Mathematical Structures in Computer Science (2025), **35**, e10, pp. 1–35
doi:10.1017/S0960129524000331

CAMBRIDGE UNIVERSITY PRESS

PAPER

Differentiable causal computations via delayed trace (extended version)

David Sprunger[1] and Shin-ya Katsumata[2]

[1]Indiana State University, Terre Haute, IN, USA and [2]National Institute of Informatics, Chiyoda, Tokyo, Japan
Corresponding author: David Sprunger; Email: david.sprunger@indstate.edu

(Received 1 December 2023; revised 26 August 2024; accepted 26 September 2024)

Abstract

We investigate causal computations, which take sequences of inputs to sequences of outputs such that the nth 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 (BPTT), a technique from machine learning based on unrolling of functions. This obtains a swath of properties for BPTT, 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.

Keywords: Delayed trace operators; artesian differential categories; recurrent neural networks; backpropagation through time; signal flow graphs

1. 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 (RNNs), compute a *stateful* and particularly a *causal* function on an infinite sequence of inputs. Here, "causal" means the output of the function at a particular index in the sequence may depend only on the inputs at that index and before.[1] These devices share a basic operational scheme, depicted in Figure 1.

Here the box labeled ϕ 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 Φ, 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 RNN has inputs of two types: data inputs and parameters. *Training* a neural network means finding parameter values θ so that when θ is fixed (by the triangular device in Figure 2 which emits θ constantly), the resulting function of data inputs has a desired behavior.

The key insight of *gradient-based training* is that the derivative of Φ with respect to θ gives an accurate prediction about how the output of Φ will change in response to a small change in

© The Author(s), 2025. Published by Cambridge University Press.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

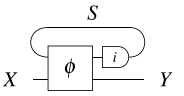


Figure 1. A basic causal device.

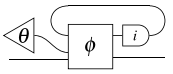


Figure 2. A simple RNN.

θ, allowing the trainer to make iterative small changes to θ which drive the network to a desired behavior.

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

The usual workaround is to first *unroll* Φ into a sequence of stateless functions, to which classical differentiation can be applied (Goodfellow et al., 2016). To be more precise, think of Φ as the solution to the recurrence relation $(s_{k+1}, y_k) = \phi(s_k, x_k)$ where $s_0 = i$. Let $\phi_S = \pi_0 \circ \phi$ and $\phi_Y = \pi_1 \circ \phi$. Then the unrolling of Φ 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 Φ is needed at an input of length k by a trainer, the gradient of ϕ_k at that input is used instead.

Taking the derivative of unrolled function is a useful procedure and is a basis of the machine learning algorithm called *backpropagation through time* (BPTT) (Werbos, 1990). However, its ad hoc nature raises some fundamental questions about the properties the procedure possesses. We study these formal properties and show that the differentiation of unrolled stateful functions enjoys a chain rule, a sum rule, a notion of partial derivative, 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 ϕ with itself, and therefore its componentwise derivative can be "re-rolled" back into a single stateful function.

1.1 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 13) and captures other stateful devices like Mealy machines and RNNs.

A distinctive feature of this construction includes the loop-with-delay gate seen in Figure 1, which we will more formally call a *delayed trace operator* (Definition 18). This delayed trace satisfies many of the properties of its better-known cousin, the trace operator of Joyal et al. (1996) (Proposition 19), but is missing the yanking condition and satisfies a modified form of dinaturality (Theorem 20).

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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* (Blute et al., 2009), then so is $\mathrm{St}(\mathbb{C})$ (Theorem 36). In particular, this differential operator matches the results obtained by unrolling-then-differentiating (Theorem 37). 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.

1.2 Differences from the conference version

Much of the material in this work previously appeared in a conference paper (Sprunger and Katsumata, 2019); here, we describe the differences. Section 2.3 revises Section II.C of the conference version (Sprunger and Katsumata, 2019) about the relationship between causal functions and morphisms in $\mathrm{St}(\mathbb{C})$. In this section, we define causal functions as families of functions depending on past inputs. We then show a new characterization of the category $\mathrm{St}(\mathbb{C})$ as the co-Kleisli category of a (generalized version of) Uustalu and Vene's non-empty list comonad on $\mathbb{C}^{\mathbb{N}}$. We remark that a similar result is shown for the category of monoidal streams (Lavore, de Felice, and Román, 2022, Theorem 6.3).

Section 5 has also been newly added. This section gives a functorial account of BPTT using the $\mathrm{St}(-)$ construction on a *reverse Cartesian differential category*. After the conference version (Sprunger and Katsumata, 2019), Cockett et al. introduced these RCDCs (Cockett et al., 2020), and Cruttwell et al. reformulated backpropagation for feedforward networks using the category of lenses (Cruttwell et al., 2022); in their work, the reverse differential operator is treated as a functor $R : \mathbb{C} \to \mathrm{Lens}(\mathbb{C})$ to the category of lenses. We refine this functor by restricting $\mathrm{Lens}(\mathbb{C})$ to the category $\mathrm{LLens}(\mathbb{C})$ of *linear lenses*. We show that the restriction $\mathbf{R}_l : \mathbb{C} \to \mathrm{LLens}(\mathbb{C})$ now preserves finite products, and the functor $\mathrm{St}(\mathbf{R}_l) : \mathrm{St}(\mathbb{C}) \to \mathrm{St}(\mathrm{LLens}(\mathbb{C}))$ computes the BPTT procedure.

2. Related Work

2.1 Categorical models of causal computations

Signal flow graphs are a widely used model of causal computation, especially in synchronous digital circuits and signal processing (Crochiere and Oppenheim, 1975; Parhi and Chen, 2013). 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 (2005, 2008), Milius (2010), Hansen et al. (2017), Basold et al. (2014), and many others achieve characterizations of computable streams by signal flow graphs. This coalgebraic perspective makes it possible to apply powerful coalgebraic techniques to analyze the behavior 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 realizing these flow graphs.

An axiomatic system for representing *digital circuits* based on monoidal category theory has been proposed by Ghica and Jung (2016) and Ghica et al., (2017). 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. (2014). Their transducers, represented as functions of

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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 (Fong et al., 2017). A notable example is Elliott (2018), 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 (Kissinger and Uijlen, 2017). Starting from a compact closed category with some extra structure, they refine it to a *-autonomous category so that morphisms there respect causal constraints.

2.2 State and feedback

Categorical constructions enhancing a basic category with some form of statefulness go back to Katis et al. (1997). Similar constructions appear in the signal flow graph literature and the growing graphical linear algebra literature (Bonchi et al., 2014, 2015, 2019). This line of work has a tradition of naming such constructions St(−). The construction given in the latter work by Bonchi et al. (2019) is very close to our St(−) construction. When it is applied to a compact closed PROP, it adjoins a one-input one-output generator to the PROP. However, it differs in that the semantics for these devices is relational, where ours is functional, and does not support arbitary initialization of delay gates.

Zanasi studies the PROP $\mathbb{IH}_R$ of *interacting Hopf algebras* over a ring R in his PhD thesis (Zanasi, 2015). The expressive power of this PROP is demonstrated by encoding various graphical systems into $\mathbb{IH}_R$ (Bonchi et al., 2014, 2015). 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 (Bonchi et al., 2014, Definition 7) and occupies a role analogous to the delay gate in our work.

At the moment of writing this paper, there are two papers that extend our St(−) construction. Carette et al. (2021) give an extension of the St(−) construction to discard categories, symmetric monoidal categories with a designated family of maps to the monoidal unit. As they are interested in quantum computing with memory, they also allow additional limited non-causal observational effects such as postselection. To accommodate these actions, they further generalize from equality in our truncation condition to a monotonicity condition by requiring a further property for the discard category (pseudo-purifiability) that forces it to be preorder-enriched. In particular, all Cartesian categories are pseudo-purifiable discard categories, and their construction specializes to ours in this case.

Lavore et al. (2022) extended our St(−) construction to a certain class of monoidal categories called productive monoidal categories. Their extension is based on their observation that stateful morphism sequences belong to the final solution of a recursively defined set. They then pursued this idea and obtained the limit characterization of the construction. Superficially, their approach is different from our approach of quotienting the stateful morphism sequences by extensional equivalences. However, when the underlying monoidal category is Cartesian, their construction coincides with the St(−) construction in this paper. They also reformulated our delayed trace operator as a feedback operator in the sense of Katis, Sabadini, and Walters, with a modification of inserting an endofunctor.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

Guardedness conditions on computations with feedback are a common theme (Abel and Pientka, 2013; Møgelberg, 2014). Goncharov and Schröder developed the theory of *guarded traced categories* to formalize this phenomena in Goncharov and Schröder (2018). The key idea is to restrict Joyal et al.'s trace operator (Joyal et al., 1996) 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.

Several research studies employ two-dimensional categorical structures to model state transition sytems. In Gadducci and Montanari's *tile model* (Gadducci and Montanari, 2000), each tile $f : A \overset{S}{\underset{S'}{\to}} B$ represents a state transition from A to B, and S and S' represents the trigger and the effect of this transition. Baez and Master consider a double category of open Petri nets and give two styles of functorial semantics (Baez and Master, 2020). In their double category, 2-cells are to relate two open Petri nets modulo renaming of open ports. In this paper, we use double category to represent two axis of information flow, namely the axis for time elapse and the axis for information processing.

2.3 Categorical models of differentiation

Inspired by the semantics of differential λ-calculus and differential proof nets by Ehrhard and Regnier (2003, 2005), Blute, Cockett, and Seely categorically formalized the differentiation operator in analysis. The formalization was first given in the categories where morphisms denote linear maps (Blute et al., 2006). Later, they introduced a new axiomatization (Blute et al., 2009) based on cartesian monoidal category where morphisms denote possibly nonlinear maps. This paper is based on the latter work and adopts more recent reformulations of differentiation operators studied in Cockett and Cruttwell (2014) and Cruttwell (2017).

3. 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 the homset for $X, Y \in |\mathbb{C}|$. We may abbreviate an identity map 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 is denoted by $\sigma_{X,Y} : X \times Y \to Y \times X$.

Bold metavariables—$\mathbf{X}$, $\mathbf{s}$, etc.—denote infinite sequences of mathematical objects, indexed by $\mathbb{N}$. The ith 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 dot $\bullet$ as a 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, $(i, [\mathbf{x}_\bullet + \mathbf{y}_\bullet])$ is $(i, \mathbf{x}_0 + \mathbf{y}_0,\ \mathbf{x}_1 + \mathbf{y}_1,\ \cdots)$ and $[(i, \mathbf{x}_\bullet + \mathbf{y}_\bullet)]$ 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 ϕ 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 $\forall i \in \mathbb{N} \cdot \mathbf{Z}_i = \mathbf{X}_i \times \mathbf{Y}_i$.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

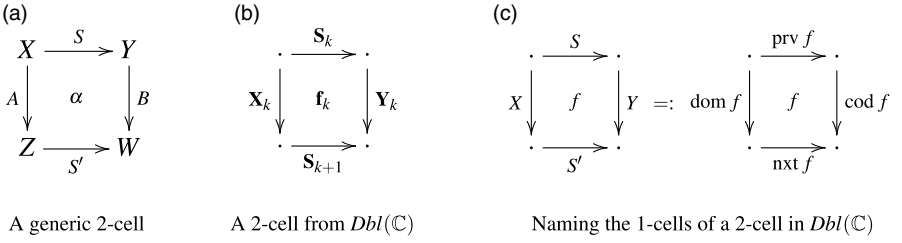


Figure 3. 2-cells in a double category.

4. Extending Cartesian Categories along Discrete Time

Before jumping into the depths of categorical abstraction, we consider 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 **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 computing a single entry in the output sequence.

These are functions taking sequences to sequences in a causal manner, but the kth output of $\mathbf{f}$ depends only on the kth 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 $\mathtt{tl} : (x_0, x_1, \dots) \mapsto (x_1, x_2, \dots)$ 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 neighboring 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 13).

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 kth 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 roles 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* (Ehresmann, 1963).

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 as in Figure 3a.

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 available 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.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

Our double category will therefore have a particularly simple structure, with 2-cells as depicted in Figure 3b. 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:*

- *The set of objects (0-cells) is* $\{\cdot\}$.
- *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 in Figure 3c, 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* ϕ *the underlying morphism of* f, *while* U *is the operation taking a 2-cell to its underlying morphism, so* $U f = \phi$.
- *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* $U(g * f) \triangleq (S' \times U g) \circ (\sigma_{S',T} \times Y) \circ (T \times U f) \circ (\sigma_{S,T} \times X)$.
- *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* $U(f; h) \triangleq (S'' \times \sigma_{Y,W}) \circ (U h \times Y) \circ (S' \times \sigma_{V,Y}) \circ (U f \times V)$.

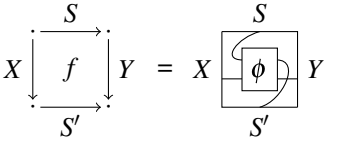
NB: vertical composition is given in relational composition order, while horizontal composition is given in functional composition order.

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 $\mathbb{C}$, we instead follow Mac Lane (Lane, 1998, p. 257) and supply an equivalent Cartesian category $\mathbb{C}^s$ defined by:

- objects are finite lists of $\mathbb{C}$-objects, and
- $\mathbb{C}^s((A_1, \cdots, A_n), (B_1, \cdots, B_m)) = \mathbb{C}\left(\prod A_i, \prod B_j\right)$.

String diagrams for the underlying $\mathbb{C}$-morphisms of horizontal and vertical composites may be helpful to digest this definition. The underlying morphisms of the horizontal composition $g * f$ and vertical composition $f; h$ are shown in Figure 4.

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



https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

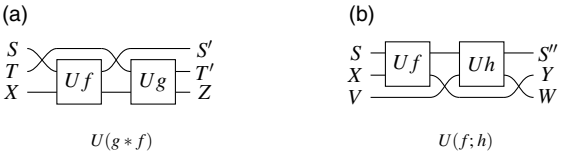


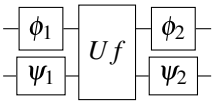
Figure 4. Underlying morphisms for composites of 2-cells.

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 $\phi^h : X \xrightarrow[1]{} Y$ 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 id^h_X are identities for horizontal composition, and similarly id^v_S are identities for vertical composition!) More practically, 2-cells of the form ϕ^h modify values only, while 2-cells of the form ϕ^v modify states only.

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 ϕ_2 and ψ_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.

4.1 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 $\mathrm{dom}\, \mathbf{s}_\bullet = \mathbf{X}_\bullet$, $\mathrm{cod}\, \mathbf{s}_\bullet = \mathbf{Y}_\bullet$, and $\mathrm{nxt}\, \mathbf{s}_\bullet = \mathrm{prv}\, \mathbf{s}_{\bullet+1}$. 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 Figure 5a.

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

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

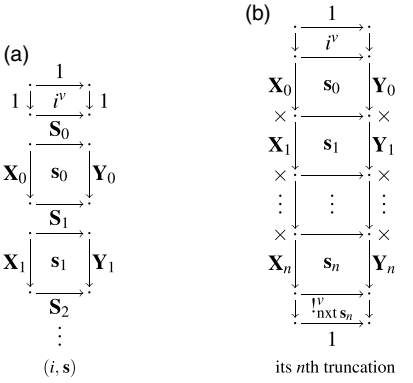


Figure 5. A stateful morphism sequence and a truncation.

value of type $\mathbf{S}_1$. 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

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

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 nth 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 Figure 5a.[2]

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.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

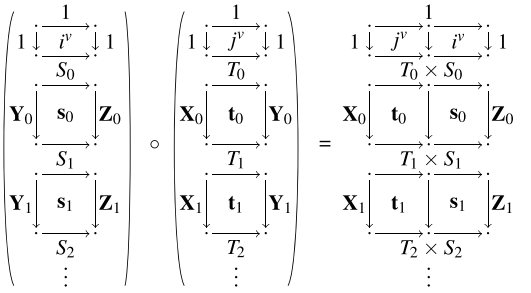


Figure 6. Composition of stateful morphism sequences.

Lemma 7 (Shim lemma).[3] *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* $\mathbf{b}_\bullet : \mathrm{prv}\,(\mathbf{s}_\bullet) \to \mathrm{prv}\,(\mathbf{t}_\bullet)$, $\mathbf{b}_0 \circ i = j$, *and* $\mathbf{s}_\bullet; \mathbf{b}^\nu_{\bullet+1} = \mathbf{b}^\nu_\bullet; \mathbf{t}_\bullet$. *Then* $(i, \mathbf{s})$ *and* $(j, \mathbf{t})$ *are extensionally equivalent.*

Proof. Show by induction that

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

□

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

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

For instance, the recurrently defined functions ϕ_n in section 1 are unrollings of a certain stateful morphism sequence involving ϕ and i.

4.2 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 Figure 6. 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)* $\mathrm{id}_\mathbf{X}$ *is the unit for the composition operation.*

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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 9 to the extensional equivalence classes by Lemma 10.*

We will justify our use of the word "causal" by establishing a connection to another formalization of causal functions in Theorem 13, 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, that is, $[1]$.

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 $n \in \{0, 1\}$. □

We extend the truncation Tc operation and the unrolling operation Un to morphisms in $\mathrm{St}(\mathbb{C})$. They are now regarded as the following functions on homsets:

$$\mathrm{Tc}_n : \mathrm{St}(\mathbb{C})(\mathbf{X}, \mathbf{Y}) \to \mathbb{C}\left(\prod_{k=0}^{n} \mathbf{X}_k, \prod_{k=0}^{n} \mathbf{Y}_k\right), \qquad \mathrm{Un}_n : \mathrm{St}(\mathbb{C})(\mathbf{X}, \mathbf{Y}) \to \mathbb{C}\left(\prod_{k=0}^{n} \mathbf{X}_k, \mathbf{Y}_n\right).$$

We also extend the $\mathrm{St}(-)$ construction to functors. Let $\mathbb{C}, \mathbb{D}$ be strict Cartesian categories and $F : \mathbb{C} \to \mathbb{D}$ be a functor that strictly preserves finite products. We define $\mathrm{St}(F) : \mathrm{St}(\mathbb{C}) \to \mathrm{St}(\mathbb{D})$ by:

$$\mathrm{St}(F)(\mathbf{X}) = [F(\mathbf{X}_\bullet)] \qquad \mathrm{St}(F)(i, \mathbf{s}) = (Fi, [F(\mathbf{s}_\bullet)]).$$

When F only preserves finite products, insert appropriate canonical isomorphisms around $F\mathbf{s}_\bullet$.

4.3 Morphisms in St($\mathbb{C}$) and Causal morphisms

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 causal computations as functions depending on past inputs and morphisms in $\mathrm{St}(\mathbb{C})$ for a strict Cartesian category $\mathbb{C}$.

What is a causal computation from $\mathbf{X}$ to $\mathbf{Y}$? Suppose f is such a computation. Then the output at time n is computed from all the inputs up to time n. Therefore the input-output relationship at time n is given by a function $f_n : \mathbf{X}_0 \times \cdots \times \mathbf{X}_n \to \mathbf{Y}_n$, and the collection of all such functions represents a causal computation from $\mathbf{X}$ to $\mathbf{Y}$. We thus regard the family $f \triangleq \{f_n\}_{n \in \mathbb{N}}$ as the entity embodying a causal computation and naturally view it as a morphism in $\mathbb{C}^{\mathbb{N}}$ (the $\mathbb{N}$-fold product category of $\mathbb{C}$).

We point out that f is a morphism in the co-Kleisli category of a comonad D on $\mathbb{C}^{\mathbb{N}}$. This comonad's functor part is given by $D(\mathbf{X})_n \triangleq \mathbf{X}_0 \times \cdots \times \mathbf{X}_n$. This is a refinement of Uustalu and Vene's non-empty list comonad capturing causal stream functions (Uustalu and Vene, 2005). The difference is that D is defined over the category $\mathbb{C}^{\mathbb{N}}$ of time-varying objects and morphisms, while their comonad is defined over a Cartesian category with countable coproducts. Indeed, Uustalu

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

and Vene's non-empty list comonad is $\Sigma \circ D \circ \Delta$, where $\Sigma \dashv \Delta : \mathbb{C} \to \mathbb{C}^{\mathbb{N}}$ is the adjunction arising from the countable coproduct in $\mathbb{C}$.

We show that the causal computations modeled by co-Kleisli morphisms of D coincides with those modeled by $\mathrm{St}(\mathbb{C})$-morphisms. This isomorphism morally states the coincidence of memoryful computations and causal computations.

Theorem 13. $(\mathbb{C}^{\mathbb{N}})_D \cong \mathrm{St}(\mathbb{C})$.

In Lavore, de Felice, and Román (2022) (Theorem 6.3), the co-Kleisli category $(\mathbb{C}^{\mathbb{N}})_D$ is shown to be isomorphic to the category of monoidal streams over $\mathbb{C}$, which is differently constructed from the $\mathrm{St}(-)$ construction.

Proof. We first decompose D into an adjunction. Let $\iota : \mathbb{N} \to \omega^{op}$ be the inclusion of the discrete poset $\mathbb{N}$ into the opposite of the poset ω of finite ordinals. The precomposition functor $- \circ \iota : \mathbb{C}^{\omega^{op}} \to \mathbb{C}^{\mathbb{N}}$ has a right adjoint given by the right Kan extension R along ι:

$$\mathbb{C}^{\mathbb{N}} \underset{-\circ\iota}{\overset{R}{\underset{\longleftarrow}{\overset{\longrightarrow}{\top}}}} \mathbb{C}^{\omega^{op}} .$$

It sends an object $\mathbf{X} \in \mathbb{C}^{\mathbb{N}}$ to the sequence of prefix projections:

$$R\mathbf{X} = \mathbf{X}_0 \xleftarrow{c_{\mathbf{X},0}} \mathbf{X}_0 \times \mathbf{X}_1 \xleftarrow{c_{\mathbf{X},1}} \mathbf{X}_0 \times \mathbf{X}_1 \times \mathbf{X}_2 \xleftarrow{c_{\mathbf{X},2}} \mathbf{X}_0 \times \mathbf{X}_1 \times \mathbf{X}_2 \times \mathbf{X}_3 \longleftarrow \cdots .$$

In particular, its object part is given by $(R\mathbf{X})_n = \prod_{i=0}^{n} \mathbf{X}_i$. We note that the projection $c_{\mathbf{X},n} : \mathbf{X}_0 \times \cdots \times \mathbf{X}_{n+1} \to \mathbf{X}_0 \times \cdots \times \mathbf{X}_n$ can be given as $c_{\mathbf{X},n} \triangleq U(\mathrm{id}^h_{\mathbf{X}_0}; \ldots ; \mathrm{id}^h_{\mathbf{X}_n}; !^h_{\mathbf{X}_{n+1}})$.

For brevity, let $L = - \circ \iota$. It holds $L \circ R(\mathbf{X}) = D(\mathbf{X})$, and D acquires a comonad structure from the adjunction $L \dashv R$. Therefore to show the theorem, it suffices to exhibit an isomorphism on homsets: $\Phi_{\mathbf{X},\mathbf{Y}} : \mathrm{St}(\mathbb{C})(\mathbf{X}, \mathbf{Y}) \cong \mathbb{C}^{\omega^{op}}(R\mathbf{X}, R\mathbf{Y})$. Define

$$(\Phi_{\mathbf{X},\mathbf{Y}}(i, \mathbf{s}))_n \triangleq \mathrm{Tc}_n(i, \mathbf{s}).$$

This is a good definition, as $(i, \mathbf{s})$ is an extensional equivalence class.

We first show that Φ (or equivalently Tc) is natural in n. By unfolding the definition, we obtain

$$c_{\mathbf{Y},n} \circ \mathrm{Tc}_{n+1}(i, \mathbf{s}) = U(i^v; \mathbf{s}_0; \ldots ; (!^h_{\mathbf{Y}_{n+1}} * \mathbf{s}_{n+1}); !^v_{S_{n+2}}).$$

The vertical composition of the last two 2-cells satisfies $U((!^h_{\mathbf{Y}_{n+1}} * s_{n+1}); !^v_{S_{n+2}}) = !^v_{S_{n+1}} * !^h_{\mathbf{X}_{n+1}}$. Therefore,

$$\begin{aligned} U(i^v; \mathbf{s}_0; \ldots ; (!^h_{\mathbf{Y}_{n+1}} * \mathbf{s}_{n+1}); !^v) &= U(i^v; \mathbf{s}_0; \ldots \mathbf{s}_n; (!^v_{S_{n+1}} * !^h_{\mathbf{X}_{n+1}})) \\ &= U(i^v; \mathbf{s}_0; \ \ldots ; \mathbf{s}_n; !^v_{S_{n+1}}) \circ U(\mathrm{id}^h_{\mathbf{X}_0}; \ \ldots ; \mathrm{id}^h_{\mathbf{X}_n}; !^h_{\mathbf{X}_{n+1}}) \\ &= \mathrm{Tc}_n(i, \mathbf{s}) \circ c_{\mathbf{X},n}. \end{aligned}$$

This ends the proof of naturality.

We next define $l_{\mathbf{X},\bullet} : (R\mathbf{X})_\bullet \to \mathbf{X}_\bullet$ to be the projection of the last component. Since we are working in a strict Cartesian category, we have $l_{\mathbf{X},0} = \mathrm{id}_{\mathbf{X}_0}$ and $\langle c_{\mathbf{X},\bullet-1}, l_{\mathbf{X},\bullet} \rangle = \mathrm{id}_{(R\mathbf{X})_\bullet}$.

We show the surjectivity of Φ. Let $\alpha : R\mathbf{X} \to R\mathbf{Y} : \omega^{op} \to \mathbb{C}$ be a natural transformation. We define 2-cells $i_\alpha : 1 \xrightarrow[1]{1} 1$ by $i_\alpha \triangleq \mathrm{id}^v_1$ and $(\mathbf{s}_\alpha)_\bullet : \mathbf{X}_\bullet \xrightarrow[(R\mathbf{X})_\bullet]{(R\mathbf{X})_{\bullet-1}} \mathbf{Y}_\bullet$ by (below δ is the diagonal morphism):

$$(\mathbf{s}_\alpha)_\bullet \triangleq (\mathrm{id}_{(R\mathbf{X})_\bullet} \times (l_{\mathbf{Y},\bullet} \circ \alpha_\bullet)) \circ \delta_{(R\mathbf{X})_\bullet} : (R\mathbf{X})_{\bullet-1} \times \mathbf{X}_\bullet = (R\mathbf{X})_\bullet \to (R\mathbf{X})_\bullet \times \mathbf{Y}_\bullet.$$

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

We show the following equality by induction, which is sufficient to show $\Phi(i_\alpha, \mathbf{s}_\alpha) = \alpha$:

$$U(i_\alpha; (\mathbf{s}_\alpha)_0; \ldots; (\mathbf{s}_\alpha)_\bullet) = (\mathrm{id}_{(R\mathbf{X})_\bullet} \times \alpha_\bullet) \circ \delta_{(R\mathbf{X})_\bullet}. \tag{3}$$

The case $n = 0$ is trivial. We show the induction case. First, a diagrammatic calculation using the induction hypothesis shows

$$U(i_\alpha; (\mathbf{s}_\alpha)_0; \ldots; (\mathbf{s}_\alpha)_{n+1}) = (\mathrm{id}_{(R\mathbf{X})_{n+1}} \times \langle \alpha_n \circ c_{\mathbf{X},n+1}, l_{\mathbf{X},n+1} \circ \alpha_{n+1} \rangle) \circ \delta_{(R\mathbf{X})_{n+1}}.$$

Then from the naturality of α, the tupled morphism part satisfies

$$\langle \alpha_n \circ c_{\mathbf{X},n+1}, l_{\mathbf{X},n+1} \circ \alpha_{n+1} \rangle = \langle c_{\mathbf{Y},n+1} \circ \alpha_{n+1} \circ, l_{\mathbf{X},n+1} \circ \alpha_{n+1} \rangle = \langle c_{\mathbf{Y},n+1}, l_{\mathbf{X},n+1} \rangle \circ \alpha_{n+1} = \alpha_{n+1}.$$

Therefore, (3) is proved.

Moreover, $(i_\alpha, \mathbf{s}_\alpha)$ is the unique extensional equivalence class yielding α. Let $(i', \mathbf{s}') \in \mathrm{St}(\mathbb{C})(A_\bullet, B_\bullet)$ be another one such that $(\Phi_{\mathbf{X},\mathbf{Y}}(i', \mathbf{s}'))_n = Tc_n(i', \mathbf{s}') = \alpha_n = Tc_n(i_\alpha, \mathbf{s}_\alpha)$. Then, $(i', \mathbf{s}')$ and $(i_\alpha, \mathbf{s}_\alpha)$ are extensionally equivalent, hence $(i', \mathbf{s}') = (i_\alpha, \mathbf{s}_\alpha)$. □

Below we let $\Gamma_\mathbb{C} : (\mathbb{C}^\mathbb{N})_D \to \mathrm{St}(\mathbb{C})$ be one side of the isomorphism of Theorem 13. It is identity on objects and sends $\mathbf{f} \in \mathbb{C}^\mathbb{N}(D\mathbf{X}, \mathbf{Y})$ to $(\mathrm{id}_1^\nu, [\langle \mathrm{id}_{(R\mathbf{X})_\bullet}, \mathbf{f}_\bullet \rangle])$. Conversely, Γ^{-1} sends an extensional equivalence class $(i, \mathbf{s})$ to $[\mathrm{Un}_\bullet(i, \mathbf{s})]$.

Next, for a functor $F : \mathbb{C} \to \mathbb{D}$ strictly preserving finite products, we define $C(F) : (\mathbb{C}^\mathbb{N})_D \to (\mathbb{D}^\mathbb{N})_D$ by:

$$C(F)(\mathbf{X}) \triangleq [F(\mathbf{X}_\bullet)], \qquad C(F)(\mathbf{f}) \triangleq [F(\mathbf{f}_\bullet)]. \tag{4}$$

Lemma 14. $\mathrm{St}(F) \circ \Gamma_\mathbb{C} = \Gamma_\mathbb{D} \circ C(F)$.

Proof. It suffices to show the above equality on morphisms. Since F strictly preserves finite products, we have

$$\begin{aligned} \mathrm{St}(F)(\Gamma_\mathbb{C}(\mathbf{f})) &= \mathrm{St}(F)(\mathrm{id}_1^\nu, [\langle \mathrm{id}_{(R\mathbf{X})_\bullet}, \mathbf{f}_\bullet \rangle]) \\ &= (\mathrm{id}_1^\nu, [\langle \mathrm{id}_{(R[F(\mathbf{X}_\bullet)])_\bullet}, \mathbf{f}_\bullet \rangle]) = \Gamma_\mathbb{D}(C(F)(F(\mathbf{f}))). \end{aligned}$$

□

4.4 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 13. 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 $\mathrm{St}_0(\mathbb{C})$.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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:*

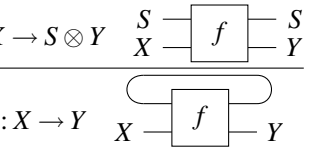
$$H_0 X = [X] \qquad\qquad H_0 f = (\mathrm{id}_1, [f^h]).$$

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.

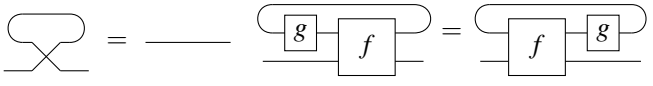
5. Delayed Trace Operator

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* (Joyal et al., 1996), 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:

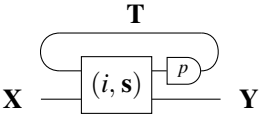
$$\frac{f : S \otimes X \to S \otimes Y}{tr^S(f) : X \to Y}$$



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 toward 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 17. *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:* ${}^{T}\lfloor f \rceil_{T'} : X \xrightarrow[S' \times T']{S \times T} Y$.

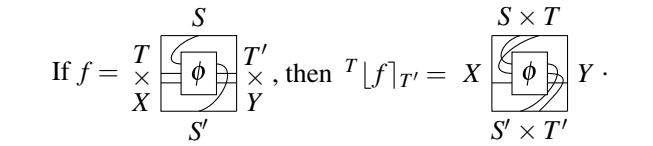
https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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

Figure 7. Equalities satisfied by delayed trace operator.

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:



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

Definition 18. *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* $\mathrm{St}(\mathbb{C})(\mathbf{X},\mathbf{Y})$:

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

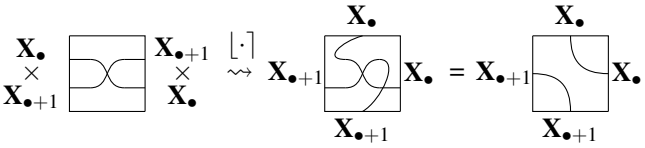
Note this operation is well defined on extensional equivalence classes of stateful morphism sequences and therefore is an operation on $\mathrm{St}(\mathbb{C})$ morphisms. The delayed trace of $\mathrm{St}(\mathbb{C})$ already differs from the standard monoidal trace in two ways: first, the domain and 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 19. *Suppose $(i,\mathbf{s}):\mathbf{T}\times\mathbf{X}\to\bigcirc\mathbf{T}\times\mathbf{Y}$ is a morphism in* $\mathrm{St}(\mathbb{C})$. *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 7, *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

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

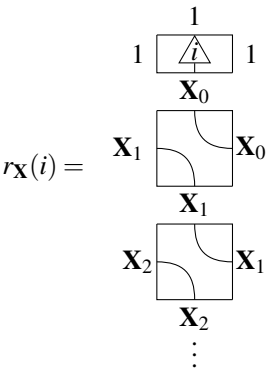
To get a better understanding of $r_{\mathbf{X}}(i)$, we first draw the value-to-state conversion in a single 2-cell in this morphism.



https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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 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* (Leiserson and Saxe, 1991), and initial states of registers is a delicate issue in this subject. The delayed trace operator satisfies the following modified dinaturality property:

Theorem 20. *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* $u : 1 \to \mathbf{U}_0$,

$$\mathrm{tr}^{\mathbf{U}}_{u}\left(((j, \mathbf{g}) \times \mathbf{X}) \circ (i, \mathbf{s})\right) = \mathrm{tr}^{\mathbf{T}}_{u'}\left((i, \mathbf{s}) \circ ((j', \bigcirc\,\mathbf{g}) \times \mathbf{Y})\right)$$

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

Proof. Let $r_i = \lfloor (\mathrm{id}^h_{Y_i} \times g_i) * s_i \rceil$. By induction, obtain the equality of this:

$$U(r_0; \cdots; r_n) \circ (\mathrm{id}_{S_{n+1} \times \prod^n Y_i} \times \sigma_{T_{n+1}, U_{n+1}}) \tag{5}$$

and this:

$$U(t_0; \cdots; t_n) \circ (\mathrm{id}_{\prod^n Y_i} \times U(g_n) \times \mathrm{id}_{T_{n+1}}). \tag{6}$$

Then observe the nth truncation of (r_i) is (5) followed by $\pi_{\prod^n Y_i}$ and similarly, the nth truncation of (t_i) is (6) followed by the same projection. Therefore, they are observationally equivalent, though not equal as sequences. □

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 21. *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.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

Theorem 22. *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.

5.1 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 23. *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* $u : 1 \to U$,

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

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

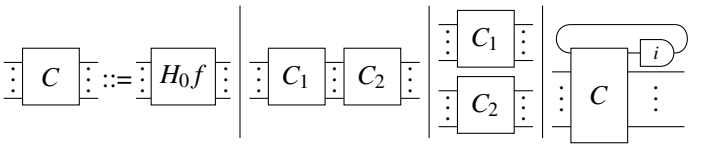
Corollary 24. *For any* $f : X \to Y$ *in* $\mathbb{C}$, *and initial state* $i : 1 \to X$, *we have* $H[f] \circ r_{[X]}(i) = r_{[Y]}(f \circ i) \circ H([f])$.

5.2 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 22 indicates that we can generate all $\mathrm{St}_0(\mathbb{C})$ morphisms with the following grammar:

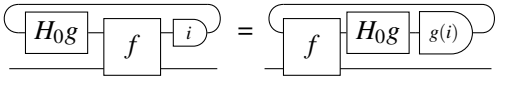
$$\varphi ::= H_0 f \mid \varphi_1 \circ \varphi_2 \mid \varphi_1 \times \varphi_2 \mid \mathrm{tr}_i^{\mathbf{S}}(\varphi)$$

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



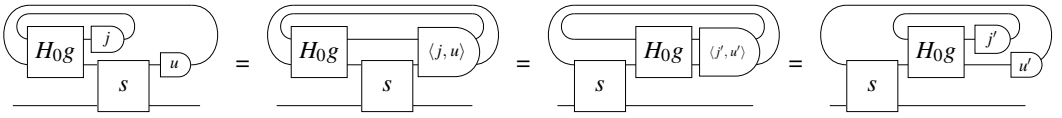
where the box labeled $H_0 f$ has m inputs and n outputs when $f : \prod_{k=1}^{m} A_k \to \prod_{k=1}^{n} B_k$. As is typical in string diagrams, $H_0 \mathrm{id}_X$ is depicted by a wire and $H_0 \sigma_{X,Y}$ by a wire crossing. Additionally, we depict $H_0 !_A$ and $H_0 \langle \mathrm{id}_A, \mathrm{id}_A \rangle$ 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 23, 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 20).

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



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 the formal axiomatization of general delayed trace operators and free categories with these structures for future work and move on to the study of the differentiability of the causal computations realized by $\mathrm{St}(\mathbb{C})$.

6. 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 et al. (2009). We begin by recalling background.

Definition 25 (Blute et al. (2009)). *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* $0_{X\times Y} = 0_X \times 0_Y$ *and* $+_{X\times Y} = (+_X \times +_Y) \circ (X \times \sigma_{Y,X} \times Y)$.

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

Example 1 (Blute et al. (2009)). *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 26. *If* $\mathbb{C}$ *is a left additive category, so is* $\mathrm{St}(\mathbb{C})$.

Proof. We already know $\mathbb{C}$ being Cartesian implies $\mathrm{St}(\mathbb{C})$ is Cartesian. A commutative monoid structure on $\mathbf{X}$ is given by componentwise monoid structure: $+_{\mathbf{X}} \triangleq [+_{\mathbf{X}_\bullet} : \mathbf{X}_\bullet \times \mathbf{X}_\bullet \xrightarrow[1]{1} \mathbf{X}_\bullet]$ and $0_{\mathbf{X}} \triangleq [0_{\mathbf{X}_\bullet} : 1 \xrightarrow[1]{1} \mathbf{X}_\bullet]$. □

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

Definition 27. *Let* $\mathbb{C}$ *be a Cartesian left additive category. For every object* X *from* $\mathbb{C}$ *[or pair of objects* (X, Y)*], let*

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

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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

Definition 28. *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 Blute et al. (2009), 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 Cockett and Cruttwell (2014) (Proposition 4.2).

Example 2 (Blute et al. (2009)). $\mathbf{Euc}_\infty$ *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 evaluated at x_2.*

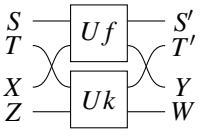
In light of the standard example, we can describe the ideas behind the CD axioms. CD1 says that the basic morphisms provided by the structure of the Cartesian left additive category are their own derivatives (i.e. $Js|_{x_2} \times x_1 = s(x_1)$), while CD2 and CD3 express the fact that $Jf|_{x_2} \times x_1$ is linear in 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. 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 29. *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:



https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

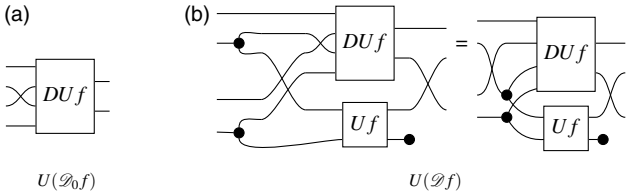


Figure 8. Underlying morphisms for Definition 31.

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 30. $(i, \mathbf{f}) \times (j, \mathbf{g}) = (\langle i, j\rangle, [\mathbf{f}_\bullet \boxtimes \mathbf{g}_\bullet])$ *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 31. *We define two endofunctions on 2-cells from* $Dbl(\mathbb{C})$. *The first,* $\mathscr{D}_0$, *takes the 2-cell* $f : X \xrightarrow[S']{S} Y$ *to the 2-cell* $\mathscr{D}_0 f : X \times X \xrightarrow[S']{S\times S} Y$ *with* $U\mathscr{D}_0 f \triangleq DUf \circ \delta_{S,X}$.

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

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

The string diagrams for the underlying morphisms of $\mathscr{D}_0$ and $\mathscr{D}$ may be easier to understand; these are shown in Figure 8.

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

Proposition 32. *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* $\mathscr{D}$:

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

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

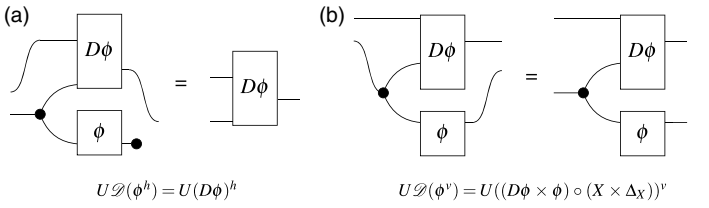


Figure 9. String diagrams for property 1.

7. $\zeta^v_S; \mathscr{D}\mathscr{D}f * \zeta^h_X = \mathscr{D}f; \zeta^v_{S'}$
8. $\delta^v_{S,S}; \mathscr{D}\mathscr{D}f * \delta^h_{X,X} = \mathscr{D}\mathscr{D}f; \delta^h_{S',S'}$

Proof. 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 (1) the same boundary 1-cells and (2) the same underlying $\mathbb{C}$-morphism.

Checking (1), namely that the source and target 1-cells of the 2-cells on each side of each of these eight equations match, is straightforward. Checking (2) 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.

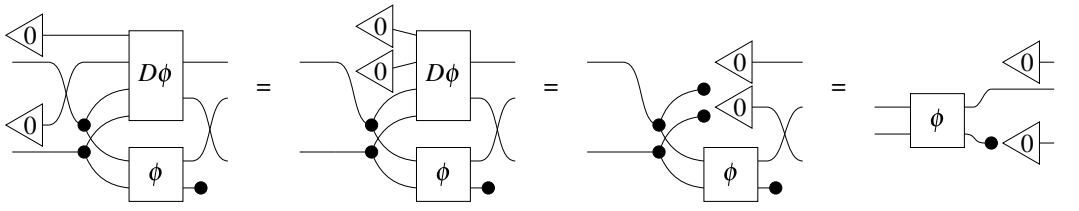
Throughout this proof, let $Uf = \phi$, $Ug = \psi$, $Uh = \xi$, and $Uk = \kappa$.

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

When ϕ is oriented horizontally, it has no state. This simplifies the diagram for the underlying morphism of $\mathscr{D}$ (from Figure 8b) considerably; properties of copy-discard show this is equal to the underlying morphism of $(D\phi)^h$. When ϕ is oriented vertically, it has no values. Omitting the value portion of $U\mathscr{D}f$ yields the left side of Figure 9b which is exactly $(D\phi \times \phi) \circ (X \times \Delta_X)$.

Property 2: $(0_S \times S)^v; \mathscr{D}f * (0_X \times X)^h = (0_Y \circ !_Y)^h * f; (0_{S'} \times S')^v$

From Lemma 3 and the diagram for the underlying morphism of $\mathscr{D}f$,

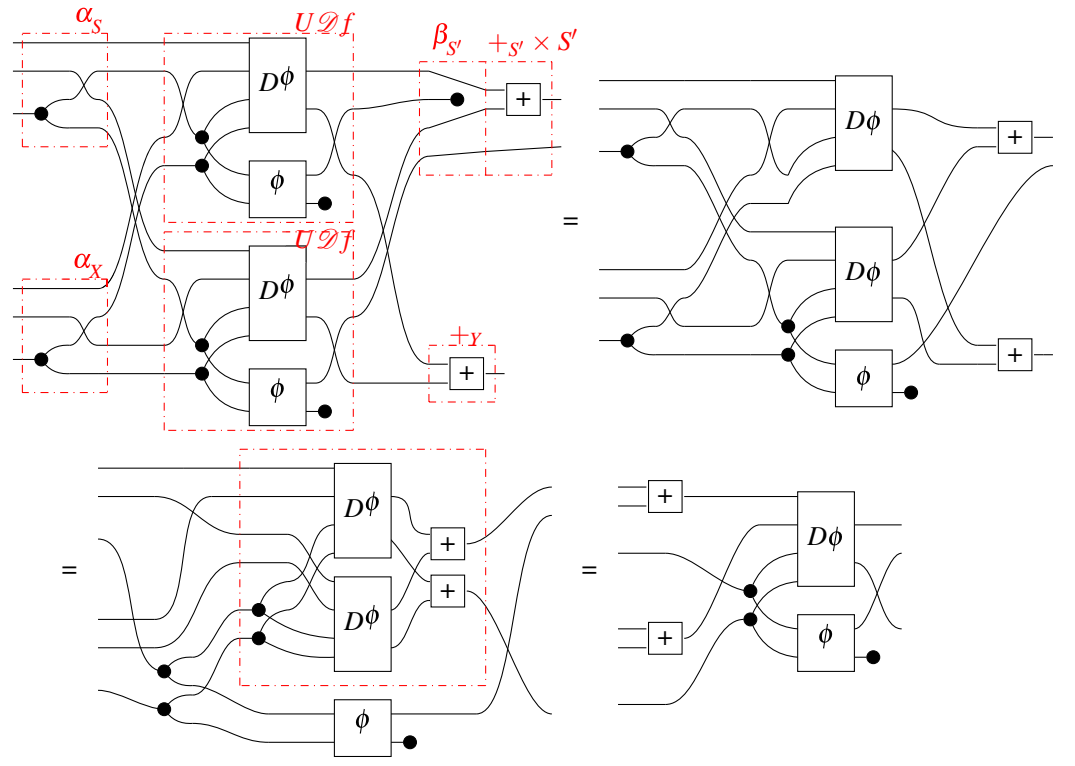


where the middle equality is by the CD2 axiom for D.

Property 3: $(+_S \times S)^v; \mathscr{D}f * (+_X \times X)^h = \alpha^v_S; (+^h_Y * \mathscr{D}f \boxtimes \mathscr{D}f * \alpha^h_X); \beta^v_{S'}; (+_{S'} \times S')^v$

We start from the right-hand side of this equation. To aid parsing this complicated string diagram, we draw a red box around parts of the diagram to serve as a label. Everything in red has no mathematical meaning—it is only there for aid of understanding.

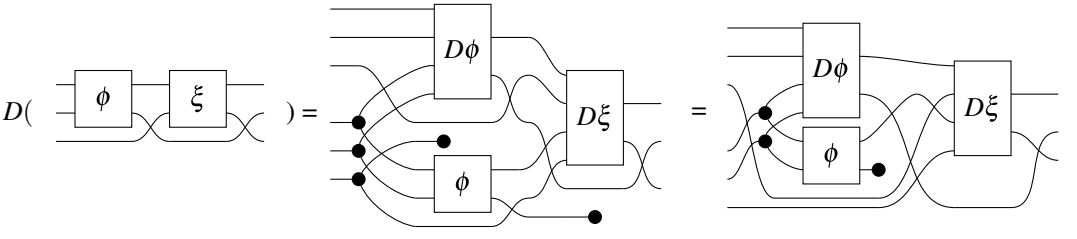
https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



In the first step, we have used properties of copying and discarding to eliminate the middle ϕ morphism, preceding copies, and succeeding discards. In the next step, we use properties of monoidal categories and properties of copying to rearrange, so the boxed region is the left-hand side of CD3 $(+_{S'\times Y} \circ (D\phi \times D\phi) \circ \alpha_{S\times X})$. In the final step, we use CD3 and further monoidal category properties.

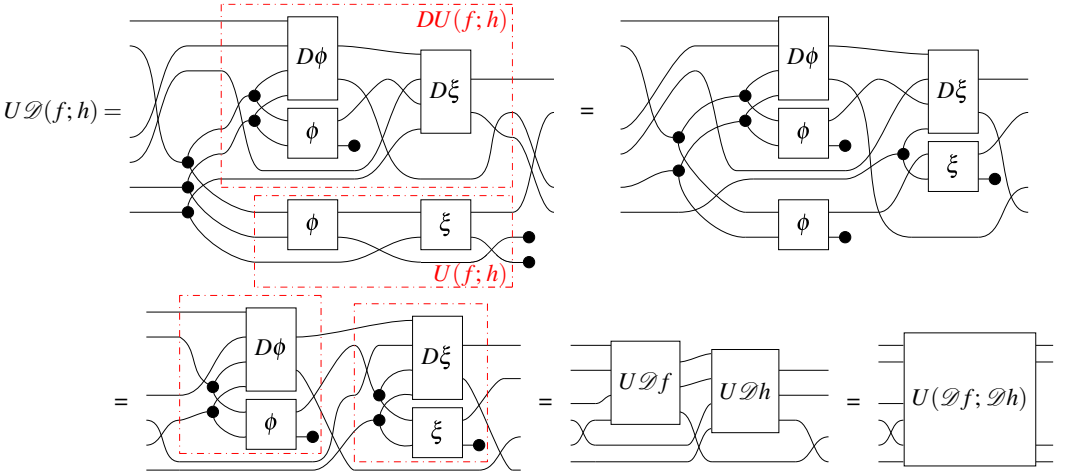
Property 4: $\mathscr{D}(f;h) = (\mathscr{D}f;\mathscr{D}h) * \delta^h_{X,Z}$.

Using the CD axioms for D, we compute the differential of the underlying morphism of $f;h$, then do some simplification:



We use that result to construct a string diagram for $U\mathscr{D}(f;h)$:

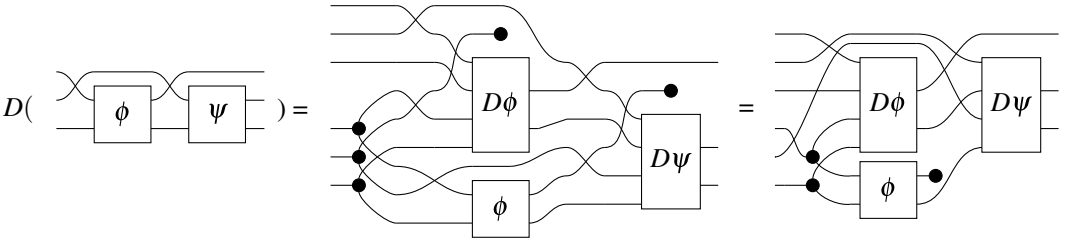
https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



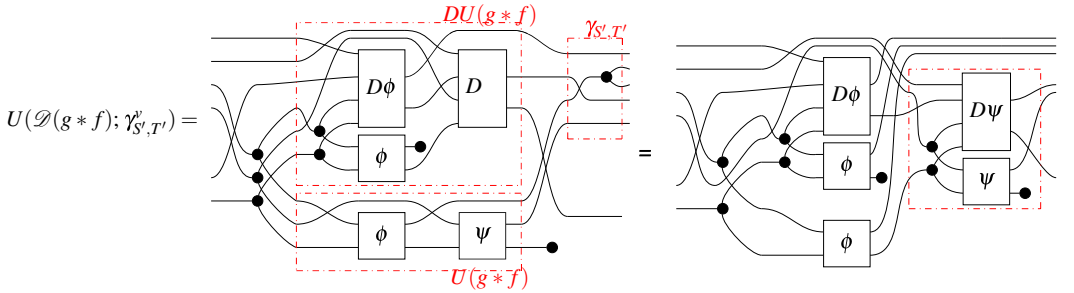
In the first step, we use properties of monoidal categories to rearrange the composition. The second step uses naturality of the copy to eliminate a ϕ, along with further monoidal rearrangements. At this point, the underlying morphisms for $\mathscr{D}f$ and $\mathscr{D}h$ have appeared, so we substitute these in the next step. Finally, we use the definition of vertical composition and arrive at the string diagram of $U((\mathscr{D}f;\mathscr{D}h) * \delta^h_{X,Z})$ (cf. Lemma 3).

Property 5: $\mathscr{D}(g * f);\gamma^v_{S',T'} = \gamma^v_{S,T};(\mathscr{D}g * (\mathscr{D}f \boxtimes f) * (X \times \Delta_X)^h)$.

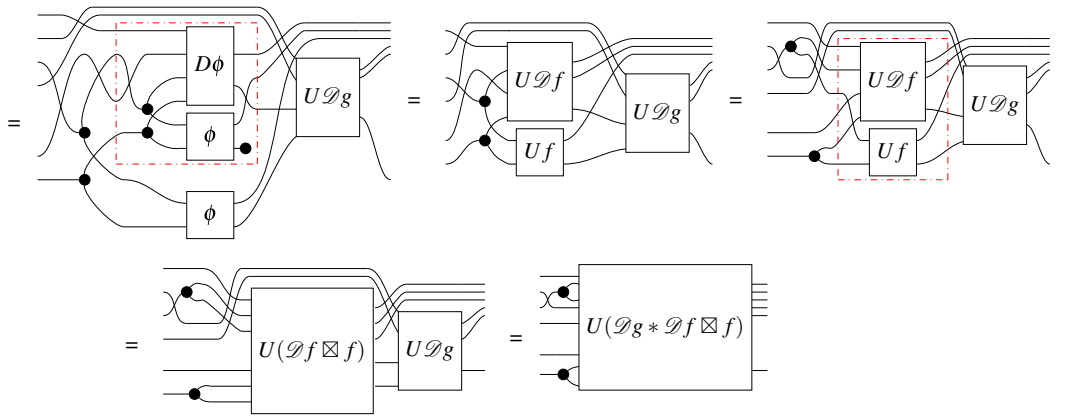
Using the CD axioms for D, we compute the differential of the underlying morphism of $g * f$:



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



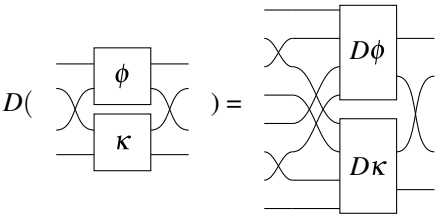
https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



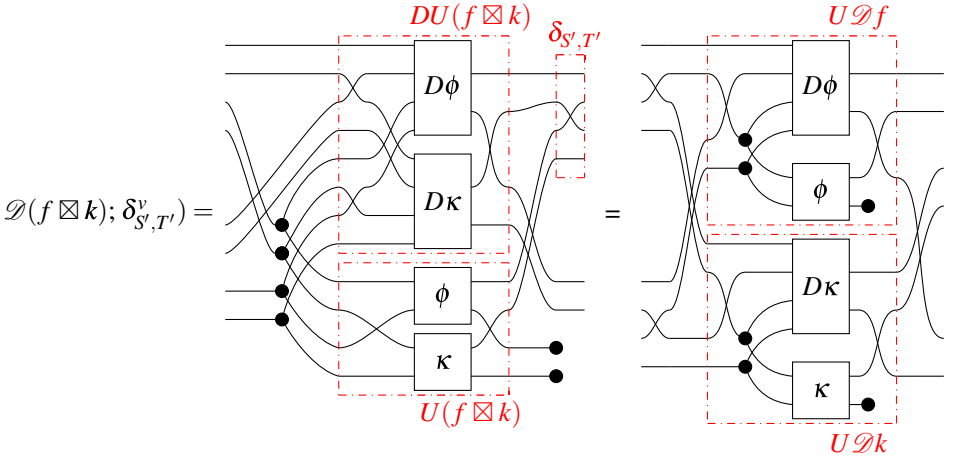
In the first row, we make extensive use of properties of copy-discard monoidal categories to realize the underlying morphisms of $\mathscr{D}g$ and $\mathscr{D}f$ (hinted in red). In the next two steps, we substitute these into the diagram. The final step in the second row is a monoidal rearrangement which finds the underlying morphism of $\mathscr{D}f \boxtimes f$ (again in red) by definition of $\boxtimes$. Then using the definition of horizontal composition in the last step, we obtain the string diagram of $U(\gamma^{v}_{S,T}; (\mathscr{D}g * (\mathscr{D}f \boxtimes f)) * (X \times \Delta_X)^{h})$.

Property 6: $\mathscr{D}(f \boxtimes k); \delta^{v}_{S',T'} = \delta^{v}_{S,T}; \mathscr{D}f \boxtimes \mathscr{D}k * \delta^{h}_{X,Z}$.

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



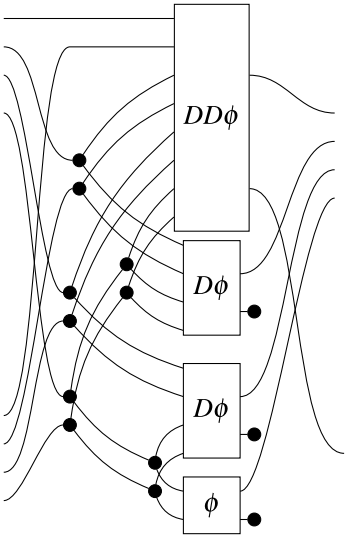
We use that result to construct a string diagram for $U(\mathscr{D}(f \times k); \delta^{v}_{S',T'})$:



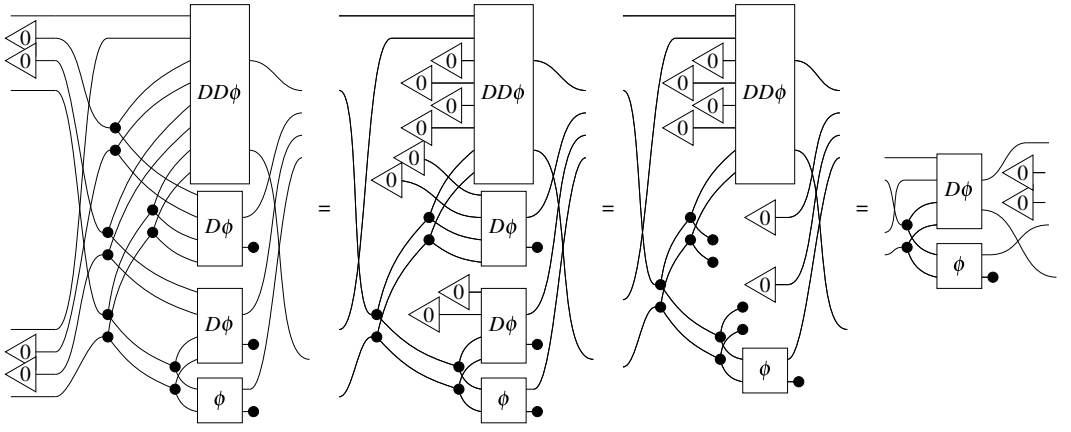
This only requires some rearrangement using axioms of monoidal categories to obtain the string diagram for the underlying morphism of $\delta^{v}_{S,T}; \mathscr{D}f \boxtimes \mathscr{D}k * \delta^{h}_{X,Z}$.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

For the next two properties, we need the underlying morphism for $\mathscr{D}\mathscr{D}f$. Using the definition of $\mathscr{D}$ and the axioms of CDCs, one can show $U\mathscr{D}\mathscr{D}f$ is given by the diagram.



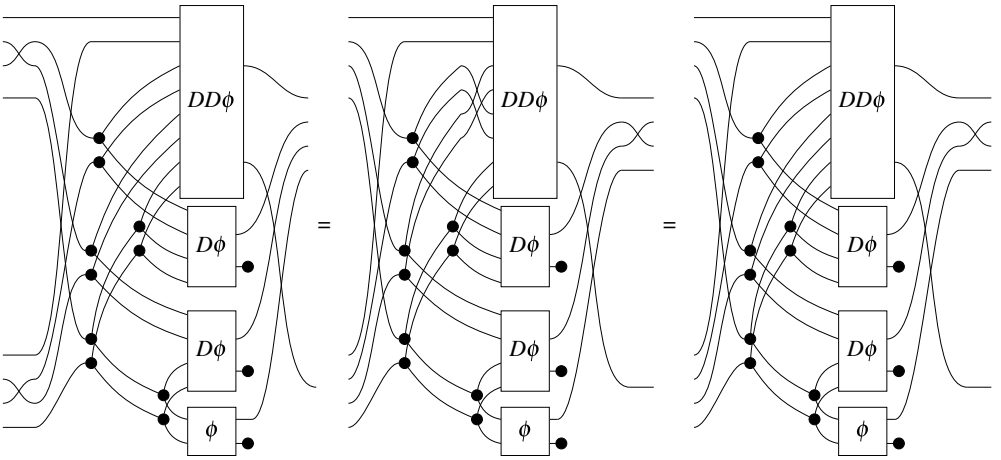
Property 7: $\zeta^v_S; \mathscr{D}\mathscr{D}f * \zeta^h_X = \mathscr{D}f; \zeta^v_{S'}$.



In the first step, we use the naturality of the copy morphisms to distribute zero maps to the appropriate places in the diagram. In the next step, we use CD2 for D twice. Finally, in the last step, we clean up some discarded copies and use CD6 for D to simplify $DD\phi \circ \zeta_{S\times X}$.

Property 8: $\delta^v_{S,S}; \mathscr{D}\mathscr{D}f * \delta^h_{X,X} = \mathscr{D}\mathscr{D}f; \delta^h_{S',S'}$.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



The first step involves properties of cartesian categories, both in swapping the order of the two $D\phi$ morphisms and also in reassociating the copying. In the second step, the swapping before $DD\phi$ is undone by invoking CD7 for D. □

An important consequence of Proposition 32(4) is the following extension to finite sequences of vertically composed 2-cells.

Lemma 33. *Let $(f_k)_{k=0}^n$ be a finite sequence of vertically composable 2-cells. Then $\mathscr{D}(f_0; \cdots ; f_n) = (\mathscr{D}f_0; \cdots ; \mathscr{D}f_n) * z^h$, where z is the unzipping isomorphism in $\mathbb{C}$ of type*

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

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

Definition 34. *The componentwise application of $\mathscr{D}$ to 2-cells in a $\mathrm{St}(\mathbb{C})$ morphism, $\mathscr{D}^* : (i, \mathbf{s}) \mapsto (\langle 0, i\rangle, [\mathscr{D}\mathbf{s}_\bullet])$, is a well-defined operation on $\mathrm{St}(\mathbb{C})$ morphisms of type*

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

To justify this definition, we must immediately check this operation is well defined on stateful morphism sequences.

Lemma 35. *$\mathscr{D}^*$ is well defined.*

Proof. Let $(i, \mathbf{s})$ and $(j, \mathbf{t})$ be extensionally equivalent sequences. We must show $(\mathscr{D}i, [\mathscr{D}\mathbf{s}_\bullet])$ and $(\mathscr{D}j, [\mathscr{D}\mathbf{t}_\bullet])$ are extensionally equivalent.

$$\begin{aligned}
\mathscr{D}i; \mathscr{D}\mathbf{s}_0; \ldots ; \mathscr{D}\mathbf{s}_n; d_{(\operatorname{nxt} s_n)^2} &= \mathscr{D}i; \mathscr{D}\mathbf{s}_0; \ldots ; \mathscr{D}\mathbf{s}_n; \mathscr{D}d_{\operatorname{nxt} s_n} \\
&= \mathscr{D}(i; \mathbf{s}_0; \ldots ; \mathbf{s}_n; d_{\operatorname{nxt} s_n}) * z^{-1} \\
&= \mathscr{D}(j; \mathbf{t}_0; \ldots ; \mathbf{t}_n; d_{\operatorname{nxt} t_n}) * z^{-1}
\end{aligned}$$

where the first line is by Proposition 32(2), the second by Lemma 33, and the last by the fact that $(i, \mathbf{s})$ and $(j, \mathbf{t})$ are extensionally equivalent. Here, it is crucial that extensionally equivalent sequences have the same source values so that the z obtained from Lemma 33 when $\varphi = \mathbf{s}$ matches the z obtained when $\varphi = \mathbf{t}$. The proof then finishes by reversing the first two steps. □

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

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

Theorem 36. *$\mathscr{D}^*$ is a Cartesian differential operator.*

The strategy for this proof is to use the properties of $\mathscr{D}$ from Proposition 32, 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:

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

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

$$\mathscr{D}(\mathbf{g}_\bullet * \mathbf{f}_\bullet); \gamma^{\nu}_{\mathbf{S}_{\bullet+1},\mathbf{T}_{\bullet+1}} = \gamma^{\nu}_{\mathbf{S}_\bullet,\mathbf{T}_\bullet}; (\mathscr{D}\mathbf{g}_\bullet * (\mathscr{D}\mathbf{f}_\bullet \boxtimes \mathbf{f}_\bullet) * (\mathbf{X}_\bullet \times \Delta_{\mathbf{X}_\bullet})^h),$$

the latter of which is a case of Proposition 32(5).

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

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

where the second line is the Shim Lemma step.

Proof. We show that $\mathscr{D}^*$ satisfies the seven axioms given in Definition 28. Most of the hard work has already been done in Proposition 32; we use the properties of $\mathscr{D}$ established there in concert with the Shim Lemma to obtain most of the CD axioms.

Let $f = (i,\mathbf{f}) : \mathbf{X} \to \mathbf{Y}$, $g = (j,\mathbf{g}) : \mathbf{Y} \to \mathbf{Z}$ and $h = (\ell,\mathbf{h}) : \mathbf{Z} \to \mathbf{W}$. Let $\mathbf{S}$, $\mathbf{T}$, and $\mathbf{U}$ be the state sequences of f, g, and h, respectively.

CD1. $\mathscr{D}^*s = s \times !_{\mathrm{dom}\,(s)}$ for $s \in \{\mathbf{X}, \sigma_{\mathbf{X},\mathbf{Y}}, !_{\mathbf{X}}, \Delta_{\mathbf{X}}, +_{\mathbf{X}}, 0_{\mathbf{X}}\}$. These s are of the form $(\mathrm{id}_1, [(\mathbf{s}_\bullet)^h])$ where each $\mathbf{s}_\bullet$ is a $\mathbb{C}$-morphism from $\{X, \sigma_{X,Y}, !_X, \Delta_X, +_X, 0_X\}$. Therefore,

$$\begin{aligned}\mathscr{D}^*s = \mathscr{D}^*(\mathrm{id}_1, [(\mathbf{s}_\bullet)^h]) &= (\mathrm{id}_1, [\mathscr{D}(\mathbf{s}_\bullet)^h]) = (\mathrm{id}_1, [(D\mathbf{s}_\bullet)^h])\\ &= (\langle \mathrm{id}_1, \mathrm{id}_1\rangle, [(\mathbf{s}_\bullet \times !_{\mathrm{dom}\,\mathbf{s}_\bullet})^h]) = (\mathrm{id}_1, [(\mathbf{s}_\bullet)^h]) \times (\mathrm{id}_1, [(!_{\mathrm{dom}\,\mathbf{s}_\bullet})^h]) = s \times !_{\mathrm{dom}\,s}\end{aligned}$$

where the last step in the first line is by Proposition 32(1), and the first step in the second line is by CD1 for D.

CD2. $\mathscr{D}^*(i,\mathbf{f})\circ(0_{\mathbf{X}}\times\mathbf{X}) = 0_{\mathbf{Y}}\circ!_{\mathbf{X}}$. By definition of $\mathscr{D}^*$, we know $\mathscr{D}^*(i,\mathbf{f})\circ(0_{\mathbf{X}}\times\mathbf{X}) = (\langle 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathbf{f}_\bullet * (0_{\mathbf{X}_\bullet}\times\mathbf{X}_\bullet)^h])$. By Proposition 32(2),

$$(0_{\mathbf{S}_0}\times\mathbf{S}_0)\circ i = \langle 0_{\mathbf{S}_0}, i\rangle \text{ and } (0_{\mathbf{Y}_\bullet}\circ!_{\mathbf{Y}_\bullet})^h * \mathbf{f}_\bullet; (0_{\mathbf{S}_{\bullet+1}}\times\mathbf{S}_{\bullet+1})^{\nu} = (0_{\mathbf{S}_\bullet}\times\mathbf{S}_\bullet)^{\nu}; \mathscr{D}\mathbf{f}_\bullet * (0_{\mathbf{X}_\bullet}\times\mathbf{X}_\bullet)^h,$$

so the shim lemma tell us

$$\begin{aligned}\mathscr{D}^*(i,\mathbf{f})\circ(0_{\mathbf{X}}\times\mathbf{X}) &= (\langle 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathbf{f}_\bullet * (0_{\mathbf{X}_\bullet}\times\mathbf{X}_\bullet)^h]) && \text{(definitions)}\\ &= (i, [(0_{\mathbf{Y}_\bullet}\circ!_{\mathbf{Y}_\bullet})^h * \mathbf{f}_\bullet]) && \text{(shim lemma)}\\ &= 0_{\mathbf{Y}}\circ!_{\mathbf{Y}}\circ(i,\mathbf{f}) && \text{(definitions)}\\ &= 0_{\mathbf{Y}}\circ!_{\mathbf{X}} && \text{(finality)}\end{aligned}$$

as desired.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

CD3. $\mathscr{D}^*(i,\mathbf{f}) \circ (+_{\mathbf{X}} \times \mathbf{X}) = +_{\mathbf{Y}} \circ (\mathscr{D}^*(i,\mathbf{f}) \times \mathscr{D}^*(i,\mathbf{f})) \circ \alpha^h_{\mathbf{X}}$. Since $(+_{\mathbf{S}_0} \times \mathbf{S}_0) \circ \langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle = \langle 0_{\mathbf{S}_0}, i\rangle$, and, by Proposition 32(3),

$$\alpha^v_{\mathbf{S}_\bullet}; (+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \boxtimes \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet}); \beta^v_{\mathbf{S}_{\bullet+1}}; (+_{\mathbf{S}_{\bullet+1}} \times \mathbf{S}_{\bullet+1})^v = (+_{\mathbf{S}_\bullet} \times \mathbf{S}_\bullet)^v; \mathscr{D}\mathbf{f}_\bullet * (+_{\mathbf{X}_\bullet} \times \mathbf{X}_\bullet)^h,$$

we can invoke the shim lemma again.

$$\begin{aligned}
&\mathscr{D}^*(i,\mathbf{f}) \circ (+_{\mathbf{X}} \times \mathbf{X}) \\
&= (\langle 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathbf{f}_\bullet * (+_{\mathbf{X}_\bullet} \times \mathbf{X}_\bullet)^h]) && \text{(definitions)}\\
&= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle, [\alpha^v_{\mathbf{S}_\bullet}; (+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \boxtimes \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet}); \beta^v_{\mathbf{S}_{\bullet+1}}]) && \text{(shim lemma)}\\
&= (\beta_{\mathbf{S}_0} \circ \langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{S}_0}, i\rangle, [\alpha^v_{\mathbf{S}_\bullet}; (+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \boxtimes \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet}); \beta^v_{\mathbf{S}_{\bullet+1}}])\\
&= (\langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{S}_0}, i\rangle, [\beta^v_{\mathbf{S}_\bullet}; \alpha^v_{\mathbf{S}_\bullet}; (+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \times \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet})]) && \text{(delayed dinaturality)}\\
&= (\langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{S}_0}, i\rangle, [(\alpha_{\mathbf{S}_\bullet} \circ \beta_{\mathbf{S}_\bullet})^v; (+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \times \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet})])\\
&= (\langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{S}_0}, i\rangle, [+^h_{\mathbf{Y}_\bullet} *\mathscr{D}\mathbf{f}_\bullet \times \mathscr{D}\mathbf{f}_\bullet * \alpha^h_{\mathbf{X}_\bullet}])
\end{aligned}$$

The last step here is a bit delicate and involves a special case of the Shim Lemma. If $\mathbf{b}$ is a sequence of idempotent maps (i.e. $\mathbf{b}_\bullet \circ \mathbf{b}_\bullet = \mathbf{b}_\bullet$) such that $\mathbf{b}_0 \circ i = i$, then $(i, [\mathbf{f}_\bullet]) = (i, [\mathbf{b}^v_\bullet; \mathbf{f}_\bullet])$. Here, $\alpha_{\mathbf{S}_\bullet} \circ \beta_{\mathbf{S}_\bullet}$ is such a sequence of idempotent maps.

CD4. $\mathscr{D}^*((j,\mathbf{g}) \circ (i,\mathbf{f})) = \mathscr{D}^*(j,\mathbf{g}) \circ (\mathscr{D}^*(i,\mathbf{f}) \times (i,\mathbf{f})) \circ (\mathbf{X} \times \Delta_{\mathbf{X}})$.
Since $\gamma_{\mathbf{S}_0,\mathbf{T}_0} \circ \langle 0_{\mathbf{S}_0}, 0_{\mathbf{T}_0}, i, j\rangle = \langle 0_{\mathbf{S}_0}, i, i, 0_{\mathbf{T}_0}, j\rangle$, and, by Proposition 32(5),

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

we use the shim lemma again.

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

CD5. $\mathscr{D}^*((i,\mathbf{f}) \times (\ell,\mathbf{h})) = (\mathscr{D}^*(i,\mathbf{f}) \times \mathscr{D}^*(\ell,\mathbf{h})) \circ \delta_{\mathbf{X},\mathbf{Z}}$.
Our invocation of the shim lemma this time uses the fact $\delta_{\mathbf{S}_0,\mathbf{U}_0} \circ \langle 0_{\mathbf{S}_0}, 0_{\mathbf{U}_0}, i, \ell\rangle = \langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{U}_0}, \ell\rangle$ and Proposition 32(6):

$$\mathscr{D}(\mathbf{f}_\bullet \boxtimes \mathbf{h}_\bullet); \delta^v_{\mathbf{S}_{\bullet+1},\mathbf{U}_{\bullet+1}} = \delta^v_{\mathbf{S}_\bullet,\mathbf{U}_\bullet}; \mathscr{D}\mathbf{f}_\bullet \boxtimes \mathscr{D}\mathbf{k}_\bullet * \delta^h_{\mathbf{X}_\bullet,\mathbf{Z}_\bullet}.$$

We can obtain this axiom now as:

$$\begin{aligned}
\mathscr{D}^*((i,\mathbf{f}) \times (\ell,\mathbf{h})) &= (\langle 0_{\mathbf{S}_0}, i, 0_{\mathbf{U}_0}, \ell\rangle, [\mathscr{D}(\mathbf{f}_\bullet \boxtimes \mathbf{h}_\bullet)]) && \text{(definitions)}\\
&= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{U}_0}, i, \ell\rangle, [\mathscr{D}\mathbf{f}_\bullet \boxtimes \mathscr{D}\mathbf{k}_\bullet * \delta^h_{\mathbf{X}_\bullet,\mathbf{Z}_\bullet}]) && \text{(shim lemma)}\\
&= (\mathscr{D}^*(i,\mathbf{f}) \times \mathscr{D}^*(\ell,\mathbf{h})) \circ \delta_{\mathbf{X}_\bullet,\mathbf{Z}_\bullet} && \text{(definitions)}
\end{aligned}$$

CD6. $\mathscr{D}^*\mathscr{D}^*(i,\mathbf{f}) \circ \zeta_{\mathbf{X}} = \mathscr{D}^*(i,\mathbf{f})$.
We invoke the shim lemma this time using the facts that $\zeta_{\mathbf{S}_0} \circ \langle 0_{\mathbf{S}_0}, i\rangle = \langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle$ and Proposition 32(7):

$$\mathscr{D}\mathbf{f}_\bullet; \zeta^v_{\mathbf{S}_{\bullet+1}} = \zeta^v_{\mathbf{S}_\bullet}; \mathscr{D}\mathscr{D}\mathbf{f}_\bullet * \zeta^h_{\mathbf{X}_\bullet}.$$

We obtain the axiom directly now,

$$\begin{aligned}
\mathscr{D}^*\mathscr{D}^*(i,\mathbf{f}) \circ \zeta_{\mathbf{X}} &= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathscr{D}\mathbf{f}_\bullet * \zeta^h_{\mathbf{X}_\bullet}]) && \text{(definitions)}\\
&= (\langle 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathbf{f}_\bullet]) && \text{(shim lemma)}\\
&= \mathscr{D}^*(i,\mathbf{f})
\end{aligned}$$

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

CD7. $\mathscr{D}^*\mathscr{D}^*(i,\mathbf{f}) \circ \delta_{\mathbf{X},\mathbf{X}} = \mathscr{D}^*\mathscr{D}^*(i,\mathbf{f})$.
We invoke the shim lemma with $\delta_{\mathbf{S}_0,\mathbf{S}_0} \circ \langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle = \langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle$ and Proposition 32(8):

$$\mathscr{D}\mathscr{D}\mathbf{f}_\bullet ; \delta^v_{\mathbf{S}_{\bullet+1},\mathbf{S}_{\bullet+1}} = \delta^v_{\mathbf{S}_\bullet,\mathbf{S}_\bullet} ; \mathscr{D}\mathscr{D}\mathbf{f}_\bullet * \delta^h_{\mathbf{X}_\bullet,\mathbf{X}_\bullet}.$$

We obtain the axiom directly now,

$$\begin{aligned}
\mathscr{D}^*\mathscr{D}^*(i,\mathbf{f}) \circ \delta_{\mathbf{X},\mathbf{X}} &= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathscr{D}\mathbf{f}_\bullet * \delta^h_{\mathbf{X}_\bullet,\mathbf{X}_\bullet}]) && \text{(definitions)}\\
&= (\langle 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, 0_{\mathbf{S}_0}, i\rangle, [\mathscr{D}\mathscr{D}\mathbf{f}_\bullet]) && \text{(shim lemma)}\\
&= \mathscr{D}^*\mathscr{D}^*(i,\mathbf{f})
\end{aligned}$$

□

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

Theorem 37. *For any morphism* $(i,\mathbf{f}) : \mathbf{A} \to \mathbf{B}$ *in* $\mathrm{St}(\mathbb{C})$,

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

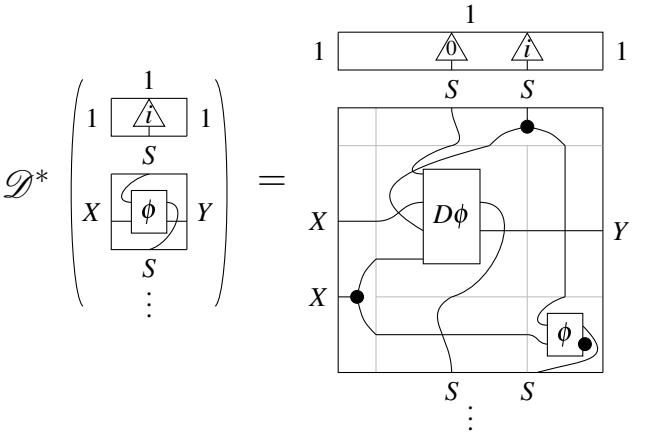
where z is the unzipping isomorphism from Lemma 33.

6.1 Differentiation of regular causal morphisms

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

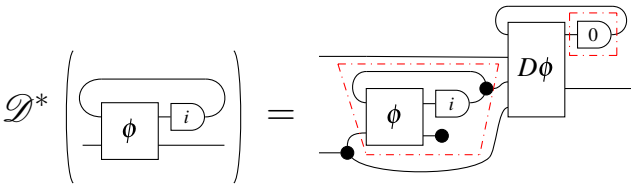
Corollary 38. *The operation* $\mathscr{D}^*$ *restricted to* $\mathrm{St}_0(\mathbb{C})$ *is a Cartesian differential operator on* $\mathrm{St}_0(\mathbb{C})$.

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 $\mathscr{D}$ 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.)

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press



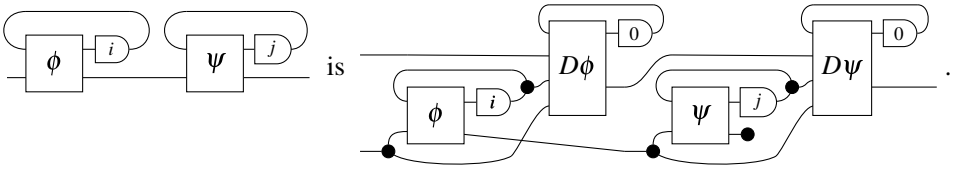
The idea of a derivative in a Cartesian differential category is to take a base point $\mathbf{x}$ as its lower argument and a small change as its upper argument $\mathbf{\Delta x}$ and return an approximation for the difference between the outputs of the function at $\mathbf{x}$ and the function at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

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 ϕ 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 ϕ device is stateless (being an underlying morphism from a 2-cell), Theorem 22 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 RNN. Theorem 37 guarantees this derivative matches precisely what we expect from the unroll-and-differentiate procedure used in BPTT. 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



We leave this exercise to the reader.

7. Backpropagation using Reverse Differential Operators

In this section, we survey developments in the literature since the publication of the conference version of this paper (Sprunger and Katsumata, 2019), and we provide further extensions of these results in categories of causal morphisms.

In 2019, Cockett et al. (2020) introduced the *Cartesian reverse differential category* (CRDC). A CRDC is a Cartesian left additive category equipped with a reverse differential combinator

$$\frac{f : A \to B}{R[f] : A \times B \to A}$$

satisfying various axioms identified in Cockett et al. (2020). It may be seen as an axiomatization of the operation $R : C^\infty(\mathbb{R}^n, \mathbb{R}^m) \to C^\infty(\mathbb{R}^n \times \mathbb{R}^m, \mathbb{R}^n)$ given by:

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

$$R[f] \triangleq \lambda(x, y) \,.\, (J_f(x))^t \cdot y$$

where $(J_f(x))^t$ is the *transpose* of the Jacobian matrix of f at point x.

The reverse differential combinator is suitable for directly obtaining the *gradient* of scalar functions. For a smooth function $l : \mathbb{R}^n \to \mathbb{R}$, its gradient vector field $grad(l) : \mathbb{R}^n \to \mathbb{R}^n$ can be expressed as (by putting $m = 1$ above)

$$grad(l)(x) = R[l](x, 1).$$

Among the axioms of the reverse differential combinator are a reverse chain rule (RD5 in Cockett et al. (2020)), stating that $R[f; g] = Rf \circ (A \times Rg) \circ \langle \pi_0, \langle f \circ \pi_0, \pi_1 \rangle \rangle$ and a "reverse parallel rule" (RD4 in Cockett et al. (2020)) $R[f \times h] = (Rf \times Rh) \circ (A \times \sigma_{C,B} \times D)$, where $f : A \to B$, $g : B \to C$ and $h : C \to D$. These rules ensure the computation of the gradient of a complex network of functions is compositional, determined by the structure of the network. This is the key in the efficient computation of gradient of large-scale neural networks.

Further refinement of this approach was presented by Cruttwell et al. (2022) using *lenses*, which capture the flavor of *bidirectional computation* in a CRDC. Slightly simplifying the definition of lens from Cruttwell et al. (2022), we obtain the following.

Definition 39. *Suppose* $\mathbb{C}$ *is a Cartesian category. The category* Lens($\mathbb{C}$) *of (simple)* $\mathbb{C}$*-lenses has objects the same as* $\mathbb{C}$*, and a morphism* $(f, f^*) : A \to B$ *consists of two* $\mathbb{C}$ *morphisms* $f : A \to B$*, called the "forward" or "get" morphism, and* $f^* : A \times B \to A$*, called the "reverse" or "put" morphism.*

The composition of morphisms $(f, f^*) : A \to B$ *and* $(g, g^*) : B \to C$ *in* Lens($\mathbb{C}$) *is defined to be* $(g \circ f, f^* \circ \langle \pi_0, g^* \circ \langle \pi_0, f \circ \pi_1 \rangle \rangle)$*. The identity on A for this composition is* (id_A, π_1).

A key insight of Cruttwell et al. is the match between composition of lenses and the reverse chain rule.

Proposition 40 (Proposition 2.7, Cruttwell et al., (2022)). *If* $\mathbb{C}$ *is a CRDC, there is a functor* $\mathbf{R} : \mathbb{C} \to$ Lens($\mathbb{C}$) *which is the identity on objects and sends* $f : A \to B$ *to* $(f, R[f])$*, where R is the reverse differential combinator in* $\mathbb{C}$.

A disadvantage of the lens approach is that Lens($\mathbb{C}$) is not generally Cartesian, even if $\mathbb{C}$ is Cartesian. In fact, Lens($\mathbb{C}$) can even fail to have a terminal object since if $(f, f^*) : A \to 1$ in Lens($\mathbb{C}$), then the reverse morphism has to be of the type $f^* : A \times 1 \to A$ in $\mathbb{C}$, and generally there may be many such maps. This is an unusual property of lenses as a semantics for gradient-based training; a priori, reverse-mode updating of parameters does not fundamentally affect the ability to copy and discard data.

The failure of Lens($\mathbb{C}$) to be Cartesian presents a challenge to connecting with this work, where we regularly require Cartesian categories. There is another a priori obstacle to connecting with causal morphisms—the apparent *acausality* of reverse-mode updating. In a reverse-mode update, errors in prediction from later stages of the computation flow "backwards in time" to produce suggested changes to parameters used in earlier steps that would reduce error.

In the remainder of this section, we show that some of these challenges may be overcome. In particular, by eliminating some morphisms from the category of lenses, we are able to recover a Cartesian category of lenses. This allows us to consider the category of stateful sequences of (linear) lenses. Though the resulting category St(LLens($\mathbb{C}$)) does not seem to be a CRDC, the functor **R** defined in Proposition 40 then becomes a product-preserving functor between Cartesian categories.

Recall from Cockett et al. (2020) that linearity in a component of a morphism is defined in any (reverse) Cartesian differential category as a morphism whose partial derivative in that variable is

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

equal to that morphism. We first restrict our category of lenses to those lenses which are linear in their reverse variable.

Definition 41. *Given a reverse Cartesian differential category* $\mathbb{C}$*, define the category of linear lenses* $\mathrm{LLens}(\mathbb{C})$ *to be the wide subcategory of* $\mathrm{Lens}(\mathbb{C})$ *retaining only the lenses* $(f, f^*) : A \to B$ *such that* $f^* : A \times B \to A$ *is linear in its B component.*

Restricting the available reverse morphisms allows $\mathrm{LLens}(\mathbb{C})$ to regain its status as Cartesian.

Lemma 42. $\mathrm{LLens}(\mathbb{C})$ *is a Cartesian category.*

Proof. The terminal object of $\mathbb{C}$ is a terminal object of $\mathrm{LLens}(\mathbb{C})$, and the unique linear lens from A to 1 is $(!_A, 0_A \circ !_A)$. Uniqueness of the forward part of the lens is clear from finality in $\mathbb{C}$. The linearity condition on the reverse part of the lens entails uniqueness, since $g : A \times 1 \to A$ is linear in the 1 component if and only if $g^{\dagger[A]\dagger[A]} = g$ by Lemma 35 of Cockett et al. (2020). Since $g^{\dagger[A]} : A \times A \to 1$ is a $\mathbb{C}$-map to the terminal object, it follows that $g^{\dagger[A]} = !_{A\times A}$. Therefore, g can only be ${!_{A\times A}}^{\dagger[A]} = 0_A \circ !_A$.

Similarly, the binary product in $\mathrm{LLens}(\mathbb{C})$ is given by the binary product in $\mathbb{C}$ with linear lens projections $(\pi_A, (1_A \times 0_B) \circ \pi_A) : A \times B \to A$ and $(\pi_B, (0_A \times 1_B) \circ \pi_B) : A \times B \to B$. □

As a result, the similarities between the reverse parallel rule and the monoidal product of lenses result in a product-preserving functor.

Lemma 43. *The functor* $\mathbf{R} : \mathbb{C} \to \mathrm{Lens}(\mathbb{C})$ *given in Proposition 40 restricts to* $\mathbf{R} : \mathbb{C} \to \mathrm{LLens}(\mathbb{C})$*. This restricted* $\mathbf{R}$ *is a product-preserving functor between Cartesian categories.*

Since $\mathrm{LLens}(\mathbb{C})$ is a Cartesian category, we can apply the $\mathrm{St}(-)$ construction to it. Since further $\mathbf{R}$ is a product-preserving functor, we can also apply the $\mathrm{St}(-)$ construction to it as well. Therefore, there is a product-preserving functor $\mathrm{St}(\mathbf{R}) : \mathrm{St}(\mathbb{C}) \to \mathrm{St}(\mathrm{LLens}(\mathbb{C}))$. This functor takes a stateful morphism sequence in $\mathbb{C}$ and returns a stateful sequence of lenses on the same infinite sequence of objects. The forward parts of these lenses are precisely the original stateful morphism sequence, and the reverse parts of these lenses are the reverse-mode derivative provided by the CRDC.

A priori, one might not expect that this sequence of reverse derivatives have significant properties. However, there is a relationship between this sequence and BPTT.

Proposition 44. *The functor* $\mathrm{St}(\mathbf{R}) : \mathrm{St}(\mathbb{C}) \to \mathrm{St}(\mathrm{LLens}(\mathbb{C}))$ *embodies the backpropagation through time in the following sense:*

$$\mathrm{Un}_\bullet(\mathrm{St}(\mathbf{R})(i, \mathbf{f})) = (\mathrm{Un}_\bullet(i, \mathbf{f}), R[\mathrm{Un}_\bullet(i, \mathbf{f})]).$$

Proof. The equality to be proven is equivalent to the morphism part of the equality of two functors: $\Gamma^{-1}(\mathrm{St}(\mathbf{R})) = C(\mathbf{R}) \circ \Gamma^{-1}$. This is proven as Lemma 14. □

At first blush, there seems to be a discrepancy here: the $\mathrm{St}(-)$ construction models causal functions on sequences, and reverse-mode differentiation is decidedly non-causal since errors from later steps backpropagate to changes for parameters in earlier steps. However, while computing the reverse-mode derivative is a non-causal process for RNNs, constructing the lens which will compute this derivative is a causal process. At each clock tick, $\mathrm{St}(\mathbf{R})$ accumulates a continuation in the reverse part of the lens such that at the end of the computation, the overall reverse morphism computes the exact reverse derivative.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

8. 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, Cockett et al. (2011) 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. The authors thank Bart Jacobs, Fabio Zanasi, Nian-Ze Lee, and Masahito Hasegawa for many useful discussions. The authors also thank JS Lemay for the pointer to Cockett and Cruttwell (2014). The authors also thank the developers of TikZiT (Aleks Kissinger, Alexander Merry, Chris Heunen, K. Johan Paulsson), which which many of these diagrams were made.

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

Competing Interests.. The author(s) declare none.

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

Notes

1 This definition for causality is drawn from the coalgebraic study of stream functions, for example, Hansen et al. (2017).
2 In Figure 5a, 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.
3 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, **b** is the shim, and it adjusts the state spaces of the two stateful morphism sequences.

References

Abel, A. and Pientka, B. (2013). Wellfounded recursion with copatterns: A unified approach to termination and productivity. In: Morrisett, G. and Uustalu, T. (eds.), *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25-27, 2013*. ACM, 185–196.

Baez, J. C. and Master, J. (2020). Open petri nets. *Mathematical Structures in Computer Science* **30** (3), 314–341.

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

Blute, R., Cockett, J. and Seely, R. (2006). Differential categories. *Mathematical Structures in Computer Science* **16** (6) 1049–1083.

Blute, R., Cockett, J. and Seely, R. (2009). Cartesian differential categories. *Theory and Applications of Categories* **22** (23) 622–672.

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

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

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

Carette, T., de Visme, M. and Perdrix, S. (2021). Graphical language with delayed trace: Picturing quantum computing with finite memory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 1–13.

Cockett, J. and Cruttwell, G. (2014). Differential structure, tangent structure, and SDG. *Applied Categorical Structures* **22** (2) 331–417.

Cockett, J., Cruttwell, G. and Gallagher, J. (2011). Differential restriction categories. *Theory and Applications of Categories* **25** (21) 537–613.

Cockett, R., Cruttwell, G., Gallagher, J., Pacaud Lemay, J.-S., MacAdam, B., Plotkin, G. and Pronk, D. (2020). Reverse derivative categories. In Fernández, M. and Muscholl, A. (eds.), *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, vol. 152. *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 18:1–18:16.

Crochiere, R. and Oppenheim, A. (1975). Analysis of linear digital networks. *Proceedings of the IEEE* 63 (4) 581–595.

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

Cruttwell, G. S. H., Gavranović, B., Ghani, N., Wilson, P. and Zanasi, F. (2022). Categorical foundations of gradient-based learning. In Sergey, I. (ed.), *Programming Languages and Systems*. Springer International Publishing, 1–28.

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

Ehrhard, T. and Regnier, L. (2003). The differential lambda-calculus. *Theoretical Computer Science* **309** (1–3) 1–41.

Ehrhard, T. and Regnier, L. (2005). Differential interaction nets. *Theoretical Computer Science* **123** 35–74.

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

Fong, B., Spivak, D. and Tuys, R. (2019). Backprop as functor: A compositional perspective on supervised learning. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–13. https://doi.org/10.1109/LICS.2019.8785665

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

Ghica, D. and Jung, A. (2016). Categorical semantics of digital circuits. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design, FMCAD '16*, FMCAD Inc., 41–48.

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

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press

Goncharov, S. and Schröder, L. (2018). Guarded traced categories. In Baier, C. and Lago, U. D. (eds.), *Foundations of Software Science and Computation Structures*. Springer International Publishing, 313–330.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. MIT Press.

Hansen, H., Kupke, C. and Rutten, J. (2017). Stream differential equations: Specification formats and solution methods. *Logical Methods in Computer Science* **13** (1) 1–52.

Hoshino, N., Muroya, K., and Hasuo, I. (2014). Memoryful geometry of interaction: From coalgebraic components to algebraic effects. In *Proceedings of the 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, Association for Computing Machinery, 52:1–52:10.

Joyal, A., Street, R. and Verity, D. (1996). Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* **119** 447–468.

Katis, P., Sabadini, N. and Walters, R. F. C. (1997). Bicategories of processes. *Journal of Pure and Applied Algebra* **115** (2), 141–178.

Kissinger, A. and Uijlen, S. (2017). 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*. IEEE Computer Society, 1–12.

Lavore, E. D., de Felice, G. and Román, M. (2022). Monoidal streams for dataflow programming. In Baier, C. and Fisman, D. (eds.), *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2-5, 2022*. ACM, 51:1–51:14.

Leiserson, C. E. and Saxe, J. B. (1991). Retiming synchronous circuitry. *Algorithmica* **6** (1) 5–35.

Mac Lane, S. (1998). *Categories for the Working Mathematician*, vol. **5**. Graduate Texts in Mathematics, Springer Science & Business Media.

Milius, S. (2010). 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*. IEEE Computer Society, 421–430.

Møgelberg, R. E. (2014). A type theory for productive coprogramming via guarded recursion. In *Proceedings of the 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, Association for Computing Machinery, 71:1–71:10.

Parhi, K. K. and Chen, Y. (2013). *Signal Flow Graphs and Data Flow Graphs*, Springer New York, 1277–1302.

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

Rutten, J. (2008). Rational streams coalgebraically. *Logical Methods in Computer Science* **4** (3) 1–22.

Sprunger, D. and Katsumata, S.-y. (2019). Differentiable causal computations via delayed trace. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–12.

Uustalu, T. and Vene, V. (2005). Signals and comonads. *Journal of Universal Computer Science* **11** (7) 1310–1326.

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

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

Cite this article: Sprunger D and Katsumata S-y (2025). Differentiable causal computations via delayed trace (extended version). *Mathematical Structures in Computer Science* **35**(e10), 1–35. https://doi.org/10.1017/S0960129524000331

https://doi.org/10.1017/S0960129524000331 Published online by Cambridge University Press