Differentiation of stateful functions

David Sprunger (with Shin-ya Katsumata, +)

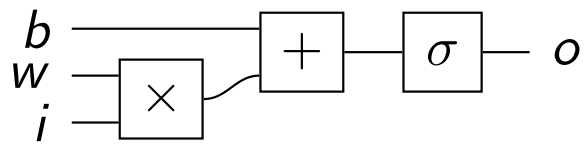
Shonan Meeting on Diagrammatic Methods
January 8, 2019

Motivation: differentiation is compositional

Classic songs from *Newton's Greatest Hits, vol. 2*:

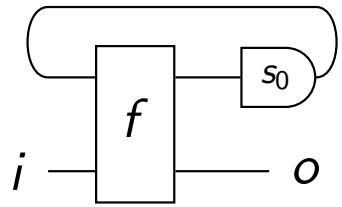
- sum rule: $D(f+g)(x) = Df(x) + Dg(x)$
- chain rule: $D(f \circ g)(x) = Df(g(x)) \circ Dg(x)$
- "cross rule": $D(f \times g)(x, y) = Df(x) \times Dg(y)$

Compositionality is useful for training (stateless) neural nets.



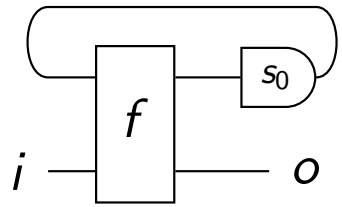
Motivation: is "differentiation" compositional?

When training recurrent neural nets, we want gradients for:



Motivation: is "differentiation" compositional?

When training recurrent neural nets, we want gradients for:

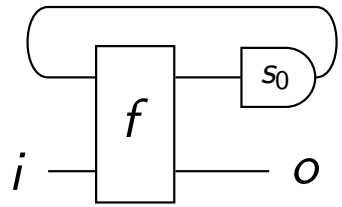


Usually this is done via "unrolling"...

$$U(tr(f)) = (f_0 : i_0 \mapsto o_0, f_1 : (i_0, i_1) \mapsto o_1, \ldots)$$

Motivation: is "differentiation" compositional?

When training recurrent neural nets, we want gradients for:



Usually this is done via "unrolling"...

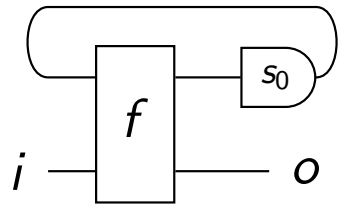
$$U(tr(f)) = (f_0 : i_0 \mapsto o_0, f_1 : (i_0, i_1) \mapsto o_1, \ldots)$$

and differentiating the resulting function sequence pointwise:

$$D(tr(f)) = map(D)(U(tr(f))) = (Df_0, Df_1, \ldots)$$

Motivation: is "differentiation" compositional?

When training recurrent neural nets, we want gradients for:



Usually this is done via "unrolling"...

$$U(tr(f)) = (f_0 : i_0 \mapsto o_0, f_1 : (i_0, i_1) \mapsto o_1, \ldots)$$

and differentiating the resulting function sequence pointwise:

$$D(tr(f)) = map(D)(U(tr(f))) = (Df_0, Df_1, \ldots)$$

Questions: Is this "differentiation"? What are we differentiating?

Background: Cartesian differential categories

[Blute, Cockett, and Seely, 2009]

A *Cartesian differential category* is:

- a Cartesian category $(\mathbb{C}, \times, 1)$, where
- every object has a chosen commutative bialgebra structure,
- (or, commutative monoid $0_X : 1 \to X$, $+_X : X \times X \to X$)
- a *differential operator* on morphisms,
  $D[f : X \to Y] = Df : X \times X \to Y$
- (and D satisfies seven axioms. . .)

Background: Cartesian differential categories

[Blute, Cockett, and Seely, 2009]
A *Cartesian differential category* is:

- a Cartesian category $(\mathbb{C}, \times, 1)$, where
- every object has a chosen commutative bialgebra structure,
- (or, commutative monoid $0_X : 1 \to X$, $+_X : X \times X \to X$)
- a *differential operator* on morphisms,
  $D[f : X \to Y] = Df : X \times X \to Y$
- (and D satisfies seven axioms. . .)

Intuition: In the standard case, $Df(\Delta x, x_0) = Jf|_{x_0} \times \Delta x$.

Goal

Starting from a CDC representing stateless computation, construct a CDC representing stateful computation.

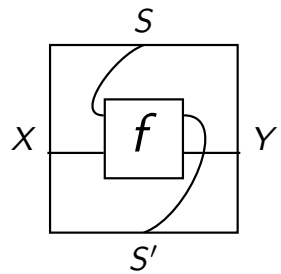
Our plan:

1. Start with Cartesian structure, separate states and values.
2. Compose along states to get stateful computations.
3. Add bialgebra structure.
4. Add differential operator.

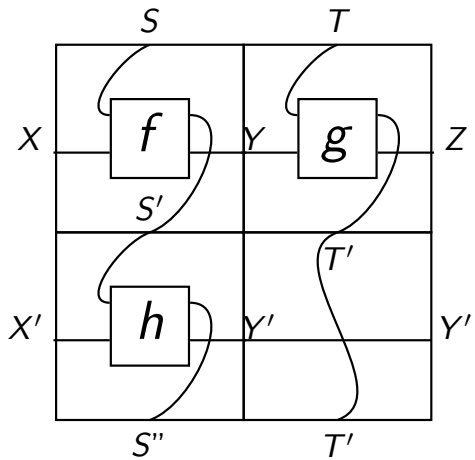
Separating states and values with double categories

Start with a (strict) Cartesian category $(\mathbb{C}, \times, 1)$.

Form a double category $Sq(\mathbb{C})$:

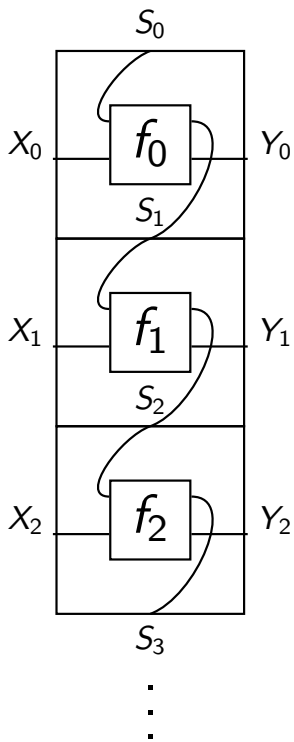


Horizontal comp. is "along values". Vertical is "along states".



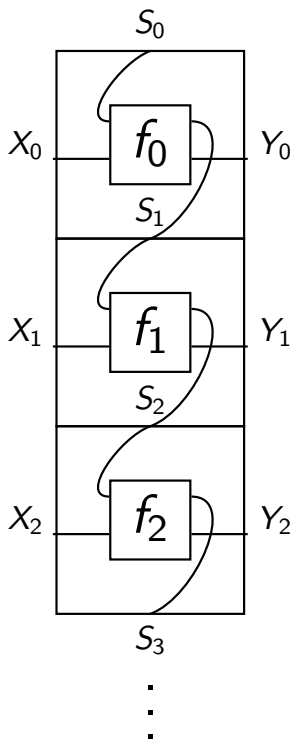
Sequences of squares

A *sequence of squares* is an infinite sequence of 2-cells that is vertically composable:

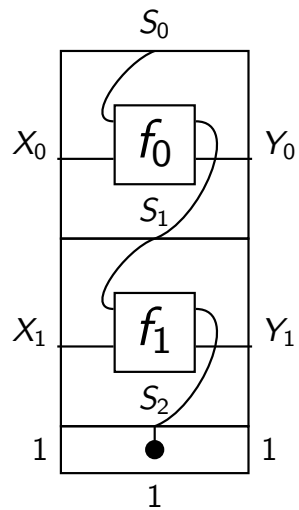


Sequences of squares

A *sequence of squares* is an infinite sequence of 2-cells that is vertically composable:



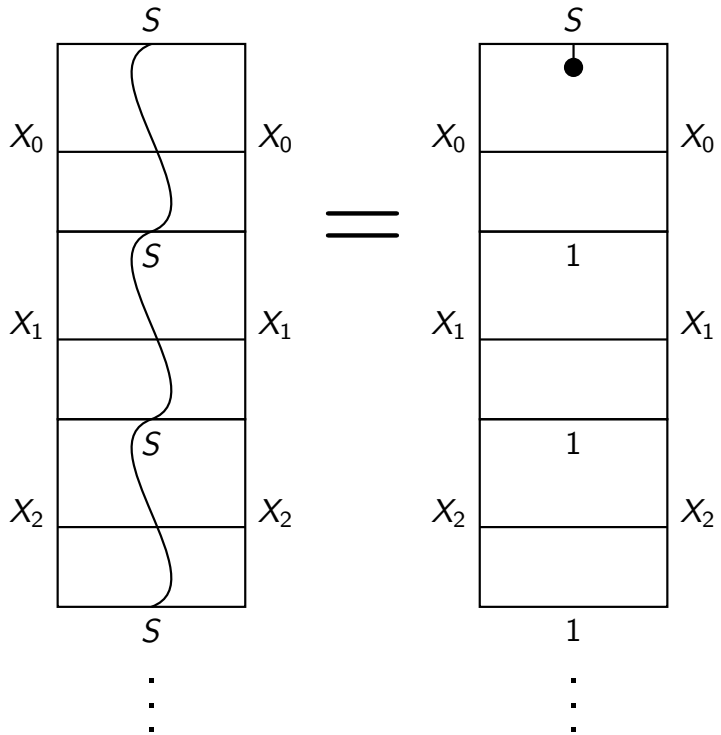
The *nth truncation* of a sequence is the composite of the first n cells followed by state discard:



Sequences of squares

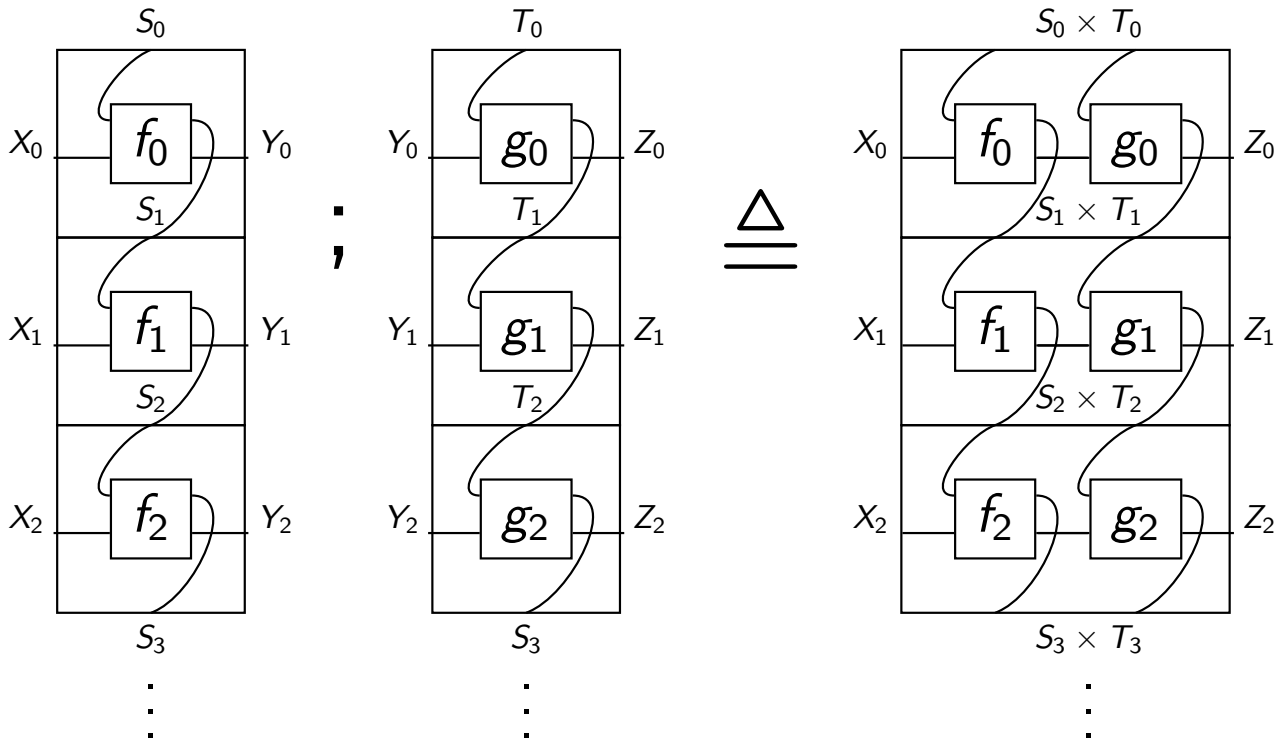
Two sequences of squares are *equivalent* if all of their truncations are equal when considered as morphisms in $\mathbb{C}$.

For example,



Composition of sequences of squares

Compose two sequences by composing 2-cells along values:



(Uninitialized) Stateful computations: $Seq(\mathbb{C})$

Definition

$Seq(\mathbb{C})$ is the category where:

- objects are infinite sequences of objects of $\mathbb{C}$: $(X_0, X_1, X_2, \ldots)$
- morphisms are (equivalence classes of) sequences of squares

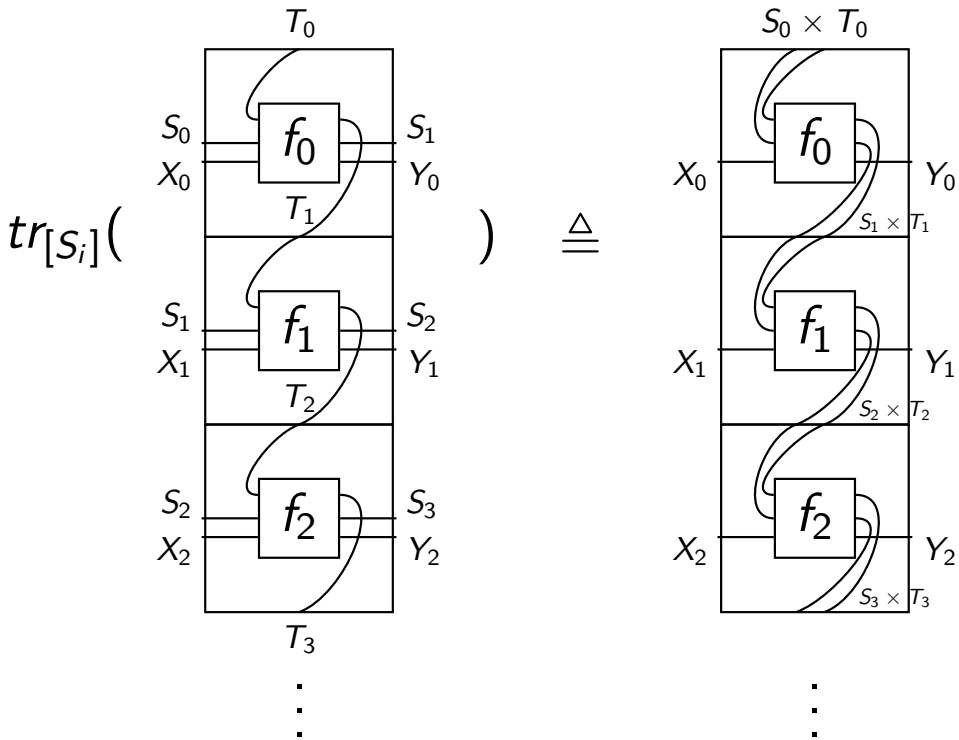
Structures from $\mathbb{C}$ are largely inherited by $Seq(\mathbb{C})$.

Proposition

~~If $\mathbb{C}$ is Cartesian, then~~ $Seq(\mathbb{C})$ is Cartesian.
If $\mathbb{C}$ is left additive, then $Seq(\mathbb{C})$ is left additive.

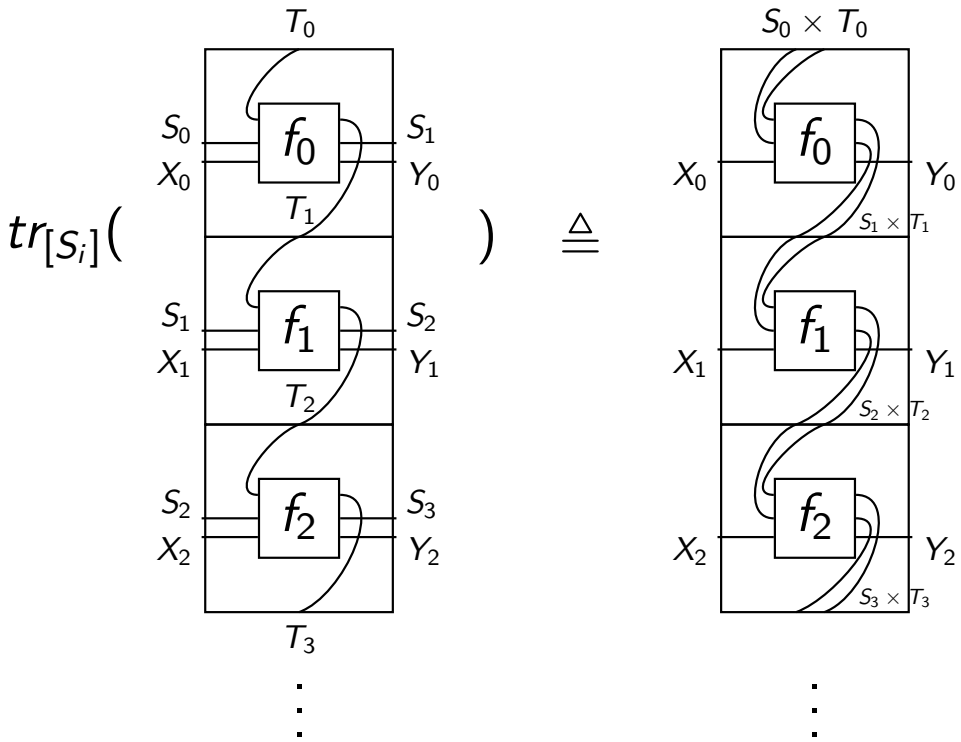
Delayed trace in $Seq(\mathbb{C})$

$Seq(\mathbb{C})$ supports a tracelike operation:



Delayed trace in $Seq(\mathbb{C})$

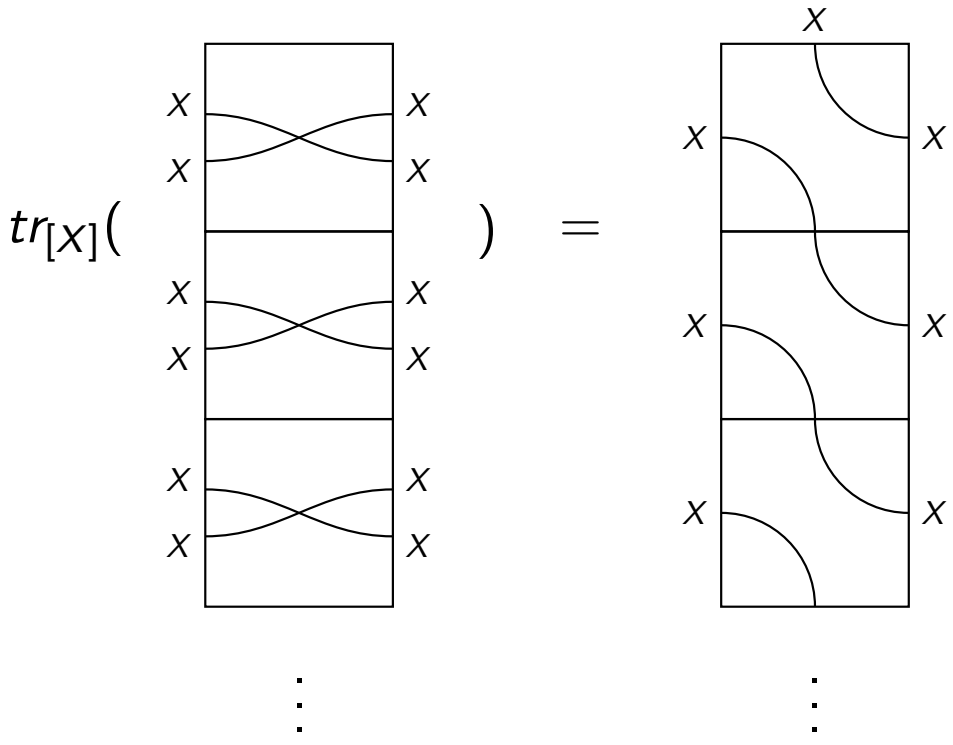
$Seq(\mathbb{C})$ supports a tracelike operation:



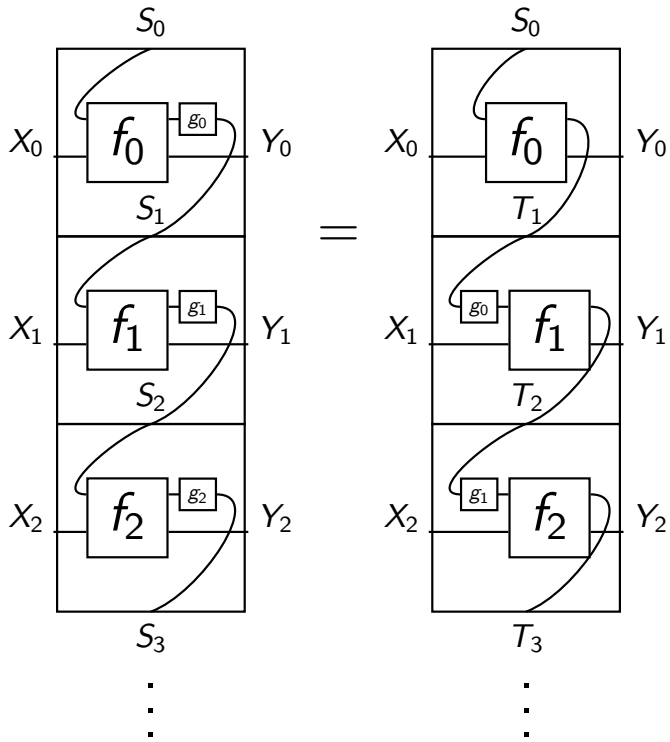
Proposition

This operation satisfies the axioms of trace except yanking and dinaturality.

~~Yanking~~ $\rightarrow$ (uninitialized) delay



~~Dinaturality~~ → Retiming



Delayed trace, generally

Definition

A *delayed (?) trace* on a symmetric monoidal category satisfies the trace axioms except yanking and dinaturality.

Idea: Given a delayed trace, *define* the delay morphism as the trace of symmetry.

Delayed trace, generally

Definition

A *delayed (?) trace* on a symmetric monoidal category satisfies the trace axioms except yanking and dinaturality.

Idea: Given a delayed trace, *define* the delay morphism as the trace of symmetry.

Extracting the abstract essence of this sequence construction is a strong interest of ours. Can you help?

Circuit applications: $Seq_0(\mathbb{C}) \hookrightarrow Seq(\mathbb{C})$

Definition

$Seq_0(\mathbb{C})$ is the subcategory of $Seq(\mathbb{C})$ where:

- objects have the form $(1, X, X, X, X, \ldots)$
- sequences have the form $(i, f, f, \ldots)$ where $i : (1, 1) \Rightarrow (S, 1)$

$Seq_0(\mathbb{C})$ is still Cartesian and retains left additive structure.

Circuit applications: $Seq_0(\mathbb{C}) \hookrightarrow Seq(\mathbb{C})$

Definition

$Seq_0(\mathbb{C})$ is the subcategory of $Seq(\mathbb{C})$ where:

- objects have the form $(1, X, X, X, X, \ldots)$
- sequences have the form $(i, f, f, \ldots)$ where $i : (1, 1) \Rightarrow (S, 1)$

$Seq_0(\mathbb{C})$ is still Cartesian and retains left additive structure.

The delayed trace described before does **not** restrict nicely to $Seq_0(\mathbb{C})$, but there are related delayed traces when initialization values are provided.

Circuit applications: $Seq_0(\mathbb{C}) \hookrightarrow Seq(\mathbb{C})$

Definition

$Seq_0(\mathbb{C})$ is the subcategory of $Seq(\mathbb{C})$ where:

- objects have the form $(1, X, X, X, X, \ldots)$
- sequences have the form $(i, f, f, \ldots)$ where $i : (1, 1) \Rightarrow (S, 1)$

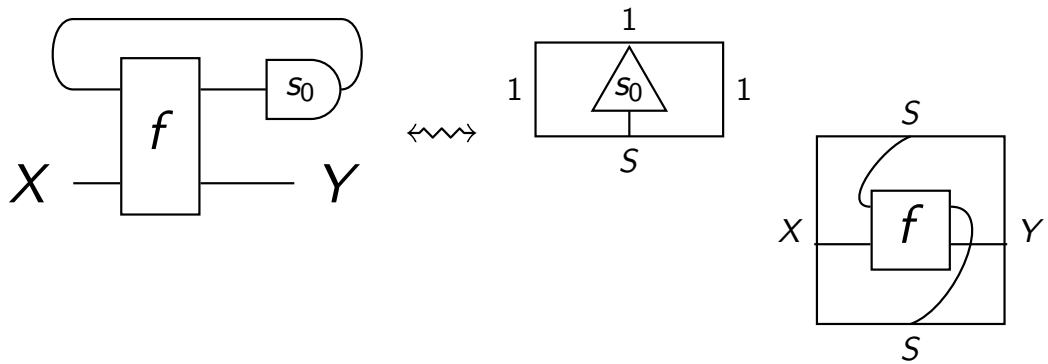
$Seq_0(\mathbb{C})$ is still Cartesian and retains left additive structure.

The delayed trace described before does **not** restrict nicely to $Seq_0(\mathbb{C})$, but there are related delayed traces when initialization values are provided.

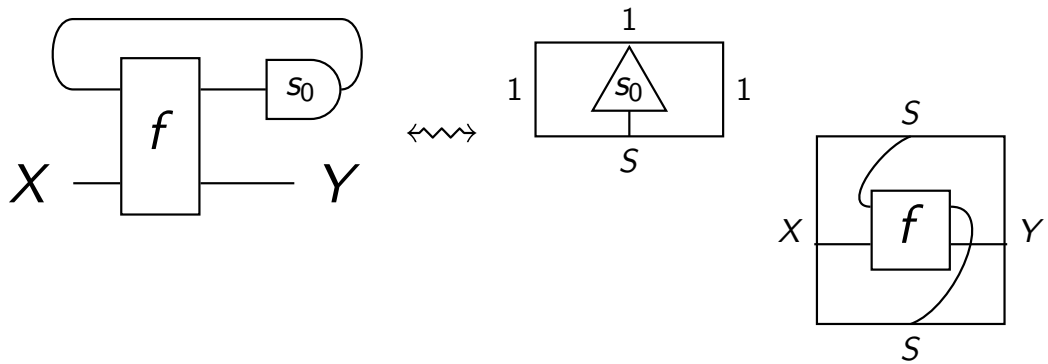
Conjecture (in progress)

If $\mathbb{C}$ has a differential operator, it lifts to $Seq_0(\mathbb{C})$.

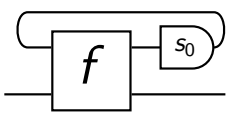
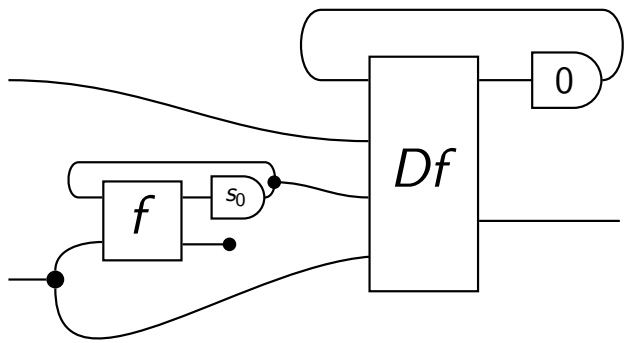
Circuits and $Seq_0(\mathbb{C})$



Circuits and $Seq_0(\mathbb{C})$



Given the conjecture on the previous slide, we could say:

$D($  $) =$ 

Further questions (roughly easy to hard)

- Convert differential operators to gradient operators?
- Equational rewrites to optimize derivative network?
- Partial functions (differential restriction categories)?
- Truncation feels like observational equivalence, what is bisimulation?
- Is network training in $Seq_0(Seq_0(\mathbb{C}))$?
- Is hyperparameter tuning in $Seq_0(Seq_0(Seq_0(\mathbb{C})))$?
- Applications in circuit design?
- Applications in control theory?
- General theory behind delayed traces?

Thanks!