Abstract—We investigate causal computations taking sequences of inputs to sequences of outputs where the $n$th output depends on the first $n$ inputs only. We model these in category theory via a construction taking a Cartesian category $\mathcal{C}$ to another category $\text{St}(\mathcal{C})$ 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 $\text{St}(\mathcal{C})$ with an implicit guardedness guarantee.

When $\mathcal{C}$ is equipped with a Cartesian differential operator, we construct a differential operator for $\text{St}(\mathcal{C})$ 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 up to that time. They share a basic operational scheme, depicted in the following diagram (which is to be read left-to-right):

```
  S  
 X  \phi  D  Y
```

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.

A recursive 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 the 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 [18]. 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 $\phi_k : X^{k+1} \rightarrow Y$ given by

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

(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 an empirically useful way to find gradients, known in the machine learning literature as backpropagation through time (BPTT) [31]. 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

Abstract—We investigate causal computations taking sequences of inputs to sequences of outputs where the $n$th output depends on the first $n$ inputs only. We model these in category theory via a construction taking a Cartesian category $\mathcal{C}$ to another category $\text{St}(\mathcal{C})$ 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 $\text{St}(\mathcal{C})$ with an implicit guardedness guarantee.

When $\mathcal{C}$ is equipped with a Cartesian differential operator, we construct a differential operator for $\text{St}(\mathcal{C})$ 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 up to that time. They share a basic operational scheme, depicted in the following diagram (which is to be read left-to-right):

```
  S  
 X  \phi  D  Y
```

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.

A recursive 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 the 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 [18]. 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 $\phi_k : X^{k+1} \rightarrow Y$ given by

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

(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 an empirically useful way to find gradients, known in the machine learning literature as backpropagation through time (BPTT) [31]. 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 \( C \), representing stateless functions, to a new category \( \text{St}(C) \) of stateful functions, particularly computations extended through discrete time (definition 11). This \( \text{St}(-) \) 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. [22] (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 the starting category \( C \) is a Cartesian differential category [28], then so is \( \text{St}(C) \) (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 \( \text{St}(C) \) is a Cartesian differential category implicitly obtains a slew of fundamental results for differentiation of stateful computations.

**Related Work**

Signal flow graphs are a widely used model of computation, especially in synchronous digital circuits and signal processing [7], [27]. 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 \( \text{St}(C) \) in this paper embodies this principle.

A line of coalgebraic study of signal flow graphs by Rutten [29], [30], Milius [25], Hansen et al. [19] and Basold et al. [2] and many others achieve characterisations of computable streams by signal flow graphs. These coalgebraic studies regard signal flow graphs as specification of coalgebraic transition systems. This 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. [15], [16]. Their system is an extension of a traced cartesian category with a few structural morphisms that implement wire join and delay gate, but their delay gates do not support arbitrary initialization. Their system can 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.

Zanasi studies the PROP \( \mathbb{H}_R \) of interacting Hopf algebras over a ring \( R \) in his PhD thesis [32]. The expressive power of this PROP is demonstrated by encoding various graphical systems into \( \mathbb{H}_R \) [4], [5]. When \( R \) is the polynomial ring \( k[x] \) over a field \( k \), the PROP \( \mathbb{H}_{k[x]} \) admits delay gates, and the trace-with-delay operation (which he called \( z \)-feedback operator) is definable [4, Definition 7]. His \( z \)-feedback operator is very close to the delayed trace operator, except that the latter supports arbitrary initial values.

Recently, Kissinger and Uijlen reformulated the concept of causality in quantum physics in a class of compact closed categories [23]. Starting from a compact closed category with some extra structure, they refine it to the \(*\)-autonomous category so that morphisms there respect causal constraints.

A category whose morphisms are realized by Mealy-machine like transducers is constructed in the memoryful GoI by Hoshino et al. [21]. Their transducers, represented as functions of type \( S \times A \rightarrow T(S \times B) \), 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.

A common theme in recursively defined computations is that to have well-defined behaviour, a recursive computation must satisfy a guardedness condition [1], [26]. Goncharov and Schröder developed the theory of guarded traced categories to formalize this phenomena in [17]. The key idea is to restrict Joyal et al.’s trace operator [22] 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, which is missing in the guarded trace operator.

The idea of using tiles as representations of computation steps is pursued in the tile models by Gadducci and Montanari [14]. In their model, each tile \( f : A \xrightarrow{S} B \) 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.

Inspired by the semantics of differential \( \lambda \)-calculus and differential proof nets by Ehrhard and Regnier [10], [11], 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 [28] 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] and [6].

There have been some recent efforts to connect category theory with machine learning, particularly backpropagation, using the fact that differentiation has a chain rule and is therefore compositional, for example [13]. A notable example is [12], where Elliot studies automatic differentiation (AD) in the context of functional programming. He gives a clean account of an AD algorithm by exploiting the functorial nature of the differentiation operator, including both a chain rule and a parallel rule to obtain a Cartesian functor.

**Preliminaries**

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

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

In general, Cartesian categories need not be strict, but working with associators etc. unnecessarily complicates the story. So whenever we mention a Cartesian category, we will assume that the associators are given.

Bold metavariables—$X, s, \text{etc.}$—denote sequences of mathematical objects, indexed by $N$. The $i$th component of a sequence is $X_i$. By $\bigotimes$ we mean the tail of $X$, namely $(0 \times X, 0 \times X_1, \cdots)$. 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 met variables $X_0, Y_0, \cdots$. By $[e]$ we mean the infinite sequence obtained by substituting $0, 1, 2 \cdots$ for $\bullet$. For instance,

\[
(i, [x_0 + y_0]) \text{ is } (i, x_0 + y_0, x_1 + y_1, \cdots)
\]

\[
(i, x_0 + y_0, y_1, x_1 + y_1, \cdots)
\]

When $e$ contains at least one dotted sequence metavariable, we may omit the outermost $[\cdots]$, so $[X_0 \times Y_0]$ may be written as $X_0 \times Y_0$. 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 $x + y + x + y + \cdots$).

A mathematical formula $\phi$ containing dotted sequence metavariables represents the conjunction $\bigwedge_{i \in N} \phi[i/\bullet]$. For instance, $Z = X_0 \times Y_0$ means $\forall i \in N. Z_i = X_i \times Y_i$.

**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 $\text{Set}^N$, the countable product category of $\text{Set}$. In this category, each morphism $f : X \to Y$ consists of independent components $f_k : X_k \to Y_k$ for all $k \in N$, 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 $f$ are independent means the $k$th output of $f$ 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 $\text{Set}(X, Y)$ for arbitrary $X, Y \in |\text{Set}|$. This class is too big—non-causal functions such as $\tau \cdot 1 : (x_0, x_1, \cdots) \mapsto (x_1, x_2, \cdots)$ are present there. Therefore, we must do something a bit more complex to obtain a class of functions somewhere between these two.

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

\[
f_k : S_k \times X_k \to S_{k+1} \times Y_k.
\]

To start this computation, we need to provide an initial state $i : 1 \rightarrow S_0$, and we call the pair $(i, f)$ a *stateful morphism sequence*. We will see causal functions are equivalence classes of these stateful morphism sequences (theorem 14).

Though these are all 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 $f_k$ 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 $f_k$ to $f_{k+1}$ 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 $X_k/Y_k$ and $S_k/S_{k+1}$—the former objects are the types of *values* flowing through $f_k$ 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* [9].

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).

\[
\begin{array}{ccc}
X & \overset{S}{\longrightarrow} & Y \\
\downarrow & \alpha & \downarrow \\
A & \overset{B}{\longrightarrow} & B
\end{array}
\]

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 (), objects from \( \mathbb{C} \) as 1-cells, representing values when oriented vertically and states when oriented horizontally, and functions \( f_k \) on states and values in the tiles.

**Definition 1** Let \( (\mathbb{C}, \times, 1) \) be a (strict) Cartesian category. The double category \( \text{DbI}(\mathbb{C}) \) 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 \( f \) by \( S \) and \( S' \), respectively. We will generally denote a 2-cell by \( f \) or \( \phi \).

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

These operations sending \( \mathbb{C} \)-morphisms to 2-cells in \( \text{DbI}(\mathbb{C}) \) are particularly useful. (Note first that \( \text{id}_{\mathbb{C}}^1 \) and \( \text{id}_{\mathbb{C}}^2 \) are the identities for horizontal composition, and similarly \( \text{id}_{\mathbb{C}}^1 \) and \( \text{id}_{\mathbb{C}}^2 \) 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 \( f : X \xrightarrow{S} Y \) is a \( \text{DbI}(\mathbb{C}) \) 2-cell, \( \phi_1 \in \mathbb{C}(T, X) \), \( \phi_2 \in \mathbb{C}(S', T') \), \( \psi_1 \in \mathbb{C}(W, X) \), and \( \psi_2 \in \mathbb{C}(Y, Z) \), then the underlying morphism of \( \phi_1 \cdot f \cdot \psi_1 \cdot \psi_2 \) has the following string diagram in \( \mathbb{C} \):

A 2-cell \( f \) of \( \text{DbI}(\mathbb{C}) \) 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:

\[
\begin{array}{c}
\text{Horizontal composition: } X \xrightarrow{S} Y = X \xrightarrow{\phi} Y \\
\intertext{Vertical composition:}
\begin{array}{c}
\text{Definition 4}
\end{array}
\end{array}
\]

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.
In this representation, the “arrow of time” starts at \( X_0 \) and points down. At the zeroth clock tick, the stateful morphism sequence receives a value at \( X_0 \), outputs a value at \( Y_0 \), 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 \((i, s) = (j, t)\) if and only if they are exactly the same sequence of 2-cells is not a suitably strong 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 \( X \rightarrow Y \) would be to compare the underlying \( C\)-morphisms of the infinite composition, meaning \((i, s) = (j, t)\) if and only if

\[
U(i^n; s_0; s_1; \cdots) = U(j^n; t_0; t_1; \cdots) : \prod_{i \in \mathbb{N}} X_i \rightarrow \prod_{j \in \mathbb{N}} Y_i.
\]

However, formalizing this infinite vertical composition is technically challenging, and \( C \) may not admit such 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 \((i, s) : X \rightarrow Y\) is the \( C\)-morphism

\[
T^n_c(i, s) \triangleq U(i^n; s_0; s_1; \cdots; s_{n-1}; \text{next } s_n) : \prod_{k=0}^{n} X_k \rightarrow \prod_{k=0}^{n} Y_k.
\]

Graphically, \( T^n_c(i, s) \) is the underlying morphism of the vertical composite 2-cell depicted in fig. 1 right.\(^\text{1}\)

**Definition 6** Two stateful morphism sequences \((i, s), (j, t) : X \rightarrow Y\) are extensionally equal iff \( T^n_c(i, s) = T^n_c(j, t) \).

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

---

\(^1\)In the figure, the 2-cells on the right have been drawn with overlapping 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.

---

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 computational processes are doing the same thing. Comparing truncations is always possible, but sometimes technically difficult. The following lemma has proven a useful method for establishing extensional equality in our experience.

**Lemma 7** (Shim lemma) Suppose \((i, s), (j, t) : X \rightarrow Y\) are stateful morphism sequences, and \( b \) is a sequence of \( C\)-morphisms such that \( b_* : \text{prv}(s_*) \rightarrow \text{prv}(t_*) \).

Then \((i, s)\) and \((j, t)\) are extensionally equivalent.

**Proof** Show by induction that

\[
i^n; s_0; \cdots; s_n; b_{n+1}^0; \text{next } t_n = j^n; t_0; \cdots; t_n; b_n^0; \text{next } t_n
\]

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

**Definition 8** Let \((i, f) : A \rightarrow B\) be a stateful morphism sequence. Its \( k \)-th unrolling is the \( k \)-th projection of the \( k \)-th truncation: \( \text{Un}_k(i, f) \triangleq \pi_k \circ T_k(i, f) : \prod_{k=0}^{n} A_k \rightarrow B_k \).

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 \( \text{id}_X \) is \((\text{id}_1, [(\text{id}_S)_n])\) for all \( X \in [C]^n \).

The composition \((i, s) \circ (j, t)\) of stateful morphism sequences \((i, s) : Y \rightarrow Z \) and \((j, t) : X \rightarrow Y\) is

\[
(i, s) \circ (j, t) \triangleq ((j, i), [s_* \ast t_*]) : X \rightarrow Z.
\]

As usual, we may denote \( \text{id}_X \) by \( X \).

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 extensional equivalence class of) \( \text{id}_X \) is the unit for the composition operation.

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

\(^2\)A sham is a little piece of material used to adjust two things so they line up better, such as a sliver of wood between a door frame and surrounding wall studs. In this case, \( b \) is the sham, and it adjusts the state spaces of the two stateful morphism sequences.
objects are $|C|^N$, that is, $\mathbb{N}$-indexed families of $C$-objects, morphisms are extensional equivalence classes of stateful morphism sequences, identities and composition are the extensions of those in Definition 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 $\text{St}(C)$.

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

$$HX = X, \quad Hf = (\text{id}_1, [f^h])$$

We call the morphisms in $\text{St}(C)$ of the form $Hf$ stateless morphisms, since they can be realized by a stateful morphism sequence with state sequence $[1]^3$.

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

**Proof** In $\text{St}(C)$, the final object is $[1]$ and the final map from $X$ is $H[1]_X$. Products and projection are also componentwise: our chosen $\text{St}(C)$ product $X \times Y$ is the sequence $[X_0 \times Y_0]$ of $C$-products, with $\pi_n \triangleq H[nX_0 \times Y_0]$ for $n \in \{0, 1\}$. □

**C. Morphisms in $\text{St}(\mathbb{S})$ and Causal Functions**

We claim that morphisms of $\text{St}(C)$ represent causal computations, whose outputs depend only on past inputs and states. To justify this claim, we compare $\text{Set}$-theoretic causal functions and morphisms in $\text{St}(\mathbb{S})$. For this, we need a precise definition of causality for functions on sequences, which we adapt from [19]. First, for $x, y \in A^N$, by $x \equiv_n y$ we mean $x$ and $y$ match in the first $n$ positions, that is, $x_i = y_i$ holds for any $i \leq n$.

**Definition 13** ([19]) Let $A$ and $B$ be sets. A function $f : A^N \rightarrow B^N$ is causal if for any $x, y \in A^N$,

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

This looks like a citation, but it means the constant sequence consisting of the terminal object of $\mathbb{S}$ in every position.

The following theorem states that $\text{St}(\mathbb{S})$ characterises causal functions on streams.

**Theorem 14** The homset $\text{St}(\mathbb{S})([A], [B])$ bijectively corresponds to the set of causal functions from $A^N$ to $B^N$.

**D. The Category $\text{St}_0(C)$ and Deterministic Mealy Machines**

The input, output, and state types for a $\text{St}(C)$ morphism can vary over time. This is a crucial property to capture all causal functions, as seen in the proof of theorem 14. However, the computational models we mentioned in the introduction, like Mealy machines, 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 $\text{St}(C)$:

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

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

It is easy to check that this restricted class of morphisms is closed under the $\text{St}(C)$-composition, hence $\text{St}_0(C)$ is a well-defined subcategory. Sequences of objects and $2$-cells in $\text{St}_0(C)$ are constant. Another way to state this uses the $\circ$ operator: $X = \bigcirc X$ if $X$ is an object of $\text{St}_0(C)$. In any case, we note the Cartesian structure of $\text{St}(C)$ restricts to $\text{St}_0(C)$.

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

$$H_0 X = [X], \quad H_0 f = (\text{id}_1, [f^h])$$

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

One useful operation on stateful morphism sequence is unrolling.

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

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

Note that the truncation operation $\text{Tr}$ can be extended to $\text{St}(C)$-morphisms, as it is well-defined on extensional equivalence classes.
III. DELAYED TRACE OPERATOR

The category \( \text{St}(\mathbb{C}) \) 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 \cite{Joyal2004}, which we briefly recall here. The trace operator is a structure on braided monoidal categories, and is defined to be a function \( \text{tr}^S : \mathbb{C}(S \otimes X, S \otimes Y) \rightarrow \mathbb{C}(X, Y) \). In the language of string diagrams, this operation is understood to form a feedback loop at a specified pair of ports:

\[
\begin{array}{c}
S \otimes X \\
\downarrow f \\
S \otimes Y
\end{array}
\xrightarrow{\text{tr}^S(f)}
\begin{array}{c}
X \\
\downarrow f \\
Y
\end{array}
\]

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

\[
\begin{array}{c}
S \\
\downarrow f \\
S
\end{array}
= 
\begin{array}{c}
\emptyset \\
\downarrow f \\
\emptyset
\end{array}
= 
\begin{array}{c}
S \\
\downarrow f \\
S
\end{array}
\]

We will show the delayed trace operator, found in \( \text{St}(\mathbb{C}) \), enjoys most of the trace operator axioms, except for 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 the delay gate in the loop path, depicted as follows:

\[
\begin{array}{c}
S \\
\downarrow f \\
S
\end{array}
\xrightarrow{\text{delay gate}}
\begin{array}{c}
X \\
\downarrow (i, f) \\
Y
\end{array}
\]

The half-round node is the delay gate, and is filled with its initial state \( p \). The delayed trace operator echoes a principle of synchronous circuit design: “all feedback loops should contain a register”.

Our first step towards a delayed trace operator on \( \text{St}(\mathbb{C}) \) 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 \( f : T \times X \xrightarrow{S} T' \times Y \) be a 2-cell in \( \text{Dbl}(\mathbb{C}) \). The value-to-state conversion of \( f \) at \( (T, T') \) is another 2-cell, denoted \( T[ f ]_{T'} \), with the same underlying morphism but different source and target 1-cells: \( T[ f ]_{T'} : X \xrightarrow{S \times T} Y \).

When the objects \( (T, T') \) involved in the conversion are clear from context, we drop them from the notation and write \( [ f ] \) for \( T[ f ]_{T'} \).

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

\[
\begin{array}{c}
S \\
\downarrow f \\
S \times T
\end{array}
\]

If \( f = T \xrightarrow{S} X \xrightarrow{S'} Y \), then \( T[ f ]_{T'} = X \xrightarrow{S \times T} Y \).

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

**Definition 19** Suppose \( (i, s) : T \times X \rightarrow \emptyset T \times Y \) is a morphism in \( \text{St}(\mathbb{C}) \). (Recall \( \emptyset T = [T_{\cdot+1}] \).) The delayed trace of \( s \) along \( T \) with an initial state \( p : 1 \rightarrow T_0 \) is the following morphism in \( \text{St}(\mathbb{C})(X, Y) \):

\[
\text{tr}^T_p (i, s) \triangleq (i, p)(T'[S \cdot T_{\cdot+1}]).
\]

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

**Proposition 20** Suppose \( (i, s) : T \times X \rightarrow \emptyset T \times Y \) is a morphism in \( \text{St}(\mathbb{C}) \). Suppose \( (h, r) : Y \rightarrow Z, (j, t) : W \rightarrow X \) and \( (f, p) : W \rightarrow Z \) are other arbitrary morphisms in \( \text{St}(\mathbb{C}) \). 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 \( \sigma_{X \cdot 0} : X \times 0 \rightarrow 0 \times X \times X \) in \( \text{St}(\mathbb{C}) \). Define its delayed trace with an initial state \( i : 1 \rightarrow X_0 \) to be

\[
r_X(i) \triangleq \text{tr}^X(i, \sigma_{X \cdot 0}) : 0 \rightarrow X.
\]

To get a better understanding of \( r_X(i) \), we first draw the value-to-state conversion in a single 2-cell in this morphism.

\[
\begin{array}{c}
X \times X_{i+1} \xrightarrow{[i]} X_{i+1} \\
\downarrow (i, f) \\
X_{i+1}
\end{array}
\xrightarrow{\text{delay gate}}
\begin{array}{c}
X \times X_{i+1} \xrightarrow{i} X_{i+1} \\
\downarrow (i, f) \\
X_{i+1}
\end{array}
\]

Doing value-to-state conversion along the whole sequence \( \sigma_{X \cdot 0} \) and supplying the initial value \( i : 1 \rightarrow X_0 \) yields:

\[
\begin{array}{c}
X_{i+1} \\
\downarrow (i, f) \\
X_{i+1}
\end{array}
\]

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 \( r_X(i) \) 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 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
[24], and there initial states of registers is a delicate issue.
The delayed trace operator satisfies the following modified
dinaturality property:

**Theorem 21** Suppose \((i, s) : T \times X \rightarrow \bigcirc U \times Y\) and \((j, g) : U \rightarrow T\) are morphisms in \(St(\mathcal{C})\). For any \(u : 1 \rightarrow U_0\),

\[
tr_U^U(((j, g) \times X) \circ (i, s)) = tr_U^U((i, s) \circ ((j', \bigcirc g) \times Y))
\]

where \(u' = \pi_1 \circ Ug_0 \circ (j, u)\) and \(j' = \pi_0 \circ Ug_0 \circ (j, u)\).

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 \(f : X \rightarrow Y\) in \(\mathcal{C}\), and initial state
\(i : 1 \rightarrow X_0\), we have \(Hf \circ r_X(i) = r_Y(f_0 \circ i) \circ H(\bigcirc \mathcal{I})\).

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

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

\[(i, s) = tr_1^{st(i, s)}(H[U(s)])\]

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 \(St_0(\mathcal{C})\)

The category \(St_0(\mathcal{C})\) is also closed under the delayed trace
operator. Since \(\mathcal{C} = \bigcirc X\) in \(St_0(\mathcal{C})\), delayed dinaturality is
even closer to true dinaturality.

**Corollary 24** Suppose \((i, s) : [T] \times [X] \rightarrow [U] \times [Y]\) is
a morphism in \(St_0(\mathcal{C})\), and \((j, [g]) : [U] \rightarrow [T]\) is another
morphism in \(St_0(\mathcal{C})\). For any initial state \(u : 1 \rightarrow U\),

\[
tr_u^{[\mathcal{C}]}(((j, [g]) \times id_X) \circ (i, [s])) = tr_u^{[\mathcal{C}]}((i, [s]) \circ ((j', [g]) \times id_Y))
\]

where \(u' = \pi_1 \circ g \circ (j, u)\) and \(j' = \pi_0 \circ g \circ (j, u)\).

**Corollary 25** For any \(f : X \rightarrow Y\) in \(\mathcal{C}\), and initial state
\(i : 1 \rightarrow X\), we have \(H[f] \circ r_X(i) = r_Y(f \circ i) \circ H([f])\).

### B. Diagrammatic reasoning about \(St_0(\mathcal{C})\) morphisms

Here we informally introduce a diagrammatic syntax for
morphisms in \(St_0(\mathcal{C})\). Theorem 23 indicates we can generate
all \(St_0(\mathcal{C})\) morphisms with the following grammar:

\[
\varphi := H_0f|\varphi_1 \circ \varphi_2| \varphi_1 \times \varphi_2| tr^S(\varphi)
\]

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

```
  \[ C := H_0f | C_1 \parallel C_2 \]
  \[ \cdots \]
```

where the box labeled \(f\) has \(m\) inputs and \(n\) outputs when
\(f : \prod_{k=1}^m A_k \rightarrow \prod_{k=1}^n B_k\). As is typical in string diagrams,
\(H_0id_X\) is depicted by a wire and \(H_0id_{XY}\) by a wire
crossing. Additionally, we depict \(H_0|A\) and \(H_0(id_A|id_A)\) with
a disconnector and copier: \(\longrightarrow\) and \(\bigcirc\).

The evident interpretation in \(St_0(\mathcal{C})\) of these diagrams
induces an equivalence on such 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:

```
  \[ H_0g \rightarrow H_0g \]
```

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

```
  \[ H_0g \quad \rightarrow \quad H_0g \]
```

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 differentiability of the causal computations
realized by \(St(\mathcal{C})\).
IV. CARTESIAN DIFFERENTIAL STRUCTURE

In this section, we investigate differentiation in \( \text{St}(\mathbb{C}) \). Our primary tool is the theory of Cartesian differential categories, introduced by Blute, Cockett, and Seely in [28]. We begin by recalling background.

Definition 26 ([28]) 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:

\[
\begin{align*}
0_{X \times Y} &= 0_X \times 0_Y \\
+_X &= (+_X +_Y) \circ (X \times \sigma_{Y,X} \times Y)
\end{align*}
\]

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

Example 27 ([28]) The category \( \text{Euc}_\infty \) 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 \( \text{St}(\mathbb{C}) \), 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 \( \text{St}(\mathbb{C}) \).

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

\[
\begin{align*}
\delta_{X,Y} &\triangleq X \times \sigma_{Y,X} \times Y \\
\alpha_X &\triangleq \delta_{X,X} \circ (X \times Y \times \Delta_X) \\
\beta_X &\triangleq X \times !X \times X \times X \\
\gamma_{X,Y} &\triangleq (X \times \Delta_X \times Y') \circ \delta_{X,Y} \\
\zeta_X &\triangleq X \times 0_{X \times X} \times X
\end{align*}
\]

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 \( D : \mathbb{C}(X,Y) \to \mathbb{C}(X \times Y, X) \), satisfying:

\[
\begin{align*}
CD1. \quad &D s = s \times 1_{\text{dom}(s)} \quad \text{for} \quad s \in \{ X, \sigma_{X,Y}, !X, \Delta_X, +_X, 0_X \} \\
CD2. \quad &D f \circ (0_X \times X) = 0_Y \circ f \\
CD3. \quad &D f \circ (+_X \times X) = +_Y \circ (D f \times D f) \circ \alpha_X \\
CD4. \quad &D (g \circ f) = D g \circ (D f \times D f) \circ (X \times \Delta_X) \\
CD5. \quad &D (f \times h) = (D f \times D h) \circ \delta_{X,Y} \\
CD6. \quad &D D f \circ \delta_X = D f \\
CD7. \quad &D D f \circ \delta_{X,X} = D D f
\end{align*}
\]

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 [28], but it is mostly straightforward to check that they are equivalent. The biggest changes are in axioms CD6 and CD7, for which we have taken alternate forms given in [6, Proposition 4.2].

Example 31 ([28]) \( \text{Euc}_\infty \) is a Cartesian differential category. The differential operator \( D \) sends a smooth function \( f : \mathbb{R}^n \to \mathbb{R}^m \) to \( D f : (x_1, x_2) \mapsto J f|_{x_2 \times x_1} \), where \( J f|_{x_2} \) is the Jacobian matrix of \( f \) evaluated 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 \( J s|_{x_2 \times x_1} = s(x_1) \)), while CD2 and CD3 express the fact that \( J f|_{x_2 \times x_1} \) 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 \( \text{St}(\mathbb{C}) \), it will be helpful to have an operation for forming parallel compositions in \( \text{St}(\mathbb{C}) \). This motivates us to define the following operation on 2-cells.

Definition 32 Let \( f : X \xrightarrow{S} Y \) and \( k : Z \xrightarrow{T} W \) be arbitrary 2-cells from \( \text{Dbl}(\mathbb{C}) \). The cross composition of \( f \) and \( k \) is another 2-cell \( f \boxtimes k : X \times Z \xrightarrow{S \times T} Y \times W \) defined by

\[
(f \boxtimes k)(x, z) = (\langle f(x), k(z) \rangle) \in Y \times W.
\]

It may be easier to understand \( \boxtimes \) composition by its underlying morphology:

\[
\begin{array}{c}
\begin{array}{c}
S \\
T
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
U f \\
U k\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
S' \\
T'
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
X \\
Y
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
Z \\
W
\end{array}
\end{array}
\end{array}
\]

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 \( \text{Dbl}(\mathbb{C}) \).

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

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

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

We can now start defining the Cartesian differential operator on \( \text{St}(\mathbb{C}) \). 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 $Dbl(C)$. The first, $D_0$, takes the 2-cell $f : X \xrightarrow{S} Y$ to the 2-cell $D_0 f : X \times X \xrightarrow{S \times S} Y$ with $UD_0 f \cong DU f \circ \delta_{X,X}$.

The second, $D$, takes $f$ to $Df : X \times X \xrightarrow{S \times S} Y$ with $Df \cong (S \times \Delta S)^c \circ (Df \otimes (LY \circ f)) \circ (X \times \Delta X)^h$.

The string diagrams for the underlying morphisms of $D_0$ and $D$ may be easier to understand. For $D_0$,

| Universe | Diagram |
|----------|---------|
| $Df$    | $DUf$  |

while for $Df$,

| Universe | Diagram |
|----------|---------|
| $UF$    | $DUf$  |

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

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

1) If $\varphi \in C(X,Y)$, then $D(\varphi^h) = (D\varphi)^h$ and $D(\varphi^v) = ((D\varphi \times \varphi) \circ (X \times \Delta X))^v$.  
2) $(0_S \times S)^v \circ Df \circ (0_X \times X)^h = (0_Y \circ \gamma_Y)^h \circ f \circ (0_S \times S')^v$.  
3) $(+_S \times S)^v \circ Df \circ (+_X \times X)^h = \alpha_0^S \circ (+_X \times X)^h \circ Df \circ \alpha_0^S \circ (+_S \times S')^v$.  
4) $Df \circ h = Df \circ Df \circ h$.  
5) $D(\varphi \circ f) = \gamma_{S',T'}^S \circ (Df \circ \gamma_{S,T}^X \circ f) \circ (X \times \Delta X)^h$.  
6) $D(f \circ k) = (Df \circ Dk) \circ \delta_{X,Z}^h$.  
7) $\zeta(X)_c \circ Df = Df \circ \zeta(X)_c$.  
8) $\delta_{S,S'}^h \circ Df = Df \circ \delta_{S,S'}^h$.  

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 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 $D((f_0 : \cdots : f_n)) = (Df_0 : \cdots : Df_n) \circ z^h$, where $z$ is the unzipping isomorphism in $C$ of type $\prod_{k=0}^n (dom f_k \times dom f_k) \rightarrow (\prod_{k=0}^n dom f_k) \times (\prod_{k=0}^n dom f_k)$.

We can now state the operator we seek on $St(C)$.

**Definition 37** The componentwise application of $D$ to 2-cells in a $St(C)$ morphism, $D^* : (i,s) \mapsto (UD(i)^v, [DS_s])$, is a well-defined operation on $St(C)$ morphisms of type

$$D^* : St(C)(X,Y) \rightarrow St(C)(X \times X, Y).$$

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

**Theorem 38** $D^*$ is a Cartesian differential operator.

The strategy for this proof is to use the properties of $D$ 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:

$$D^*((j,g) \circ (i,f)) = D^*(j,g) \circ (D^*(i,f) \times (i,f)) \circ (X \times \Delta X).$$

The key step in proving this is invoking the Shim Lemma. We have two conditions to check for this invocation:

$$f_{S_0, T_0} \circ (0_{S_0}, 0_{T_0}, i, j) = (0_{S_0}, i, i, 0_{T_0}, j)$$

and

$$D((g_a \circ f_a) : (D_{S_a} \circ f_a) \circ (X_a \times \Delta X_a)^h),$$

which is a case of Proposition 35(5).

We can now prove $CD^*$:

$$D^*((j,g) \circ (i,f)) = ((0_{S_0}, 0_{T_0}, i, j), [D(a \circ f_a)]) = (0_{S_0}, i, i, 0_{T_0}, j), [D_{S_0} \circ (D_{S_0} \circ f_a)] \circ (X_a \times \Delta X_a)^h)$$

$$D^*(j,g) \circ (D^*(i,f) \times (i,f)) \circ (X \times \Delta X)$$

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 $(i,f) : A \rightarrow B$ in $St(C)$,

$$\text{Un}_{n}(D^*(i,f)) = D(\text{Un}_{n}(i,f)) \circ z : \prod_{n=0}^k (A_n \times A_n) \rightarrow B_k,$$

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

V. DIFFERENTIATION OF CAUSAL MORPHISMS

For our applications, we note that $D^*$ restricts to $St_0(C)$.

**Corollary 40** The operation $D^*$ restricted to $St_0(C)$ is a Cartesian differential operator on $St_0(C)$.

Using this differential operator in $St_0(C)$, we can find the derivative of a stateful function as another stateful function. From the definition of $D$ on 2-cells, we know:
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 St(−) construction, taking a category and augmenting it with morphisms representing causal functions of sequences. The special subcategory 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 C is a Cartesian differential category, so is St(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 often 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 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 $\text{St}(\mathbb{C})$ is as well. Therefore, $\text{St}(\text{St}(\mathbb{C}))$ is also a Cartesian differential category. Morphisms in $\mathbb{C}$ process individual inputs and morphisms in $\text{St}(\mathbb{C})$ process sequences of inputs, so morphisms in $\text{St}(\text{St}(\mathbb{C}))$ process sequences of sequences of inputs. Similarly, while $\text{St}(\mathbb{C})$ adds delay gates whose values can change as elements of its input sequence are processed, $\text{St}(\text{St}(\mathbb{C}))$ 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 $\text{St}(\text{St}(\text{St}(\mathbb{C})))$ 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 $\text{St}(\mathbb{C})$.

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 [6]. Thanks also to Ze Lee and Masahito Hasegawa for many useful discussions.

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

REFERENCES

[1] Andreas Abel and Brigitte Pientka. Wellfounded recursion with co-patterns: 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] Henning Basold, Marcello Bonsangue, Helle Hvid Hansen, and Jan Rutten. (Co)algebraic Characterizations of Signal Flow Graphs, pages 124–145. Springer International Publishing, Cham, 2014.

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

[4] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In Paolo Baldan and Daniele 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.

[5] Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In Srijan K. Rajamani and David 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.

[6] J. R. B. Cockett and G. S. H. Cruvellier. Differential structure, tangent structure, and sdg. Applied Categorical Structures, 22(2):331-417, Apr 2014.

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

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

[9] Charles Ehresmann. Categories structures. Annales scientiﬁques de l’´cole Normale Supérieure, 80(4):349–426, 1963.

[10] Thomas Ehrlund and Laurent Regnier. The differential lambda-calculus. Theor. Comput. Sci., 309(1-3):1–41, 2003.

[11] Thomas Ehrlund and Laurent Regnier. Differential interaction nets. Electr. Notes Theor. Comput. Sci., 123:35–74, 2005.

[12] Conal Elliott. The simple essence of automatic differentiation. PACMPL, 2(ICC):70:1–70:29, 2019.

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

[14] Fabio Gadducci and Ugo Montanari. The tile model. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, Proof, Language and Interaction, Essays in Honour of Robin Milner, pages 133–166. The MIT Press, 2000.

[15] Dan R. Ghica and Achim Jung. Categorical semantics of digital circuits. In Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design, FM CAD ’16, pages 41–48, Austin, TX, 2016. FMCAD Inc.

[16] Dan R. Ghica, Achim Jung, and Ali clumsy Lopez. Diagrammatic semantics for digital circuits. In Valentin Goranko and Mads 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.

[17] Sergey Goncharov and Lutz 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.

[18] Jan Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[19] Helle Hvid Hansen, Clemens Kupke, and Jan Rutten. Stream differential equations: Specification formats and solution methods. Logical Methods in Computer Science, 13(1), 2017.

[20] 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/SIGEE Symposium on Logic in Computer Science (LICS), CSL-LICS ’14, Vienna, Austria, July 14 - 18, 2014. ACM, 2014.

[21] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Henzinger and Miller [20], pages 52:1–52:10.

[22] Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. Math. Proc. Camb. Phil. Soc., 119:447–468, 1996.

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

[24] Charles E. Leiserson and James B. Saxe. Retiming synchronous dataflow networks. Journal of the ACM (JACM), 30(1):245–264, July 1983.

[25] Stefan Milus. A sound and complete calculus for finite stream circuits. Theory and Applications of Categories, 24:1–24:16. University of Koblenz-Landau, 2010.

[26] Rasmus Ejlers Møgelberg. A type theory for productive coprogramming via guarded recursion. In Henzinger and Miller [20], pages 71:1–71:10.

[27] Keshab K. Parhi and Yanni Chen. Retiming synchronous digital networks and memory circuits. The McGraw-Hill Companies, Inc., 1995.

[28] Robin Cockett Richard F. Blute and Robert A.G. Seely. Cartesian differential categories. In Conal Elliott, editor, Automatic Differentiation - Applications, Trends, and New Directions, volume 83 of Lecture Notes in Computer Science, pages 1277–1302. Springer New York, New York, NY, 2013.

[29] Robin Cockett Richard F. Blute and Robert A.G. Seely. Cartesian differential categories. Theory and Applications of Categories, 22(23):622–672, 2009.

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

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

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

[33] Fabio 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.
A. Proof of theorem 14

a) Injectivity: Without loss of generality, we assume $A$ is non-empty and take $\bot \in A$. We define helper functions

$$(-)_{\leq k} : A^\omega \to A^{k+1}$$

by

$$a_{\leq k} = (a_0, \ldots, a_k), \quad (a_0, \ldots, a_k)^{k+} = a_0, \ldots, a_k, \bot.$$ 

**Lemma 41** For any causal function $f : A^\omega \to B^\omega$, we have $(f((a_{\leq k})^{k+}))_k = (f(a))_k$.

**Proof** Notice that $(a_{\leq k})^{k+} \equiv_k a$. From causality, we have $f((a_{\leq k})^{k+}) \equiv_k f(a)$, and hence $(f((a_{\leq k})^{k+}))_k = (f(a))_k$. $\square$

Let $f : A^N \to B^N$ be a causal function. We define $s(f) = (id_1, f)$ by

$$s(f) : A^k \times A \to A^{k+1} \times B$$

$$s(f)_k(x, a) = ((x, a), (f((x, a)^{k+}))_k)$$

**Lemma 42** For any $k \in \mathbb{N}$ and $a \in A^\omega$, we have $U(i; f_0; \ldots; f_k)(a_{\leq k}) = (a_{\leq k}, f(a))_{\leq k}$.

**Proof** When $k = 0$,

$$T_{c_0}(id_1, f)(a_{\leq 0}) = (a_0, f((a_0)^{0+}))_0 = (a_0, f(a))_{\leq 0}.$$ 

Suppose that $U(i; f_0; \ldots; f_k)(a_{\leq k}) = (a_{\leq k}, f(a))_{\leq k}$ holds. Then

$$U(i; f_0; \ldots; f_k; f_{k+1})(a_{\leq k+1}) = U(i; f_0; \ldots; f_k)(a_{\leq k})$$

$$= U(i; f_0; \ldots; f_k; s''(x, b')) = U f_{k+1}(s'', a_{k+1}) \in (s''(x, b'))$$

$$= U f_{k+1}(a_{\leq k}, a_{k+1}) \in (s'', (f(a))_{\leq k}, b'))$$

$$= (a_{\leq k+1}, (f(a))_{\leq k}, f((a_{\leq k+1})^{k+}))_{k+1}$$

$$= (a_{\leq k+1}, f(a))_{\leq k+1}$$

$$= (a_{\leq k+1}, f(a))_{\leq k+1}.$$ 

$\square$

**Corollary 43** $T_{c_k}(id_1, f)(a_{\leq k}) = f(a)_{\leq k}$.

Suppose that $s(f)$ and $s(g)$ are extentionally equivalent. That is, $T_{c_0}(id_1, f) = T_{c_0}(id_1, g)$. Then for any $a \in A^N$, we have

$$f(a)_{\leq} = T_{c_k}(id_1, f)(a_{\leq k}) = T_{c_k}(id_1, g)(a_{\leq k}) = g(a)_{\leq k}.$$ 

Therefore $f = g$.

b) Surjectivity: Let $(i, f) : [A] \to [B]$ be a stateful morphism sequence. We define a function $g : A^\omega \to B^\omega$ by

$$g(a)_k = (Tr_k(i, f)(a_{\leq k}))_k.$$ 

We show that $s(g)$ and $(i, f)$ are extentionally equivalent. From the definition, $s(g) = (id_1, g)$ where

$$g_k : A^k \times A \to A^{k+1} \times B, $$

$$g_k(x, a) = ((x, a), (g((x, a)^{k+}))_k)$$

$$= ((x, a), (Tr_k(i, f)((x, a)^{k+}))_{\leq k})$$

$$= ((x, a), (Tr_k(i, f)(x, a))_k).$$

We show $(id_1, g)$ and $(i, f)$ are extensionally equivalent. For this, we inductively show that for any $x \in A^{k+1}$, we have

$$U(id; g_0; \ldots; g_k)(x) = (x, Tr_k(i, f)(x)).$$ 

Note that this immediately entails $Tr_k(id; g) = Tr_k(i, f)$. Let $x \in A^k$ and $a_k \in A$.

When $k = 0$, it is obvious.
When \( k > 0 \), we show

\[
U(\text{id}_1; g_0; \ldots; g_k)(x, a_k) = \text{let}(s, y) = U(\text{id}_1; g_0; \ldots; g_{k-1})(x) \text{ in }
\text{let}(s', b) = g_k(s, a_k) \text{ in } (s', (y, b))
\]

\[
= \text{let}(s, y) = (x, \text{Tr}_{k-1}(i, f))(x) \text{ in }
\text{let}(s', b) = g_k(s, a_k) \text{ in } (s', (y, b))
\]

\[
= \text{let } y = \text{Tr}_{k-1}(i, f)(x) \text{ in }
\text{let}(s', b) = g_k(x, a_k) \text{ in } (s', (y, b))
\]

\[
= \text{let } y = \text{Tr}_{k-1}(i, f)(x) \text{ in }
\text{let}(s', b) = (\text{Tr}_k(i, f)(x, a_k))_{\text{in}}(s', (y, b))
\]

\[
= \text{let } y = \text{Tr}_{k-1}(i, f)(x) \text{ in }
\text{let } y = \text{Tr}_k(i, f)(x, a_k)_{\text{in}}(x, a_k, y, b)
\]

\[
= \text{let } y = \text{Tr}_{k-1}(i, f)(x) \text{ in }
\text{let } y = \text{Tr}_k(i, f)(x, a_k)_{\text{in}}(x, a_k, y, b)
\]

\[
= ((x, a_k), \text{Tr}_k(i, f)(x, a_k)).
\]

**B. Proof of theorem 21**

Let \( r_i = [(\text{id}_{Y_i} \times g_i) * s_i] \). By induction, obtain the equality of this:

\[
U(r_0; \ldots; r_n) \circ (\text{id}_{X_{n+1} \times \prod Y_i} \times \sigma_{T_{n+1}, U_{n+1}})
\]

and this:

\[
U(t_0; \ldots; t_n) \circ (\text{id}_{\prod Y_i} \times \sigma_{T_{n+1}, X_{n+1}}).
\]

Then observe the \( n \)th truncation of \( t_i \) is (3) followed by \( \pi_{\prod Y_i} \) and similarly, the \( n \)th truncation of \( r_i \) is (4) followed by the same projection. Therefore, they are observationally equivalent, though not equal as sequences.

**C. Proofs from section IV**

1) **Lemma 28**: We already know \( \mathbb{C} \) being Cartesian implies \( \text{St}(\mathbb{C}) \) is Cartesian. A commutative monoid structure on \( X \) is given by componentwise monoid structure: \( +X \triangleq [+X_i : X_i \times X_i \xrightarrow{1} X_i] \) and \( 0X \triangleq [0X_i : 1 \xrightarrow{1} X_i] \).

2) **Properties of \( D \)**: It is straightforward to check that all the properties we claim are well-typed, in the sense that the source and target 1-cells of the 2-cells on each side match. It remains to check that the underlying morphisms of the 2-cells in each claimed property match. This is not an effortless task; we have two kinds of composition and will be making heavy use of the axioms of Cartesian differential operators. We find it easiest to do this reasoning with string diagrams, so this is what we will present here.

When the string diagrams below are particularly complicated, we may draw a dotted red box around a region or include some red text. Anything found in red has no mathematical meaning—it is only there to help break down a complex diagram or foreshadow a major substitution.

Throughout this proof, let \( Uf = \phi, Ug = \psi, Uh = \xi, \) and \( Uk = \kappa \).

**Property 1**: If \( \phi \in \mathbb{C}(X, Y) \), then \( D(\phi^h) = (D\phi)^h \) and \( D(\phi^v) = ((D\phi \times \phi) \circ (X \times \Delta_X))^v \).

When \( \phi \) is oriented horizontally, it has no state. Omitting the wires corresponding to state in the diagram for the underlying morphism of \( D \), we find

\[
\begin{align*}
\begin{tikzpicture}
  
  \node (D) at (0, 0) {$D\phi$};
  \node (phi) at (-1, -2) {$\phi$};
  \node (Dphi) at (1, -2) {$D\phi$};

  \draw [->] (D) -- (phi);
  \draw [->] (phi) -- (Dphi);

\end{tikzpicture}
\end{align*}
\]
which matches the underlying morphism of \((D\phi)^h\). On the other hand, when \(\phi\) is oriented vertically, it has no values. Omitting the value wires in the same diagram, we find

which again matches.

**Property 2:** \((0_S \times S)^v; Df \ast (0_X \times X)^h = (0_Y \circ \iota)^h \ast f; (0_{S'} \times S')^v\)

From Lemma 3 and the diagram for the underlying morphism of \(Df\),

where the middle equality is by the CD2 axiom for \(D\).

**Property 3:** \((+_S \times S)^v; Df \ast (+_X \times X)^h = \alpha_S^v; (+_X \ast Df \otimes Df \ast \alpha_X^v); \beta_S^v; (+_{S'} \times S')^v\)

This property requires the use of CD3 for \(D\).

**Property 4:** \(D(f; h) = (Df; Dh) \ast \delta_{K,X}^v\).

Using the CD axioms for \(D\), we compute the differential of the underlying morphism of \(f; h\):
We use that result to construct a string diagram for $UD(f; h)$:

\[
UD(f; h) = \begin{array}{c}
D\phi \\
\phi \\
D\xi \\
\xi \\
\end{array} 
\begin{array}{c}
D\phi \\
\phi \\
D\xi \\
\xi \\
\end{array} = \begin{array}{c}
D\phi \\
\phi \\
D\xi \\
\xi \\
\end{array} = \begin{array}{c}
U(Df; Dh) \\
U(f; h) \\
U(Df; Dh) \\
\end{array}
\]

This is the string diagram of $U((Df; Dh) * \delta^h_{X,Z})$ by Lemma 3.

**Property 5:** $D(g * f); \gamma^g_{S,T'} = \gamma^g_{S,T} * (Dg * (Df \otimes f) * (X \times \Delta X)^h)$.

Using the CD axioms for $D$, we compute the differential of the underlying morphism of $g * f$:

\[
D(\begin{array}{c}
\phi \\
\psi \\
\end{array}) = \begin{array}{c}
D\phi \\
\phi \\
D\psi \\
\psi \\
\end{array} = \begin{array}{c}
D\phi \\
\phi \\
D\psi \\
\psi \\
\end{array}
\]

We use that result to construct a string diagram for $U(D(g * f); \gamma^g_{S,T'})$:

\[
U(D(g * f); \gamma^g_{S,T'}) = \begin{array}{c}
D\phi \\
\phi \\
D\psi \\
\psi \\
\end{array} = \begin{array}{c}
D\phi \\
\phi \\
D\psi \\
\psi \\
\end{array} = \begin{array}{c}
U(Dg) \\
U(f) \\
U(Dg) \\
\end{array}
\]

\[
U(Dg) = \begin{array}{c}
D\phi \\
\phi \\
U(f) \\
\end{array} = \begin{array}{c}
D\phi \\
\phi \\
U(f) \\
\end{array} = \begin{array}{c}
U(Df \otimes f) \\
\end{array}
\]

\[
U(Dg) = \begin{array}{c}
D\phi \\
\phi \\
U(f) \\
\end{array} = \begin{array}{c}
D\phi \\
\phi \\
U(f) \\
\end{array} = \begin{array}{c}
U(Dg \otimes Df \otimes f) \\
\end{array}
\]
This is the string diagram of \( U(\gamma_{S,T}; (Dg \ast (Df \boxtimes f)) \ast (X \times \Delta_X)^h) \) by Lemma 3.

**Property 6:** \( D(f \boxtimes k); \delta_{S',T'}^v = \delta_{S,T}^v; Df \boxtimes Dk \ast \delta_{X,Z}^h. \)

Using the CD axioms for \( D \), we compute the differential of the underlying morphism of \( f \boxtimes k \):

\[
D(\begin{array}{c}
\phi \\
\kappa
\end{array}) = \begin{array}{c}
D(\begin{array}{c}
\phi \\
\kappa
\end{array})
\end{array}
\]

We use that result to construct a string diagram for \( U(D(f \times k); \delta_{S',T'}^v) \):

\[
D\begin{array}{c}
\phi \\
\kappa
\end{array} = \begin{array}{c}
D\begin{array}{c}
\phi \\
\kappa
\end{array}
\end{array}
\]

**Property 7:** \( \zeta_S^v; DDf \ast \zeta_{k}^h = Df; \zeta_{S'}^v. \)

**Property 8:** \( \delta_{S,S}^v; DDf \ast \delta_{X,X}^h = DDf; \delta_{S',S'}^h. \)

For these two properties, we need the underlying morphism for \( DDf \). This can be found in the diagram below. We do not show all the steps, hopefully the methodology is clear enough for the impossibly interested reader to check the claim. For property 7, you need to use properties CD2 and CD6 for \( D \). For property 8, you need to use CD7 for \( D \).

![Diagram](image)

3) **Well-definedness of \( D^* \):**

**Lemma 44** \( D^* \) is well-defined.

**Proof** Let \((i, s)\) and \((j, t)\) be extensionally equivalent sequences. We must show \((Di, [D_{s*}])\) and \((Dj, [Dt*])\) are extensionally
CD3. \( D_i; D_{s_0}; \ldots; D_{s_n}; d_{(\text{ext } s_n)} z = D_i; D_{s_0}; \ldots; D_{s_n}; D_{d_{\text{ext } s_n}} = D(i; s_0; \ldots; s_n; d_{\text{ext } s_n}) \ast z^{-1} = D(j; t_0; \ldots; t_m; d_{\text{ext } t_n}) \ast z^{-1} \)

where the first line is by Proposition 35(2), the second by Lemma 36, and the last by the fact that \((i, s)\) and \((j, t)\) are extensionally equivalent. Here it is crucial that extensionally equivalent sequences have the same source values, so that the \(z\) obtained from Lemma 36 when \(\varphi = s\) matches the \(z\) obtained when \(\varphi = t\). The proof then finishes by reversing the first two steps. \(\square\)

4) \(D^*\) is a Cartesian differential operator for \(\text{St}(\mathbb{C})\) (Proposition 38):

We show that \(D^*\) satisfies the seven axioms given in Definition 30. Most of the hard work has already been done in Proposition 35; we use the properties of \(D\) established there in concert with the Shim Lemma to obtain most of the CD axioms.

Let \(f = (i, f) : X \rightarrow Y\), \(g = (j, g) : Y \rightarrow Z\) and \(h = (f, h) : Z \rightarrow W\). Let \(S\), \(T\), and \(U\) be the state sequences of \(f\), \(g\), and \(h\), respectively.

CD1. \(D^* s = s \ast \ast \text{dom(s)}\) for \(s \in \{X, \sigma X, Y, !X, \Delta X, + X, 0X\}\).

These \(s\) are of the form \((\text{id}_i, [(s^h)_i])\) where each \(s^h\) is a \(C\)-morphism from \(\{X, \sigma X, Y, !X, \Delta X, + X, 0X\}\). Therefore,

\[
D^* s = D^* (\text{id}_i, [(s^h)_i]) = (\text{id}_i, [D(s^h)]) = (\text{id}_i, [(D(s^h)_i)])
\]

\[
= (\text{id}_i, (s^h \ast \ast \text{dom(s)})) = (\text{id}_i, [(s^h)_i]) \ast s \ast \ast \text{dom(s)}\]

Where the last step in the first line is by Proposition 35(1), and the first step in the second line is by CD1 for \(D\).

CD2. \(D^* (i, f) \circ (0X \times X) = 0Y \circ \ast X\).

By definition of \(D^*\), we know \(D^* (i, f) \circ (0X \times X) = (0s_b, i), [Df \ast (0X \times X^h)]\). By Proposition 35(2),

\[(0s_b \times S_0) \circ i = (0s_b, i) \text{ and } (0Y \circ \ast Y^h \circ f; (0s_{b+1} \times S_{b+1})^v = (0s_b \times S^v) \circ Df \ast (0X \times X^h),\]

so the shim lemma tell us

\[
D^* (i, f) \circ (0X \times X) = (0s_b, i), [Df \ast (0X \times X^h)]
\]

(definitions)

\[
= (i, [(0Y \circ \ast Y^h \circ f_s)])
\]

(shim lemma)

\[
= 0Y \circ \ast Y \circ (i, f)
\]

(definitions)

\[
= 0Y \circ \ast X
\]

(finality)

as desired.

CD3. \(D^* (i, f) \circ (+X \times X) = +Y \circ (D^* (i, f) \times D^* (i, f) \circ \alpha^X_X).\)

Since \(+s_b \times S_0) \circ (0s_b, 0s_b, i) = (0s_b, i)\), and, by Proposition 35(3),

\[
\alpha^S_b; (+Y \circ Df_0 \otimes Df_0 \circ \alpha^X_X) = (\beta^S_{b+1}; (+s_{b+1} \times S_{b+1})^v = (+s_b \times S^v) \circ Df \ast (+X \times X^h),\]

we can invoke the shim lemma again.

\[
D^* (i, f) \circ (+X \times X) = ((0s_b, i), [Df \ast (+X \times X^h)]
\]

(definitions)

\[
= ((0s_b, 0s_b, i), [\alpha^S_b; (+Y \circ Df \otimes Df \circ \alpha^X_X)])
\]

(shim lemma)

\[
= (\beta^S_b \circ (0s_b, i), 0s_b, i), [\alpha^S_b; (+Y \circ Df_0 \otimes Df_0 \circ \alpha^X_X)])
\]

(definition)

\[
= (\beta^S_b, (0s_b, i); \alpha^S_b; (+Y \circ Df \otimes Df \circ \alpha^X_X)])
\]

(delayed dinaturality)

\[
= ((0s_b, i), 0s_b, i), [\alpha^S_b; (+Y \circ Df \otimes Df \circ \alpha^X_X)])
\]

\[
= ((0s_b, i), 0s_b, i), [\alpha^S_b; (+Y \circ Df \otimes Df \circ \alpha^X_X)])
\]

The last step here is a bit delicate, and involves a special case of the Shim Lemma. If \(b^0 \circ i = i\), then \((i, [f_s]) = (i, [b^0 \circ f_s])\). Here \(\alpha^S_b; \circ \beta^S_b\) is such a sequence of idempotent maps.
CD4. $D^*(i, j) = D^*(j, i) = \delta_X$. 
Since $\gamma_{S_0, T_0} \circ (0_{S_0}, 0_{T_0}) = (0_{S_0}, 0_{T_0}, i, j)$, and by Proposition 35(5),
\[
D(\delta) \cdot \gamma_{S_0}^n \cdot T_{\ast+1} = \gamma_{S_0}^n \cdot T_{\ast}; (D(\delta) * (D\xi \otimes \xi)) = (X \times \Delta_X)^h
\]
we use the shim lemma again.
\[
D^*(i, j) = ((0_{S_0}, 0_{T_0}, i, j), [D(\delta) * \xi])
\]
\[
= ((0_{S_0}, i, i, 0_{T_0}, j), [D(\delta) * (D\xi \otimes \xi) * (X \times \Delta_X)^h])
\]
We invoke the shim lemma with
\[
D^*(j, i) = ((0_{T_0}, i), [D(\delta) * \xi])
\]
\[
= ((0_{T_0}, i, i), [D(\delta) * (D\xi \otimes \xi) * (X \times \Delta_X)])
\]
\[
= D^*(j, i) \circ (D^*(i, j) \times (i, j)) \circ (X \times \Delta_X)
\]
CD5. $D^*(i, j) \times (\ell, h) = D^*(i, j) \times D^*(\ell, h) \circ \delta_X$. 
Our invocation of the shim lemma this time uses the fact $\delta_{S_0, T_0} \circ (0_{S_0}, 0_{T_0}, i, j) = (0_{S_0}, 0_{T_0}, i, j)$ and Proposition 35(6)
\[
D(\delta) \cdot \delta_{S_0, T_0} = \delta_{S_0, T_0} \cdot D(\delta) \circ \delta_X.
\]
We can obtain this axiom now as
\[
D^*(i, j) \times (\ell, h) = ((0_{S_0}, 0_{T_0}, i, j), [D(\delta) \otimes \delta])
\]
\[
= ((0_{S_0}, 0_{T_0}, i, j), [D(\delta) \otimes \delta \circ \delta_X])
\]
We invoke the shim lemma this time using the facts that $\delta_{S_0} \circ (0_{S_0}, i) = (0_{S_0}, 0_{S_0}, 0_{S_0}, i)$ and Proposition 35(7)
\[
D(\delta) \cdot \delta_{S_0, T_0} = \delta_{S_0, T_0} \cdot D(\delta) \circ \delta_X.
\]
We obtain the axiom directly now,
\[
D^*(i, j) \circ \delta_X = ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\delta_X \circ \delta])
\]
\[
= ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\delta_X])
\]
\[
= D^*(i, j)
\]
CD6. $D^* D^*(i, j) \circ \zeta_X = D^*(i, j)$. 
We invoke the shim lemma this time using the facts that $\zeta_{S_0} \circ (0_{S_0}, i) = (0_{S_0}, 0_{S_0}, 0_{S_0}, i)$ and Proposition 35(8)
\[
D(\zeta) \cdot \zeta_{S_0, T_0} = \zeta_{S_0, T_0} \cdot D(\zeta) \circ \zeta_X.
\]
We obtain the axiom directly now,
\[
D^* D^*(i, j) \circ \zeta_X = ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\zeta_X \circ \zeta])
\]
\[
= ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\zeta_X])
\]
\[
= D^* (i, j)
\]
CD7. $D^* D^*(i, j) \circ \delta_X = D^* D^*(i, j)$. 
We invoke the shim lemma with $\delta_{S_0, S_0} \circ (0_{S_0}, 0_{S_0}, 0_{S_0}, i) = (0_{S_0}, 0_{S_0}, 0_{S_0}, i)$ and Proposition 35(9)
\[
D(\delta) \cdot \delta_{S_0, T_0} = \delta_{S_0, T_0} \cdot D(\delta) \circ \delta_X.
\]
We obtain the axiom directly now,
\[
D^* D^*(i, j) \circ \delta_X = ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\delta_X \circ \delta])
\]
\[
= ((0_{S_0}, 0_{S_0}, 0_{S_0}, i), [\delta_X])
\]
\[
= D^* D^*(i, j)
\]