\begin{aligned} \mathrm{obs}(f) = [\langle (c_n \triangleleft \mathrm{id}) \colon (X_0 \times Y_0 \times \cdots \times X_{n-1} \times Y_{n-1}) \times X_n \to \\ (X_0 \times Y_0 \times \cdots \times X_n \times Y_n) \times Y_n \rangle]_{\approx}, \end{aligned}$$

which is independent of the chosen family of conditionals.

Proof. Any two families of conditionals for f give rise to the same n-stage processes in Stoch (by Proposition A.24). Being a productive category, observational sequences are determined by their n-stage processes. □

Proposition A.26. *An observational sequence in* Stoch*,*

$$[\langle g_n \colon M_{n-1} \otimes X_n \to M_n \otimes Y_n \rangle]_{\approx} \in \mathbf{Obs}(\mathbb{X}, \mathbb{Y})$$

gives rise to a stochastic process $\mathrm{proc}(g) \in \mathrm{StochProc}(\mathbb{X}, \mathbb{Y})$ *defined by* $\mathrm{proc}(g)_n = \underline{g_0}; \underline{g_1}; \dots; \underline{g_n}; \underline{\varepsilon_{M_n}}$.

Proof. The symmetric monoidal category Stoch is productive: by Lemma D.3, observational sequences are determined by their n-stage truncations

$$\langle g_0| \dots |g_n| \in \mathbf{Stage}_n(\mathbb{X}, \mathbb{Y}).$$

Each n-stage truncation gives rise to the n-th component of the stochastic process, $\mathrm{proc}(g)_n = \underline{g_0}; \underline{g_1}; \dots; \underline{g_n}; \underline{\varepsilon_{M_n}}$, and this is well-defined: composing the morphisms is invariant to *sliding equivalence*, and the last discarding map is natural.

It only remains to show that they satisfy the marginalisation property. Indeed,

$$\begin{aligned}\mathrm{proc}(g)_{n+1}; \underline{\varepsilon_{n+1}} &= \underline{g_0}; \underline{g_1}; \dots; \underline{g_{n+1}}; \underline{\varepsilon_{M_{n+1}}}; \underline{\varepsilon_{Y_{n+1}}} \\ &= \underline{g_0}; \underline{g_1}; \dots; \underline{g_n}; \underline{\varepsilon_{M_n}} \\ &= \mathrm{proc}(g)_n.\end{aligned}$$

Thus, $\mathrm{proc}(g)$ is a stochastic process in StochProc$(\mathbb{X}, \mathbb{Y})$. □

Proposition A.27. *Let $f \in$ StochProc$(\mathbb{X}, \mathbb{Y})$, we have that $\mathrm{proc}(\mathrm{obs}(f)) = f$.*

Proof. Indeed, for c_n some family of conditionals,

$$f_n = c_0 \triangleleft \dots \triangleleft c_n = (c_0 \triangleleft \mathrm{id}); \dots; (c_n \triangleleft \mathrm{id}); \varepsilon_{M_n}.$$ □

Theorem A.28. *Observational sequences in Stoch are in bijection with stochastic processes.*

Proof. The function obs is injective by Proposition A.27. We only need to show it is also surjective.

We will prove that any n-stage process $\langle g_0| \dots |g_n|$ can be equivalently written in the form $\langle (c_0 \triangleleft \mathrm{id})| \dots |(c_n \triangleleft \mathrm{id})|$. We proceed by induction. Given any $\langle g_0|$ we use conditionals and dinaturality to rewrite it as

$$\langle g_0| = \langle c_0 \triangleleft c_M| = \langle c_0 \triangleleft \mathrm{id}| .$$

Given any $\langle c_0 \triangleleft \mathrm{id}| \dots |c_n \triangleleft \mathrm{id}|g_{n+1}|$, we use again conditionals and dinaturality to rewrite it as

$$\begin{aligned}\langle c_0 \triangleleft \mathrm{id}| \dots |c_n \triangleleft \mathrm{id}|g_{n+1}| = \\ \langle c_0 \triangleleft \mathrm{id}| \dots |c_n \triangleleft \mathrm{id}|c_{n+1} \triangleleft c_M| = \\ \langle c_0 \triangleleft \mathrm{id}| \dots |c_n \triangleleft \mathrm{id}|c_{n+1} \triangleleft \mathrm{id}| .\end{aligned}$$

We have shown that obs is both injective and surjective. □

Theorem A.29 (From Theorem 7.2)**.** *The category Stoch of stochastic processes is monoidally isomorphic to the category Stream over Stoch.*

Proof. We have shown in Theorem A.28 that proc is a bijection. Let us show that it preserves compositions. Indeed,

$$\begin{aligned}\mathrm{proc}(g;h)_n &= \underline{g_0}; \underline{h_0}; \dots; \underline{g_n}; \underline{h_n}; \underline{\varepsilon_{M_n \otimes N_n}} \\ &= \underline{g_0}; \dots; \underline{g_n}; \underline{h_0}; \dots; \underline{h_n}; \underline{(\varepsilon_{M_n} \otimes \varepsilon_{N_n})} \\ &= \underline{g_0 \dots g_n}; \underline{\varepsilon_{M_n}}; \underline{h_0 \dots h_n}; \underline{\varepsilon_{N_n}} \\ &= \mathrm{proc}(g)_n; \mathrm{proc}(h)_n.\end{aligned}$$

It also trivially preserves the identity. It induces thus an identity-on-objects functor which is moreover an equivalence of categories. Let us finally show that it preserves tensoring of morphisms.

$$\begin{aligned}\mathrm{proc}(g \otimes h)_n &= \underline{(g_0 \otimes h_0)}; \dots; \underline{(g_n \otimes h_n)}; \underline{\varepsilon_{M_n \otimes N_n}} \\ &= \underline{(g_0 \otimes h_0)}; \dots \underline{((g_n \varepsilon_{M_n}) \otimes (h_n \varepsilon_{N_n}))} \\ &= \underline{(g_0 \dots g_n \varepsilon_{M_n})} \otimes \underline{(h_0 \dots h_n \varepsilon_{N_n})} \\ &= \mathrm{proc}(g)_n \otimes \mathrm{proc}(h)_n.\end{aligned}$$

It is thus also a monoidal equivalence. □

B Coend Calculus and Profunctors

Coend calculus is the name given to the a branch of category theory that describes the behaviour of certain colimits called *coends*. MacLane [55] and Loregian [53] give complete presentations of coend calculus.

Definition B.1. *Coends* are the coequalizers of the action of morphisms on both arguments of a profunctor.

$$\mathrm{coend}(P) := \mathrm{coeq}\left(\coprod_{f\colon B \to A} P(A,B) \rightrightarrows \coprod_{X \in \mathbf{C}} P(X,X) \right).$$

Coends are usually denoted with a superscripted integral, drawing on an analogy with classical calculus.

$$\int^{X \in \mathbf{C}} P(X,X) = \mathrm{coend}(P).$$

Proposition B.2 (Yoneda reduction)**.** *Let* C *be any category and let $F\colon \mathbf{C} \to \mathbf{Set}$ be a functor; the following isomorphism holds for any given object $A \in \mathbf{C}$.*

$$\int^{X \in \mathbf{C}} \hom(X, A) \times FX \cong FA.$$

Following the analogy with classical analysis, the hom *profunctor works as a Dirac's delta.*

Proposition B.3 (Fubini rule)**.** *Coends commute between them; that is, there exists a natural isomorphism*

$$\int^{X_1 \in \mathbf{C}} \int^{X_2 \in \mathbf{C}} P(X_1, X_2, X_1, X_2) \cong \int^{X_2 \in \mathbf{C}} \int^{X_1 \in \mathbf{C}} P(X_1, X_2, X_1, X_2).$$

In fact, they are both isomorphic to the coend over the product category,

$$\int^{(X_1, X_2) \in \mathbf{C} \times \mathbf{C}} P(X_1, X_2, X_1, X_2).$$

Following the analogy with classical analysis, coends follow the Fubini rule for integrals.

A profunctor from a category A to a category B is a functor $P\colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$. They can be seen as a categorification of the concept of *relations*, functions $A \times B \to 2$. Under this analogy, existential quantifiers correspond to *coends*. The

canonical example of a profunctor is, $\hom\colon \mathrm{A}^{op} \times \mathrm{A} \to \mathrm{Set}$, the profunctor that returns the set of morphisms between two objects. Many operations relating families of processes are more easily defined in terms of profunctors: for instance, sequential composition connects the outputs of a family of processes to the outputs of another family.

Definition B.4 (Sequential composition)**.** Two profunctors $P\colon \mathrm{A}^{op}\times\mathrm{B} \to \mathrm{Set}$ and $Q\colon \mathrm{B}^{op}\times\mathrm{C} \to \mathrm{Set}$ compose sequentially into a profunctor $P \diamond Q\colon \mathrm{A}^{op} \times \mathrm{C} \to \mathrm{Set}$ defined by

$$(P \diamond Q)(A, C) = \int^{B\in\mathrm{B}} P(A, B) \times Q(B, C).$$

The hom-profunctor $\hom\colon \mathrm{A}^{op}\times\mathrm{A} \to \mathrm{Set}$ that returns the set of morphisms between two objects is the unit for sequential composition. Sequential composition is associative up to isomorphism.

Definition B.5 (Parallel composition)**.** Two profunctors $P\colon \mathrm{A}_1^{op} \times \mathrm{B}_1 \to \mathrm{Set}$ and $Q\colon \mathrm{A}_2^{op} \times \mathrm{B}_2 \to \mathrm{Set}$ compose *in parallel* into a profunctor $P \times Q\colon \mathrm{A}_1^{op} \times \mathrm{A}_2^{op} \times \mathrm{B}_1 \times \mathrm{B}_2 \to \mathrm{Set}$ defined by

$$(P \times Q)(A, A', B, B') = P(A, B) \times Q(A', B').$$

Definition B.6 (Intensional communicating composition)**.** Let A, B, C be categories and let B have a monoidal structure. Let $P\colon \mathrm{A}^{op} \times \mathrm{B} \to \mathrm{Set}$ and $Q\colon \mathrm{B}^{op} \times \mathrm{C}^{\mathbb{N}} \to \mathrm{Set}$ be a pair of profunctors. Their *intensional communicating composition* is the profunctor $P \boxdot Q\colon \mathrm{A}^{op} \times \mathrm{B}^{op} \times \mathrm{B} \times \mathrm{C} \to \mathrm{Set}$ defined as

$$(P \boxdot Q)(A, B; B', C) = \sum_{M\in\mathrm{B}} P(A, B \otimes M) \times Q(M \otimes B', C).$$

Remark B.7. Let C be a monoidal category and let $P\colon \mathrm{C}^{op} \times \mathrm{C} \to \mathrm{Set}$ and $Q\colon [\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}] \to \mathrm{Set}$ be a pair of profunctors. Note that $[\mathbb{N}, \mathrm{C}] \cong \mathrm{C} \times [\mathbb{N}, \mathrm{C}]$, and so the second profunctor can be interpreted as having type $Q\colon \mathrm{C}^{op} \times ([\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}]) \to \mathrm{Set}$. In this case, their intensional communicating composition is defined by

$$(P \boxdot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \sum_{M\in\mathrm{C}} P(X_0, M \otimes Y_0) \times Q(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

This is the composition we use when we describe the endofunctor $(\hom \boxdot \bullet)\colon [[\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}], \mathrm{Set}] \to [[\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}], \mathrm{Set}]$.

$$(\hom \boxdot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \sum_{M\in\mathrm{C}} \hom(X_0, M \otimes Y_0) \times Q(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Definition B.8 (Communicating profunctor composition)**.** Let A, B, C be categories and let B have a monoidal structure. Two profunctors $P\colon \mathrm{A}^{op} \times \mathrm{B} \to \mathrm{Set}$ and $Q\colon \mathrm{B}^{op} \times \mathrm{C} \to \mathrm{Set}$ compose communicating along B into the profunctor $(P \odot Q)\colon \mathrm{A}^{op} \times \mathrm{B} \times \mathrm{B}^{op} \times \mathrm{C} \to \mathrm{Set}$ defined by

$$(P \odot Q)(A, B; B', C) = \int^{M} P(A, B \otimes M) \times Q(M \otimes B', C).$$

The profunctors $\hom(I, \bullet)\colon \mathrm{B} \to \mathrm{Set}$ and $\hom(\bullet, I)\colon \mathrm{B}^{op} \to \mathrm{Set}$ are left and right units with respect to communicating composition. The communicating composition of three profunctors $P\colon \mathrm{A}^{op} \times \mathrm{B} \to \mathrm{Set}$, $Q\colon \mathrm{B}^{op} \times \mathrm{C} \to \mathrm{Set}$ and $R\colon \mathrm{C}^{op} \times \mathrm{D} \to \mathrm{Set}$ is associative up to isomorphism and a representative can be written simply by $(P \odot Q \odot R)\colon \mathrm{A}^{op} \times \mathrm{B} \times \mathrm{B}^{op} \times \mathrm{C} \times \mathrm{C}^{op} \times \mathrm{D} \to \mathrm{Set}$, where both B and C are assumed to have a monoidal structure.

Remark B.9. This is the composition we use when we describe the endofunctor $(\hom \odot \bullet)\colon [[\mathbb{N}, \mathrm{C}]^{op}\times[\mathbb{N}, \mathrm{C}], \mathrm{Set}] \to [[\mathbb{N}, \mathrm{C}]^{op} \times [\mathbb{N}, \mathrm{C}], \mathrm{Set}]$.

$$(\hom \odot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \int^{M\in\mathrm{C}} \hom(X_0, M\otimes Y_0)\times Q(M\cdot\mathbb{X}^+, \mathbb{Y}^+).$$

B.1 Initial algebras, final coalgebras

Definition B.10 (Algebras and coalgebras)**.** Let C be a category and let $F\colon \mathrm{C} \to \mathrm{C}$ be an endofunctor. An *algebra* (X, α) is an object $X \in \mathrm{C}$, together with a morphism $\alpha\colon FX \to X$. A *coalgebra* (Y, β) is an object $Y \in \mathrm{C}$, together with a morphism $\beta\colon Y \to FY$.

An *algebra morphism* $f\colon (X, \alpha) \to (X', \alpha')$ is a morphism $f\colon X \to X'$ such that the diagram on the left commutes. A *coalgebra morphism* $g\colon (Y, \beta) \to (Y', \beta')$ is a morphism $f\colon Y \to Y'$ such that the diagram on the right commutes.

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FX' \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha'} \\ X & \xrightarrow{f} & X' \end{array} \qquad \begin{array}{ccc} Y & \xrightarrow{g} & Y' \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle\beta'} \\ FY & \xrightarrow{Fg} & FY' \end{array}$$

Algebras for an endofunctor form a category with algebra morphisms between them. The initial algebra is the initial object in this category. Coalgebras for an endofunctor form a category with coalgebra morphisms between them. The final coalgebra is the terminal object in this category.

Definition B.11 (Fixpoints of an endofunctor)**.** Let C be a category and let $F\colon \mathrm{C} \to \mathrm{C}$ be an endofunctor. A *fixpoint* is an algebra (X, α) such that $\alpha\colon FX \to X$ is an isomorphism. Equivalently, a fixpoint is a coalgebra (Y, β) such that $\beta\colon Y \to FY$ is an isomorphism.

Fixpoints form a category with algebra morphisms (or, equivalently, coalgebra morphisms) between them.

Theorem B.12 (Lambek, [50])**.** *The final coalgebra of a functor is a fixpoint. As a consequence, when it exists, it is the final fixpoint.*

Theorem B.13 (Adamek, [4])**.** *Let* D *be a category with a final object* 1 *and* ω*-shaped limits. Let* $F\colon \mathrm{D} \to \mathrm{D}$ *be an endofunctor. We write* $L = \lim_n F^n 1$ *for the limit of the following* ω*-chain, which is called the* terminal sequence*.*

$$1 \xleftarrow{!} F1 \xleftarrow{F!} FF1 \xleftarrow{FF!} FFF1 \xleftarrow{FFF!} \dots$$

Assume that F preserves this limit, meaning that the canonical morphism $FL \to L$ is an isomorphism. Then, L is the final F-coalgebra.

B.2 Size concerns, limits and colimits

Remark B.14. We call Set to the category of sets and functions below a certain Grothendieck universe. We do take colimits (and coends) over this category without creating size issues: we can be sure of their existence in our metatheoretic category of sets.

Proposition B.15. *Terminal coalgebras exist in* Set. *More generally, the category of sets below a certain regular uncountable cardinal is algebraically complete and cocomplete; meaning that every* Set*-endofunctor has a terminal coalgebra and an initial algebra. See [5, Theorem 13].*

Theorem B.16 (Coproducts commute with connected limits)**.** *Let I be a set, understood as a discrete category, and let* A *be a connected category with $F : I \times \mathsf{A} \to \mathrm{Set}$ a functor. The canonical morphism*

$$\sum_{i \in I} \lim_{a \in A} F(i,a) \to \lim_{a \in A} \sum_{i \in I} F(i,a)$$

is an isomorphism.

In particular, let $F_n : I \to \mathrm{Set}$ be a family of functors indexed by the natural numbers with a family of natural transformations $\alpha_n : F_{n+1} \to F_n$. The canonical morphism

$$\sum_{i \in I} \lim_{n \in \mathbb{N}} F_n(i) \to \lim_{n \in \mathbb{N}} \sum_{i \in I} F_n(i)$$

is an isomorphism.

Proof. Note that there are no morphisms between any two indices $i, j \in I$. Once some $i \in I$ is chosen in any factor of the connected limit, it forces any other factor to also choose $i \in I$. This makes the local choice of $i \in I$ be equivalent to the global choice of $i \in I$. □

C The List$^+$ opmonoidal comonad and Fox's theorem

Proposition C.1. *Cartesian monoidal categories are productive.*

Proof. Let $\langle \alpha | \in \mathbf{Stage}_1(\mathbb{X}, \mathbb{Y})$. For some given representative $\alpha : X_0 \to M \otimes Y_0$, we define the two projections $\alpha_Y = \alpha; \underline{\blacklozenge_M} : X_0 \to Y_0$ and $\alpha_M = \alpha; \underline{\blacklozenge_Y}$. The second projection α_M depends on the specific representative α we have chosen; however, the first projection α_Y is defined independently of the specific representative α, as a consequence of naturality of the discarding map (see Fox's theorem for cartesian monoidal categories Theorem C.5). We define $\alpha_0 = \delta_{X_0}; \underline{\alpha_Y}$. Then, we can factor any representative as $\alpha = \alpha_0; \underline{\alpha_M}$ (see Figure 19). Now, assume that we have two representatives $\langle \alpha_i | = \langle \alpha_j |$ for which $\langle \alpha_i; u | = \langle \alpha_j; v |$. By naturality of the discarding map, $\alpha_i; \underline{\blacklozenge} = \alpha_j; \underline{\blacklozenge}$, and we call this map α_Y. Again

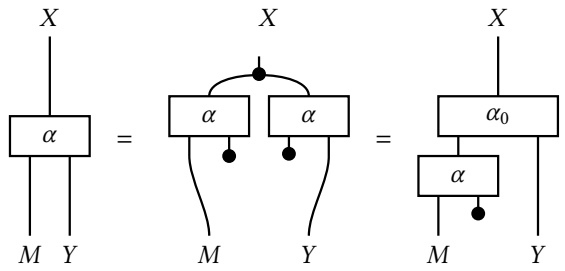


Figure 19. Productivity for cartesian categories.

by naturality of the discarding map, $\alpha_i; u; \underline{\blacklozenge} = \alpha_j; v; \underline{\blacklozenge}$, and discarding the output in Y, we get that $\alpha_{M,i}; u; \underline{\blacklozenge} = \alpha_{M,j}; v; \underline{\blacklozenge}$, which implies $\langle \alpha_{M,i}; u | = \langle \alpha_{M,j}; v |$. □

Definition C.2 (Opmonoidal comonad)**.** In a monoidal category $(\mathsf{C}, \otimes, I)$, a comonad (R, ε, δ) is an *opmonoidal comonad* when the endofunctor $R : \mathsf{C} \to \mathsf{C}$ is oplax monoidal with laxators $\psi_{X,Y} : R(X \otimes Y) \to RX \otimes RY$ and $\psi^I : RI \to I$, and both the counit $\varepsilon_X : RX \to X$ and the comultiplication $\delta_X : RX \to RRX$ are monoidal natural transformations.

Explicitly, $\varepsilon_I = \psi_0$, $\varepsilon_{X \otimes Y} = \psi_{X,Y}; (\varepsilon_X \otimes \varepsilon_Y)$, $\delta_I; \psi_0; \psi_0 = \psi_0$ and $\delta_{X \otimes Y}; \psi_{X,Y}; \psi_{RX,RY} = \psi_{X,Y}; (\delta_X \otimes \delta_Y)$.

Alternatively, an *opmonoidal comonad* is a comonoid in the bicategory **MonOplax** of oplax monoidal functors with composition and monoidal natural transformations between them.

Definition C.3. Let $(\mathsf{C}, \otimes, I)$ be a symetric monoidal category. There is a functor $\mathbf{List}^+ : \mathsf{C} \to \mathsf{C}$ defined on objects by

$$\mathbf{List}^+(X)_n := \bigotimes_{i=0}^{n} X_i.$$

This functor is monoidal, with oplaxators $\psi_0^+ : \mathbf{List}^+(I) \to I$ and $\psi_{X,Y} : \mathbf{List}^+(X \otimes Y) \to \mathbf{List}^+(X) \otimes \mathbf{List}^+(Y)$ given by symmetries, associators and unitors.

Theorem C.4 (From Theorem 6.1)**.** *The opmonoidal functor* $\mathbf{List}^+$ *has an opmonoidal comonad structure if and only if its base monoidal category $(\mathsf{C}, \otimes, I)$ is cartesian monoidal.*

Proof. When C is cartesian, we can construct the comonad structure using projections $\prod_{i=0}^{n} X_i \to X_n$ and copying together with braidings $\prod_{i=0}^{n} X_i \to \prod_{i=0}^{n} \prod_{k=0}^{i} X_k$. These are monoidal natural transformations making $\mathbf{List}^+$ a monoidal comonad.

Suppose (L, ε, δ) is an opmonoidal comonad structure. This means it has families of natural transformations

$$\delta_n : \bigotimes_{i=0}^{n} X_i \to \bigotimes_{i=0}^{n} \bigotimes_{k=0}^{i} X_k \text{ and } \varepsilon_n : \bigotimes_{i=0}^{n} X_i \to X_n.$$

We will use these to construct a uniform counital comagma structure on every object of the category. By a refined version of Fox's theorem (Theorem C.7), this will imply that C is cartesian monoidal.

Let $X \in \mathrm{C}$ be any object. Choosing $n = 2$, $X_0 = X$ and $X_1 = I$; and using coherence maps, we get $\delta_2 \colon X \to X \otimes X$ and $\varepsilon_2 \colon X \to I$. These are coassociative, counital, natural and uniform because the corresponding transformations δ and ε are themselves coassociative, counital, natural and monoidal. This induces a uniform comagma structure in every object $(X, \delta_2, \varepsilon_2)$; with this structure, every morphism of the category is a comagma homomorphism because δ_2 and ε_2 are natural. □

Theorem C.5 (Fox's theorem [29])**.** *A symmetric monoidal category* $(\mathrm{C}, \otimes, I)$ *is cartesian monoidal if and only if every object* $X \in \mathrm{C}$ *has a cocommutative comonoid structure* $(X, \varepsilon_X, \delta_X)$*, every morphism of the category* $f \colon X \to Y$ *is a comonoid homomorphism, and this structure is uniform across the monoidal category: meaning that* $\varepsilon_{X\otimes Y} = \varepsilon_X \otimes \varepsilon_Y$*, that* $\varepsilon_I = \mathrm{id}$*, that* $\delta_I = \mathrm{id}$ *and that* $\delta_{X\otimes Y} = (\delta_X \otimes \delta_Y); (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.

Remark C.6. Most sources ask the comonoid structure in Fox's theorem (Theorem C.5) to be cocommutative [28, 29]. However, cocommutativity and coassociativity of the comonoid structure are implied by the fact that the structure is uniform and natural. We present an original refined version of Fox's theorem.

Theorem C.7 (Refined Fox's theorem)**.** *A symmetric monoidal category* $(\mathrm{C}, \otimes, I)$ *is cartesian monoidal if and only if every object* $X \in \mathrm{C}$ *has a counital comagma structure* $(X, \varepsilon_X, \delta_X)$*, or* $(X, \swarrow_X, \curvearrowright_X)$*, every morphism of the category* $f \colon X \to Y$ *is a comagma homomorphism, and this structure is uniform across the monoidal category: meaning that* $\varepsilon_{X\otimes Y} = \varepsilon_X \otimes \varepsilon_Y$*,* $\varepsilon_I = \mathrm{id}$*,* $\delta_I = \mathrm{id}$ *and* $\delta_{X\otimes Y} = (\delta_X \otimes \delta_Y); (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.

Proof. We prove that such a comagma structure is necessarily coassociative and cocommutative. Note that any comagma homomorphism $f \colon A \to B$ must satisfy $\delta_A; (f \otimes f) = f; \delta_B$. In particular, $\delta_X \colon X \to X \otimes X$ must itself be a comagma homomorphism (see Figure 20), meaning that

$$\delta_X; (\delta_X \otimes \delta_X) = \delta_X; \delta_{X\otimes X} = \delta_X; (\delta_X \otimes \delta_X); (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id}), \tag{3}$$

where the second equality follows by uniformity.

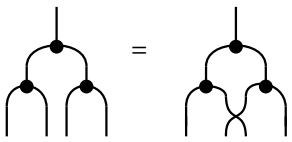


Figure 20. Comultiplication is a comagma homomorphism.

Now, we prove cocommutativity (Figure 21): composing both sides of Equation (3) with $(\epsilon_X \otimes \mathrm{id} \otimes \mathrm{id} \otimes \epsilon_X)$ discards the two external outputs and gives $\delta_X = \delta_X; \sigma_X$.

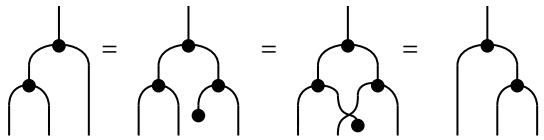


Figure 22. Coassociativity

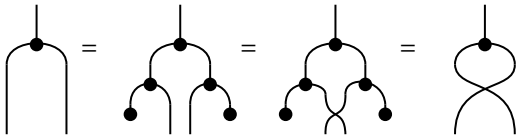


Figure 21. Cocommutativity

Now, we prove coassociativity (Figure 22): composing both sides of Equation (3) with $(\mathrm{id} \otimes \epsilon_X \otimes \mathrm{id} \otimes \mathrm{id})$ discards one of the middle outputs and gives $\delta_X; (\mathrm{id} \otimes \delta_X) = \delta_X; (\delta_X \otimes \mathrm{id})$.

A coassociative and cocommutative comagma is a cocommutative comonoid. We can then apply the classical form of Fox's theorem (Theorem C.5). □

Distributive laws. One could hope to add effects such as probability or non-determinism to set-based streams via the bikleisli category arising from a monad-comonad distributive law $\mathbf{List}^+ \circ \mathbb{T} \Rightarrow \mathbb{T} \circ \mathbf{List}^+$ [7, 64], as proposed by Uustalu and Vene [75]. This would correspond to a lifting of the $\mathbf{List}^+$ comonad to the kleisli category of some commutative monad T; the arrows $\mathbb{X} \to \mathbb{Y}$ of such a category would look as follows,

$$f_n \colon X_1 \times \cdots \times X_n \to TY_n.$$

However, we have already shown that this will not result in a monoidal comonad whenever $\mathsf{kl}(T)$ is not cartesian. To see explicitly what fails, we use the string diagrams to show how composition should work for the case $n = 2$ (Figure 23). This composition is not associative or unital whenever the kleisli category does not have natural comultiplications or counits, respectively (Figures 24 and 25).

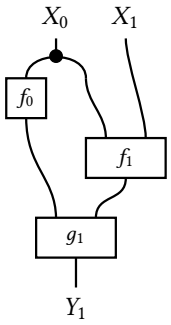


Figure 23. Composition in the case $n = 2$.

D Productive categories

Definition D.1 (Truncating coherently)**.** Let $f_n^k \colon X_n \otimes M_{n-1} \to Y_n \otimes M_n$ be a family of families of morphisms of increasing length, indexed by $k \in \mathbb{N}$ and $n \leqslant k$. We say that this family

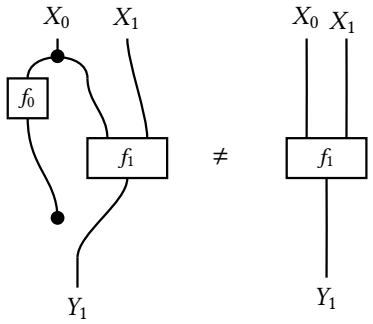


Figure 24. Failure of unitality if discarding is not natural, as it happens, for instance, with partial functions.

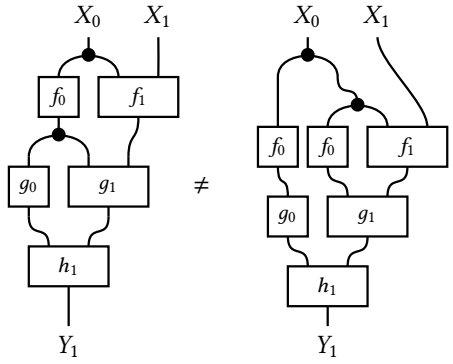


Figure 25. Failure of associativity if copying is not natural, as it happens, for instance, with stochastic functions.

truncates coherently if $\langle f_0^p| \dots |f_n^p| = \langle f_0^q| \dots |f_n^q|$ for each $p, q \in \mathbb{N}$ and each $n \leqslant \min\{p, q\}$.

Lemma D.2 (Factoring a family of processes)**.** *In a productive category, let $(\langle f_0^k| \dots |f_k^k|)_{k\in\mathbb{N}}$ be a sequence of sequences that truncates coherently. Then, there exists a sequence h_i with $s_{i-1}^k f_i^k = h_i s_i^k$ such that, for each $k \in \mathbb{N}$ and each $n \leqslant k$, $\langle f_0^k| \dots |f_n^k| = \langle h_0| \dots |h_n|$. Moreover, this family h_i is such that $\langle h_0 \dots h_n s_n^p u| = \langle h_0 \dots h_n s_n^q v|$ implies $\langle s_n^p u| = \langle s_n^q v|$.*

Proof. We construct the family by induction. In the case $n = 0$, we use that the family truncates coherently to have that $\langle f_0^p| = \langle f_0^q|$ and thus, by productivity, create an h_0 with $f_0^k = h_0 s_0^k$ such that $\langle h_0 s_0^p u| = \langle h_0 s_0^q v|$ implies $\langle s_0^p u| = \langle s_0^q v|$.

In the general case, assume we already have constructed $h_0, \dots, h_{n-1}$ with $s_{i-1}^k f_i^k = h_i s_i^k$ such that, for each $k \in \mathbb{N}$ and $\langle f_0^k| \dots |f_{n-1}^k| = \langle h_0| \dots |h_{n-1}|$. Moreover, $\langle h_0 \dots h_{n-1} s_{n-1}^p u| = \langle h_0 \dots h_{n-1} s_{n-1}^q v|$ implies $\langle s_{n-1}^p u| = \langle s_{n-1}^q v|$.

In this case, we use the fact that composition "along a bar" is dinatural: $\langle f_0^p| \dots |f_n^p| = \langle f_0^q| \dots |f_n^q|$ implies that $\langle \underline{f_0^p \dots f_n^p}| = \langle \underline{f_0^q \dots f_n^q}|$. This can be then rewritten as

$$\langle h_0 \dots h_{n-1} s_{n-1}^p f_n^p| = \langle h_0 \dots h_{n-1} s_{n-1}^q f_n^q|,$$

which in turn implies $\langle s_{n-1}^p f_n^p| = \langle s_{n-1}^q f_n^q|$. By productivity, there exists h_n with $s_{n-1}^k f_n^k = h_n s_n^k$ such that $\langle f_0^k| \dots |f_n^k| = \langle h_0| \dots |h_n|$.

Finally, assume that $\langle h_0 \dots h_{n-1} h_n s_n^p u| = \langle h_0 \dots h_{n-1} h_n s_n^q v|$. Thus, we have $\langle h_0 \dots h_{n-1} s_{n-1}^p f_n^p u| = \langle h_0 \dots h_{n-1} s_{n-1}^q f_n^q v|$ and $\langle s_{n-1}^p f_n^p u| = \langle s_{n-1}^q f_n^q v|$. This can be rewritten as $\langle h_n s_n^p u| = \langle h_n s_n^q v|$, which in turn implies $\langle s_n^p u| = \langle s_n^q v|$. We have shown that the h_n that we constructed satisfies the desired property. □

Lemma D.3. *In a productive category, the set of observational sequencesis isomorphic to the limit of the terminal sequence of the endofunctor* (hom ⊙ •) *via the canonical map between them.*

Proof. We start by noting that observational equivalence of sequences is, by definition, the same thing as being equal under the canonical map to the limit of the terminal sequence

$$\lim_n \int^{M_0,\dots,M_n} \prod_{i=0}^{n} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

We will show that this canonical map is surjective. That means that the domain quotiented by equality under the map is isomorphic to the codomain, q.e.d.

Indeed, given any family f_n^k that truncates coherently, we can apply Lemma D.2 to find a sequence h_i such that $\langle f_0^k| \dots |f_n^k| = \langle h_0| \dots |h_n|$. This means that it is the image of the stateful sequence h_i. □

Lemma D.4 (Factoring two processes)**.** *In a productive category, let $\langle f_0| = \langle g_0|$. Then there exists h_0 with $f_0 = h_0 s_0$ and $g_0 = h_0 t_0$ such that*

$$\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$$

implies the existence of a family h_i together with s_i and t_i such that $s_{i-1} f_i = h_i s_i$ and $t_{i-1} g_i = h_i t_i$; and moreover, such that

$$\langle h_0 \dots h_n s_n u| = \langle h_0 \dots h_n t_n v| \text{ implies } \langle s_n u| = \langle t_n v|.$$

Proof. By productivity, we can find such a factorization $f_0 = h_0 s_0$ and $g_0 = h_0 t_0$.

Assume now that we have a family of morphisms such that $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$. We proceed by induction on n, the size of the family. The case $n = 0$ follows from the definition of productive category.

In the general case, we will construct the relevant h_n. The assumption $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$ implies, in particular, that $\langle f_0| \dots |f_{n-1}| = \langle g_0| \dots |g_{n-1}|$. Thus, by induction hypothesis, there exist $h_1, \dots, h_{n-1}$ together with $s_{i-1} f_i = h_i s_i$ and $t_{i-1} g_i = h_i t_i$, such that

$$\langle h_0 \dots h_n s_{n-1} u| = \langle h_0 \dots h_n t_{n-1} v| \text{ implies } \langle s_{n-1} u| = \langle t_{n-1} v|.$$

We know that $\langle f_0 \dots f_n| = \langle g_0 \dots g_n|$ and thus,

$$\langle h_0 \dots h_{n-1} s_{n-1} f_n| = \langle h_0 \dots h_{n-1} t_{n-1} g_n|,$$

which, by induction hypothesis, implies $\langle s_{n-1} f_n| = \langle t_{n-1} g_n|$. By productivity, there exists h_n with $s_{n-1} f_n = h_n s_n$ and $t_{n-1} g_n = h_n t_n$ such that $\langle h_n s_n u| = \langle h_n t_n v|$ implies $\langle s_n u| = \langle t_n v|$.

Finally, assume that $\langle h_0 \dots h_{n-1} h_n s_n u| = \langle h_0 \dots h_{n-1} h_n t_n v|$. Thus, we have $\langle h_0 \dots h_{n-1} s_{n-1} f_n u| = \langle h_0 \dots h_{n-1} t_{n-1} g_n v|$ and $\langle s_{n-1} f_n u| = \langle t_{n-1} g_n v|$. This can be rewritten as $\langle h_n s_n u| = \langle h_n t_n v|$, which in turn implies $\langle s_n u| = \langle t_n v|$. We have shown that the h_n that we constructed satisfies the desired property. □

Lemma D.5 (Removing the first step)**.** *In a productive category, let $\langle f_0| = \langle g_0|$. Then there exists h with $f_0 = hs$ and $g_0 = ht$ such that $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$ implies $\langle sf_1| \dots |f_n| = \langle tg_1| \dots |g_n|$.*

Proof. By Lemma D.4, we obtain a factorization $f_0 = hs$ and $g_0 = ht$. Moreover, each time that we have $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$, we can obtain a family h_i together with s_i and t_i such that $s_{i-1} f_i = h_i s_i$ and $t_{i-1} f_i = h_i t_i$. Using the fact that $\langle h_n s_n| = \langle h_n t_n|$, we have that $\langle h_1| \dots |h_n s_n| = \langle h_1| \dots |h_n t_n|$, which can be rewritten using dinaturality as $\langle s_0 f_1| \dots |f_n| = \langle t_0 g_1| \dots |g_n|$. □

Lemma D.6. *In a productive category, the final coalgebra of the endofunctor $(\hom \odot \bullet)$ does exist and it is given by the limit of the terminal sequence*

$$L := \lim_n \int^{M_0,\dots,M_n} \prod_{i=0}^{n} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

Proof. We will apply Theorem 1.5. The endofunctor $(\hom \odot \bullet)$ acts on the category $[([\mathbb{N}, \mathrm{C}])^{op} \times [\mathbb{N}, \mathrm{C}], \mathrm{Set}]$, which, being a presheaf category, has all small limits. We will show that there is an isomorphism $\hom \odot L \cong L$ given by the canonical morphism between them.

First, note that the set $L(\mathbb{X}; \mathbb{Y})$ is, explicitly,

$$\lim_n \int^{M_1,\dots,M_n} \prod_{i=1}^{n} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

A generic element from this set is a *sequence of sequences of increasing length*. Moreover, the sequences must *truncate coherently* (Definition D.1).

Secondly, note that the set $(\hom \odot L)(\mathbb{X}; \mathbb{Y})$ is, explicitly,

$$\int^{M_0} \hom(X_0, Y_0 \otimes M_0) \times \lim_n \int^{M_1,\dots,M_n} \prod_{i=1}^{n} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

A generic element from this set is of the form

$$\langle f| (\langle f_1^k| \dots |f_k^k|)_{k \in \mathbb{N}}|,$$

that is, a pair consisting on a first morphism $f : X_0 \to Y_0 \otimes M_0$ and a family of sequences $(\langle f_1^k| \dots |f_k^k|)$, quotiented by dinaturality of M_0 and truncating coherently. The canonical map to $L(\mathbb{X}; \mathbb{Y})$ maps this generic element to the family of sequences $(\langle f_0 | f_1^k| \dots |f_k^k|)_{k \in \mathbb{N}}$, which truncates coherently because the previous family did and we are precomposing with f_0, which is dinatural.

Thirdly, this map is injective. Imagine a pair of elements $\langle f_0| (\langle f_1^k| \dots |f_k^k|)_{k \in \mathbb{N}}|$ and $\langle g_0| (\langle g_1^k| \dots |g_k^k|)_{k \in \mathbb{N}}|$ that have the same image, meaning that, for each $k \in \mathbb{N}$,

$$\langle f_0 | f_1^k| \dots |f_k^k| = \langle g_0 | g_1^k| \dots |g_k^k|.$$

By Lemma D.5, we can find h with $f_0 = hs$ and $g_0 = ht$ such that, for each $k \in \mathbb{N}$, $\langle sf_1^k| \dots |f_k^k| = \langle tg_1^k| \dots |g_k^k|$. Thus,

$$\langle f_0| (\langle f_1^k| \dots |f_k^k|)_{k \in \mathbb{N}}| = \langle h| (\langle sf_1^k| \dots |f_k^k|)_{k \in \mathbb{N}}| =$$
$$\langle h| (\langle tg_1^k| \dots |g_k^k|)_{k \in \mathbb{N}}| = \langle g_0| (\langle tg_1^k| \dots |g_k^k|)_{k \in \mathbb{N}}|.$$

Finally, this map is also surjective. From Lemma D.2, it follows that any family that truncates coherently can be equivalently written as $\langle h_0| \dots |h_n|_{n \in \mathbb{N}}$, which is the image of the element $\langle h_0| \langle h_1| \dots |h_n|_{n \in \mathbb{N}}|$. □

Theorem D.7 (From Theorem 4.11)**.** *In a productive category, the final coalgebra of the endofunctor $(\hom \odot \bullet)$ exists and it is given by the set of stateful sequences quotiented by observational equivalence.*

$$\left(\int^{M \in [\mathbb{N}, \mathrm{C}]} \prod_{i=0}^{\infty} \hom(X_i \otimes M_{i-1}, Y_i \otimes M_i) \right) \Big/ \approx$$

Proof. By Lemma D.6, we know that the final coalgebra exists and is given by the limit of the terminal sequence. By Lemma D.3, we know that it is isomorphic to the set of stateful sequences quotiented by observational equivalence. □

E Monoidal streams

Lemma E.1. *Sequential composition of streams with memories (Definition 5.3) is associative. Given three streams*

- $f \in \mathrm{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$,
- $g \in \mathrm{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$,
- *and* $h \in \mathrm{Stream}(B \cdot \mathbb{Z}, \mathbb{W})$;

we can compose them in two different ways,

- $(f^A; g^B); h^C \in \mathrm{Stream}((A \otimes B) \otimes C \cdot \mathbb{X}, \mathbb{W})$, *or*
- $f^A; (g^B; h^C) \in \mathrm{Stream}(A \otimes (B \otimes C) \cdot \mathbb{X}, \mathbb{W})$.

We claim that

$$((f^A; g^B); h^C) = \alpha_{A,B,C} \cdot (f^A; (g^B; h^C)).$$

Proof. First, we note that both sides of the equation represent streams with different memories.

- $M((f^A; g^B); h^C) = (M(f) \otimes M(g)) \otimes M(h)$,
- $M(f^A; (g^B; h^C)) = M(f) \otimes (M(g) \otimes M(h))$.

We will prove they are related by dinaturality over the associator α. We know that $\mathrm{now}((f^A; g^B); h^C) = \mathrm{now}(f^A; (g^B; h^C))$ by string diagrams (see Figure 26). Then, by coinduction, we know that

$$(\mathrm{later}(f)^{M(f)}; \mathrm{later}(g)^{M(g)}); \mathrm{later}(h)^{M(h)} =$$
$$\alpha \cdot (\mathrm{later}(f)^{M(f)}; (\mathrm{later}(g)^{M(g)}; \mathrm{later}(h)^{M(h)})),$$

that is, $\mathrm{later}((f^A; g^B); h^C) = \alpha \cdot \mathrm{later}(f^A; (g^B; h^C))$. □

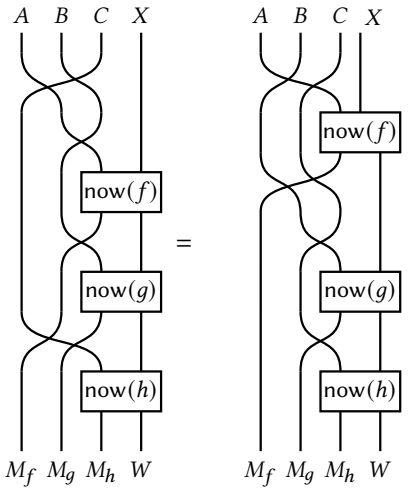


Figure 26. Associativity for sequential composition

Lemma E.2. *Sequential composition of streams (Definition 5.3) is associative. Given three streams*

- $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$,
- $g \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$,
- *and* $h \in \mathsf{Stream}(\mathbb{Z}, \mathbb{W})$;

we claim that $((f;g);h) = (f;(g;h))$.

Proof. Direct consequence of Lemma E.1, after considering the appropriate coherence morphisms. □

Lemma E.3. *Parallel composition of streams with memories is functorial with regards to sequential composition of streams with memories. Given four streams*

- $f \in \mathsf{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$,
- $f' \in \mathsf{Stream}(A' \cdot \mathbb{X}', \mathbb{Y}')$,
- $g \in \mathsf{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$, *and*
- $g' \in \mathsf{Stream}(B' \cdot \mathbb{Y}', \mathbb{Z}')$,

we can compose them in two different ways,

- $(f^A \otimes f'^{A'})^{(A \otimes A')} ; (g^B \otimes g'^{B'})^{(B \otimes B')}$, *and*
- $(f^A ; g^B)^{(A \otimes B)} \otimes (f'^{A'} ; g'^{B'})^{(A' \otimes B')}$,

having slightly different types, respectively,

- $\mathsf{Stream}((A \otimes A') \otimes (B \otimes B') \cdot \mathbb{X} \otimes \mathbb{X}', \mathbb{Z} \otimes \mathbb{Z}')$, *and*
- $\mathsf{Stream}((A \otimes B) \otimes (A' \otimes B') \cdot \mathbb{X} \otimes \mathbb{X}', \mathbb{Z} \otimes \mathbb{Z}')$.

We claim that

$$(f^A \otimes f'^{A'})^{(A \otimes A')} ; (g^B \otimes g'^{B'})^{(B \otimes B')} =$$
$$\sigma_{A',B} \cdot (f^A ; g^B)^{(A \otimes B)} \otimes (f'^{A'} ; g'^{B'})^{(A' \otimes B')}.$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent strams with different memories.

- $M(LHS) = (M(f) \otimes M(f')) \otimes (M(g) \otimes M(g'))$,
- $M(RHS) = (M(f) \otimes M(g)) \otimes (M(f') \otimes M(g'))$.

We will prove they are related by dinaturality over the symmetry σ. We know that $\mathsf{now}(LHS); \sigma = \mathsf{now}(RHS)$ by string diagrams (see Figure 27). Then, by coinduction, we know that $\mathsf{later}(LHS) = \sigma \cdot \mathsf{later}(RHS)$. □

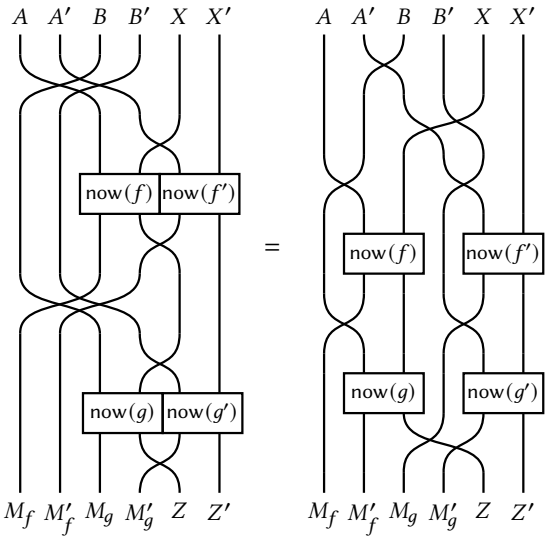


Figure 27. Functoriality of parallel composition.

Lemma E.4. *Parallel composition of streams is functorial with respect to sequential composition of streams. Given four streams*

- $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$,
- $f' \in \mathsf{Stream}(\mathbb{X}', \mathbb{Y}')$,
- $g \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$, *and*
- $g' \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$;

we claim that $(f \otimes f'); (g \otimes g') = (f;g) \otimes (f';g')$.

Proof. Direct consequence of Lemma E.3, after considering the appropriate coherence morphisms. □

Theorem E.5 (see [65])**.** *Monoidal streams over a symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a symmetric monoidal category* Stream.

Proof. Sequential composition of streams (Definition 5.3) is associative (Lemma E.2) and unital with respect to identities. Parallel composition is bifunctorial with respect to sequential composition (Lemma E.4); this determines a bifunctor, which is the tensor of the monoidal category. The coherence morphisms and the symmetry can be included from sets, so they still satisfy the pentagon and triangle equations. □

Lemma E.6. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *with memories satisfies the tightening axiom (A1). Given three streams*

- $u \in \mathsf{Stream}(A \cdot \mathbb{X}', \mathbb{X})$,
- $f \in \mathsf{Stream}(B \otimes T \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $v \in \mathsf{Stream}(C \cdot \mathbb{Y}, \mathbb{Y}')$;

we claim that

$$\mathbf{fbk}^S(u^A; f^B; v^C) = \sigma \cdot u^A; \mathbf{fbk}^S(f^{B \otimes T}); v^C.$$

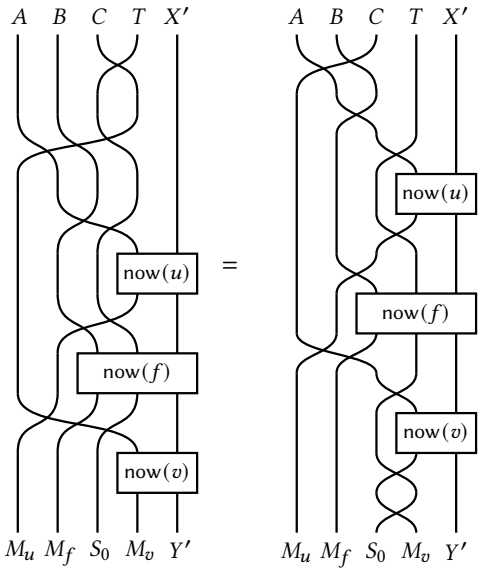


Figure 28. The tightening axiom (A1).

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = A \otimes B \otimes C \otimes T$,
- $M(RHS) = A \otimes B \otimes T \otimes C$.

We will prove that they are related by dinaturality over the symmetry σ. We know that $\mathsf{now}(LHS); \sigma = \mathsf{now}(RHS)$ by string diagrams (see Figure 28). Then, by coinduction, we know that $\mathsf{later}(LHS) = \sigma \cdot \mathsf{later}(RHS)$. □

Lemma E.7. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *satisfies the tightening axiom (A1). Given streams*

- $u \in \mathsf{Stream}(\mathbb{X}', \mathbb{X})$,
- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $v \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Y}')$;

we claim that $\mathbf{fbk}^S(u; f; v) = u; \mathbf{fbk}^S(f); v$.

Proof. Consequence of Lemma E.6, after applying the necessary coherence morphisms. □

Lemma E.8. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *with memories satisfies the vanishing axiom (A2). Given a stream*

- $f \in \mathsf{Stream}(A \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that $\mathbf{fbk}^I(f^A) = \rho \cdot f$.

Proof. First, we note that both sides of the equation represent streams with different memories, $M(\mathbf{fbk}^I(f^A)) = M(f) \otimes I$. We will prove that they are related by dinaturality over the right unitor ρ. We know that $\mathsf{now}(\mathbf{fbk}^I(f^A)) = \mathsf{now}(f)$ by definition. Then, by coinduction, we konw that $\mathsf{later}(\mathbf{fbk}^I(f^A)) = \rho \cdot \mathsf{later}(f)$. □

Lemma E.9. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *satisfies the vanishing axiom (A2). Given a stream* $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *we claim that* $\mathbf{fbk}^I(f) = f$.

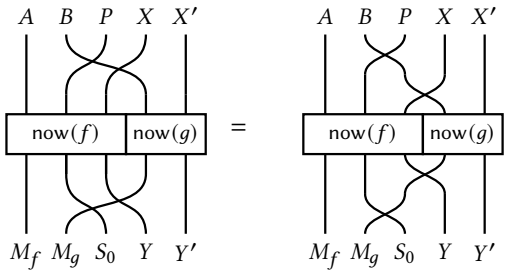


Figure 29. The strength axiom (A4).

Proof. Consequence of Lemma E.8, after applying the necessary coherence morphisms. □

Lemma E.10. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *with memories satisfies the joining axiom (A3). Given a stream*

- $f \in \mathsf{Stream}((A \otimes P \otimes Q) \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that

$$\mathbf{fbk}^{S\otimes T}(f^{A\otimes(P\otimes Q)}) = \alpha \cdot \mathbf{fbk}^T(\sigma \cdot \mathbf{fbk}^S(\sigma \cdot f^{(A\otimes Q)\otimes P})).$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent strams with different memories.

- $M(LHS) = M(f) \otimes (S_0 \otimes T_0)$,
- $M(RHS) = (M(f) \otimes S_0) \otimes T_0$.

We will prove that they are related by dinaturality over the associator α. We know that $\mathsf{now}(LHS); \alpha = \mathsf{now}(RHS)$ by definition. Then, by coinduction, we know that $\mathsf{later}(LHS) = \alpha \cdot \mathsf{later}(RHS)$. □

Lemma E.11. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *satisfies the joining axiom (A3). Given a stream*

- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that $\mathbf{fbk}^{S\otimes T}(f) = \mathbf{fbk}^T(\mathbf{fbk}^S(f))$.

Proof. Consequence of Lemma E.10, after applying the necessary coherence morphisms. □

Lemma E.12. *The structure* $(\mathsf{Stream}, \mathbf{fbk})$ *with memories satisfies the strength axiom (A4). Given two streams*

- $f \in \mathsf{Stream}((A \otimes P) \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $g \in \mathsf{Stream}(B \cdot \mathbb{X}', \mathbb{Y}')$,

we claim that

$$\alpha \cdot \mathbf{fbk}^S(f^A \otimes g^B) = \mathbf{fbk}^S(f^{A\otimes P})^{A\otimes P} \otimes g^B.$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = (M(f) \otimes S_0) \otimes M(g)$,
- $M(RHS) = M(f) \otimes (S_0 \otimes M(g))$.

We will prove that they are related by dinaturality over the symmetry α. We know that $\mathsf{now}(LHS) = \mathsf{now}(RHS); \alpha$ by string diagrams (see Figure 29). Then, by coinduction, we know that $\alpha \cdot \mathsf{later}(LHS) = \mathsf{later}(RHS)$. □

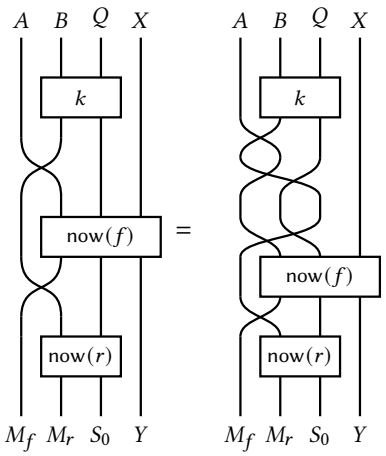


Figure 30. The sliding axiom (A5).

Lemma E.13. *The structure* (Stream, **fbk**) *with memories satisfies the strength axiom (A4). Given two streams*

- $f \in \mathsf{Stream}(\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $g \in \mathsf{Stream}(\mathbb{X}', \mathbb{Y}')$,

we claim that

$$\alpha \cdot \mathbf{fbk}^S(f \otimes g) = \mathbf{fbk}^S(f) \otimes g.$$

Proof. Consequence of Lemma E.12, after applying the necessary coherence morphisms. □

Lemma E.14. *The structure* (Stream, **fbk**) *with memories satisfies the sliding axiom (A5). Given two streams*

- $f \in \mathsf{Stream}(A \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{T} \otimes \mathbb{Y})$, *and*
- $r \in \mathsf{Stream}(C \cdot \mathbb{T}, \mathbb{S})$

we claim that, for each $k \colon B \otimes Q \to C \otimes P$,

$$k \cdot \mathbf{fbk}^{\mathbb{S}}(f^{A\otimes P}; (r^C \otimes \mathrm{id})) = \mathbf{fbk}^{\mathbb{T}}(k \cdot (\partial r^C \otimes \mathrm{id}); f^{A\otimes P}).$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = M(f) \otimes M(r) \otimes S_0$,
- $M(RHS) = M(r) \otimes M(f) \otimes T_0$.

We will prove that they are related by dinaturality over the symmetry and the first action of r, that is, $\sigma; (\mathsf{now}(r) \otimes \mathrm{id})$. We know that $\mathsf{now}(LHS) = \mathsf{now}(RHS); (\sigma; (\mathrm{id} \otimes \mathsf{now}(r)))$ by string diagrams (see Figure 30). Using coinduction,

$$\begin{aligned}
&(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathsf{later}(LHS) \\
=&(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathbf{fbk}^{\mathbb{S}}(\mathsf{later}(f)^{M(f)\otimes S_0}; (\mathsf{later}(r)^{M(r)} \otimes \mathrm{id})) \\
=&\mathbf{fbk}^{\mathbb{T}}((\sigma; \underline{\mathsf{now}(r)}) \cdot (\partial\,\mathsf{later}(r)^{M(r)} \otimes \mathrm{id}); \mathsf{later}(f)^{M(f)\otimes S_0}) \\
=&\mathbf{fbk}^{\mathbb{T}}(\sigma \cdot (\mathsf{later}(\partial r)^{M(r)} \otimes \mathrm{id}); \mathsf{later}(f)^{M(f)\otimes S_0}) \\
=&\,\mathsf{later}(RHS),
\end{aligned}$$

we show that $(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathsf{later}(LHS) = \mathsf{later}(RHS)$. □

Lemma E.15. *The structure* (Stream, **fbk**) *satisfies the sliding axiom (A5). Given two streams*

$$\textsc{Gen}\ \frac{f \in \mathcal{G}(A_1, \ldots, A_n; B) \qquad \Gamma_1 \vdash x_1 : A_1 \ldots \Gamma_n \vdash x_n : A_n}{\mathsf{Shuf}(\Gamma_1, \ldots, \Gamma_n; \Gamma) \vdash f(x_1, \ldots, x_n) : B}$$

$$\textsc{Pair}\ \frac{\Gamma_1 \vdash x_1 : A_1 \qquad \ldots \qquad \Gamma_n \vdash x_n : A_n}{\mathsf{Shuf}(\Gamma, \Delta) \vdash [x_1, \ldots, x_n] : A_1 \otimes \cdots \otimes A_n} \qquad \textsc{Var}\ \frac{}{x : A \vdash x : A}$$

$$\textsc{Split}\ \frac{\Delta \vdash m : A_1 \otimes \cdots \otimes A_n \qquad \Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash z : C}{\mathsf{Shuf}(\Gamma_1, \ldots, \Gamma_n) \vdash \textsc{Split}\ m \to [x_1, \ldots, x_n]\ \textsc{in}\ z : C}$$

Figure 31. Type theory of symm. monoidal categories [70].

- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{T} \otimes \mathbb{Y})$, *and*
- $r \in \mathsf{Stream}(\mathbb{T}, \mathbb{S})$

we claim that, $\mathbf{fbk}^{\mathbb{S}}(f; (r \otimes \mathrm{id})) = \mathbf{fbk}^{\mathbb{T}}((\partial r \otimes \mathrm{id}); f)$.

Proof. Consequence of Lemma E.14, after applying the necessary coherence morphisms. □

Theorem E.16 (From Theorem 5.10)**.** *Monoidal streams over a symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a* ∂*-feedback monoidal category* (Stream, **fbk**).

Proof. We have proven that it satisfies all the feedback axioms:

- the tightening axiom by Lemma E.7,
- the vanishing axiom by Lemma E.9,
- the joining axiom by Lemma E.11,
- the strength axiom by Lemma E.13,
- and the sliding axiom by Lemma E.15.

Thus, it is a feedback structure □

F Type theory

As our base type theory, we consider the that of symmetric monoidal categories over some generators forming a multigraph $\mathcal{G}$, as described by Shulman [70]. Our only change will be to consider an *unbiased presentation*, meaning that we consider n-ary tensor products instead of only binary and 0-ary (the monoidal unit). This reduces the number of rules: the binary case and the 0-ary case of the usual presentation are taken care of by a single n-ary case.

Definition F.1 (Shuffling contexts [70])**.** A *shuffle of contexts* $\mathsf{Shuf}(\Gamma_1, \ldots, \Gamma_n)$ is the result of a permutation of $\bigsqcup_{i=0}^{n} \Gamma_n$ that leaves invariant the internal order of each Γ_i. Shuffles allow us to derive morphisms that make use of the symmetry without introducing redundancy in our type theory.

Substitution is admissible in the type theory for symmetric monoidal categories [70]. It can be inductively defined: we write $P[m/x]$ for the substitution of the variable x by a term m inside the term p. Using substitution, we can state a pair of β/η-reduction equalities for the terms of our type theory.

- Split $[E_1, \dots, E_n] \to [x_1, \dots, x_n]$ in $z \equiv z[E_1/x_1, \dots]$
- Split $[E_1, \dots, E_n] \to [x_1, \dots, x_n]$ in $N[[x_1, \dots, x_n]/u]$ $\equiv N[[E_1, \dots, E_n]/u]$

F.1 Type theory for a strong monoidal endofunctor

Definition F.2 (Signature)**.** Let T be a set of basic types. We write

$$T_\partial^\star := \{\partial^{n_1} t_1 \partial^{n_2} t_2 \dots \partial^{n_k} t_k \mid n_1, \dots, n_k \in \mathbb{N}, t_i \in T\}$$

for the free monoid-with-an-endomorphism $\partial : T_\partial^\star \to T_\partial^\star$ over T. The **signature** for a type theory of monoidal categories with a monoidal endofunctor is given by a pair of functions $s, t : O \rightrightarrows T_\partial^\star$, assigning source and target to every generator from a set O.

Example F.3. We will usually include two families of generators in our theory. For each type $t_0 \in T$ that can be copied, we have a copy $\in O$ generator with $s(\text{copy}) = t_0 \in T_\partial^\star$ and $t(\text{copy}) = t_0 \cdot t_0 \in T_\partial^\star$.

Definition F.4. The type theory of symmetric monoidal categories with a symmetric monoidal endofunctor over a signature $O \rightrightarrows T_\partial^\star$ is the type theory of symmetric monoidal categories extended with an operator ∂ on types such that

$$\partial[] = [] \quad \text{and} \quad \partial(\Gamma, x{:}A) = \partial\Gamma, x{:}\partial A,$$

and the following Delay introduction rule.

$$\frac{\Gamma \vdash x : A}{\partial\Gamma \vdash \textsc{Delay}(x) : \partial A}\ \textsc{Delay}$$

The delay operator is a monoid homomorphism on types, satisfying $\partial I \equiv I$ and $\partial(A \otimes B) \equiv \partial A \otimes \partial B$. The Delay introduction rule satisfies the following conversion equalities, that state that it acts as a functor preserving the monoidal structure.

- $\textsc{Delay}(x) \equiv x$ for x a variable,
- $[\textsc{Delay}(e_1), \dots, \textsc{Delay}(e_n)] \equiv \textsc{Delay}([e_1, \dots, e_n])$,
- Split $\textsc{Delay}(m) \to [e_1, \dots, e_n]$ in $\textsc{Delay}(z)$ $\equiv \textsc{Delay}(\text{Split } m \to [e_1, \dots, e_n] \text{ in } z)$,

We choose not to explicitly state the delay rule when writing terms of the type theory (as we do in Section 8.2), as it does not cause ambiguity with the typing. However, we do write it when typechecking.

F.2 Type Theory for Delayed Feedback

We finally augment the type theory of symmetric monoidal categories with a monoidal endofunctor by adding the following derivation rule.

$$\frac{\Gamma, s : \partial S \vdash x(s) : S \otimes A}{\Gamma \vdash \text{Fbk } s.\ x(s) : A}\ \textsc{Fbk}$$

Following the axioms of categories with delayed feedback and their normalization theorem, we introduce rules that follow from the feedback axioms. Note that these rules simplify multiple applications of feedback into a single application at the head of the term.

1. $g(\textsc{Fbk}\ s \text{ in } x(s)) \equiv$ $\textsc{Fbk}\ s \text{ in Let } [t, b] \leftarrow x(s) \text{ in } [t, g(b)]$
2. $[\textsc{Fbk}\ s \text{ in } x(s), \textsc{Fbk}\ t \text{ in } y(t)] \equiv \textsc{Fbk}\ m \text{ in}$ Split $m \to [s, t]$ in Split $x(s) \to [u, v]$ in Split $y(t) \to [u', v']$ in $[[u, u'], [v, v']]$
3. Split $m \to [x_1, \dots, x_n]$ in $\textsc{Fbk}\ s \text{ in } z \equiv$ $\textsc{Fbk}\ s \text{ in Split } m \to [x_1, \dots, x_n] \text{ in } z$
4. Split $(\textsc{Fbk}\ u \text{ in } z(u)) \to [x_1, \dots, x_n]$ in $m \equiv$ $\textsc{Fbk}\ u \text{ in Split } z(u) \to [v, n] \text{ in}$ Split $n \to [x_1, \dots, x_n]$ in $[v, m]$
5. $x \equiv \textsc{Fbk}\ i \text{ in Split } i \to [] \text{ in } [t, x]$
6. $\textsc{Fbk}\ x \text{ in } \textsc{Fbk}\ y \text{ in } m(x, y) \equiv$ $\textsc{Fbk}\ n \text{ in Split } n \to [x, y] \text{ in } m(x, y)$

Finally, we introduce an equality representing the *sliding axiom* (Figure 7). Having all of these rules together means that we can always rewrite a term in the type theory with feedback as a term of the type theory of symmetric monoidal categories *up to sliding*. These are precisely the morphisms of the $\text{St}(\bullet)$-construction, the free category with feedback.

1. $\textsc{Fbk}\ s \text{ in } f(\textsc{Delay}(h(s)), x) \equiv$ $\textsc{Fbk}\ t \text{ in Split } f(x, t) \to [y, s] \text{ in } [y, h(s)]$

F.3 Categories with copy and syntax sugar

Definition F.5. A *category with copying* is a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ in which every object $A \in \mathsf{C}$ has a (non-necessarily natural) coassociative and cocommutative comultiplication $\delta_A = (\curlywedge)_A : A \to A \otimes A$, called the "copy".

Every cartesian category and every kleisli category of a Set-based commutative monad is a category with copying. In our type theory, this is translated into a Copy generator acting as follows.

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash \textsc{Copy}(x) : A \otimes A}\ \textsc{Copy}$$

Definition F.6. A *category with ∂-merging* is a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ with a symmetric monoidal endofunctor $\partial : \mathsf{C} \to \mathsf{C}$ in which every object $A \in \mathsf{C}$ has an associated morphism $\phi_A : A \otimes \partial A \to A$.

In our type theory, this is translated by a Fby generator acting as follows.

$$\frac{\Gamma \vdash x : A \qquad \Delta \vdash y : \partial(A)}{\text{Shuf}(\Gamma, \Delta) \vdash x\ \textsc{Fby}\ y : A}\ \textsc{Fby}$$

We allow three pieces of syntax sugar in our language, suited only for the case of categories with copying. These

make the language more Lucid-like without changing its formal description.

1. We allow multiple occurences of a variable, implicily copying it.
2. We apply Delay rules where needed for type-checking, without explicitly writing the rule.
3. Recursive definitions are syntax for the Fbk rule and the Copy rule. That is,

$$M = x(M) \quad \text{means} \quad M = \text{Fbk } m \text{ in } \text{Copy}(x(m)).$$

4. We use Wait to declare an implicit feedback loop.

$$\text{Wait}(x) \quad \text{means} \quad \text{Fbk } y \text{ in } [x, y].$$

G Implementation

We use the **Haskell** [39] programming language for computations. We use `Arrows` [40] to represent monoidal categories with an identity-on-objects monoidal functor from our base category of Haskell types and functions. Notations for arrows [61] have been explained in terms of Freyd categories [63]. In particular, the **`loop`** notation is closely related to feedback, as it is usually employed to capture *traces*.

Our definition of monoidal streams follows Definition 5.1.

```
type Stream c = StreamWithMemory c ()

data StreamWithMemory c n x y where
  StreamWithMemory :: (Arrow c) =>
     c (n , x) (m , y)
    -> StreamWithMemory c m x y
    -> StreamWithMemory c n x y
```

Their sequential and parallel composition (**`comp`** and **`tensor`**) follow from Definitions 5.3 and 5.5. In Appendix G.2 we describe both first on the now part; and then trivially extended by coinduction.

$$\text{now}(f ;_N g) = (\sigma \otimes \text{id}_A); (\text{id} \otimes \text{now}(f)); (\sigma \otimes \text{id}_B); (\text{id} \otimes \text{now}(g)).$$

$$\text{now}(f \otimes_N g) = (\text{id} \otimes \sigma \otimes \text{id}); (\text{now}(f) \otimes \text{now}(g)); (\text{id} \otimes \sigma \otimes \text{id}).$$

Example G.1 (Fibonacci example). The code for **`fibonacci`** in Appendix G.2 follows the definition in Section 6.3. We can execute it to obtain the first 10 numbers from the Fibonacci sequence.

```
> take 10 <$> run fibonacci
Identity [0,1,1,2,3,5,8,13,21,34]
```

Example G.2 (Random walk example). The code for **`walk`** in Appendix G.2 follows the definition in Example 7.4. We can execute it multiple times to obtain different random walks starting from 0.

```
> take 10 <$> run walk
[0,1,0,-1,-2,-1,-2,-3,-2,-3]
> take 10 <$> run walk
[0,1,2,1,2,1,2,3,4,5]
> take 10 <$> run walk
[0,-1,-2,-1,-2,-1,0,-1,0,-1]
```

Example G.3 (Ehrenfest model). The code for **`ehrenfest`** in Appendix G.2 follows the definition in Figure 12. We can execute it to simulate the Ehrenfest model.

```
> take 10 <$> run ehrenfest
[([2,3,4],[1]),([2,3],[1,4]),
([2,3,4],[1]),([2,4],[1,3]),
([2],[1,3,4]),([2,4],[1,3]),
([2],[1,3,4]),([2,3],[1,4]),
([3],[1,2,4]),([2,3],[1,4])]
```

G.1 Term derivations

Example G.4 (Fibonacci).

$$\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathbb{N}_0} \quad \dfrac{\dfrac{}{f : \mathbb{N} \vdash f : \mathbb{N}}}{f : \mathbb{N} \vdash \textsc{Copy}(f) : \mathbb{N} \otimes \mathbb{N}} \quad \dfrac{\dfrac{}{f_1 : \mathbb{N} \vdash f_1 : \mathbb{N}} \quad \dfrac{\dfrac{}{\vdash 1 : \mathbb{N}_0} \quad \dfrac{\dfrac{}{f_2 : \mathbb{N} \vdash f_2 : \mathbb{N}}}{f_2 : \mathbb{N} \vdash \textsc{Wait}(f_2) : \partial\mathbb{N}}}{f_2 : \mathbb{N} \vdash 1 \textsc{ Fby } \textsc{Wait}(f_2) : \mathbb{N}}}{f_1 : \mathbb{N}, f_2 : \mathbb{N} \vdash f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2) : \mathbb{N}}}{f : \mathbb{N} \vdash \textsc{Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2)) : \mathbb{N}}}{f : \partial\mathbb{N} \vdash \textsc{Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2)) : \partial\mathbb{N}}}{f : \partial\mathbb{N} \vdash 0 \textsc{ Fby Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2)) : \mathbb{N}}$$
$$\dfrac{f : \partial\mathbb{N} \vdash 0 \textsc{ Fby Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2)) : \mathbb{N}}{f : \partial\mathbb{N} \vdash \textsc{Copy}(0 \textsc{ Fby Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2))) : \mathbb{N} \otimes \mathbb{N}}$$
$$\dfrac{f : \partial\mathbb{N} \vdash \textsc{Copy}(0 \textsc{ Fby Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2))) : \mathbb{N} \otimes \mathbb{N}}{\vdash : \textsc{Fbk } f \textsc{ in } \textsc{Copy}(0 \textsc{ Fby Split } \textsc{Copy}(f) \to [f_1, f_2] \textsc{ in } (f_1 + 1 \textsc{ Fby } \textsc{Wait}(f_2))) : \mathbb{N}}$$

Example G.5 (Random walk).

$$\dfrac{\dfrac{}{\vdash 0 : \mathbb{N}_0} \quad \dfrac{\dfrac{}{\vdash \textsc{Uniform}(-1, 1) : \mathbb{N}} \quad \dfrac{}{w : \partial\mathbb{N} \vdash w : \partial\mathbb{N}}}{w : \partial\mathbb{N} \vdash \textsc{Uniform}(-1, 1) + w : \partial\mathbb{N}}}{w : \partial\mathbb{N} \vdash 0 \textsc{ Fby } (\textsc{Uniform}(-1, 1) + w) : \mathbb{N}}$$
$$\dfrac{w : \partial\mathbb{N} \vdash 0 \textsc{ Fby } (\textsc{Uniform}(-1, 1) + w) : \mathbb{N}}{w : \partial\mathbb{N} \vdash \textsc{Copy}(0 \textsc{ Fby } (\textsc{Uniform}(-1, 1) + w)) : \mathbb{N} \otimes \mathbb{N}}$$
$$\dfrac{w : \partial\mathbb{N} \vdash \textsc{Copy}(0 \textsc{ Fby } (\textsc{Uniform}(-1, 1) + w)) : \mathbb{N} \otimes \mathbb{N}}{\vdash \textsc{Fbk } w \textsc{ in } \textsc{Copy}(0 \textsc{ Fby } (\textsc{Uniform}(-1, 1) + w)) : \mathbb{N}}$$

Example G.6 (Ehrenfest model).

$$\dfrac{\dfrac{}{\vdash (1,2,3,4) : \mathrm{Urn}_0 \quad \vdash () : \mathrm{Urn}_0} \quad \dfrac{\dfrac{}{u : \mathrm{Urn} \otimes \mathrm{Urn} \vdash u : \mathrm{Urn} \otimes \mathrm{Urn}} \quad \dfrac{\dfrac{}{\vdash \textsc{Uniform} : \mathbb{N}}}{\vdash \textsc{Copy}(\textsc{Uniform}) : \mathbb{N} \otimes \mathbb{N}} \quad \dfrac{\dfrac{}{n_1 : \mathbb{N} \vdash n_1 : \mathbb{N}} \quad \dfrac{}{u_1 : \mathrm{Urn} \vdash u_1 : \mathrm{Urn}}}{n_1 : \mathbb{N}, u_1 : \mathrm{Urn} \vdash \textsc{Move}(n_1, u_1) : \mathrm{Urn}} \quad \dfrac{\dfrac{}{n_2 : \mathbb{N} \vdash n_2 : \mathbb{N}} \quad \dfrac{}{u_2 : \mathrm{Urn} \vdash u_2 : \mathrm{Urn}}}{n_2 : \mathbb{N}, u_2 : \mathrm{Urn} \vdash \textsc{Move}(n_2, u_2) : \mathrm{Urn}}}{n_1 : \mathbb{N}, n_2 : \mathbb{N}, u_1 : \mathrm{Urn}, u_2 : \mathrm{Urn} \vdash [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}}{u_1 : \mathrm{Urn}, u_2 : \mathrm{Urn} \vdash \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}}{u : \mathrm{Urn} \otimes \mathrm{Urn} \vdash \textsc{Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}}{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash \textsc{Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \partial(\mathrm{Urn} \otimes \mathrm{Urn})}$$
$$\dfrac{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash \textsc{Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash [(1,2,3,4), ()] \textsc{ Fby Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}$$
$$\dfrac{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash [(1,2,3,4), ()] \textsc{ Fby Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathrm{Urn} \otimes \mathrm{Urn}}{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash \textsc{Copy}([(1,2,3,4), ()] \textsc{ Fby Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]) : (\mathrm{Urn} \otimes \mathrm{Urn}) \otimes (\mathrm{Urn} \otimes \mathrm{Urn})}$$
$$\dfrac{u : \partial(\mathrm{Urn} \otimes \mathrm{Urn}) \vdash \textsc{Copy}([(1,2,3,4), ()] \textsc{ Fby Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]) : (\mathrm{Urn} \otimes \mathrm{Urn}) \otimes (\mathrm{Urn} \otimes \mathrm{Urn})}{\vdash \textsc{Fbk } u \textsc{ in } \textsc{Copy}([(1,2,3,4), ()] \textsc{ Fby Split } u \to [u_1, u_2] \textsc{ in } \textsc{Split } \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2] \textsc{ in } [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]) : \mathrm{Urn} \otimes \mathrm{Urn}}$$

G.2 Code

The following code has been compiled under GHCi, version 8.6.5. The "random" library may need to be installed.

```
-- | Monoidal Streams for Dataflow Programming.

-- Anonymous.

-- The following code implements the category of monoidal streams over a
-- monoidal category with an identity-on-objects functor from the
-- (pseudo)category of Haskell types.

-- During the manuscript, we have needed to perform some computations: for
-- instance, to see that according to our definitions the Fibonacci morphism we
-- describe really computes the Fibonacci sequence. Computations of this kind
-- are difficult and tedious to write and to justify, and the reader may find
-- difficult to reproduce them. Instead of explicitly writing these
-- computations, we implement them and we provide the necessary code so that the
-- reader can verify the result from the computation.

-- Morphisms in a monoidal category are written in "Arrow" notation, using (>>>)
-- for sequential composition and (***) for parallel composition. Coherence
-- morphisms need to be written explicitly, we usually write them at the side of
-- the diagram.

{-# LANGUAGE GADTs                    #-}
{-# LANGUAGE TypeSynonymInstances     #-}
{-# LANGUAGE FlexibleInstances       #-}

module Main where

import Prelude hiding (id)
import Data.Functor.Identity
import Data.List
import Control.Category
import Control.Arrow
import System.Random
import System.IO.Unsafe


-- Fixpoint equation for monoidal streams. Figure 5.
type Stream c = StreamWithMemory c ()

data StreamWithMemory c n x y where
  StreamWithMemory :: (Arrow c) =>
     c (n , x) (m , y)
    -> StreamWithMemory c m x y
    -> StreamWithMemory c n x y


--------------
-- EXAMPLES --
--------------
fibonacci :: Stream (Kleisli Identity) () Int
fibonacci = fbk                            $ runitS
```

```
  >>> copy                                  >>> lunitinvS *** id
  >>> delay (k1 *** wait) *** id
  >>> delay fby *** id
  >>> plus                                  >>> lunitinvS
  >>> k0 *** id
  >>> fby
  >>> copy

walk :: Stream (Kleisli IO) (()) (Int)
walk = fbk
    $ (id *** unif)
  >>> plus                                  >>> lunitinvS
  >>> k0 *** id
  >>> fby
  >>> copy
 where
   unif :: Stream (Kleisli IO) () Int
   unif = lift $ Kleisli (\() -> do
      boolean <- randomIO
      return $ if boolean then 1 else -1)

type Urn = [Int]

ehrenfest :: Stream (Kleisli IO) (()) (Urn,Urn)
ehrenfest = fbk                             $ runitS>>> lunitinv *** lunitinv
  >>> (full *** idS) *** (empty *** idS)    >>> runitinv
  >>> (fby *** fby) *** unif
  >>> idS *** copy                          >>> associnv >>> idS *** assoc
  >>> idS *** (sigma *** idS)               >>> idS *** associnv >>> assoc
  >>> move *** move
  >>> copy *** copy                          >>> associnv >>> idS *** assoc
  >>> idS *** (sigma *** idS)               >>> idS *** associnv >>> assoc
  where
    unif :: Stream (Kleisli IO) () Int
    unif = lift $ Kleisli (\() -> randomRIO (1,4))

    empty :: Stream (Kleisli IO) () Urn
    empty = lift $ arr (\() -> [])

    full :: Stream (Kleisli IO) () Urn
    full = lift $ arr (\() -> [1,2,3,4])

    move :: Stream (Kleisli IO) (Urn, Int) Urn
    move = lift $ arr (\(u,i) ->
      if elem i u
        then (delete i u)
        else (insert i u))

--- take 10 <$> run fibonacci
--- take 10 <$> run walk
--- take 10 <$> run ehrenfest
```

```
---------------------------
-- THE FEEDBACK CATEGORY --
---------------------------

compS :: (Arrow c) =>
  StreamWithMemory c m x y ->
  StreamWithMemory c n y z ->
  StreamWithMemory c (m , n) x z
compS
  (StreamWithMemory fnow flater)
  (StreamWithMemory gnow glater) =
  StreamWithMemory (sequentialComposition fnow gnow) (compS flater glater)
 where

   -- Definition 5.2.
   -- Sequential composition, "now".
   sequentialComposition :: Arrow c
     => c (m , x) (p , y)
     -> c (n , y) (q , z)
     -> c ((m,n),x) ((p,q),z)
   sequentialComposition f g =
        sigma  *** id    >>> associnv
    >>> id *** f       >>> assoc
    >>> sigma  *** id  >>> associnv
    >>> id *** g       >>> assoc

comp :: (Arrow c) => Stream c x y -> Stream c y z -> Stream c x z
comp f g = lact lunitinv (compS f g)



tensorS :: (Arrow c) =>
  StreamWithMemory c p x y ->
  StreamWithMemory c p' x' y' ->
  StreamWithMemory c (p , p') (x,x') (y,y')
tensorS
  (StreamWithMemory fnow flater)
  (StreamWithMemory gnow glater) =
  StreamWithMemory (parallelCompTosition fnow gnow) (tensorS flater glater)
 where

   -- Definition 5.3. Parallel compTosition.
   parallelCompTosition :: Arrow c
     => c (m,x) (p,z)
     -> c (n,y) (q,w)
     -> c ((m,n),(x,y)) ((p,q),(z,w))
   parallelCompTosition f g =
         associnv                   >>> id *** assoc
     >>> (id *** (sigma *** id))  >>> id *** associnv >>> assoc
     >>> (f *** g)                >>> associnv           >>> id *** assoc
     >>> (id *** (sigma *** id))  >>> id *** associnv >>> assoc

tensor :: Arrow c => Stream c x y -> Stream c x' y' -> Stream c (x,x') (y,y')
tensor f g = lact lunitinv (tensorS f g)
```

```
lact :: (Arrow c) => c n m -> StreamWithMemory c m x y -> StreamWithMemory c n x y
lact f (StreamWithMemory now later) = StreamWithMemory ((f *** id) >>> now) later


fbkS :: (Arrow c) =>
  StreamWithMemory c m (s,x) (s,y) ->
  StreamWithMemory c (m, s) x y
fbkS (StreamWithMemory now later) =
  StreamWithMemory (nowFeedback now) (fbkS later)
 where

   -- Definition 5.7. Feedback operation.
   nowFeedback :: (Arrow c) => c (m,(s,x)) (n,(t,y)) -> c ((m,s),x) ((n,t),y)
   nowFeedback f = associnv >>> f >>> assoc

fbk :: (Arrow c) => Stream c (s,x) (s,y) -> Stream c x y
fbk t = lact (arr (\() -> ((),undefined))) (fbkS t)

idS :: (Arrow c) => Stream c x x
idS = StreamWithMemory (id) idS


lift :: (Arrow c) => c x y -> Stream c x y
lift f = StreamWithMemory (id *** f) (lift f)

liftarr :: (Arrow c) => (x -> y) -> Stream c x y
liftarr s = lift $ arr s

instance (Arrow c) => Category (Stream c) where
  id = idS
  (.) f g = comp g f

instance (Arrow c) => Arrow (Stream c) where
  arr = liftarr
  (***) = tensor

instance (Arrow c) => ArrowLoop (Stream c) where
  loop f = fbk $ sigma >>> f >>> sigma


delay :: (Arrow c) => Stream c x y -> Stream c x y
delay f = StreamWithMemory (id *** undefined)  f




------------
-- ARROWS --
------------
assoc :: Arrow c => c (x,(y,z)) ((x,y),z)
assoc = arr $ \(x,(y,z)) -> ((x,y),z)
assocS :: Arrow c => Stream c (x,(y,z)) ((x,y),z)
```

```
assocS = lift assoc

associnv :: Arrow c => c ((x,y),z) (x,(y,z))
associnv = arr $ \((x,y),z) -> (x,(y,z))
associnvS :: Arrow c => Stream c ((x,y),z) (x,(y,z))
associnvS = lift $ associnv

lunit :: Arrow c => c ((),a) a
lunit = arr $ \((),a) -> a
lunitS :: Arrow c => Stream c ((),a) a
lunitS = lift $ lunit

lunitinv :: Arrow c => c a ((),a)
lunitinv = arr $ \a -> ((),a)
lunitinvS :: Arrow c => Stream c a ((),a)
lunitinvS = lift $ lunitinv

runit :: Arrow c => c (a,()) a
runit = arr $ \(a,()) -> a
runitS :: Arrow c => Stream c (a,()) a
runitS = lift $ runit

runitinv :: Arrow c => c a (a,())
runitinv = arr $ \a -> (a,())
runitinvS :: Arrow c => Stream c a (a,())
runitinvS = lift $ runitinv


sigma :: Arrow c => c (x,y) (y,x)
sigma = arr $ \(x,y) -> (y,x)

sigmaS :: Arrow c => Stream c (x,y) (y,x)
sigmaS = lift $ sigma


----------------
-- GENERATORS --
----------------
fby :: (Monad t) => Stream (Kleisli t) (a , a) a
fby = StreamWithMemory (Kleisli $ \((),(x,y)) -> pure ((),x)) (lift (arr snd))

copy :: (Monad t) => Stream (Kleisli t) a (a,a)
copy = lift (Kleisli $ \a -> pure (a,a))


k0,k1,k2 :: (Arrow c) => Stream c () Int
k0 = lift $ arr (\() -> 0)
k1 = lift $ arr (\() -> 1)
k2 = lift $ arr (\() -> 2)

plus :: (Arrow c) => Stream c (Int,Int) Int
plus = lift $ arr (\(a,b) -> a + b)

wait :: (Arrow c) => Stream c a a
```

```
wait = fbk sigmaS

------------
-- SYSTEM --
------------

class (Monad m) => IOMonad m where unsafeRun :: m a -> m a
instance IOMonad IO where unsafeRun = unsafeInterleaveIO
instance IOMonad Identity where unsafeRun = id


runUnsafeWithMemory :: (IOMonad t) => m -> StreamWithMemory (Kleisli t) m a b -> [a] -> t [b]
runUnsafeWithMemory m (StreamWithMemory (Kleisli now) later) (a:as) = do
  (n , b)<- now (m , a)
  l <- unsafeRun $ runUnsafeWithMemory n later as
  pure (b : l)

runUnsafe :: (IOMonad t) => Stream (Kleisli t) a b -> [a] -> t [b]
runUnsafe = runUnsafeWithMemory ()

run :: (IOMonad t) => Stream (Kleisli t) () a -> t [a]
run s = runUnsafe s (repeat ())

----------------------------------------

main :: IO ()
main = return ()
```