# Differentiable Causal Computations via Delayed Trace

David Sprunger
National Institute of Informatics Tokyo, Japan 100-0003
Email: sprunger@nii.ac.jp

Shin-ya Katsumata
National Institute of Informatics Tokyo, Japan 100-0003
Email: s-katsumata@nii.ac.jp

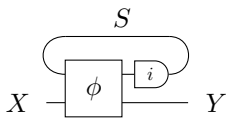
***Abstract*—We investigate causal computations, which take sequences of inputs to sequences of outputs such that the $n$th output depends on the first $n$ inputs only. We model these in category theory via a construction taking a Cartesian category $\mathbb{C}$ to another category $\mathrm{St}(\mathbb{C})$ with a novel trace-like operation called "delayed trace", which misses yanking and dinaturality axioms of the usual trace. The delayed trace operation provides a feedback mechanism in $\mathrm{St}(\mathbb{C})$ with an implicit guardedness guarantee.**

**When $\mathbb{C}$ is equipped with a Cartesian differential operator, we construct a differential operator for $\mathrm{St}(\mathbb{C})$ using an abstract version of backpropagation through time, a technique from machine learning based on unrolling of functions. This obtains a swath of properties for backpropagation through time, including a chain rule and Schwartz theorem. Our differential operator is also able to compute the derivative of a stateful network without requiring the network to be unrolled.**

***Index Terms*—delayed trace operators, Cartesian differential categories, recurrent neural networks, backpropagation through time, signal flow graphs**

## I. Introduction

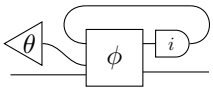
Many objects of study in computer science, such as Mealy machines, clocked digital circuits, signal flow graphs, discrete-time feedback loops, and recurrent neural networks, compute a *stateful* and particularly a *causal* function of their inputs, meaning the output of the function at a particular time may depend on not only the current input but also all inputs received by the device up to that time. They share a basic operational scheme, depicted in the following diagram (which is to be read left-to-right):




Here the box labeled $\phi$ is a (sub)device which takes an $S$-value at its upper left interface and an $X$-value at its lower left interface and produces output $S$- and $Y$-values at its right interfaces. The differently-shaped box labeled $i$ is our depiction of a *delay gate*, a device which stores the value provided to its left boundary and emits it one step later at its right boundary, initially emitting the value $i$. The whole device, which we call $\Phi$, receives sequences of $X$-valued inputs at the left and emits sequences of $Y$-valued outputs at the right, storing its internal state in the delay gate.

978-1-7281-3608-0/19/$31.00 

A recurrent neural network has inputs of two types: data inputs and parameters. *Training* a neural network means finding parameter values $\theta$ so that when $\theta$ is fixed (in the diagram below by the triangular device which emits $\theta$ constantly), the resulting function of data inputs has a desired behavior.




The key insight of *gradient-based training* is that the derivative of $\Phi$ with respect to $\theta$ gives an accurate prediction about how the output of $\Phi$ will change in response to a small change in $\theta$, allowing the trainer to make iterative small changes to $\theta$ to drive the network to a desired behavior.

This idea works perfectly for feedforward (stateless) neural networks. Recurrent neural networks require a workaround, however, due to the fact that classical differentiation does not work on stateful functions (or must be performed in an infinite dimensional vector space).

The usual workaround is to first *unroll* $\Phi$ into a sequence of stateless functions, to which classical differentiation can be applied [21]. To be more precise, think of $\Phi$ as the solution to the following recurrence relation:

$$(s_{k+1}, y_k) = \phi(s_k, x_k) \text{ where } s_0 = i.$$

Let $\phi_S = \pi_0 \circ \phi$ and $\phi_Y = \pi_1 \circ \phi$. Then the unrolling of $\Phi$ is the sequence $\phi_k : X^{k+1} \to Y$ given by

$$\begin{aligned} \phi_0(x_0) &= \phi_Y(i, x_0) \\ \phi_1(x_0, x_1) &= \phi_Y(\phi_S(i, x_0), x_1) \\ \phi_2(x_0, x_1, x_2) &= \phi_Y(\phi_S(\phi_S(i, x_0), x_1), x_2) \\ &\vdots \end{aligned} \tag{1}$$

When the gradient of $\Phi$ is needed at an input of length $k$ by a trainer, the gradient of $\phi_k$ at that input is used instead.

This is a useful procedure, known in the machine learning literature as *backpropagation through time* (BPTT) [34]. However, its ad-hoc nature raises some fundamental questions, the principal one we address here being: **Does BPTT have the usual properties of differentiation, or is it just a process involving differentiation?** That is, does this unroll-then-differentiate procedure have a chain rule, a sum rule, a notion of partial derivative, etc., or is it merely an empirically useful process using derivatives?

We show that BPTT has the properties of differentiation mentioned above and more. In particular, we are able to state the derivative of a stateful function as another stateful function, rather than a sequence of stateless functions. Roughly speaking, we accomplish this by taking advantage of the fact that the unrolling above is an iterated composition of $\phi$ with itself, and therefore its componentwise derivative can be "re-rolled" back into a single stateful function.

**Outline.** Our first main contribution is to give a construction which extends any given (Cartesian) category $\mathbb{C}$, representing stateless functions, to a new category $\mathrm{St}(\mathbb{C})$ of stateful functions, particularly computations extended through discrete time (definition 11). This $\mathrm{St}(-)$ construction captures causal functions as a special instance (theorem 14), and captures other stateful devices like Mealy machines and recurrent neural networks.

A distinctive feature of this construction includes the loop-with-delay gate seen in the first diagram, which we will more formally call a *delayed trace operator* (definition 19). This delayed trace satisfies many of the properties of its better-known cousin, the trace operator of Joyal et al. [25] (proposition 20), but is missing the yanking condition and satisfies a modified form of dinaturality (theorem 21).

Our second major contribution is to give an abstract form of differentiation in this category of stateful computations. A key result of this paper is that if $\mathbb{C}$ is a *Cartesian differential category* [4], then so is $\mathrm{St}(\mathbb{C})$ (theorem 38). In particular, this differential operator matches the results obtained by unrolling-then-differentiating as in BPTT (theorem 39). The definition of Cartesian differential categories packages many of the classic properties of derivatives in a convenient abstract unit. Hence, showing that $\mathrm{St}(\mathbb{C})$ is a Cartesian differential category implicitly obtains a slew of fundamental results for differentiation of stateful computations.

## Related Work

**Categorical models of causal computations.** *Signal flow graphs* are a widely used model of causal computation, especially in synchronous digital circuits and signal processing [10], [31]. The formation of loop paths in signal flow graphs are often restricted so that each loop path must go through at least one (initialized) delay gate. The delayed trace operator in $\mathrm{St}(\mathbb{C})$ in this paper embodies this principle.

A line of coalgebraic study of signal flow graphs by Rutten [32], [33], Milius [29], Hansen et al. [22] and Basold et al. [2] and many others achieve characterisations of computable streams by signal flow graphs. This coalgebraic perspective makes it possible to apply powerful coalgebraic techniques to analyse the behaviour of signal flow graphs. Our categorical work, on the other hand, regards signal flow graphs as morphisms in a certain category, and focuses on the categorical structures realising these flow graphs.

An axiomatic system for representing *digital circuits* based on monoidal category theory has been proposed by Ghica et al. [18], [19]. Their system is an extension of a traced cartesian category with a few structural morphisms that implement wire join and delay gate. Their delay gates do not support arbitrary initialization, but they can also represent interesting well-defined digital circuits using general loops without delay gates. The precise relationship between their axiomatic system and our categorical construction is not clear yet, and it is an interesting topic to investigate.

A category whose morphisms are realized by *Mealy machine*-like transducers is constructed in the memoryful GoI by Hoshino et al. [24]. Their transducers, represented as functions of type $S \times A \to T(S \times B)$, extend deterministic Mealy machines with the ability to perform computational effects represented by the monad $T$. The machine type considered in our work does not support these abstract computational effects. Another technical difference from our work is that the monoidal structure on their category of transducers is based on finite coproducts in order to realize the particle-style trace operator for the GoI interpretation, whereas our work uses finite products.

Connections between category theory and *machine learning* have recently attracted significant interest, for example [16]. A notable example is [15], where Elliot studies automatic differentiation (AD) from the perspective of functional programming and category theory, obtaining Cartesian functors representing many AD algorithms.

Recently, Kissinger and Uijlen reformulated the concept of causality in *quantum physics* in a class of compact closed categories [27]. Starting from a compact closed category with some extra structure, they refine it to a *-autonomous category so that morphisms there respect causal constraints.

**State and feedback.** Categorical constructions enhancing a basic category with some form of statefulness go back to Katis, Sabadini and Walters [26]. Similar constructions appear in the signal flow graph literature and the growing graphical linear algebra literature [5]. This line of work has a tradition of naming such constructions $\mathrm{St}(-)$. The construction given in the latter work [5] is very close to our $\mathrm{St}$ construction. When it is applied to a compact closed PROP, it adjoins a one-input one-output generator to the PROP.

Zanasi studies the PROP $\mathbb{IH}_R$ of *interacting Hopf algebras* over a ring $R$ in his PhD thesis [35]. The expressive power of this PROP is demonstrated by encoding various graphical systems into $\mathbb{IH}_R$ [6], [7]. When $R$ is the polynomial ring $k[x]$ over a field $k$, the PROP $\mathbb{IH}_{k[x]}$ admits delay gates, and the trace-with-delay operation (which he called *z-feedback operator*) is definable [6, Definition 7] and occupies a role analogous to the delay gate in our work.

Guardedness conditions on computations with feedback are a common theme [1], [30]. Goncharov and Schröder developed the theory of *guarded traced categories* to formalize this phenomena in [20]. The key idea is to restrict Joyal et al.'s trace operator [25] to a class of *guarded morphisms*, which are an abstractly given class of morphisms satisfying the guardedness condition. It is interesting to see the relationship between guarded trace operator and the delayed trace operator, and the key in this comparison is the treatment of the initial state.

The idea of using tiles as representations of computation steps is pursued in the *tile models* by Gadducci and Montanari [17]. In their model, each tile $f : A \xrightarrow[S']{S} B$ represents a state transition from $A$ to $B$, while $S$ and $S'$ are the trigger of and effect of this transition, respectively. In our work, $S$ and $S'$ denote types of values stored across clock ticks.

**Categorical models of differentiation.** Inspired by the semantics of differential $\lambda$-calculus and differential proof nets by Ehrhard and Regnier [13], [14], Blute, Cockett and Seely categorically formalized the differentiation operator in analysis. The formalization was first given in the categories where morphisms denote linear maps [3]. Later, they introduced a new axiomatization [4] based on cartesian monoidal category where morphisms denote possibly non-linear maps. This paper is based on the latter work, and adopts more recent reformulations of differentiation operators studied in [8], [11].

## Preliminaries

We assume familiarity with basic category theory. If $\mathbb{C}$ is a category, we write $|\mathbb{C}|$ to denote its objects, and $\mathbb{C}(X, Y)$ to denote a homset for $X, Y \in |\mathbb{C}|$. We may abbreviate an identity map $\mathrm{id}_X$ to the name of its object, $X$.

If $\mathbb{C}$ is a cartesian category, we write $1$ for its terminal object, $!_X : X \to 1$ for the unique maps to 1, and $\times$ for the product bifunctor. The tupling of morphisms $f_i : Y \to X_i$ for $i \in \{0, 1\}$ is denoted by $\langle f_0, f_1 \rangle$. Projections are denoted by $\pi_i^{X_0, X_1} : X_0 \times X_1 \to X_i$ ($i \in \{0, 1\}$), and we drop the superscript when it is obvious from context. The symmetry map on products is $\sigma_{X,Y} : X \times Y \to Y \times X$.

Bold metavariables—$\mathbf{X}, \mathbf{s}$, etc.—denote sequences of mathematical objects, indexed by $\mathbb{N}$. The $i$th component of a sequence is $\mathbf{X}_i$. By $\bigcirc \mathbf{X}$ we mean the tail of $\mathbf{X}$, namely $\bigcirc \mathbf{X} = \mathbf{X}_1, \mathbf{X}_2, \cdots$. In addition to Roman-letter subscripts, we use a bullet $\bullet$ as an special index variable, which can be bound by the sequence-forming bracket notation given next.

Let $e$ be an expression containing some dotted sequence metavariables $\mathbf{X}_\bullet, \mathbf{Y}_\bullet \cdots$. By $[e]$ we mean the infinite sequence obtained by substituting $0, 1, 2 \cdots$ for $\bullet$. For instance,

$$(i, [\mathbf{x}_\bullet + \mathbf{y}_\bullet]) \text{ is } (i, \mathbf{x}_0 + \mathbf{y}_0,\ \mathbf{x}_1 + \mathbf{y}_1,\ \cdots)$$
$$[(i, \mathbf{x}_\bullet + \mathbf{y}_\bullet)] \text{ is } (i, \mathbf{x}_0 + \mathbf{y}_0),\ (i, \mathbf{x}_1 + \mathbf{y}_1),\ \cdots$$

When $e$ contains at least one dotted sequence metavariable, we may omit the outermost $[-]$, so $[\mathbf{X}_\bullet \times \mathbf{Y}_\bullet]$ may be written as $\mathbf{X}_\bullet \times \mathbf{Y}_\bullet$. This omission is not allowed when $e$ contains no such variable; otherwise we would confuse ordinary expressions (like $x + y$) and constant infinite sequences (like $[x + y] = x + y, x + y, \cdots$).

A mathematical formula $\phi$ containing dotted sequence metavariables represents the conjunction $\bigwedge_{i \in \mathbb{N}} \phi[i/\bullet]$. For instance, $\mathbf{Z}_\bullet = \mathbf{X}_\bullet \times \mathbf{Y}_\bullet$ means $\forall i \in \mathbb{N}\ .\ \mathbf{Z}_i = \mathbf{X}_i \times \mathbf{Y}_i$.

## II. Extending Cartesian Categories along Discrete Time

Before jumping into the depths of categorical abstraction, we take a moment to think about different kinds of functions on sequences and particularly where causal functions lie.

One natural way to obtain functions on sequences is to consider the category $\mathbf{Set}^{\mathbb{N}}$, the countable product category of $\mathbf{Set}$. In this category, each morphism $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ consists of independent components $\mathbf{f}_k : \mathbf{X}_k \to \mathbf{Y}_k$ for all $k \in \mathbb{N}$, each of which compute a single entry in the output sequence.

These are certainly functions taking sequences to sequences in a causal manner, but the fact that each of the components of $\mathbf{f}$ are independent means the $k$th output of $\mathbf{f}$ depends only on the $k$th input, not on all inputs before $k$. Therefore, some causal functions of sequences, such as computing a running average, are missing from this class.

Another natural idea would be to take all the functions in homsets $\mathbf{Set}(\prod \mathbf{X}, \prod \mathbf{Y})$ for arbitrary $\mathbf{X}, \mathbf{Y} \in |\mathbf{Set}^{\mathbb{N}}|$. This class is too big—non-causal functions such as `tl` : $(x_0, x_1, \ldots) \mapsto (x_1, x_2, \ldots)$ are present there. Therefore, we must do something a bit more complex to obtain the class of causal functions somewhere between these two.

To obtain causal functions, we return to our original idea, $\mathbf{Set}^{\mathbb{N}}$, and add objects in the domain and codomain of each component of $\mathbf{f}$ representing communication channels with its neighbouring components, like

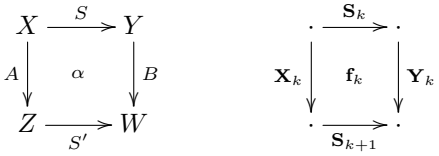
$$\mathbf{f}_k : \mathbf{S}_k \times \mathbf{X}_k \to \mathbf{S}_{k+1} \times \mathbf{Y}_k. \tag{2}$$

To start this computation, we also need to provide an initial state $i : 1 \to \mathbf{S}_0$. We call the pair $(i, \mathbf{f})$ a *stateful morphism sequence*. We will see causal functions are equivalence classes of these stateful morphism sequences (theorem 14).

Though these are functions on sequences, it will often be convenient to pretend that these sequences are produced one element at a time, synchronized by some clock signal. Thus, since the function $\mathbf{f}_k$ above computes the $k$th element in the sequence, we may refer to it as producing a value *at clock tick* $k$. Similarly, we refer to the element of state passed from $\mathbf{f}_k$ to $\mathbf{f}_{k+1}$ as being kept *across clock ticks*, and other such language. In this way, computing functions of sequences can also be thought of as performing discrete timed computations.

There is a clear distinction between the role of $\mathbf{X}_k/\mathbf{Y}_k$ and $\mathbf{S}_k/\mathbf{S}_{k+1}$—the former objects are the types of *values* flowing through $\mathbf{f}_k$ at clock tick $k$, while the latter objects are the types of *states* passed across clock ticks. We organize these two different kinds of information flow using special two-dimensional categories called *double categories* [12].

Roughly speaking, double categories consist of 0-cells (objects), two types of 1-cells (horizontal and vertical morphisms), and 2-cells (tiles) which go between pairs of horizontal and vertical 1-cells. These 2-cells are often drawn like below (left).



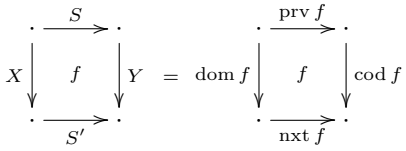

These tiles can be composed along either common vertical 1-cells (*horizontal composition*) or common horizontal 1-cells (*vertical composition*). Having these two distinct types of composition is the essential and only reason for using a double

category in this paper, so that we can use one composition for composition *within* a clock tick and the other for composition *across* clock ticks. We will not be using any results of higher category theory or further higher-dimensional abstractions.

Our double category will therefore have a particularly simple structure, with 2-cells as above (right). We have a dummy 0-cell ($\cdot$), objects from $\mathbb{C}$ as 1-cells, representing values when oriented vertically and states when oriented horizontally, and functions $\mathbf{f}_k$ on states and values in the tiles.

**Definition 1** *Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category. The double category $Dbl(\mathbb{C})$ is defined as follows:*

- *$\cdot$ is the only object (0-cell)*
- *Horizontal and vertical 1-cells are both given by objects of $\mathbb{C}$, composed with $\times$, and have $1$ as the identity.*
- *A 2-cell $f$ with source horizontal 1-cell $S$, source vertical 1-cell $X$, target horizontal 1-cell $S'$, and target vertical 1-cell $Y$ is a morphism $\phi \in \mathbb{C}(S \times X, S' \times Y)$.*




*As indicated above, we denote the source and target 1-cells of $f$—$S, X, S'$, and $Y$—by $\mathrm{prv}\, f, \mathrm{dom}\, f, \mathrm{nxt}\, f$, and $\mathrm{cod}\, f$, respectively. We will generally denote a 2-cell by $f : X \xrightarrow[S']{S} Y$. We call $\phi$ the* underlying morphism *of $f$, while $U$ is the operation taking a 2-cell to its underlying morphism, so $Uf = \phi$.*

- *The horizontal composition of 2-cells, say the $f$ above before $g : Y \xrightarrow[T']{T} Z$, is $g{*}f : X \xrightarrow[S' \times T']{S \times T} Z$ with underlying morphism*

$$U(g{*}f) \triangleq (S' \times Ug) \circ (\sigma_{S',T} \times Y) \circ (T \times Uf) \circ (\sigma_{S,T} \times X).$$

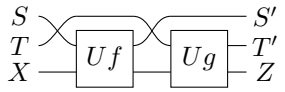
- *The vertical composition of 2-cells, say the $f$ above before $h : V \xrightarrow[S'']{S'} W$, is $f; h : X \times V \xrightarrow[S'']{S} Y \times W$ with underlying morphism*

$$U(f;h) \triangleq (S'' \times \sigma_{Y,W}) \circ (Uh \times Y) \circ (S' \times \sigma_{V,Y}) \circ (Uf \times V).$$

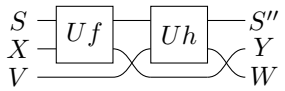
*NB: vertical composition is given in relational composition order while horizontal composition is given in functional composition order.*

The strictness of $\times$ is important in this definition so that 1-cell composition is associative. It is possible that a better definition for $Dbl(\mathbb{C})$ might obviate this technical restriction, but for now we sacrifice a bit of generality to avoid working with associators and unitors. When we want to use this construction on a non-strict Cartesian category (like $\mathbf{Set}$), we can always find an equivalent Cartesian category which *is* strict, so this is not too big of a sacrifice anyway.

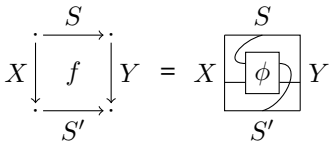
String diagrams for the underlying $\mathbb{C}$-morphisms of horizontal and vertical composites may be helpful to digest this definition. The underlying morphism of the horizontal composition $g * f$ is:




While for vertical composition, we have $U(f;h)$ as below:




A 2-cell $f$ of $Dbl(\mathbb{C})$ is determined by its underlying morphism $\phi$ from $\mathbb{C}$. To stress this, we often draw $\phi$ inside the tile, with its inputs and outputs connected to corresponding edges:

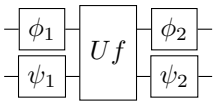



Horizontal composition of 2-cells is composition along values like $\mathbf{X}_\bullet$ or $\mathbf{Y}_\bullet$, and we think of as occurring within a single clock tick. Vertical composition is composition along states like $\mathbf{S}_\bullet$, and occurs across clock ticks.

**Definition 2** *For $\phi \in \mathbb{C}(X, Y)$, the 2-cells $\phi^h : X \xrightarrow[1]{1} Y$ and $\phi^v : 1 \xrightarrow[Y]{X} 1$ have $U(\phi^h) \triangleq \phi \triangleq U(\phi^v)$.*

These operations sending $\mathbb{C}$-morphisms to 2-cells in $Dbl(\mathbb{C})$ are particularly useful. (Note first that $\mathrm{id}^h_X$ are the identities for horizontal composition, and similarly $\mathrm{id}^v_S$ are the identities for vertical composition!) More practically, 2-cells of the form $\phi^h$ modify values only, while 2-cells of the form $\phi^v$ modify states only, as shown in the following lemma.

**Lemma 3** *If $f : X \xrightarrow[S']{S} Y$ is a $Dbl(\mathbb{C})$ 2-cell, $\phi_1 \in \mathbb{C}(T, S)$, $\phi_2 \in \mathbb{C}(S', T')$, $\psi_1 \in \mathbb{C}(W, X)$, and $\psi_2 \in \mathbb{C}(Y, Z)$, then the underlying morphism of $\phi^v_1; (\psi^h_2 * f * \psi^h_1); \phi^v_2$ has the following string diagram in $\mathbb{C}$:*




Note that if $\phi_2$ and $\psi_2$ are identities, the composed 2-cell above is denoted $\phi^v_1; f * \psi^h_1$. This compact notation for precomposition in both dimensions is a powerful notational advantage of having the ; and $*$ operators take their arguments in different orders.

### A. Stateful Morphism Sequences and Extensional Equivalence

Each 2-cell of the double category $Dbl(\mathbb{C})$ represents an individual component computing a single output value in a time-extended computation, like that of eq. (2). To represent a whole causal computation, we collect together countably many of these components into a *stateful morphism sequence*.

**Definition 4** *Let $\mathbf{X}, \mathbf{Y} \in |\mathbb{C}|^{\mathbb{N}}$ be sequences of $\mathbb{C}$-objects. A* stateful morphism sequence *of type $\mathbf{X} \to \mathbf{Y}$ is a pair $(i, \mathbf{s})$ of a sequence $\mathbf{s}$ of 2-cells in $Dbl(\mathbb{C})$ and a $\mathbb{C}$-morphism $i : 1 \to \mathrm{prv}\, \mathbf{s}_0$ such that*

$$\mathrm{dom}\, \mathbf{s}_\bullet = \mathbf{X}_\bullet, \quad \mathrm{cod}\, \mathbf{s}_\bullet = \mathbf{Y}_\bullet, \quad \mathrm{nxt}\, \mathbf{s}_\bullet = \mathrm{prv}\, \mathbf{s}_{\bullet+1}.$$

*The* state sequence *of* $(i, \mathbf{s})$ *is* $\mathrm{st}(i, \mathbf{s}) \triangleq [\mathrm{prv}(\mathbf{s}_\bullet)]$.

Note the last condition implies $\mathbf{s}_\bullet ; \mathbf{s}_{\bullet+1}$ exists, which allows each component to pass state to the next.

A stateful morphism sequence can be thought of as an infinite tower of 2-cells, each layer of which is vertically composable with adjacent layers, as depicted in fig. 1 left.

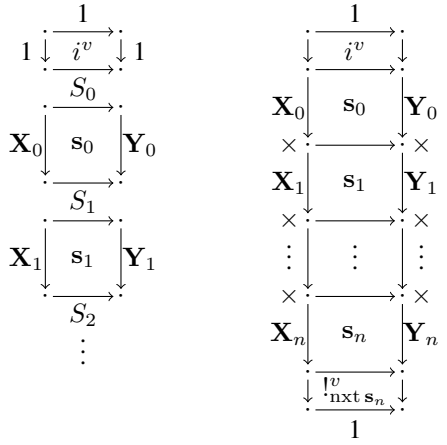



Fig. 1. The stateful morphism sequence $(i, \mathbf{s})$ and its $n$th truncation

In this representation, the "arrow of time" starts at $\mathbf{X}_0$ and points down. At the zeroth clock tick, the stateful morphism sequence receives a value at $\mathbf{X}_0$, outputs a value at $\mathbf{Y}_0$, and sets a state value of type $S_1$. Then at the first clock tick, the first layer of the stateful morphism sequence executes, using the state previously prepared by the zeroth layer.

Since we intend the state maintained by these sequences to be internal, saying $(i, \mathbf{s}) = (j, \mathbf{t})$ if and only if they are exactly the same sequence of 2-cells is not a suitable notion of equality. Ideally, if we could form the infinite vertical composition of 2-cells, the natural definition of equality of two stateful morphism sequences of type $\mathbf{X} \to \mathbf{Y}$ would be to compare the underlying $\mathbb{C}$-morphisms of the infinite composition, meaning $(i, \mathbf{s}) = (j, \mathbf{t})$ if and only if

$$U(i^v; \mathbf{s}_0; \mathbf{s}_1; \cdots) = U(j^v; \mathbf{t}_0; \mathbf{t}_1; \cdots) : \prod_{i \in \mathbb{N}} \mathbf{X}_i \to \prod_{i \in \mathbb{N}} \mathbf{Y}_i.$$

However, formalizing this infinite vertical composition is technically challenging, and $\mathbb{C}$ may not have countable products. We therefore instead require that all finite initial segments of the sequence match using a *truncation operation*.

**Definition 5** *The* $n$th truncation *of a stateful morphism sequence* $(i, \mathbf{s}) : \mathbf{X} \to \mathbf{Y}$ *is the* $\mathbb{C}$*-morphism*

$$\mathrm{Tc}_n(i, \mathbf{s}) \triangleq U(i^v; \mathbf{s}_0; \cdots; \mathbf{s}_n; !^v_{\mathrm{nxt}\,\mathbf{s}_n}) : \prod_{k=0}^{n} \mathbf{X}_k \to \prod_{k=0}^{n} \mathbf{Y}_k.$$

Graphically, $\mathrm{Tc}_n(i, \mathbf{s})$ is the underlying morphism of the vertical composite 2-cell depicted in fig. 1 right.[1]

[1]In fig. 1, the 2-cells on the right have been drawn with common horizontal 1-cells and vertical 1-cells composed with $\times$ to indicate the 2-cells have been composed vertically, whereas on the left the 2-cells are separate since the full infinite vertical composition may not be possible.

**Definition 6** *Two stateful morphism sequences* $(i, \mathbf{s}), (j, \mathbf{t}) : \mathbf{X} \to \mathbf{Y}$ *are* extensionally equal *iff* $\mathrm{Tc}_\bullet(i, \mathbf{s}) = \mathrm{Tc}_\bullet(j, \mathbf{t})$.

It is easy to verify that extensional equality between stateful morphism sequences is an equivalence relation.

The state sequences of extensionally equivalent stateful morphisms sequences can be different, which is good because it matches our intention and bad because it can be harder to decide whether two computation sequences are equal. Comparing truncations is always possible, but sometimes technically difficult. The following lemma has proven a useful method for establishing extensional equality.

**Lemma 7 (Shim lemma)** [2] *Suppose* $(i, \mathbf{s}), (j, \mathbf{t}) : \mathbf{X} \to \mathbf{Y}$ *are stateful morphism sequences, and* $\mathbf{b}$ *is a sequence of* $\mathbb{C}$*-morphisms such that* $\mathbf{b}_\bullet : \mathrm{prv}(\mathbf{s}_\bullet) \to \mathrm{prv}(\mathbf{t}_\bullet)$*,*

$$\mathbf{b}_0 \circ i = j, \text{ and } \mathbf{s}_\bullet ; \mathbf{b}^v_{\bullet+1} = \mathbf{b}^v_\bullet ; \mathbf{t}_\bullet.$$

*Then* $(i, \mathbf{s})$ *and* $(j, \mathbf{t})$ *are extensionally equivalent.*

**Proof** Show by induction that

$$i^v; \mathbf{s}_0; \cdots; \mathbf{s}_n; \mathbf{b}^v_{n+1}; !^v_{\mathrm{nxt}\,\mathbf{t}_n} = j^v; \mathbf{t}_0; \cdots; \mathbf{t}_n; !^v_{\mathrm{nxt}\,\mathbf{t}_n}$$

Unrolling and truncation are related operations, and in fact we can extend unrolling to general stateful morphism sequences.

**Definition 8** *Let* $(i, \mathbf{f}) : \mathbf{A} \to \mathbf{B}$ *be a stateful morphism sequence. Its* $k$-th unrolling *is the* $k$*th projection of the* $k$*th truncation:* $\mathrm{Un}_k(i, \mathbf{f}) \triangleq \pi_k \circ \mathrm{Tc}_k(i, \mathbf{f}) : \prod_{n=0}^{k} \mathbf{A}_n \to \mathbf{B}_k$.

For instance, the recurrently defined functions $\phi_k$ in section I are unrollings of a certain stateful morphism sequence involving $\phi$ and $i$.

### *B. Category of Causal Morphisms*

We are ready to construct our category of causal morphisms using stateful morphism sequences and extensional equality between them.

**Definition 9** *The* identity stateful morphism sequence $\mathrm{id}_{\mathbf{X}}$ *is* $(\mathrm{id}_1, [(\mathrm{id}_{\mathbf{X}_\bullet})^h])$ *for all* $\mathbf{X} \in |\mathbb{C}|^{\mathbb{N}}$.

*The* composition *of two stateful morphism sequences* $(i, \mathbf{s}) : \mathbf{Y} \to \mathbf{Z}$ *and* $(j, \mathbf{t}) : \mathbf{X} \to \mathbf{Y}$ *is*

$$(i, \mathbf{s}) \circ (j, \mathbf{t}) \triangleq (\langle j, i \rangle, [\mathbf{s}_\bullet * \mathbf{t}_\bullet]) : \mathbf{X} \to \mathbf{Z}.$$

As usual, we may denote $\mathrm{id}_{\mathbf{X}}$ by $\mathbf{X}$.

In our "tower of 2-cells" representation, the composition of stateful morphism sequences is in fig. 2. Note the state sequence of the composite is the componentwise product of the original state sequences.

**Lemma 10** *Composition of stateful morphism sequences is well-defined on extensional equivalence classes. Further, (the*

[2]A shim is a little piece of material used to align two items, such as a sliver of wood between a door frame and surrounding wall studs. In this case, $\mathbf{b}$ is the shim, and it adjusts the state spaces of the two stateful morphism sequences.

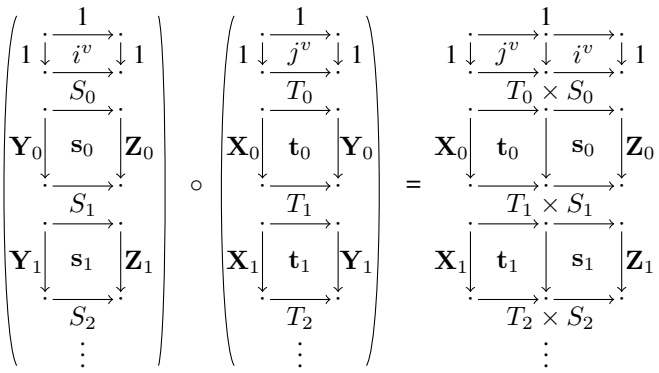


Fig. 2. Composition of stateful morphism sequences

*extensional equivalence class of)* $\mathrm{id}_{\mathbf{X}}$ *is the unit for the composition operation.*

**Definition 11** *Given a Cartesian category* $\mathbb{C}$*, its* causal extension *is a category* $\mathrm{St}(\mathbb{C})$ *where*

- *objects are* $|\mathbb{C}|^{\mathbb{N}}$*, that is,* $\mathbb{N}$*-indexed families of* $\mathbb{C}$*-objects,*
- *morphisms are extensional equivalence classes of stateful morphism sequences,*
- *identities and composition are the extensions of those in definition theorem 9 to the extensional equivalence classes by lemma 10.*

We will justify our use of the word "causal" by establishing a connection to the existing notion of causal functions in theorem 14, but first we establish some properties of $\mathrm{St}(\mathbb{C})$.

The category $\mathbb{C}^{\mathbb{N}}$ is naturally included into $\mathrm{St}(\mathbb{C})$ via the functor $H : \mathbb{C}^{\mathbb{N}} \to \mathrm{St}(\mathbb{C})$:

$$H\mathbf{X} = \mathbf{X}, \qquad H\mathbf{f} = (\mathrm{id}_1, [\mathbf{f}^h_\bullet]).$$

We call the morphisms in $\mathrm{St}(\mathbb{C})$ of the form $H\mathbf{f}$ *stateless morphisms*, since they can be realized by a stateful morphism sequence whose state sequence is constantly 1, i.e. $[1]$.

**Proposition 12** $\mathrm{St}(\mathbb{C})$ *is Cartesian, and* $H$ *is finite-product preserving.*

**Proof** In $\mathrm{St}(\mathbb{C})$, the final object is $[1]$ and the final map from $\mathbf{X}$ is $H[!_{\mathbf{X}_\bullet}]$. Products and projection are also componentwise: our chosen $\mathrm{St}(\mathbb{C})$ product $\mathbf{X} \times \mathbf{Y}$ is the sequence $[\mathbf{X}_\bullet \times \mathbf{Y}_\bullet]$ of $\mathbb{C}$ products, with $\pi_n \triangleq H[\pi_n^{\mathbf{X}_\bullet, \mathbf{Y}_\bullet}]$ for $n \in \{0, 1\}$. □

### C. Morphisms in $\mathrm{St}(\mathbf{Set})$ and Causal Functions

We claim that morphisms of $\mathrm{St}(\mathbb{C})$ represent *causal* computations, whose outputs depend only on past inputs and states. To justify this claim, we compare $\mathbf{Set}$-theoretic causal functions and morphisms in $\mathrm{St}(\mathbf{Set})$. Note that $\mathbf{Set}$ is not a *strict* Cartesian category but is still Cartesian. Therefore, to apply our constructions we use a strictified (but categorically equivalent) version of $\mathbf{Set}$, which we will denote by $\mathbf{Set}^s$.

First, we need a precise definition of causality for functions on sequences, which we adapt from [22]. First, for $\mathbf{x}, \mathbf{y} \in A^{\mathbb{N}}$, by $\mathbf{x} \equiv_n \mathbf{y}$ we mean $\mathbf{x}$ and $\mathbf{y}$ match in the first $n$ positions, that is, $\mathbf{x}_i = \mathbf{y}_i$ holds for any $i \leq n$.

**Definition 13 ([22])** *Let* $A$ *and* $B$ *be sets. A function* $f : A^{\mathbb{N}} \to B^{\mathbb{N}}$ *is* causal *if for any* $\mathbf{x}, \mathbf{y} \in A^{\mathbb{N}}$*,*

$$\forall n \in \mathbb{N} \,.\, \mathbf{x} \equiv_n \mathbf{y} \implies f(\mathbf{x}) \equiv_n f(\mathbf{y}).$$

Causal functions are closed under composition and so there is a category, $\mathbf{Caus}$, whose objects are sets and morphisms from $A$ to $B$ are causal functions $f : A^{\mathbb{N}} \to B^{\mathbb{N}}$.

**Theorem 14** *The full subcategory of* $\mathrm{St}(\mathbf{Set}^s)$ *generated by objects of the form* $[A]$ *for* $A \in |\mathbf{Set}^s|$ *is isomorphic to* $\mathbf{Caus}$*.*

The proof can be found in the appendix.

### D. The Category $\mathrm{St}_0(\mathbb{C})$ and Deterministic Mealy Machines

The input, output, and state types for a $\mathrm{St}(\mathbb{C})$ morphisms can vary over time. This is a crucial property to capture all causal functions, as can be seen in the proof of theorem 14. However, many computational models are more regular, having fixed input, output, and state types, and additionally executing the same function at each time step. Thus it may appear we have overgeneralized. Luckily, we can recover these regular causal functions in a subcategory of $\mathrm{St}(\mathbb{C})$:

**Definition 15** *The subcategory* $\mathrm{St}_0(\mathbb{C})$ *of* $\mathrm{St}(\mathbb{C})$ *has:*

- *objects of the form* $[X]$ *for some* $X \in |\mathbb{C}|$*, and*
- *morphisms the (extensional equivalence classes of) stateful morphism sequences of the form* $(i, [f])$ *for some 2-cell* $f : X \xrightarrow[S]{S} Y$*.*

It is easy to check that this restricted class of morphisms is closed under the $\mathrm{St}(\mathbb{C})$-composition, hence $\mathrm{St}_0(\mathbb{C})$ is a well-defined subcategory. We note the Cartesian structure of $\mathrm{St}(\mathbb{C})$ restricts to $\mathrm{St}_0(\mathbb{C})$.

**Proposition 16** *The category* $\mathrm{St}_0(\mathbb{C})$ *is Cartesian, and the functor* $H_0 : \mathbb{C} \to \mathrm{St}_0(\mathbb{C})$ *is finite-product preserving.*

$$H_0 X = [X] \qquad H_0 f = (\mathrm{id}_1, [f^h]).$$

Morphisms of $\mathrm{St}_0(\mathbf{Set}^s)$ may be identified as the causal functions that can be computed by *deterministic Mealy machines*, possibly with infinitely many states. Suppose $(i, [f]) : [X] \to [Y]$ is a morphism in $\mathrm{St}_0(\mathbf{Set}^s)$. The set $S = \mathrm{cod}\, i$ is the set of states of the Mealy machine, $i : 1 \to S$ is the initial state, and the function $f : S \times X \to S \times Y$ is the deterministic transition-and-output function computing the next state and output from the current state and input. The composition of morphisms in $\mathrm{St}_0(\mathbf{Set}^s)$ corresponds to the *series (cascade) composition* of Mealy machines.

One useful operation on stateful morphism sequences is *unrolling*.

**Definition 17** *Let* $(i, \mathbf{f}) : \mathbf{A} \to \mathbf{B}$ *be a stateful morphism sequence. Its* $k$-th unrolling *is the* $k$*th projection of the* $k$*th truncation:* $\mathrm{Un}_k(i, \mathbf{f}) \triangleq \pi_k \circ \mathrm{Tc}_k(i, \mathbf{f}) : \prod_{n=0}^{k} \mathbf{A}_n \to \mathbf{B}_k$*.*

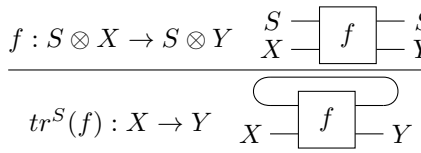
For instance, the recurrently defined functions $\phi_k$ in eq. (1) in section I are unrollings: $\phi_k = \mathrm{Un}_k(i, [\phi])$.

Note that the truncation operation $\mathrm{Tc}$ can be extended to $\mathrm{St}(\mathbb{C})$-morphisms, as it is well-defined on extensional equivalence classes.

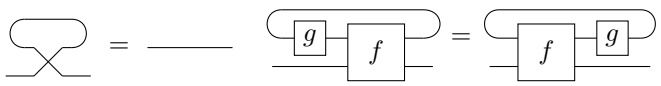
## III. Delayed Trace Operator

The category $\mathrm{St}(\mathbb{C})$ carries interesting structure that may not be present in $\mathbb{C}$—it has a *delayed trace operator*. This is related to Joyal et al.'s *trace operator* [25], which we briefly recall here. The trace operator is a structure on braided monoidal categories, namely a collection of functions $tr^S : \mathbb{C}(S \otimes X, S \otimes Y) \to \mathbb{C}(X, Y)$. In the language of string diagrams, this operation forms a feedback loop at a specified pair of ports:

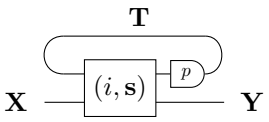
$$\frac{f : S \otimes X \to S \otimes Y}{tr^S(f) : X \to Y}$$




Interpreted as string diagrams, the equational axioms of the trace operator identify intuitively equivalent diagrams involving feedback loops. Two characteristic axioms are *yanking* (left) and *dinaturality* (right):




We will show the delayed trace operator, found in $\mathrm{St}(\mathbb{C})$, satisfies the trace operator axioms except yanking and dinaturality. In fact, the delayed trace of the symmetry yields the morphism that acts as a *delay gate*. Therefore the delayed trace (as its name suggests) may be naturally regarded as an operation that forms a feedback loop *and* inserts a delay gate in the loop path. This echoes a principle of synchronous circuit design: "all feedback loops should contain a register". We depict this operation as follows:



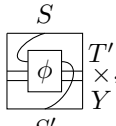

In this diagram, the half-round node is the delay gate and is filled with its initial state $p$.

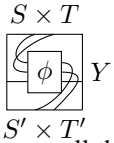
Our first step towards a delayed trace operator on $\mathrm{St}(\mathbb{C})$ is to introduce an operation on 2-cells that converts parts of the value types into the state space of a computation step.

**Definition 18** *Let $f : T \times X \xrightarrow[S']{S} T' \times Y$ be a 2-cell in $Dbl(\mathbb{C})$. The* value-to-state conversion of $f$ at $(T, T')$ *is another 2-cell, denoted ${}^{T}\lfloor f \rceil_{T'}$, with the same underlying morphism but different source and target 1-cells: ${}^{T}\lfloor f \rceil_{T'} : X \xrightarrow[S' \times T']{S \times T} Y$.*

*When the objects $(T, T')$ involved in the conversion are clear from context, we drop them from the notation and write $\lfloor f \rceil$ for ${}^{T}\lfloor f \rceil_{T'}$.*

The value-to-state conversion is depicted inside the tile:

If $f =$ 

, then ${}^{T}\lfloor f \rceil_{T'} =$ 

.

The pointwise application of this operation to all the 2-cells in a stateful morphism sequence is the delayed trace operator.

**Definition 19** *Suppose $(i, \mathbf{s}) : \mathbf{T} \times \mathbf{X} \to \bigcirc \mathbf{T} \times \mathbf{Y}$ is a morphism in $\mathrm{St}(\mathbb{C})$. (Recall $\bigcirc \mathbf{T} = [\mathbf{T}_{\bullet+1}]$.) The* delayed trace of $\mathbf{s}$ along $\mathbf{T}$ with an initial state $p : 1 \to \mathbf{T}_0$ *is the following morphism in $\mathrm{St}(\mathbb{C})(\mathbf{X}, \mathbf{Y})$:*

$$\mathrm{tr}_p^{\mathbf{T}}(i, \mathbf{s}) \triangleq (\langle i, p \rangle, [{}^{\mathbf{T}_\bullet}\lfloor \mathbf{s}_\bullet \rceil_{\mathbf{T}_{\bullet+1}}]).$$

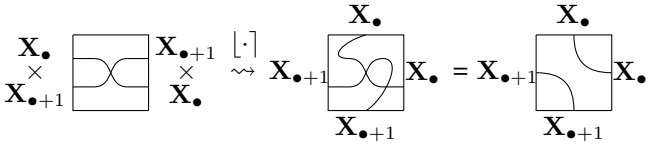
Note this operation is well-defined on extensional equivalence classes of stateful morphism sequences, and therefore is an operation on $\mathrm{St}(\mathbb{C})$ morphisms. The delayed trace of $\mathrm{St}(\mathbb{C})$ already differs from the standard monoidal trace in two ways: first, the domain and codomain types that are bound ($\mathbf{T}$ and $\bigcirc \mathbf{T}$) do not match, and second, the delayed trace also requires a designated global element $p$ called the initial state. Despite these differences, many of the trace axioms holds for the delayed trace operator.

**Proposition 20** *Suppose $(i, \mathbf{s}) : \mathbf{T} \times \mathbf{X} \to \bigcirc \mathbf{T} \times \mathbf{Y}$ is a morphism in $\mathrm{St}(\mathbb{C})$. Suppose $(h, \mathbf{r}) : \mathbf{Y} \to \mathbf{Z}$, $(j, \mathbf{t}) : \mathbf{W} \to \mathbf{X}$ and $(f, \mathbf{p}) : \mathbf{W} \to \mathbf{Z}$ are other arbitrary morphisms in $\mathrm{St}(\mathbb{C})$. Five standard axioms of monoidal trace, presented in Figure 3, hold of delayed trace.*

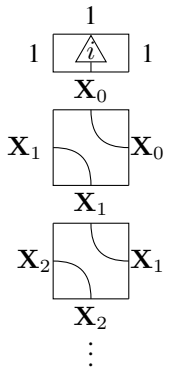
The yanking axiom of the trace operator fails for the delayed trace operator. Consider the symmetry morphism $\sigma_{\mathbf{X}, \bigcirc \mathbf{X}} : \mathbf{X} \times \bigcirc \mathbf{X} \to \bigcirc \mathbf{X} \times \mathbf{X}$ in $\mathrm{St}(\mathbb{C})$. Define its delayed trace with an initial state $i : 1 \to \mathbf{X}_0$ to be

$$r_{\mathbf{X}}(i) \triangleq \mathrm{tr}_i^{\mathbf{X}}(\sigma_{\mathbf{X}, \bigcirc \mathbf{X}}) : \bigcirc \mathbf{X} \to \mathbf{X}$$

To get a better understanding of $r_{\mathbf{X}}(i)$, we first draw the value-to-state conversion in a single 2-cell in this morphism.




Doing value-to-state conversion along the whole sequence $\sigma_{\mathbf{X}, \bigcirc \mathbf{X}}$ and supplying the initial value $i : 1 \to \mathbf{X}_0$ yields:

$r_{\mathbf{X}}(i) =$ 


We can see that the input at clock tick $k$ is output at clock tick $k + 1$. Therefore, instead of the identity, which is what $r_{\mathbf{X}}(i)$ would be if the yanking axiom held, we have a morphism that operates as a *delay gate*.

The dinaturality axiom of the trace operator also fails for the delayed trace operator. Dinaturality corresponds to sliding circuits from one end of a feedback loop to the other, but

$$
\begin{array}{lrcl}
\text{Target naturality} & (h,\mathbf{r}) \circ \mathrm{tr}_p^{\mathbf{T}}(i,\mathbf{s}) & = & \mathrm{tr}_p^{\mathbf{T}}((\mathrm{id}_{\bigcirc \mathbf{T}} \times (h,\mathbf{r})) \circ (i,\mathbf{s})) \\
\text{Source naturality} & \mathrm{tr}_p^{\mathbf{T}}(i,\mathbf{s}) \circ (j,\mathbf{t}) & = & \mathrm{tr}_p^{\mathbf{T}}((i,\mathbf{s}) \circ (\mathrm{id}_{\mathbf{T}} \times (j,\mathbf{t}))) \\
\text{Superposing} & \mathrm{tr}_q^{\mathbf{T}}(i,\mathbf{s}) \times (f,\mathbf{p}) & = & \mathrm{tr}_q^{\mathbf{T}}((i,\mathbf{s}) \times (f,\mathbf{p})) \\
\text{Vanishing 1} & \mathrm{tr}_{\mathrm{id}_1}^{[1]}(i,\mathbf{s}) & = & (i,\mathbf{s}) \\
\text{Vanishing } \times & \mathrm{tr}_q^{\mathbf{V}}(\mathrm{tr}_p^{\mathbf{U}}(i,\mathbf{s})) & = & \mathrm{tr}_{\langle q,p\rangle}^{[\mathbf{V}_\bullet \times \mathbf{U}_\bullet]}(i,\mathbf{s})
\end{array}
$$

Fig. 3. Equalities Satisfied by Delayed Trace Operator

doing so with a delay gate in the loop affects the gate's initial state. In digital circuit design, this kind of operation is called *retiming* [28], and initial states of registers is a delicate issue in this subject. The delayed trace operator satisfies the following modified dinaturality property:

**Theorem 21** *Suppose* $(i,\mathbf{s}) : \mathbf{T} \times \mathbf{X} \to \bigcirc \mathbf{U} \times \mathbf{Y}$ *and* $(j,\mathbf{g}) : \mathbf{U} \to \mathbf{T}$ *are morphisms in* $\mathrm{St}(\mathbb{C})$*. For any* $u : 1 \to \mathbf{U}_0$*,*

$$
\begin{aligned}
&\mathrm{tr}_u^{\mathbf{U}}(((j,\mathbf{g}) \times \mathbf{X}) \circ (i,\mathbf{s})) \\
&= \mathrm{tr}_{u'}^{\mathbf{T}}((i,\mathbf{s}) \circ ((j', \bigcirc \mathbf{g}) \times \mathbf{Y}))
\end{aligned}
$$

*where* $\langle j', u' \rangle = U\mathbf{g}_0 \circ \langle j, u \rangle$*.*

A special case of this modified dinaturality is an abstract version of circuit retiming, which allows us to commute properly initialized delay gates and stateless morphisms.

**Corollary 22** *For any* $\mathbf{f} : \mathbf{X} \to \mathbf{Y}$ *in* $\mathbb{C}^{\mathbb{N}}$*, and initial state* $i : 1 \to \mathbf{X}_0$*, we have* $H\mathbf{f} \circ r_{\mathbf{X}}(i) = r_{\mathbf{Y}}(\mathbf{f}_0 \circ i) \circ H(\bigcirc \mathbf{f})$*.*

The following representation result says that every morphism in $\mathrm{St}(\mathbb{C})$ can be obtained as the delayed trace of a stateless morphism.

**Theorem 23** *For any morphism* $(i,\mathbf{s})$ *in* $\mathrm{St}(\mathbb{C})$*, the following equality holds:*

$$
(i,\mathbf{s}) = \mathrm{tr}_i^{\mathrm{st}(i,\mathbf{s})}(H[U(\mathbf{s}_\bullet)])
$$

This theorem is our formalization of folklore knowledge that every synchronous digital circuit can be written as a single combinational (stateless) circuit plus a feedback loop with a register.

### A. Delayed Trace in $\mathrm{St}_0(\mathbb{C})$

The category $\mathrm{St}_0(\mathbb{C})$ is also closed under the delayed trace operator. Since $\mathbf{X} = \bigcirc \mathbf{X}$ in $\mathrm{St}_0(\mathbb{C})$, delayed dinaturality is even closer to true dinaturality.

**Corollary 24** *Suppose* $(i,[s]) : [T] \times [X] \to [U] \times [Y]$ *is a morphism in* $\mathrm{St}_0(\mathbb{C})$*, and* $(j,[g]) : [U] \to [T]$ *is another morphism in* $\mathrm{St}_0(\mathbb{C})$*. For any initial state* $u : 1 \to U$*,*

$$
\begin{aligned}
&\mathrm{tr}_u^{[U]}(((j,[g]) \times \mathrm{id}_{[X]}) \circ (i,[s])) \\
&= \mathrm{tr}_{u'}^{[T]}((i,[s]) \circ ((j',[g]) \times \mathrm{id}_{[Y]}))
\end{aligned}
$$

*where* $\langle j', u' \rangle = g \circ \langle j, u \rangle$*.*

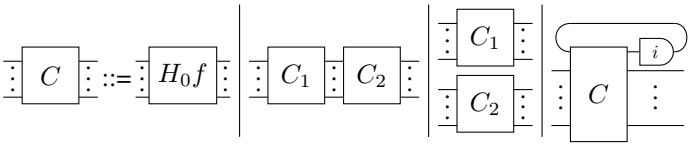
**Corollary 25** *For any* $f : X \to Y$ *in* $\mathbb{C}$*, and initial state* $i : 1 \to X$*, we have* $H[f] \circ r_{[X]}(i) = r_{[Y]}(f \circ i) \circ H([f])$*.*

### B. Diagrammatic reasoning about $\mathrm{St}_0(\mathbb{C})$ morphisms

Here we informally introduce a diagrammatic syntax for morphisms in $\mathrm{St}_0(\mathbb{C})$. Theorem 23 indicates that we can generate all $\mathrm{St}_0(\mathbb{C})$ morphisms with the following grammar:

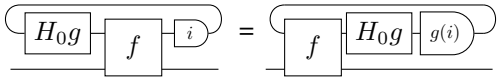
$$
\varphi ::= H_0 f | \varphi_1 \circ \varphi_2 | \varphi_1 \times \varphi_2 | \mathrm{tr}_i^{\mathbf{S}}(\varphi)
$$

where $f$ is a $\mathbb{C}$-morphism. We generate circuit diagrams with a parallel 2-dimensional grammar:

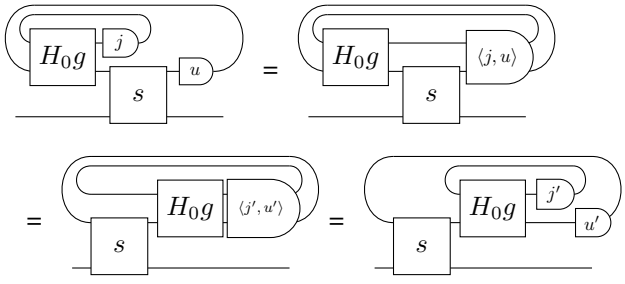



where the box labeled $H_0 f$ has $m$ inputs and $n$ outputs when $f : \prod_{k=1}^{m} A_k \to \prod_{k=1}^{n} B_k$. As is typical in string diagrams, $H_0 \mathrm{id}_X$ is depicted by a wire and $H_0 \sigma_{X,Y}$ by a wire crossing. Additionally, we depict $H_0 !_A$ and $H_0 \langle \mathrm{id}_A, \mathrm{id}_A \rangle$ with a discarder and copier: —● and —◀ .

The evident interpretation in $\mathrm{St}_0(\mathbb{C})$ of these diagrams induces an equivalence on the diagrams. For instance, as a special case of corollary 24, sliding a stateless node along a loop is possible by changing the value in the delay gate:




As an example of diagrammatic reasoning, we show that this simple delayed dinaturality plus superposing allows us to obtained delayed dinaturality for stateful circuits (theorem 21).




More formal treatment of this diagrammatic equational system can be done through the construction of the free cartesian category with the delayed trace operator. We reserve this formal axiomatization for future work, and move on to the study of the differentiablity of the causal computations realized by $\mathrm{St}(\mathbb{C})$.

## IV. Cartesian Differential Structure

In this section, we investigate differentiation in $\mathrm{St}(\mathbb{C})$. Our primary tool is the theory of Cartesian differential categories, introduced by Blute, Cockett, and Seely in [4]. We begin by recalling background.

**Definition 26 ([4])** *A* left additive category *is a Cartesian category such that every object has a designated commutative monoid structure, which we write $+_X : X \times X \to X$ and $0_X : 1 \to X$. These commutative monoids must be compatible with the Cartesian structure of the category by satisfying:*

$$0_{X\times Y} = 0_X \times 0_Y$$
$$+_{X\times Y} = (+_X \times +_Y) \circ (X \times \sigma_{Y,X} \times Y)$$

The vector space structure on Euclidean spaces is a classic example of left additive structure.

**Example 27 ([4])** *The category $\mathbf{Euc}_\infty$, whose objects are $\mathbb{R}^n$ for $n \geq 0$ and morphisms are smooth functions, is a left additive category where $+_{\mathbb{R}^n}$ is the sum of vectors in $\mathbb{R}^n$ and $0_{\mathbb{R}^n}$ is the zero vector in $\mathbb{R}^n$.*

To obtain left additive structure for $\mathrm{St}(\mathbb{C})$, it suffices to take sequences of the corresponding pieces of left additive structure for $\mathbb{C}$, much like how the Cartesian structure of $\mathbb{C}$ lifted.

**Lemma 28** *If $\mathbb{C}$ is a left additive category, so is $\mathrm{St}(\mathbb{C})$.*

Next, we introduce some helpful families of morphisms present in every Cartesian left additive category that are useful for condensing later definitions.

**Definition 29** *Let $\mathbb{C}$ be a Cartesian left additive category. For every object $X$ from $\mathbb{C}$ [or pair of objects $(X, Y)$], let*

- $\delta_{X,Y} \triangleq X \times \sigma_{Y,X} \times Y$
- $\alpha_X \triangleq \delta_{X,X} \circ (X \times X \times \Delta_X)$
- $\beta_X \triangleq X \times !_X \times X \times X$
- $\gamma_{X,Y} \triangleq (X \times \Delta_X \times Y \times Y) \circ \delta_{X,Y}$
- $\zeta_X \triangleq X \times 0_{X\times X} \times X$

Now we are ready to describe the central object of our study this section, Cartesian differential categories.

**Definition 30** *A* Cartesian differential category *is a left additive category $\mathbb{C}$ with a* Cartesian differential operator *$D : \mathbb{C}(X, Y) \to \mathbb{C}(X \times X, Y)$, satisfying:*

*CD1.* $Ds = s \times !_{\mathrm{dom}(s)}$ *for* $s \in \{X, \sigma_{X,Y}, !_X, \Delta_X, +_X, 0_X\}$
*CD2.* $Df \circ (0_X \times X) = 0_Y \circ !_X$
*CD3.* $Df \circ (+_X \times X) = +_Y \circ (Df \times Df) \circ \alpha_X$
*CD4.* $D(g \circ f) = Dg \circ (Df \times f) \circ (X \times \Delta_X)$
*CD5.* $D(f \times h) = (Df \times Dh) \circ \delta_{X,V}$
*CD6.* $DDf \circ \zeta_X = Df$
*CD7.* $DDf \circ \delta_{X,X} = DDf$
*for all $f : X \to Y$, $g : Y \to Z$, and $h : V \to W$.*

This definition of a Cartesian differential category is not exactly that of [4], but it is mostly straightfoward to check that they are equivalent. The biggest changes are in axioms CD6 and CD7, for which we have taken alternate forms given in [8, Proposition 4.2].

**Example 31 ([4])** $\mathbf{Euc}_\infty$ *is a Cartesian differential category. The differential operator $D$ sends a smooth function $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (x_1, x_2) \mapsto Jf|_{x_2} \times x_1$, where $Jf|_{x_2}$ is the Jacobian matrix of $f$ at $x_2$.*

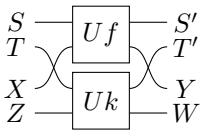
In light of the standard example, we can describe the ideas behind the CD axioms. CD1 says that the basic morphisms provided by the structure of the Cartesian left additive category are linear (in the sense that $Js|_{x_2} \times x_1 = s(x_1)$), while CD2 and CD3 express the fact that $Jf|_{x_2} \times x_1$ is linear (in the sense of linear algebra) in its $x_1$ argument. CD4 is the chain rule, while CD5 says the derivative of a parallel composition is the parallel composition of derivatives. CD6 and CD7 have to do with partial derivatives: CD7 is the symmetry of partial derivatives, and CD6 is trickier to describe exactly, but is related to the linearity of partial derivatives.

Many of the CD axioms mention the parallel composition of morphisms with $\times$. When we state these in $\mathrm{St}(\mathbb{C})$, it will be helpful to have an operation for forming parallel compositions. This motivates us to define the following operation on 2-cells.

**Definition 32** *Let $f : X \xrightarrow[S']{S} Y$ and $k : Z \xrightarrow[T']{T} W$ be arbitrary 2-cells from $Dbl(\mathbb{C})$. The* cross composition *of $f$ and $k$ is another 2-cell $f \boxtimes k : X \times Z \xrightarrow[S'\times T']{S\times T} Y \times W$ defined by*

$$f \boxtimes k = ({}^{T}\lfloor (T \times Y)^h \rceil_{T}; k) * (f; {}^{S'}\lfloor (S' \times Z)^h \rceil_{S'}).$$

It may be easier to understand cross composition by its underlying morphism:




The idea of this operation is to execute two 2-cells in parallel, without their states or values interacting with each other. We are purposefully avoiding using $\times$ for $\boxtimes$ so as not to imply there is some kind of Cartesian structure on the double category $Dbl(\mathbb{C})$.

To avoid using too many grouping symbols when disambiguating 2-cell expressions involving ;, $*$, and $\boxtimes$ we will say $\boxtimes$ binds tightest, then ;, and last $*$, so $f; g \boxtimes h * k$ means $(f; (g \boxtimes h)) * k$.

As desired, this operation implements Cartesian product in $\mathrm{St}(\mathbb{C})$.

**Lemma 33** $(i, \mathbf{f}) \times (j, \mathbf{g}) = (\langle i, j\rangle, [\mathbf{f}_\bullet \boxtimes \mathbf{g}_\bullet])$ *for all* $\mathrm{St}(\mathbb{C})$ *morphisms $(i, \mathbf{f})$ and $(j, \mathbf{g})$.*

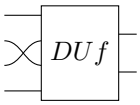
We can now start defining the Cartesian differential operator on $\mathrm{St}(\mathbb{C})$. For the remainder of this section we assume $\mathbb{C}$ is a Cartesian differential category and let $D$ be its differential operator. We start by defining our differential operator within a time step, by giving some operations on 2-cells.

**Definition 34** *We define two endofunctions on 2-cells from $Dbl(\mathbb{C})$. The first, $\mathcal{D}_0$, takes the 2-cell $f : X \xrightarrow[S']{S} Y$ to the 2-cell $\mathcal{D}_0 f : X \times X \xrightarrow[S']{S\times S} Y$ with $U\mathcal{D}_0 f \triangleq DUf \circ \delta_{S,X}$.*

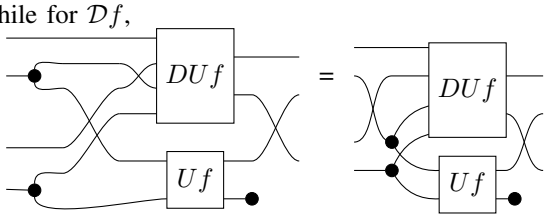
*The second, $\mathcal{D}$, takes $f$ to $\mathcal{D}f : X \times X \xrightarrow[S' \times S']{S \times S} Y$ with*

$$\mathcal{D}f \triangleq (S \times \Delta_S)^v; (\mathcal{D}_0 f \boxtimes (!_Y{}^h * f)) * (X \times \Delta_X)^h.$$

The string diagrams for the underlying morphisms of $\mathcal{D}_0$ and $\mathcal{D}$ may be easier to understand. For $\mathcal{D}_0$,




while for $\mathcal{D}f$,




The Cartesian differential operator on $\mathrm{St}(\mathbb{C})$ is based on $\mathcal{D}$, and so to prove that it is a differential operator, we need some properties of $\mathcal{D}$.

**Proposition 35** *Let $f : X \xrightarrow[S']{S} Y$, $g : Y \xrightarrow[T']{T} Z$, $h : Z \xrightarrow[S'']{S'} W$, and $k : Z \xrightarrow[T']{T} W$ be arbitrary 2-cells. The following are properties of $\mathcal{D}$:*

1) *If $\varphi \in \mathbb{C}(X, Y)$, then $\mathcal{D}(\varphi^h) = (D\varphi)^h$ and $\mathcal{D}(\varphi^v) = ((D\varphi \times \varphi) \circ (X \times \Delta_X))^v$.*
2) $(0_S \times S)^v; \mathcal{D}f * (0_X \times X)^h = (0_Y \circ !_Y)^h * f; (0_{S'} \times S')^v$
3) $(+_S \times S)^v; \mathcal{D}f * (+_X \times X)^h = \alpha^v_S; (+^h_Y * \mathcal{D}f \boxtimes \mathcal{D}f * \alpha^h_X); \beta^v_{S'}; (+_{S'} \times S')^v$
4) $\mathcal{D}(f; h) = (\mathcal{D}f; \mathcal{D}h) * \delta^h_{X,Z}$
5) $\mathcal{D}(g * f); \gamma^v_{S',T'} = \gamma^v_{S,T}; (\mathcal{D}g * (\mathcal{D}f \boxtimes f) * (X \times \Delta_X)^h)$
6) $\mathcal{D}(f \boxtimes k); \delta^v_{S',T'} = \delta^v_{S,T}; \mathcal{D}f \boxtimes \mathcal{D}k * \delta^h_{X,Z}$
7) $\zeta^v_S; \mathcal{D}\mathcal{D}f * \zeta^h_X = \mathcal{D}f; \zeta^v_{S'}$
8) $\delta^v_{S,S}; \mathcal{D}\mathcal{D}f * \delta^h_{X,X} = \mathcal{D}\mathcal{D}f; \delta^h_{S',S'}$

The method to prove these properties is conceptually simple: use the definitions of the operations on 2-cells (and properties of left additive categories and CD axioms) to check that both sides of each equation have the same boundary 1-cells and the same underlying $\mathbb{C}$-morphism. Practically, the underlying morphisms are complex, so this turns into an intense string diagram exercise, which can be found in the appendix.

An important consequence of Proposition 35(4) is the following extension to finite sequences of vertically composed 2-cells.

**Lemma 36** *Let $(f_k)_{k=0}^n$ be a finite sequence of vertically composable 2-cells. Then $\mathcal{D}(f_0; \cdots; f_n) = (\mathcal{D}f_0; \cdots; \mathcal{D}f_n) * z^h$, where $z$ is the unzipping isomorphism in $\mathbb{C}$ of type*

$$\textstyle\prod_{k=0}^n (\mathrm{dom}\, f_k \times \mathrm{dom}\, f_k) \to (\prod_{k=0}^n \mathrm{dom}\, f_k) \times (\prod_{k=0}^n \mathrm{dom}\, f_k).$$

We can now state the operator we seek on $\mathrm{St}(\mathbb{C})$.

**Definition 37** *The componentwise application of $\mathcal{D}$ to 2-cells in a $\mathrm{St}(\mathbb{C})$ morphism, $\mathcal{D}^* : (i, \mathbf{s}) \mapsto (\langle 0, i\rangle, [\mathcal{D}\mathbf{s}_\bullet])$, is a well-defined operation on $\mathrm{St}(\mathbb{C})$ morphisms of type*

$$\mathcal{D}^* : \mathrm{St}(\mathbb{C})(\mathbf{X}, \mathbf{Y}) \to \mathrm{St}(\mathbb{C})(\mathbf{X} \times \mathbf{X}, \mathbf{Y}).$$

A key contribution of this work is the fact that this operation is actually a Cartesian differential operator.

**Theorem 38** *$\mathcal{D}^*$ is a Cartesian differential operator.*

The strategy for this proof is to use the properties of $\mathcal{D}$ from Proposition 35, which were selected to be used with the Shim Lemma to obtain the CD axioms. For example, in this context, CD4 (the chain rule) states:

$$\mathcal{D}^*((j, \mathbf{g}) \circ (i, \mathbf{f})) = \mathcal{D}^*(j, \mathbf{g}) \circ (\mathcal{D}^*(i, \mathbf{f}) \times (i, \mathbf{f})) \circ (\mathbf{X} \times \Delta_{\mathbf{X}}).$$

The key step in proving this is invoking the Shim Lemma with $\mathbf{b}_\bullet = \gamma_{\mathbf{S}_\bullet, \mathbf{T}_\bullet}$. We have two conditions to check for this invocation: $\gamma_{\mathbf{S}_0, \mathbf{T}_0} \circ \langle 0_{\mathbf{S}_0}, 0_{\mathbf{T}_0}, i, j\rangle = \langle 0_{\mathbf{S}_0}, i, i, 0_{\mathbf{T}_0}, j\rangle$ and

$$\begin{aligned}&\mathcal{D}(\mathbf{g}_\bullet * \mathbf{f}_\bullet); \gamma^v_{\mathbf{S}_{\bullet+1}, \mathbf{T}_{\bullet+1}} \\ &\quad = \gamma^v_{\mathbf{S}_\bullet, \mathbf{T}_\bullet}; (\mathcal{D}\mathbf{g}_\bullet * (\mathcal{D}\mathbf{f}_\bullet \boxtimes \mathbf{f}_\bullet) * (\mathbf{X}_\bullet \times \Delta_{\mathbf{X}_\bullet})^h),\end{aligned}$$

the latter of which is a case of Proposition 35(5).

We can now prove CD4 for $\mathcal{D}^*$:

$$\begin{aligned}\mathcal{D}^*((j, \mathbf{g}) \circ (i, \mathbf{f})) &= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{T}_0}, i, j\rangle, [\mathcal{D}(\mathbf{g}_\bullet * \mathbf{f}_\bullet)]) \\ &= (\langle 0_{\mathbf{S}_0}, i, i, 0_{\mathbf{T}_0}, j\rangle, [\mathcal{D}\mathbf{g}_\bullet * (\mathcal{D}\mathbf{f}_\bullet \boxtimes \mathbf{f}_\bullet) * (\mathbf{X}_\bullet \times \Delta_{\mathbf{X}_\bullet})^h]) \\ &= \mathcal{D}^*(j, \mathbf{g}) \circ (\mathcal{D}^*(i, \mathbf{f}) \times (i, \mathbf{f})) \circ (\mathbf{X} \times \Delta_{\mathbf{X}})\end{aligned}$$

where the second line is the Shim Lemma step. The other axioms are similar and can be found in the appendix.

The following result demonstrates that our differential operator matches (up to isomorphism) the unroll-and-differentiate procedure used in backpropagation through time.

**Theorem 39** *For any morphism $(i, \mathbf{f}) : \mathbf{A} \to \mathbf{B}$ in $\mathrm{St}(\mathbb{C})$,*

$$\mathrm{Un}_k(\mathcal{D}^*(i, \mathbf{f})) = D(\mathrm{Un}_k(i, \mathbf{f})) \circ z : \prod_{n=0}^{k} (\mathbf{A}_n \times \mathbf{A}_n) \to \mathbf{B}_k,$$

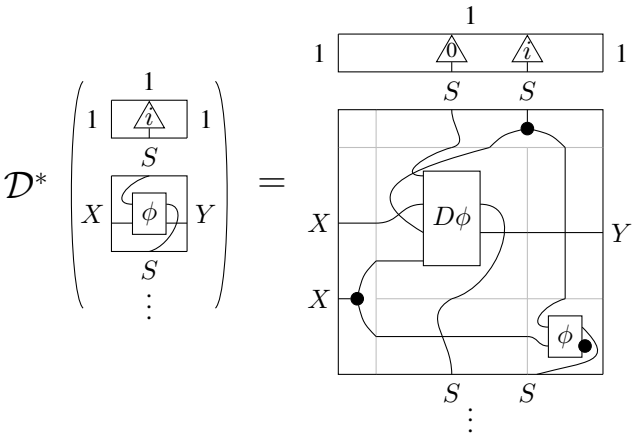
*where $z$ is the unzipping isomorphism from lemma 36.*

## V. Differentiation of causal morphisms

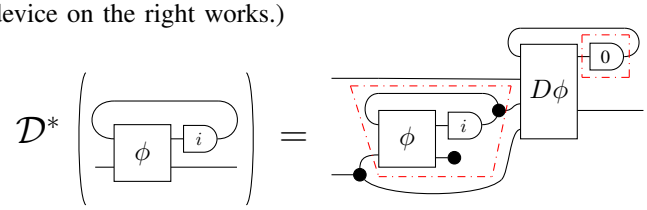
For our applications, we note that $\mathcal{D}^*$ restricts to $\mathrm{St}_0(\mathbb{C})$.

**Corollary 40** *The operation $\mathcal{D}^*$ restricted to $\mathrm{St}_0(\mathbb{C})$ is a Cartesian differential operator on $\mathrm{St}_0(\mathbb{C})$.*

Using this differential operator in $\mathrm{St}_0(\mathbb{C})$, we can find the derivative of a stateful function as another stateful function. From the definition of $\mathcal{D}$ on 2-cells, we know:



Translating this fact along the correspondence between circuit diagrams and morphisms in $\mathrm{St}_0(\mathbb{C})$, we obtain the following diagram as the derivative of our simple stateful function. (The red dashed boxes do not have any mathematical meaning; they are only there so we can describe how the device on the right works.)



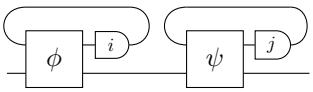

The idea of a derivative in a Cartesian differential category is to take a base point $\mathbf{x}$ as its lower argument and a small change as its upper argument $\mathbf{\Delta x}$ and return an approximation for the difference between the outputs of the function at $\mathbf{x}$ and the function at $\mathbf{x} + \mathbf{\Delta x}$.

Here is how the device obtained above accomplishes this. The red trapezoidal region is a copy of the original device which maintains the current state of the function in the delay gate initialized with $i$. It uses this state itself to maintain this invariant, and supplies a copy to the derivative of the combinational part, $D\phi$. Therefore, the bottom two arguments received by the $D\phi$ subdevice are the state and value inputs $\phi$ would receive.

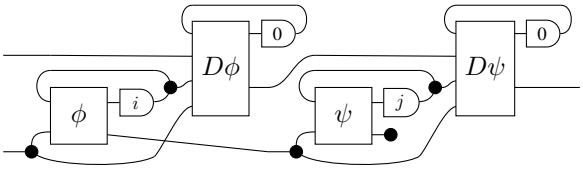
In the upper delay gate (initialized to 0, also boxed in red), the device accumulates its best approximation for the **difference between states** between the original device executed at $\mathbf{x}$ and at $\mathbf{x} + \mathbf{\Delta x}$, using the current state and input values, the approximate state change supplied from the upper delay gate, and the value change supplied at the upper input (above the red trapezoid). Meanwhile, the output wire to the left reports the best approximation for the difference in outputs to the environment.

Though it may seem we have taken a slightly special case by assuming the $\phi$ device is stateless (being an underlying morphism from a 2-cell), theorem 23 ensures *all* $\mathrm{St}_0(\mathbb{C})$ morphisms can be written in this form. So in fact this is a fully abstract circuit diagram for derivatives in $\mathrm{St}_0(\mathbb{C})$.

Taking $\mathbb{C} = \mathbf{Euc}_\infty$, this string diagram specializes to the derivative of a recurrent neural network. Theorem 39 guarantees this derivative matches precisely what we expect from the unroll-and-differentiate procedure used in backpropagation through time. However, the extra structure we have discovered for this procedure, namely that $\mathrm{St}(\mathbf{Euc}_\infty)$ is a Cartesian differential category, give us many useful properties. For example, the derivative of




is




We leave this exercise to the reader.

## VI. Conclusion and Future Directions

We have shown how to treat differentiation of stateful functions via two pieces of categorical machinery. First, we described the $\mathrm{St}(-)$ construction, taking a category and augmenting it with morphisms representing causal functions of sequences. The special subcategory $\mathrm{St}_0(-)$ consists of constant stateful morphism sequences, which perform the same computation at each clock tick, much like a Mealy machine.

Second, we showed that this causal construction also admits an abstract form of differentiation. Our key technical results showed that if a category $\mathbb{C}$ is a Cartesian differential category, so is $\mathrm{St}(\mathbb{C})$. In particular, this allows us to give a finite representation of the derivative of a causal function. In addition to being much more compact than the well-known unroll-then-differentiate approach, the structure of Cartesian differential categories ensure this differentiation operation has many useful properties of derivatives of undergraduate calculus, including a chain rule.

We believe that experimentation in machine learning will use differentiation and gradients in many new and interesting contexts. We also believe the abstract nature of Cartesian differential categories will prove very valuable for organizing the theory behind this growing field.

Though we would like to say our abstract treatment of differentiation can be used directly by machine learning practitioners, it appears this is not the case yet. The derivative of a morphism in a Cartesian differential category is not the same as having an explicit Jacobian or gradient. A gradient can be recovered from this morphism by applying it to all the basis vectors, but when there are millions of parameters in a machine learning model, this idea is computationally disastrous. We think that by adding some structure to Cartesian differential categories, such as a designated closed subcategory, we could give a theoretical treatment allowing for more explicit representation of Jacobians.

Another issue with our work is that machine learning practitioners often use functions which are not smooth, not differentiable, and even sometimes partial! While this wrinkle is easy enough to overcome in practice so long as it is encountered sufficiently rarely, to theoreticians it can be more of a challenge. Enhancing this work with differential restriction categories [9] might be a good way forward.

An interesting observation that points to potential further applications in machine learning is the following. We know that $\mathbb{C}$ being Cartesian differential category implies $\mathrm{St}_0(\mathbb{C})$ is as well. Therefore, $\mathrm{St}_0(\mathrm{St}_0(\mathbb{C}))$ is **also** a Cartesian differential category. Morphisms in $\mathbb{C}$ process individual inputs and morphisms in $\mathrm{St}_0(\mathbb{C})$ process sequences of inputs, so morphisms in $\mathrm{St}_0(\mathrm{St}_0(\mathbb{C}))$ process sequences of sequences of inputs. Similarly, while $\mathrm{St}_0(\mathbb{C})$ adds delay gates whose values can change as elements of its input sequence are processed, $\mathrm{St}_0(\mathrm{St}_0(\mathbb{C}))$ will add *meta-delay gates* whose values change after a single sequence in its input sequence of sequences has been processed. This behavior of meta-delay gates seems a

lot like parameter updating after processing an example in the training of neural networks. Further iterating this construction to $\mathrm{St}_0(\mathrm{St}_0(\mathrm{St}_0(\mathbb{C})))$ may be a good way to model a hyperparameter tuning process.

Somewhat removed from potential machine learning applications, we are also curious about the further development of the theory of delayed traces. In particular, it seems there are quite a few interesting delayed traces besides the one we described for $\mathrm{St}(\mathbb{C})$.

## Acknowledgments

We are very grateful to Bart Jacobs, Fabio Zanasi, Nian-Ze Lee and Masahito Hasegawa for many useful discussions. Thanks to JS Lemay for the pointer to [8]. Thanks also to the developers of TikZiT (Aleks Kissinger, Alexander Merry, Chris Heunen, K. Johan Paulsson), which which many of these diagrams were made.

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

## References

[1] A. Abel and B. Pientka. Wellfounded recursion with copatterns: a unified approach to termination and productivity. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 185–196. ACM, 2013.

[2] H. Basold, M. Bonsangue, H.H. Hansen, and J.J.M.M. Rutten. *(Co)Algebraic Characterizations of Signal Flow Graphs*, pages 124–145. Springer International Publishing, Cham, 2014.

[3] R.F. Blute, J.R.B. Cockett, and R.A.G. Seely. Differential categories. *Mathematical Structures in Computer Science*, 16(6):1049–1083, 2006.

[4] R.F. Blute, J.R.B. Cockett, and R.A.G. Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23):622–672, 2009.

[5] F. Bonchi, J. Holland, R. Piedeleu, P. Sobociński, and F. Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proceedings of the ACM on Programming Languages*, 3(POPL):128, Jan 2019.

[6] F. Bonchi, P. Sobociński, and F. Zanasi. A categorical semantics of signal flow graphs. In P. Baldan and D. Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 435–450. Springer, 2014.

[7] F. Bonchi, P. Sobociński, and F. Zanasi. Full abstraction for signal flow graphs. In S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 515–526. ACM, 2015.

[8] J.R.B. Cockett and G.S.H. Cruttwell. Differential structure, tangent structure, and SDG. *Applied Categorical Structures*, 22(2):331–417, Apr 2014.

[9] J.R.B. Cockett, G.S.H. Cruttwell, and J.D. Gallagher. Differential restriction categories. *Theory and Applications of Categories*, 25(21):537–613, 2011.

[10] R.E. Crochiere and A.V. Oppenheim. Analysis of linear digital networks. *Proceedings of the IEEE*, 63(4):581–595, April 1975.

[11] G.S.H. Cruttwell. Cartesian differential categories revisited. *Mathematical Structures in Computer Science*, 27(1):70–91, 2017.

[12] C. Ehresmann. Catégories structurées. *Annales scientifiques de l'École Normale Supérieure*, 80(4):349–426, 1963.

[13] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.

[14] T. Ehrhard and L. Regnier. Differential interaction nets. *Electr. Notes Theor. Comput. Sci.*, 123:35–74, 2005.

[15] C. Elliott. The simple essence of automatic differentiation. *PACMPL*, 2(ICFP):70:1–70:29, 2018.

[16] B. Fong, D. Spivak, and R. Tuyéras. Backprop as functor: A compositional perspective on supervised learning. See arxiv.org/abs/1711.10455, 2017.

[17] F. Gadducci and U. Montanari. The tile model. In G.D. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 133–166. The MIT Press, 2000.

[18] D.R. Ghica and A. Jung. Categorical semantics of digital circuits. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, FMCAD '16, pages 41–48, Austin, TX, 2016. FMCAD Inc.

[19] D.R. Ghica, A. Jung, and A. Lopez. Diagrammatic semantics for digital circuits. In V. Goranko and M. Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[20] S. Goncharov and L. Schröder. Guarded traced categories. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 313–330, Cham, 2018. Springer International Publishing.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] H.H. Hansen, C. Kupke, and J.J.M.M. Rutten. Stream differential equations: Specification formats and solution methods. *Logical Methods in Computer Science*, 13(1), 2017.

[23] T.A. Henzinger and D. 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*. ACM, 2014.

[24] N. Hoshino, K. Muroya, and I. Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Henzinger and Miller [23], pages 52:1–52:10.

[25] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119:447–468, 1996.

[26] P. Katis, N. Sabadini, and R.F.C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141178, Feb 1997.

[27] A. Kissinger and S. Uijlen. A categorical semantics for causal structure. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[28] Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, Jun 1991.

[29] S. Milius. A sound and complete calculus for finite stream circuits. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 421–430. IEEE Computer Society, 2010.

[30] R.E. Møgelberg. A type theory for productive coprogramming via guarded recursion. In Henzinger and Miller [23], pages 71:1–71:10.

[31] K.K. Parhi and Y. Chen. *Signal Flow Graphs and Data Flow Graphs*, pages 1277–1302. Springer New York, New York, NY, 2013.

[32] J.J.M.M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.

[33] J.J.M.M. Rutten. Rational streams coalgebraically. *Logical Methods in Computer Science*, 4(3), 2008.

[34] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct 1990.

[35] F. Zanasi. *Interacting Hopf Algebras - the Theory of Linear Systems. (Interacting Hopf Algebras - la théorie des systèmes linéaires)*. PhD thesis, École normale supérieure de Lyon, France, 2015.