Differentiation of stateful functions

Shin-ya Katsumata and David Sprunger*

ERATO MMSD Colloquium
February 7, 2019

Outline

Outline

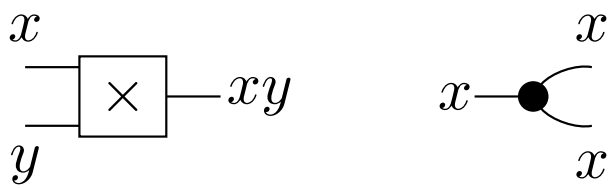
Ordinary functions as diagrams

We depict functions as boxes with input/output wires. For example, multiplication $\times : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and copying $\Delta_{\mathbb{R}} : \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ are:

Ordinary functions as diagrams

We depict functions as boxes with input/output wires. For example, multiplication $\times : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and copying $\Delta_{\mathbb{R}} : \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ are:

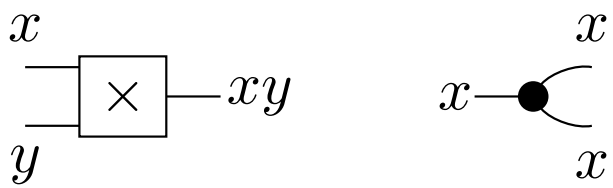
We may also put values on the wires for scratchwork, so



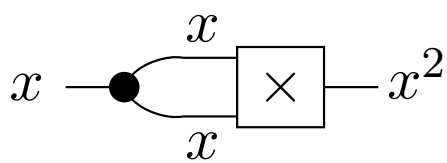
Ordinary functions as diagrams

We depict functions as boxes with input/output wires. For example, multiplication $\times : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and copying $\Delta_{\mathbb{R}} : \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ are:

We may also put values on the wires for scratchwork, so



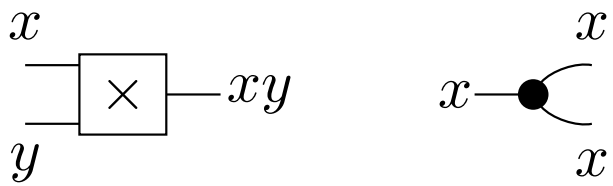
We depict sequential composition by:



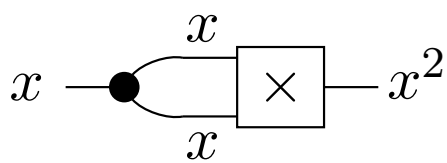
Ordinary functions as diagrams

We depict functions as boxes with input/output wires. For example, multiplication $\times : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and copying $\Delta_{\mathbb{R}} : \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ are:

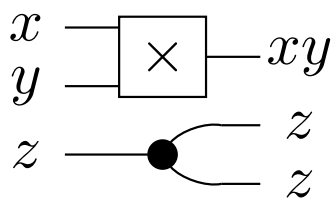
We may also put values on the wires for scratchwork, so



We depict sequential composition by:

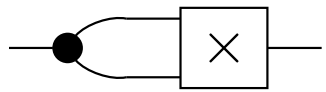


We can also compose functions in parallel:



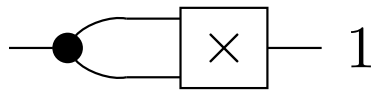
Functions on sequences, part I

Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	
2	
3	
⋮	

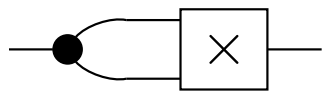
Functions on sequences, part I

Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	 1
2	
3	
⋮	

Functions on sequences, part I

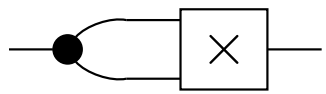
Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	1
2	
3	
⋮	

Functions on sequences, part I

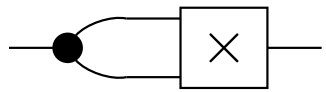
Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	1
2	4
3	
⋮	



Functions on sequences, part I

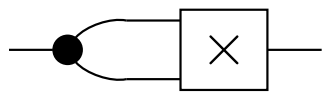
Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	1
2	4
3	
⋮	

Functions on sequences, part I

Ordinary functions can be upgraded to functions on sequences:

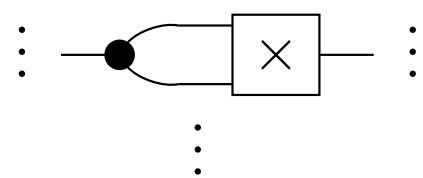
Input	**Output**
1	1
2	4
3	9
⋮	



Functions on sequences, part I

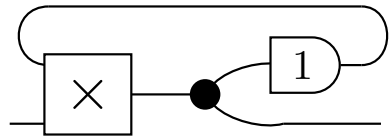
Ordinary functions can be upgraded to functions on sequences:

Input	**Output**
1	1
2	4
3	9



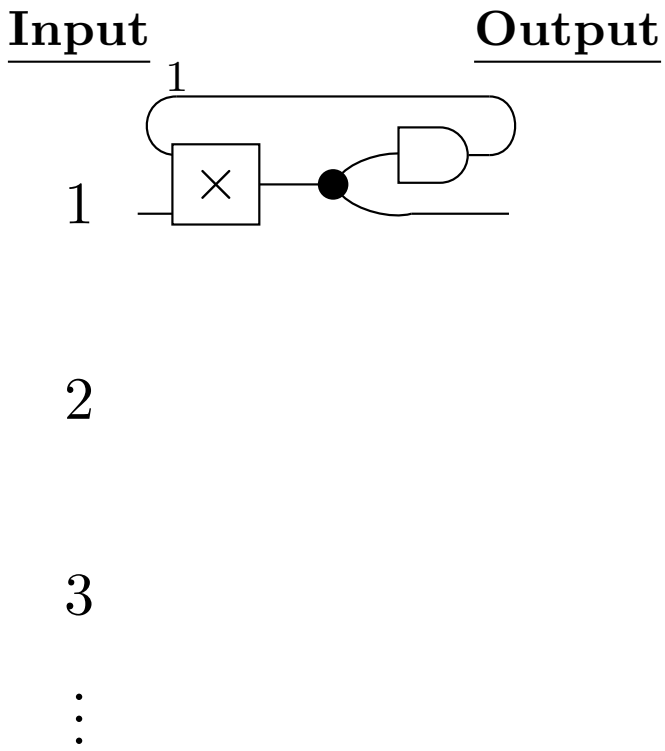
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	
2	
3	
⋮	

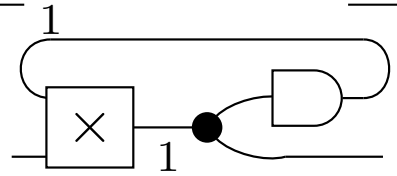
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:



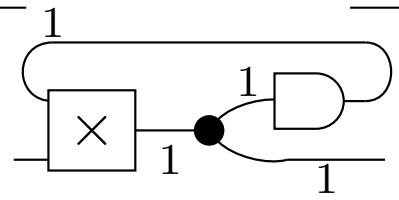
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	
2	
3	

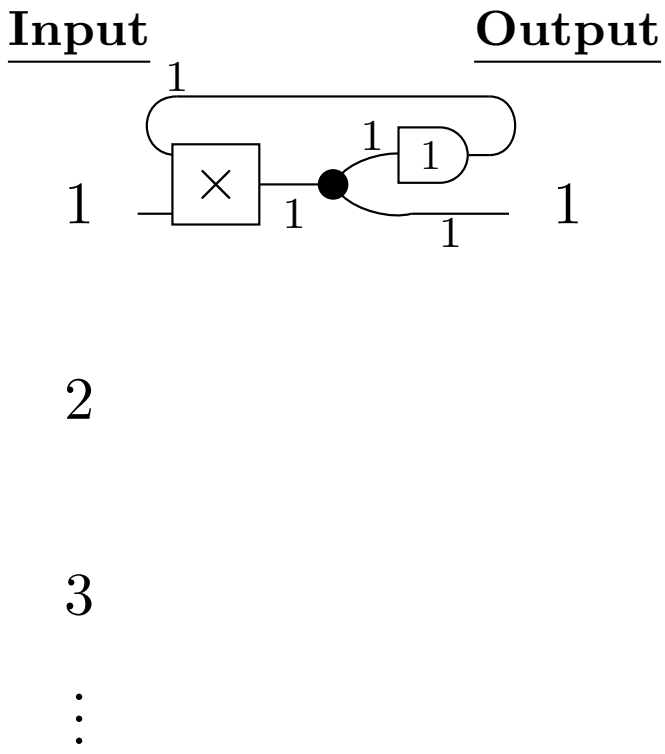
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	
2	
3	
⋮	

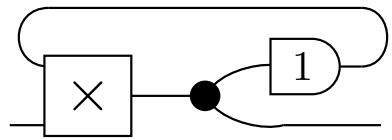
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:



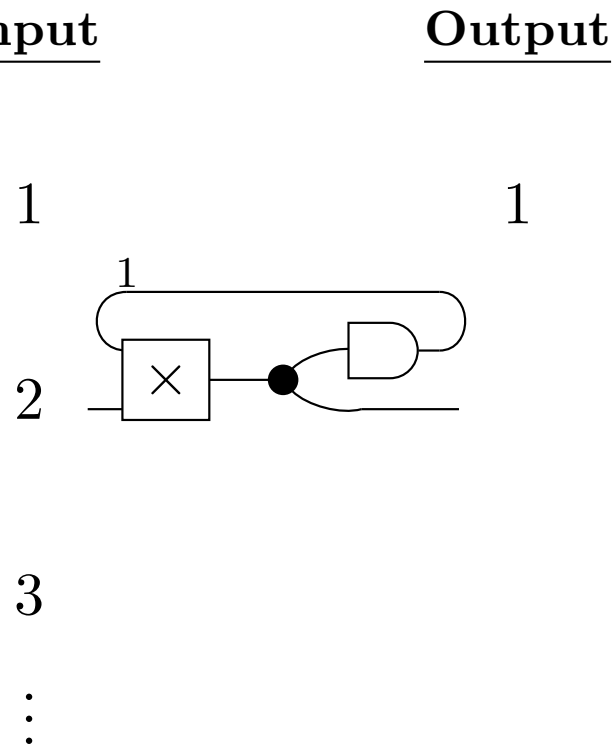
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	1
2	
3	
⋮	

Functions on sequences, part II

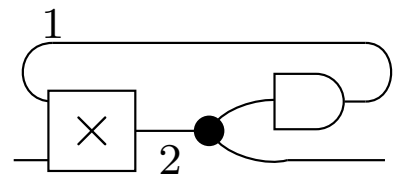
We also use state and feedback mechanisms for sequence functions:



Functions on sequences, part II

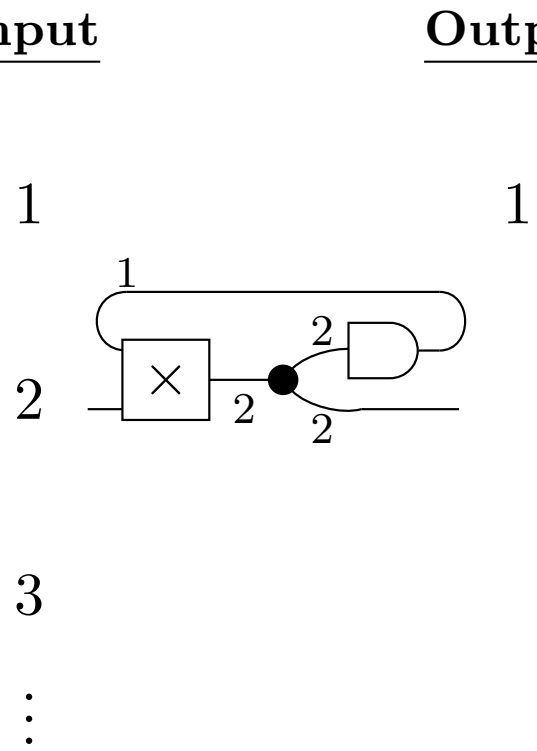
We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	1
2	
3	
⋮	



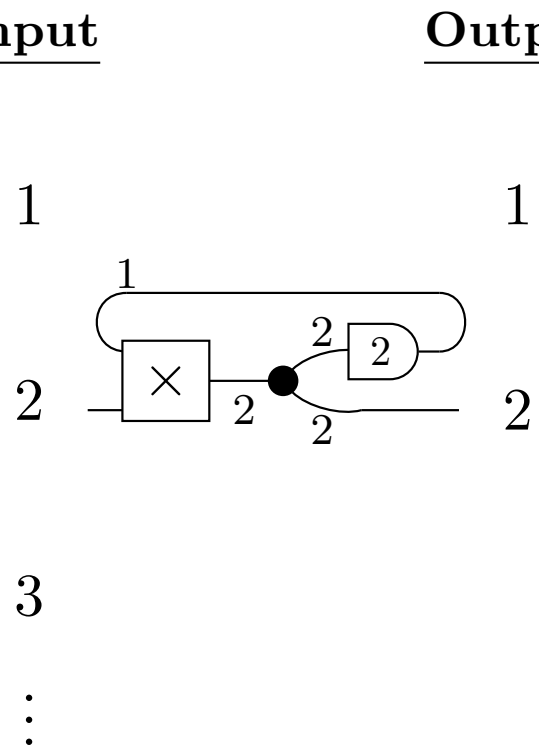
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:



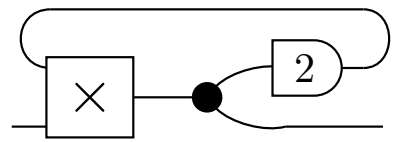
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:



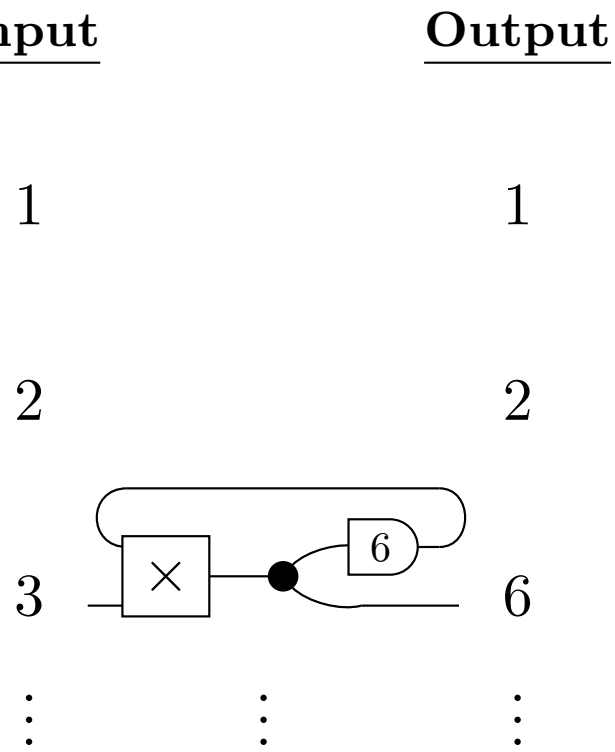
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:

Input	**Output**
1	1
2	2
3	
⋮	

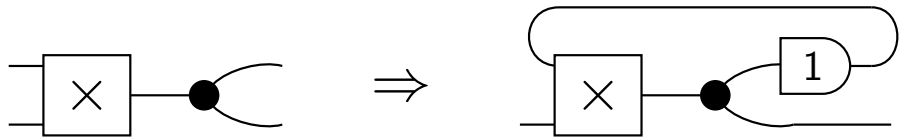
Functions on sequences, part II

We also use state and feedback mechanisms for sequence functions:



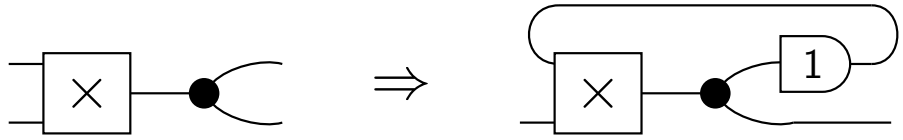
What is this diagram?

What do diagrams like this, or operational semantics shown previously, remind you of?

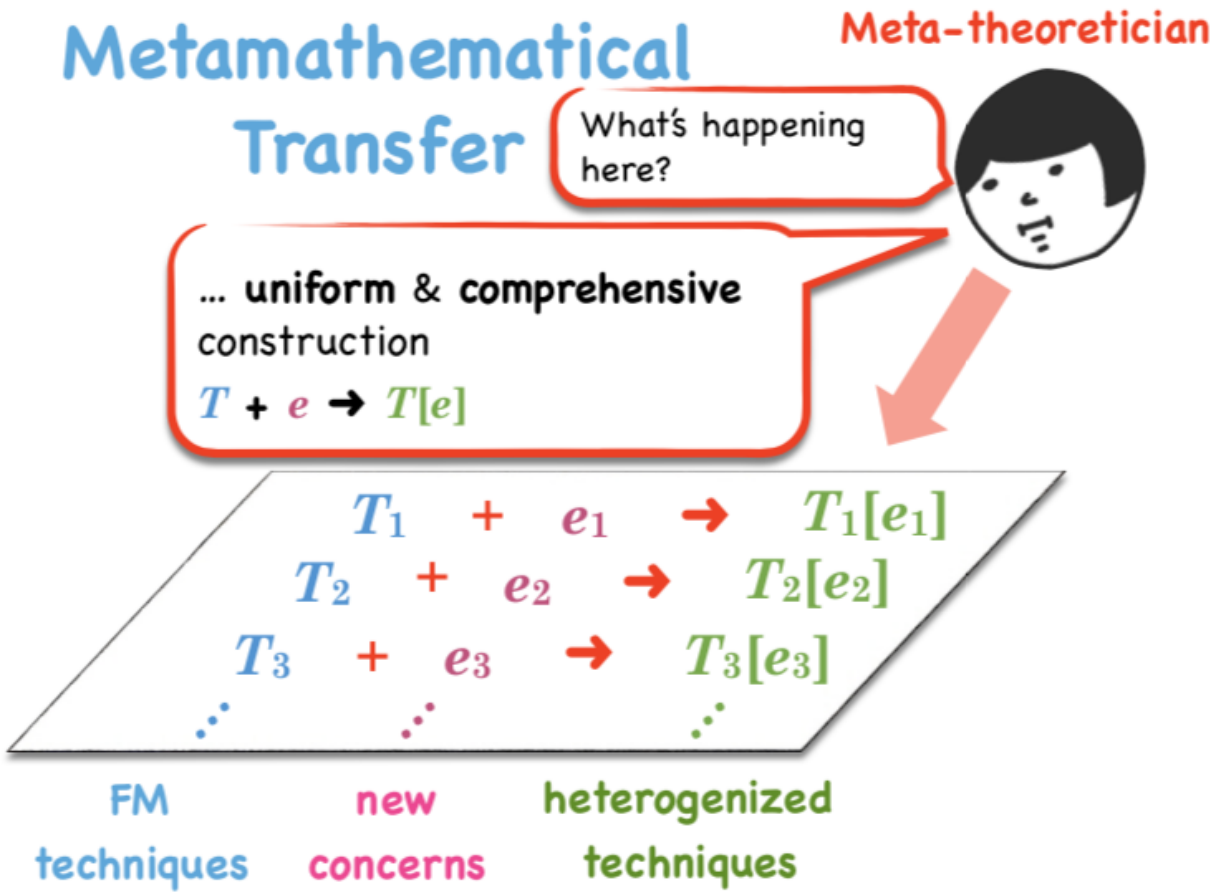


What is this diagram?

What do diagrams like this, or operational semantics shown previously, remind you of?



ahem



Outline

Backpropagation & backpropagation through time

Backpropagation: an efficient implementation for computing derivatives.

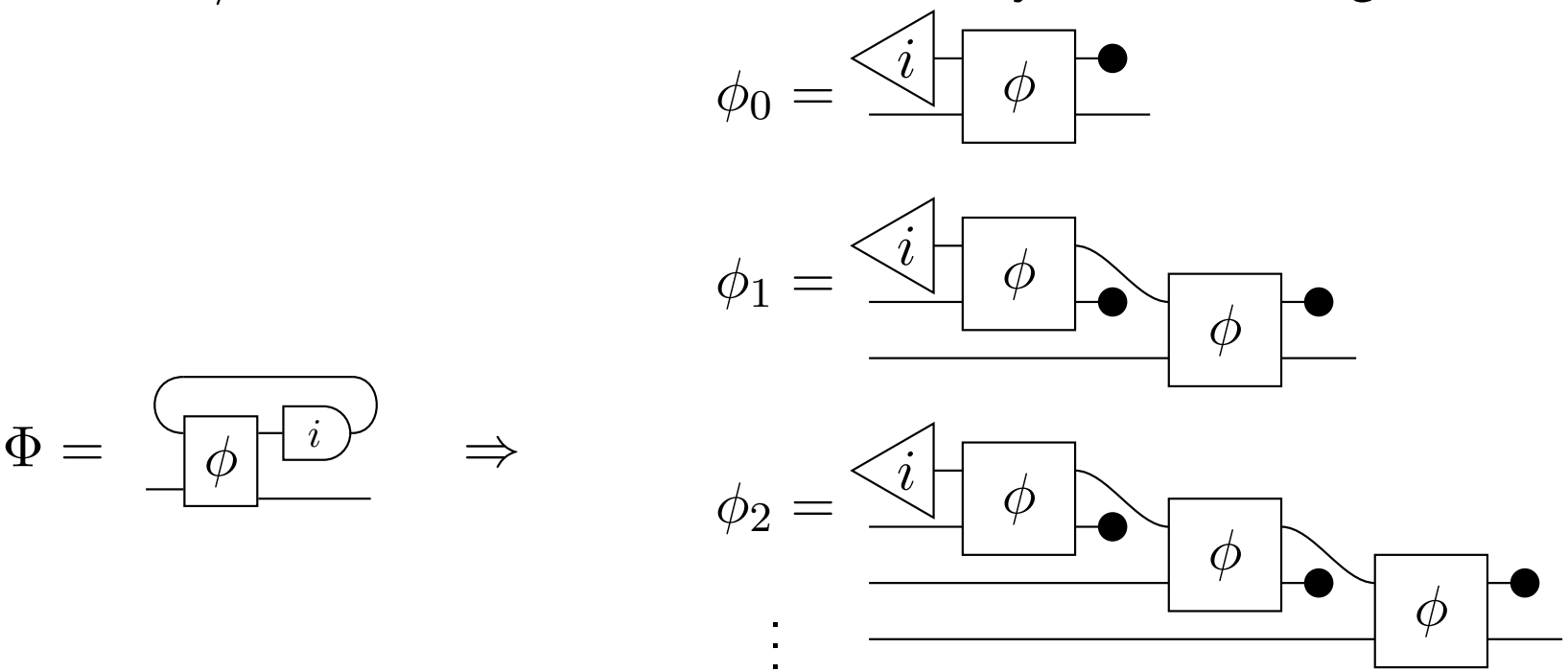
Backpropagation & backpropagation through time

Backpropagation: an efficient implementation for computing derivatives. Works only for stateless/feedforward networks.

Backpropagation & backpropagation through time

Backpropagation: an efficient implementation for computing derivatives. Works only for stateless/feedforward networks.

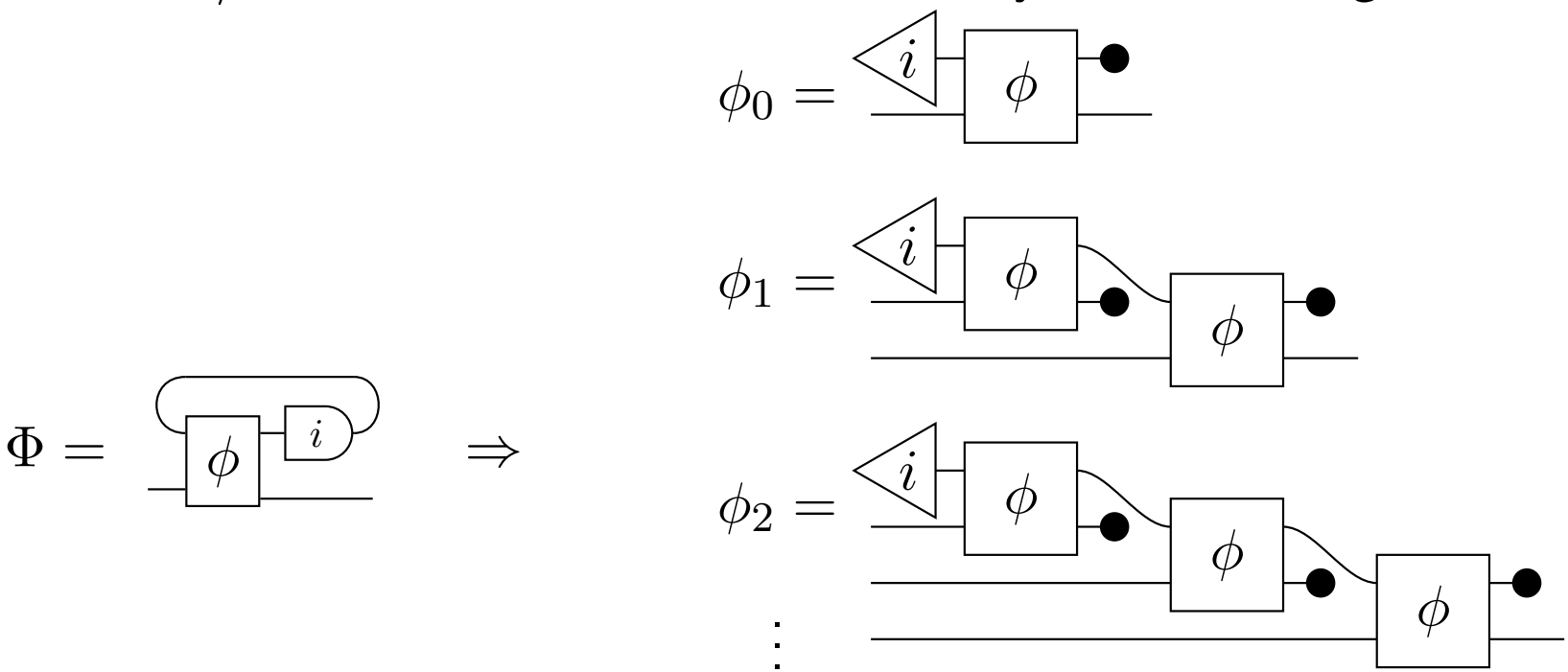
Stateful/recurrent networks are trained by first *unrolling:*



Backpropagation & backpropagation through time

Backpropagation: an efficient implementation for computing derivatives. Works only for stateless/feedforward networks.

Stateful/recurrent networks are trained by first *unrolling:*



Backpropagation through time (BPTT): Whenever the derivative of Φ is needed at an input of length $k+1$, the derivative of ϕ_k is supplied instead.

Does BPTT make sense, or is it just a hack?

Does BPTT make sense, or is it just a hack?

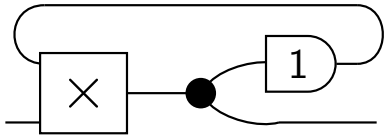
"Making sense" means having the usual properties of derivatives:

1. a sum rule,
2. a chain rule,
3. being linear when evaluated at any base point,
4. symmetry of mixed partial derivatives,

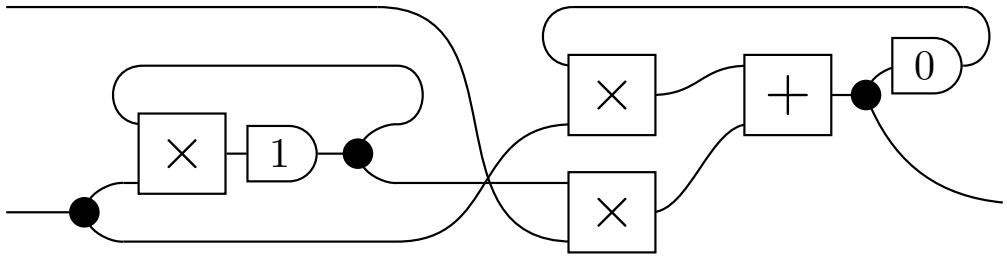
⋮

Outline

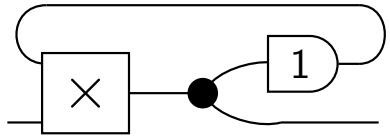
Goals of this talk (specific to general)

A. The derivative of 

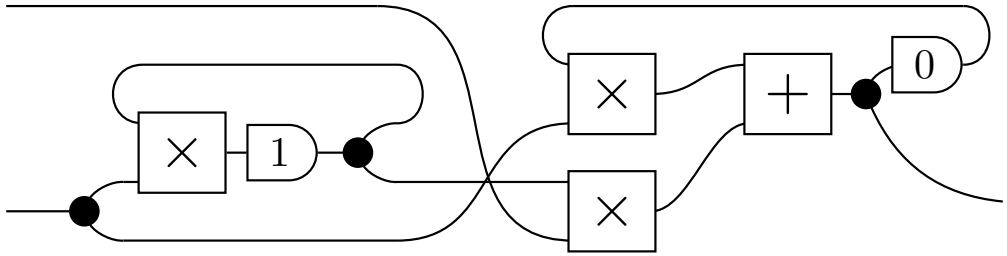
is this monster



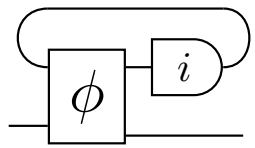
Goals of this talk (specific to general)

A. The derivative of 

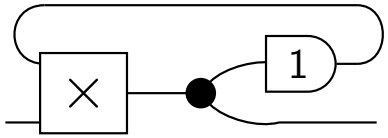
is this monster



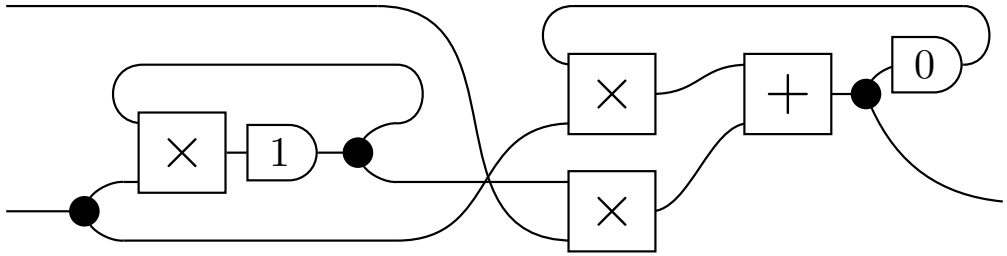
B. There is a general rule for derivatives of stateful functions



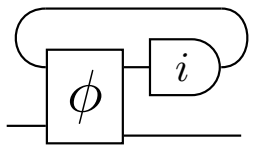
Goals of this talk (specific to general)

A. The derivative of 

is this monster

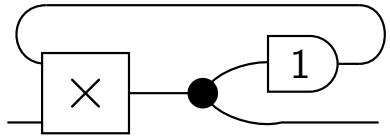


B. There is a general rule for derivatives of stateful functions

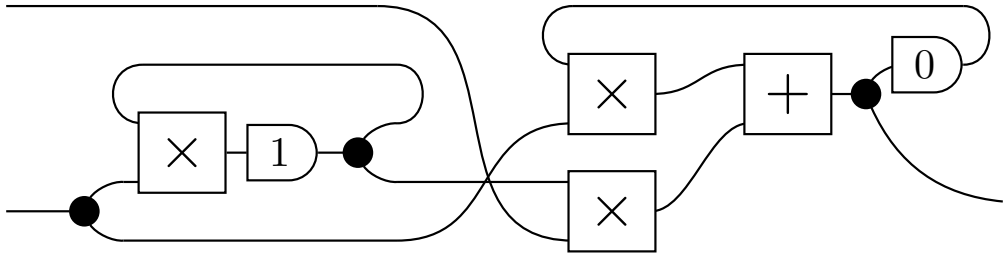


C. We understand many properties of this rule.

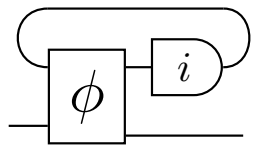
Goals of this talk (specific to general)

A. The derivative of 

is this monster



B. There is a general rule for derivatives of stateful functions



C. We understand many properties of this rule.

⋮

Z. ~~Profit??~~ Get paper??

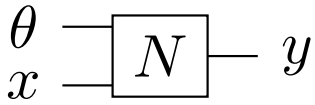
Goal Z & C: How do we get ¥¥¥?

Goal Z & C: How do we get ¥¥¥?

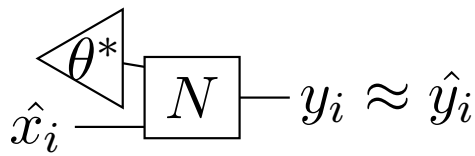
Answer: Use it for machine learning.

Goal Z & C: How do we get ¥¥¥?

Answer: Use it for machine learning.

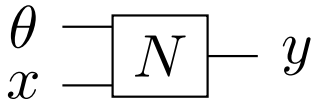
A *neural network* looks like 

. *Training* a neural network means finding θ^* so that:

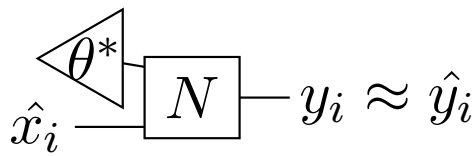


Goal Z & C: How do we get ¥¥¥?

Answer: Use it for machine learning.

A *neural network* looks like 

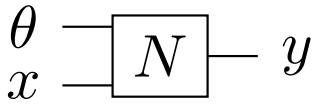
. *Training* a neural network means finding θ^* so that:



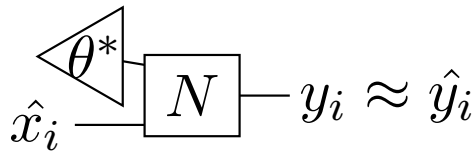
Gradient-based training algorithms are based on the insight that $\frac{\partial N}{\partial \theta}$ is a good approximation for the change in y that results from a small change in θ. This allows us to make smart updates to θ^*.

Goal Z & C: How do we get ¥¥¥?

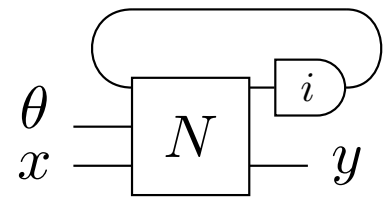
Answer: Use it for machine learning.

A *neural network* looks like 

. *Training* a neural network means finding θ^* so that:



Gradient-based training algorithms are based on the insight that $\frac{\partial N}{\partial \theta}$ is a good approximation for the change in y that results from a small change in θ. This allows us to make smart updates to θ^*.

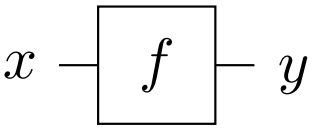
Upshot: If we can find (partial) derivatives of 

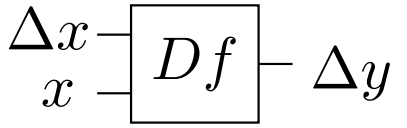
,
we can train recurrent neural networks. Using **properties** of these derivatives, maybe we can do it more efficiently than before.

Goal B: Derivatives in diagram form

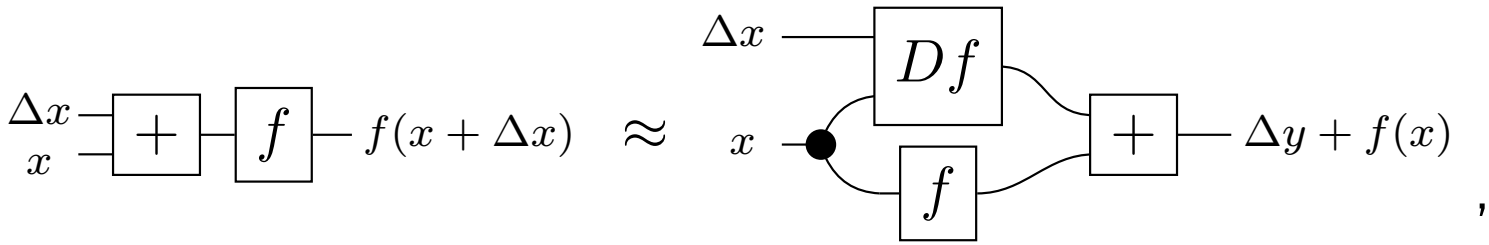
The derivative of x — f — y is $\begin{matrix}\Delta x \\ x\end{matrix}$ — Df — Δy, which gives the best linear approximation to ϕ at a base point x.

Goal B: Derivatives in diagram form

The derivative of 

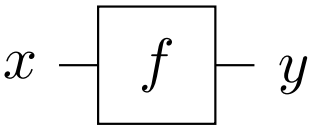
is 

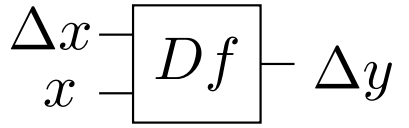
, which gives the best linear approximation to ϕ at a base point x. That is

1. 

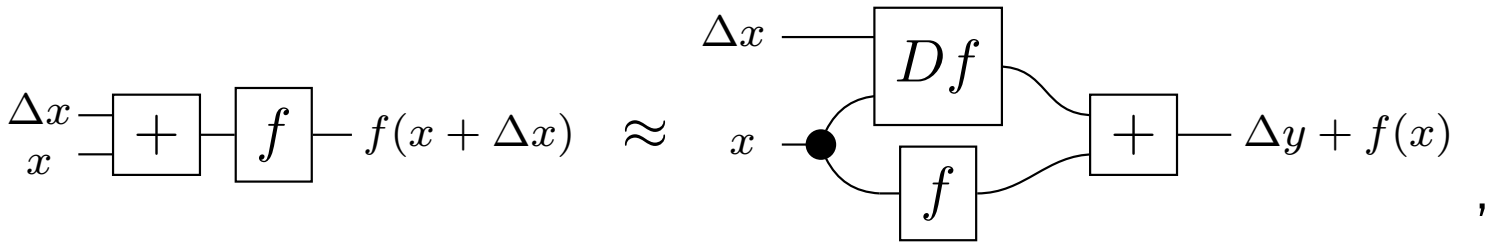
,

Goal B: Derivatives in diagram form

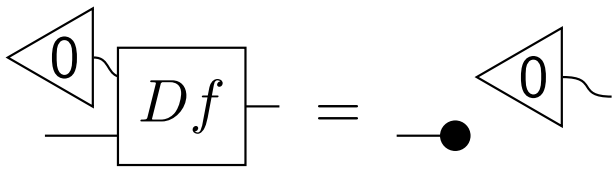
The derivative of 

is 

, which gives the best linear approximation to ϕ at a base point x. That is

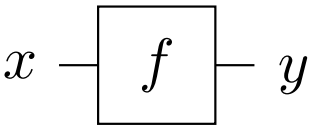
1. 

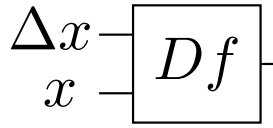
,

2. 

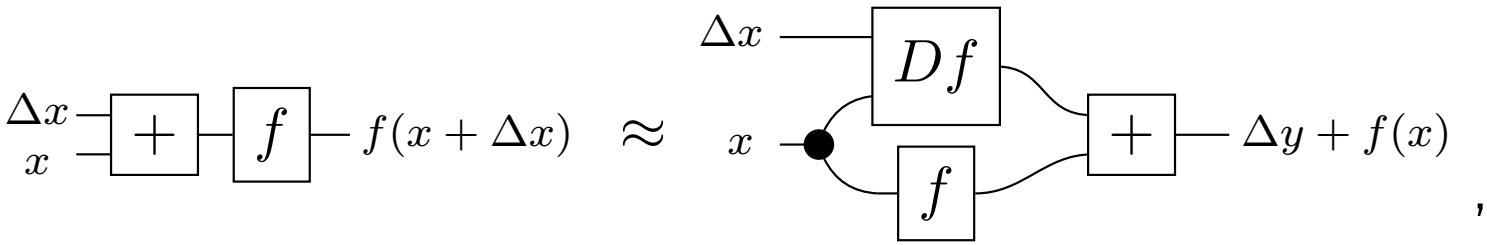
, and

Goal B: Derivatives in diagram form

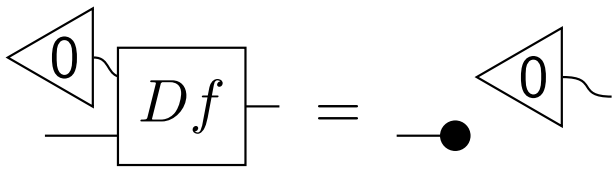
The derivative of 

is 

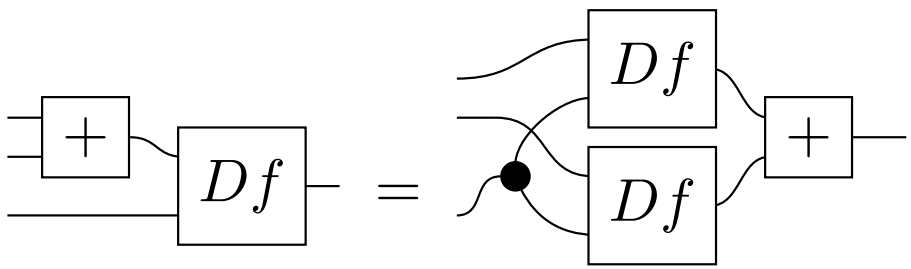
, which gives the best linear approximation to ϕ at a base point x. That is

1. 

,

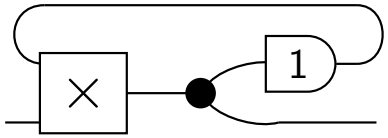
2. 

, and

3. 

.

Goal A: Does the monster approximate output change?

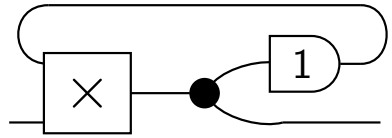
Let $\Phi =$ 

. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

What is $D\Phi((0.1, 1), (0.1, 1), (0.1, 1), \ldots)$?

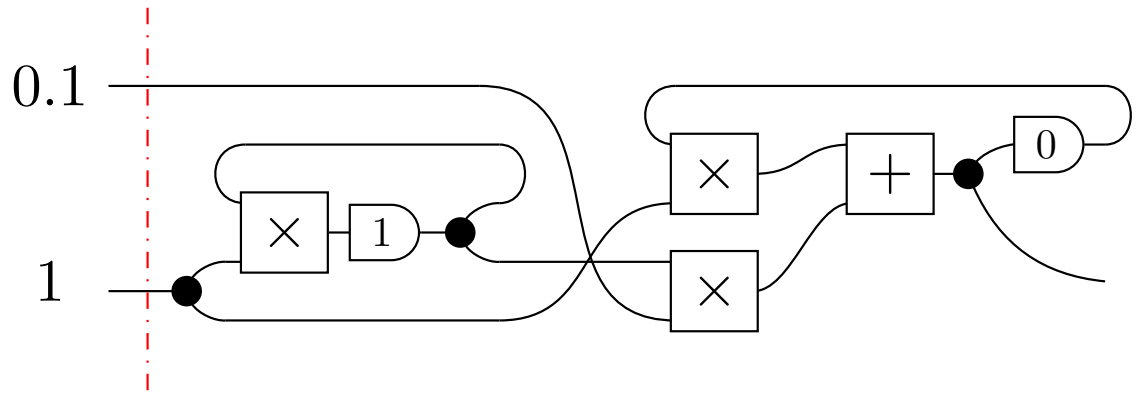
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

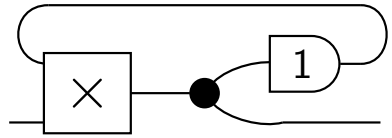
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

What is $D\Phi(\mathbf{(0.1,\ 1)}, (0.1, 1), (0.1, 1), \ldots)$?



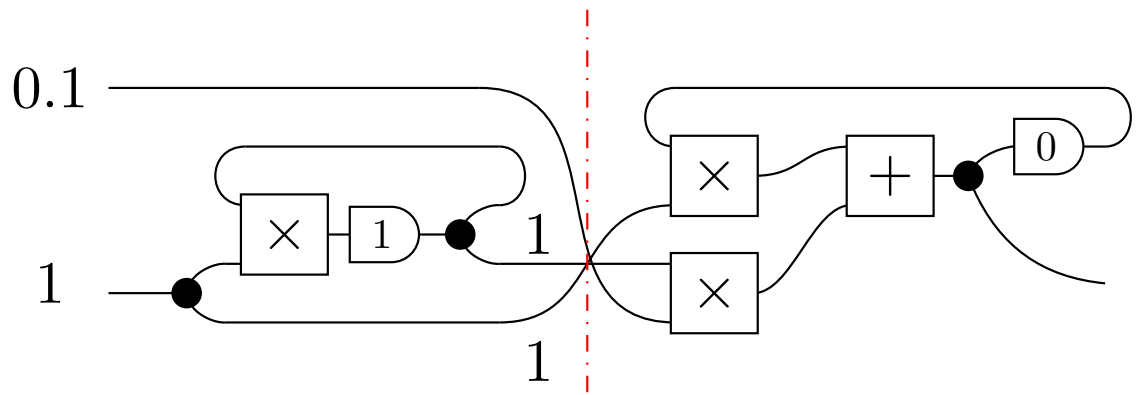
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

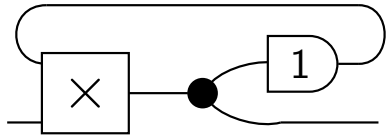
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

What is $D\Phi(\mathbf{(0.1,\ 1)}, (0.1, 1), (0.1, 1), \ldots)$?



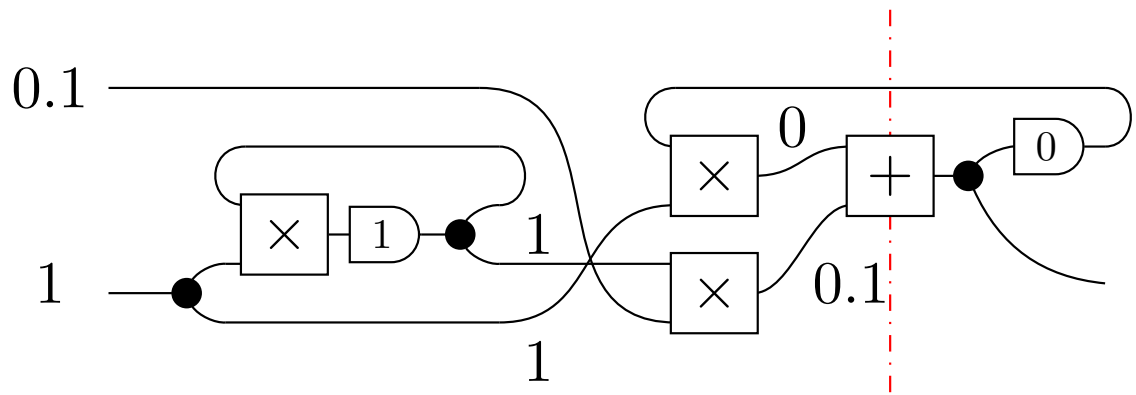
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

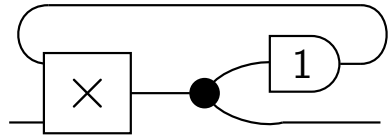
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

What is $D\Phi(\mathbf{(0.1,\ 1)}, (0.1, 1), (0.1, 1), \ldots)$?



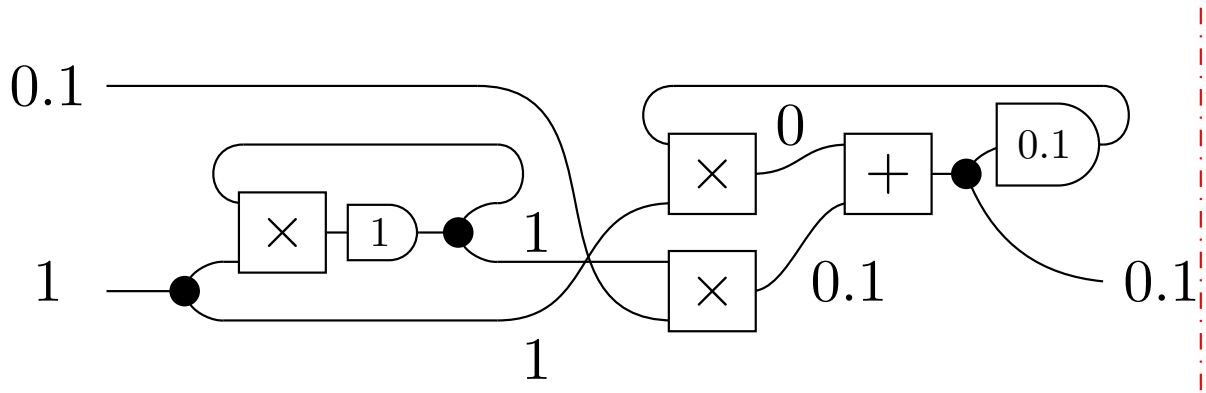
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

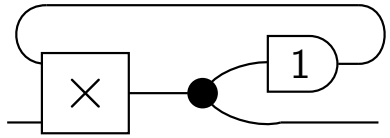
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi(\mathbf{(0.1,\ 1)}, (0.1, 1), (0.1, 1), \ldots) = (\mathbf{0.1,}\ \ldots).$$



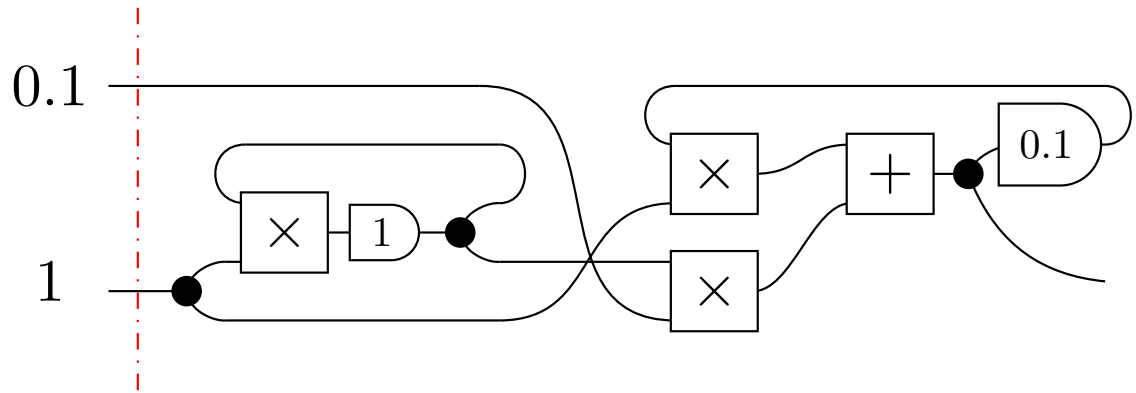
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

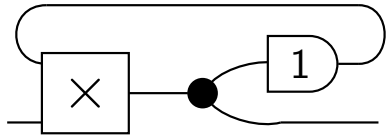
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), \mathbf{(0.1,\ 1)}, (0.1, 1), \ldots) = (0.1,\ \ldots).$$



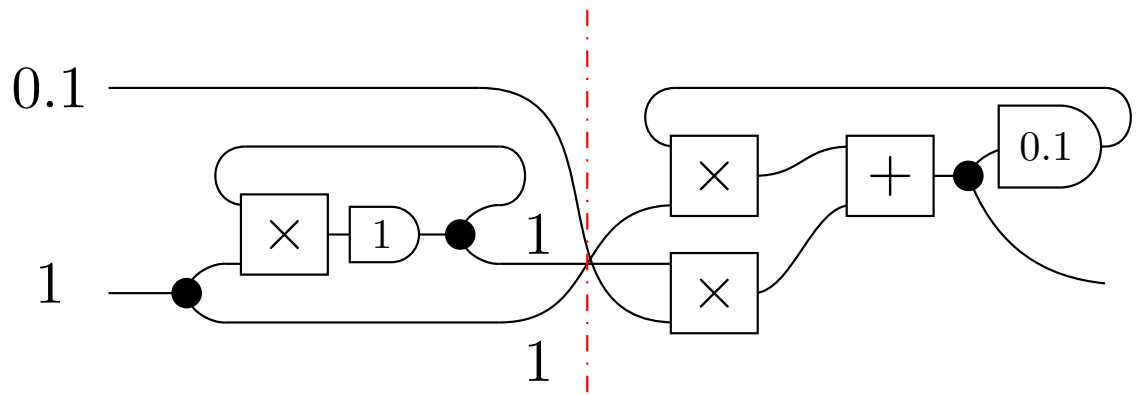
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

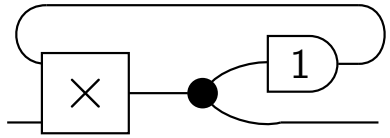
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), \mathbf{(0.1,\ 1)}, (0.1, 1), \ldots) = (0.1,\ \ldots).$$



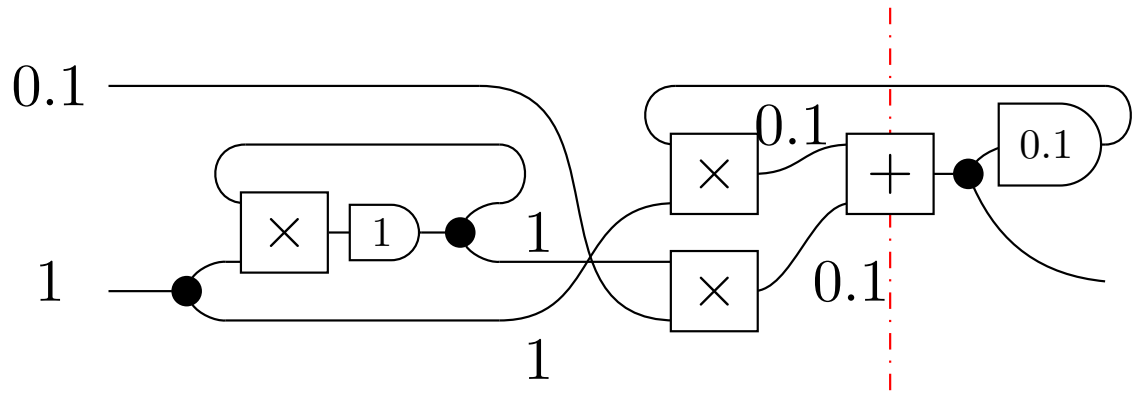
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

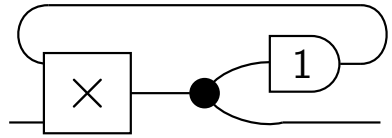
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), \mathbf{(0.1,\ 1)}, (0.1, 1), \ldots) = (0.1,\ \ldots).$$



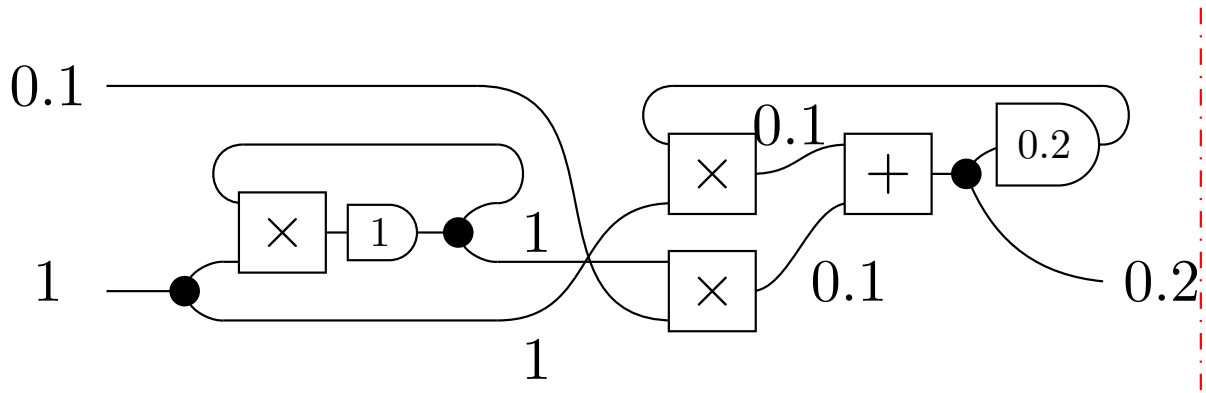
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

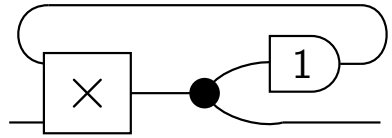
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), \mathbf{(0.1,\ 1)}, (0.1, 1), \ldots) = (0.1, \mathbf{0.2,}\ \ldots).$$



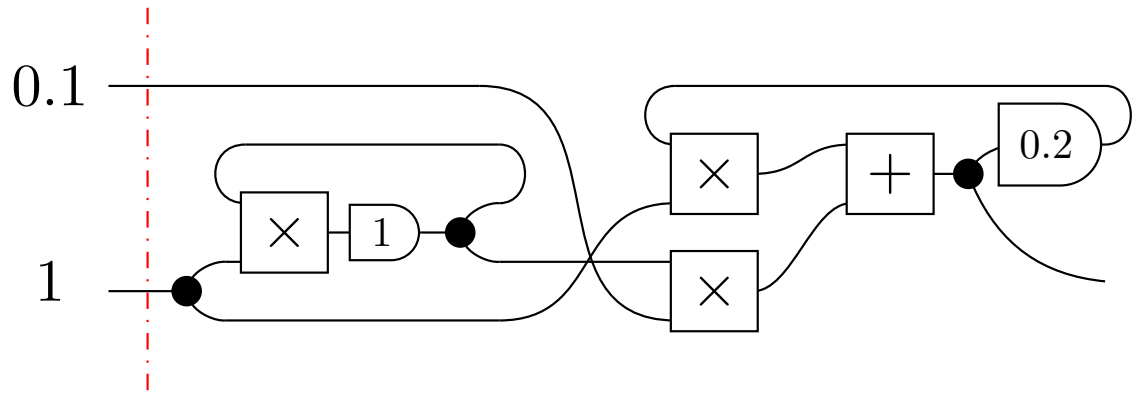
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

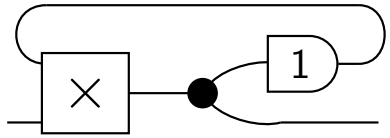
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), (0.1, 1), \mathbf{(0.1,\ 1)}, \ldots) = (0.1,\ 0.2,\ \ldots).$$



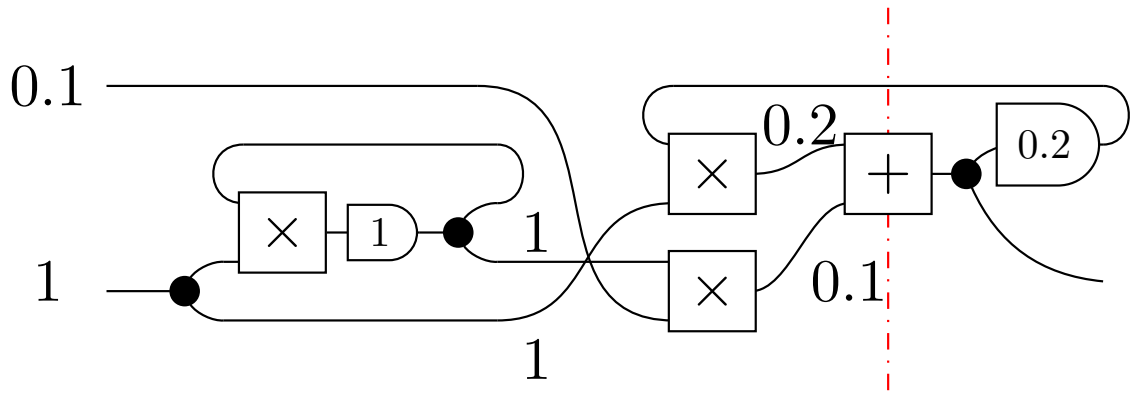
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

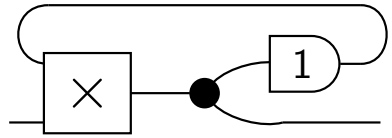
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

$$D\Phi((0.1, 1), (0.1, 1), \mathbf{(0.1,\ 1)}, \ldots) = (0.1,\ 0.2,\ \ldots).$$



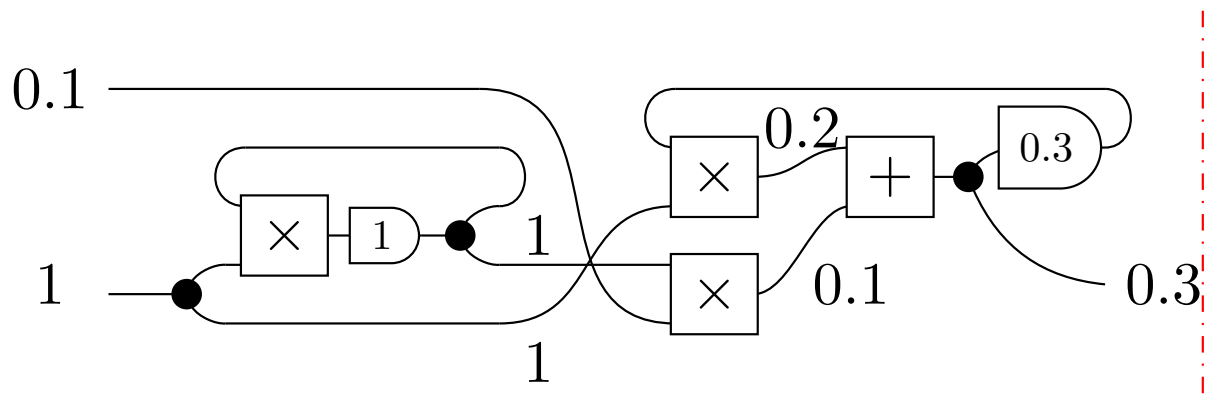
Goal A: Does the monster approximate output change?

Let $\Phi =$ 

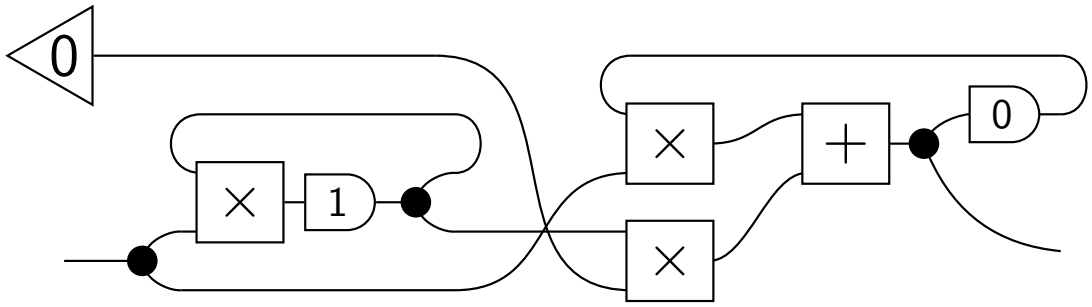
. Take $\mathbf{x} = (1, 1, 1, \ldots)$, and $\mathbf{\Delta x} = (0.1, 0.1, 0.1, \ldots)$. Then

$$\Phi(\mathbf{x}) = \Phi(1, 1, 1, \ldots) = (1, 1, 1, \ldots),$$
$$\Phi(\mathbf{x} + \mathbf{\Delta x}) = \Phi(1.1, 1.1, 1.1, \ldots) = (1.1, 1.21, 1.331, \ldots)$$

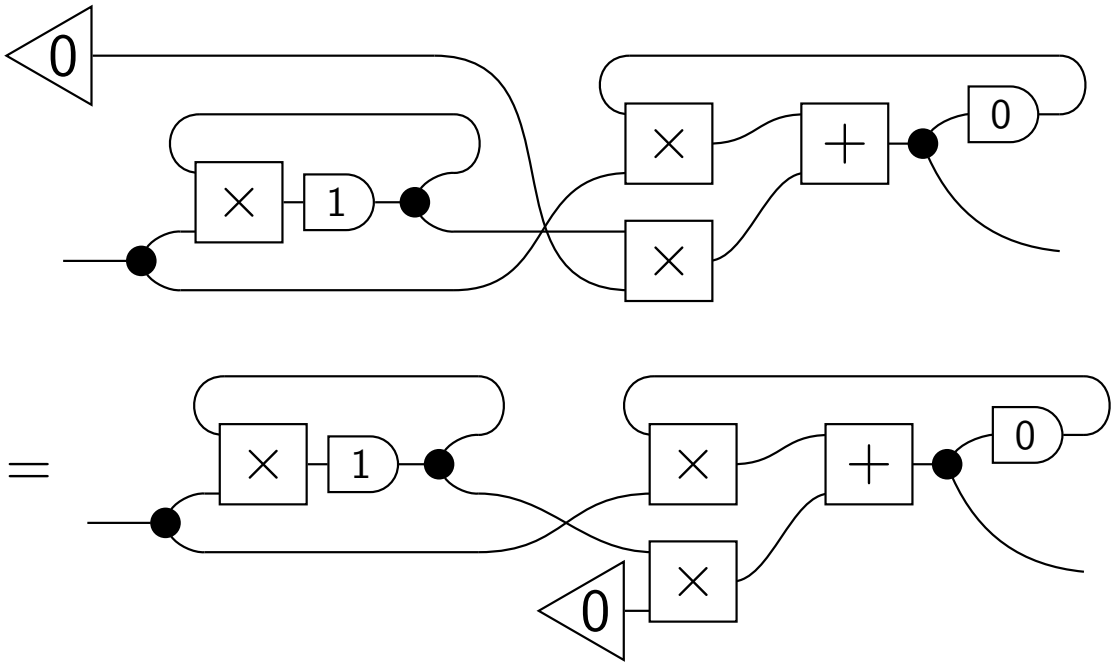
$$D\Phi((0.1, 1), (0.1, 1), \mathbf{(0.1,\ 1)}, \ldots) = (0.1,\ 0.2,\ \mathbf{0.3,}\ \ldots).$$



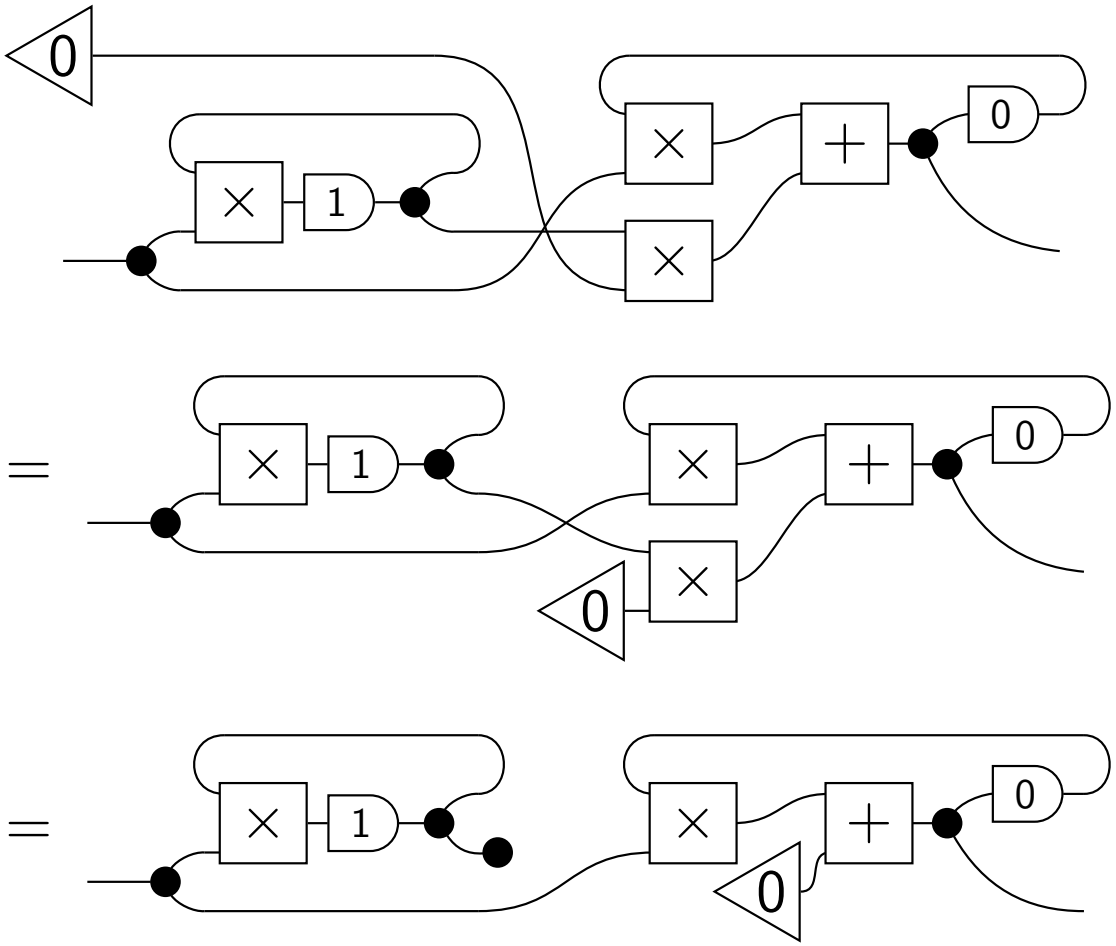
Goal A: Is $D\Phi(0, \mathbf{x}) = 0$?



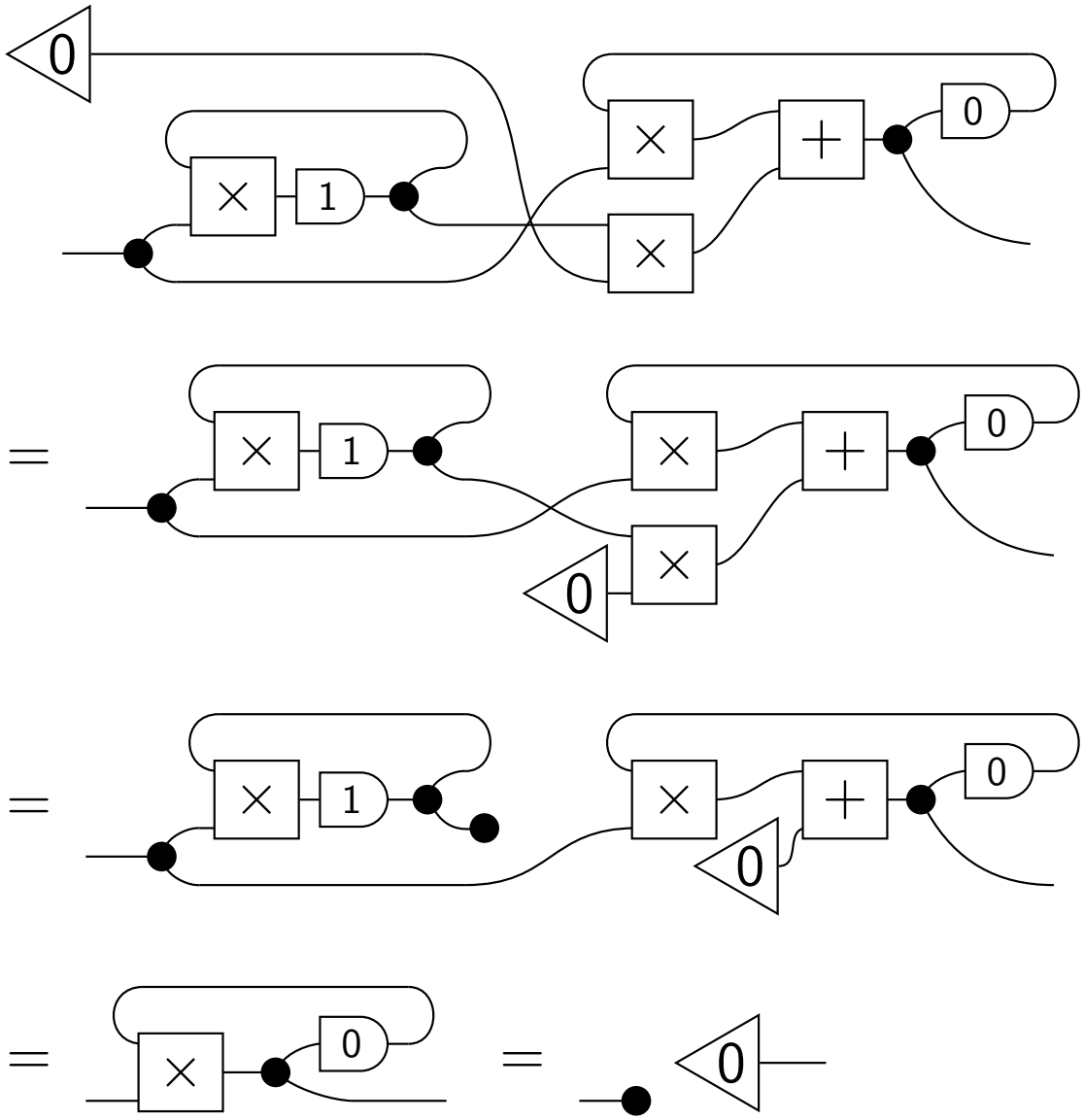
Goal A: Is $D\Phi(0, \mathbf{x}) = 0$?



Goal A: Is $D\Phi(0, \mathbf{x}) = 0$?



Goal A: Is $D\Phi(0, \mathbf{x}) = 0$?



Outline

Main ideas

We imagine that we are starting from a (strictified Cartesian) category $\mathbb{C}$ whose morphisms represent single computations, which we aim to extend to computations on sequences.

Main ideas

We imagine that we are starting from a (strictified Cartesian) category $\mathbb{C}$ whose morphisms represent single computations, which we aim to extend to computations on sequences.

We want to capture *causal* functions on sequences, meaning the nth element of the output sequence depends only on the first n elements of the input sequence.

Main ideas

We imagine that we are starting from a (strictified Cartesian) category $\mathbb{C}$ whose morphisms represent single computations, which we aim to extend to computations on sequences.

We want to capture *causal* functions on sequences, meaning the nth element of the output sequence depends only on the first n elements of the input sequence.

There are three steps:

1. describe a single step of the sequence computation, including a mechanism for sending and receiving state,
2. chain these single steps together to get a full sequence, and
3. quotient these computations by their observable behaviour.

Step 1: Single computation steps

We separate inputs and outputs of a $\mathbb{C}$-morphism into two types: *values*, exchanged with the environment, and *states*, received from (sent to) the previous (next) step of the computation.

Step 1: Single computation steps

We separate inputs and outputs of a $\mathbb{C}$-morphism into two types: *values*, exchanged with the environment, and *states*, received from (sent to) the previous (next) step of the computation.

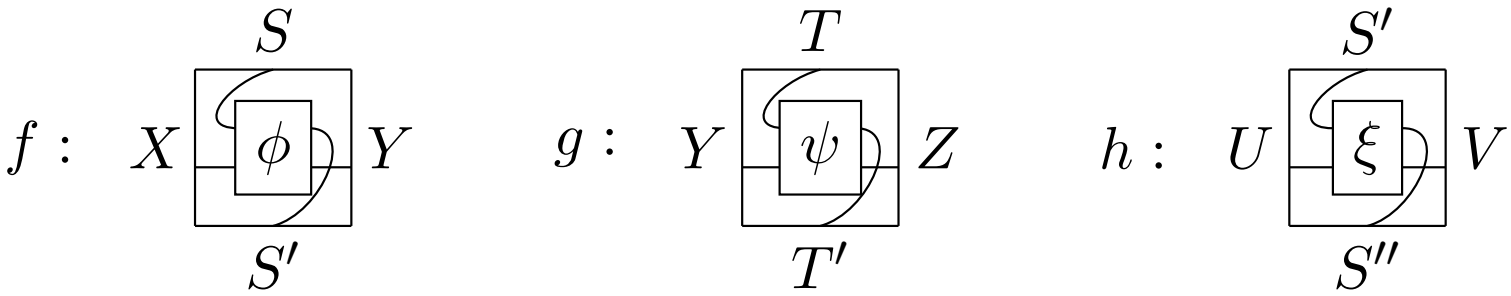
Definition

A *computation step* $f : X \xrightarrow[S']{S} Y$ is four objects and a morphism:

1. S — the input state received from the previous step
2. X — the input value received from the environment
3. S' — the output state sent to the next step
4. Y — the output value sent to the environment
5. $\phi : S \times X \to S' \times Y$ — the $\mathbb{C}$-morphism doing the computation

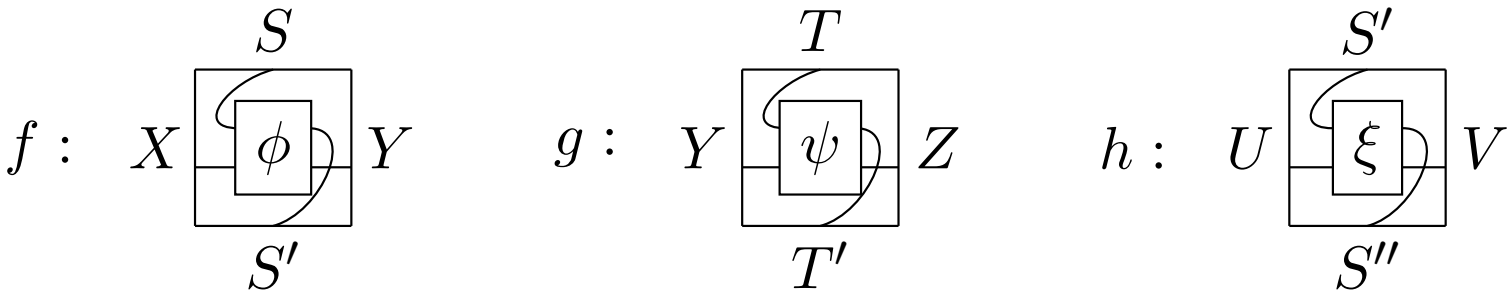
Step 1: Single computation steps

We can draw a single step like so:

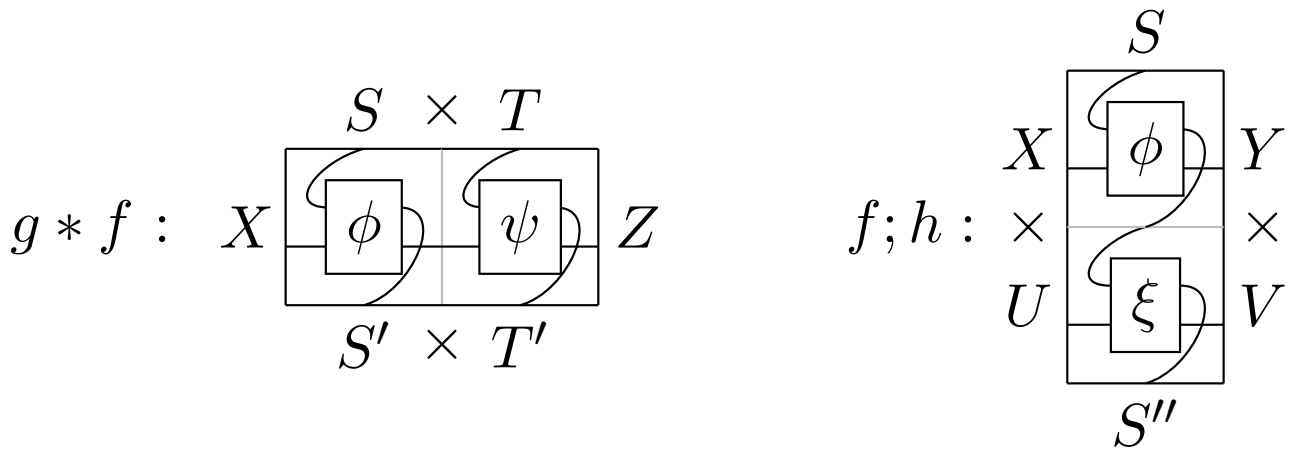


Step 1: Single computation steps

We can draw a single step like so:

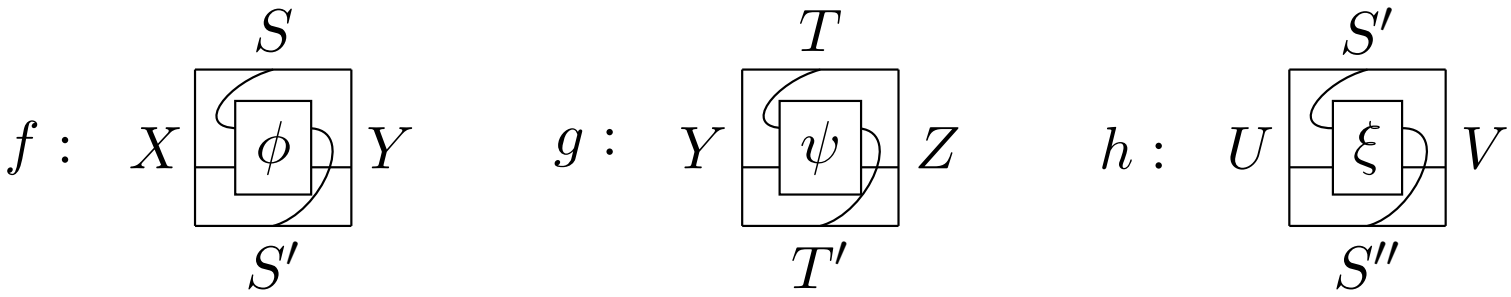


Computation steps can be composed in two different ways:

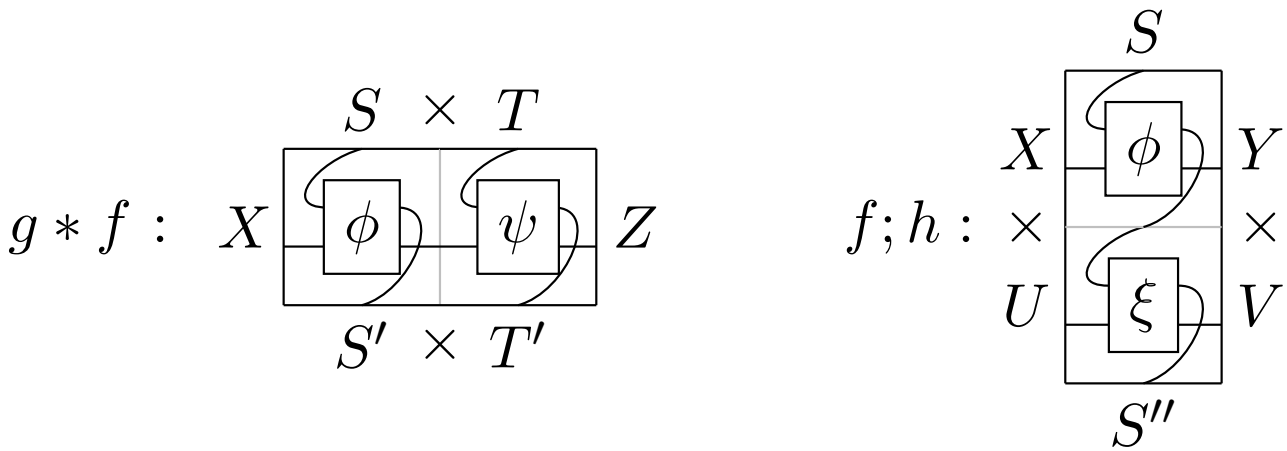


Step 1: Single computation steps

We can draw a single step like so:



Computation steps can be composed in two different ways:



psst . . . double category

Step 2: Computation sequences

Next, we chain together infinitely many of these computation steps to compute a whole sequence of outputs.

Definition

A *computation sequence* $\mathbf{f} = (i, [f_k])$ is an infinite sequence of computation steps $f_k : X_k \xrightarrow[S_{k+1}]{S_k} Y_k$, and an initial state $i : 1 \xrightarrow[S_0]{1} 1$. We say $\mathbf{f}$ takes sequences of type $[X_k] = (X_0, X_1, \ldots, X_n, \ldots)$ to sequences of type $[Y_k] = (Y_0, Y_1, \ldots, Y_n, \ldots)$.

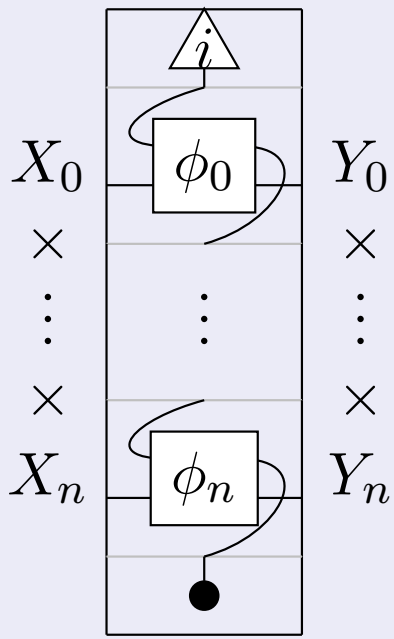
Note that f_k and f_{k+1} are vertically composable, as are i and f_0.

Step 3: Comparing computation sequences

Two computation sequences might have different state spaces and still compute the same function. We need to make some identifications. . .

Definition

The n*th truncation* of a computation sequence is the morphism of the vertical composite of the first $n+1$ steps:



Step 3: Comparing computation sequences

Definition

Two computation sequences $\mathbf{f}, \mathbf{g} : [X_k] \to [Y_k]$ are *extensionally equivalent* means they have the same nth truncation for all $n \in \mathbb{N}$.

Definition

A *stateful (sequence) function* is an extensional equivalence class of computation sequences.

Definition

If $\mathbb{C}$ is a (strict) Cartesian category, then its *stateful extension* is a category $\mathrm{St}(\mathbb{C})$ where

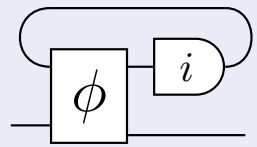
- objects are $[X_k]$, i.e. infinite sequences of objects in $\mathbb{C}$ and
- morphisms are stateful functions $\mathbf{f} : [X_k] \to [Y_k]$.

Sanity checks

Theorem

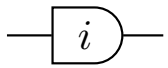
The homset $\mathrm{St}(Set)([A],[B])$ *is in 1-1 correspondence with the set of causal functions from* A^ω *to* B^ω*.*

Theorem

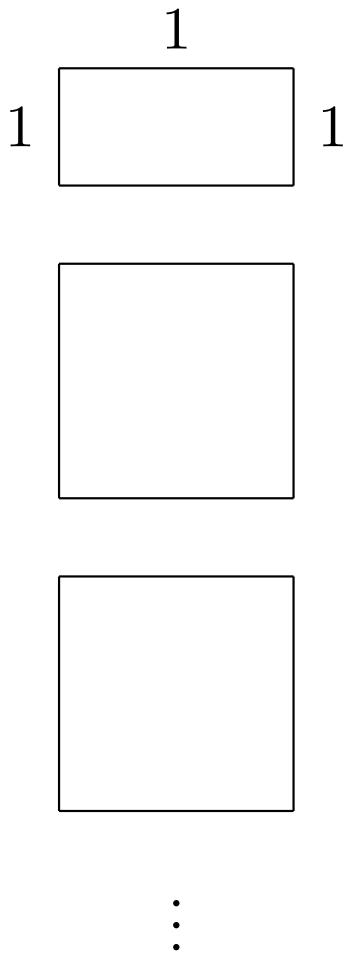
The sequence of n*th truncations of* $(i,[\phi])$*, each projected to their last component, is the unrolling of* 

.

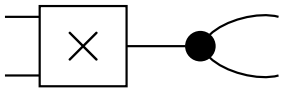
Fill-in-the-blank

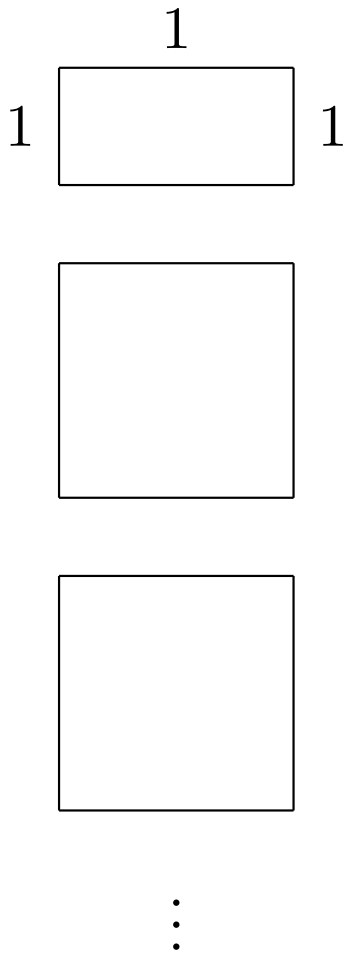
How would you implement 

as a computation sequence?

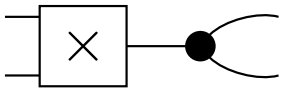


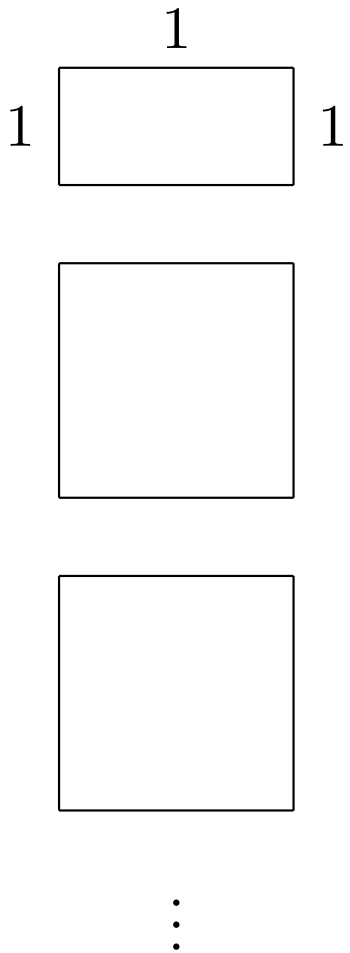
Fill-in-the-blank

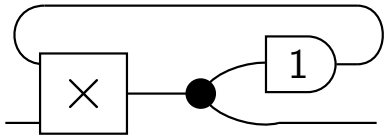
How would you implement  as a computation sequence?

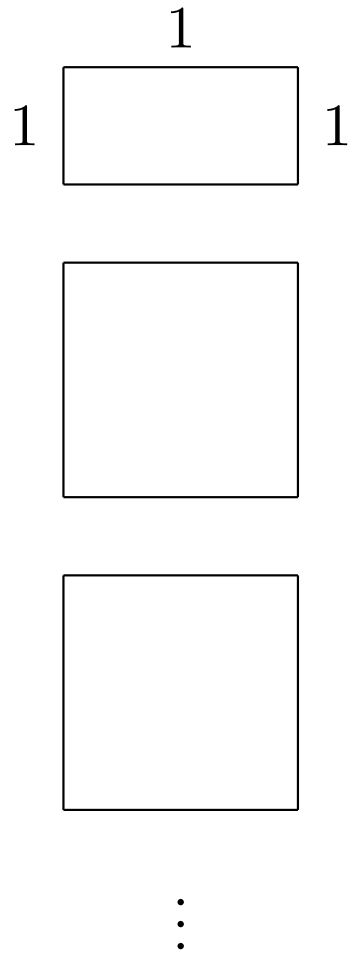


Fill-in-the-blank

How would you implement  as a computation sequence?



How would you implement  as a computation sequence?

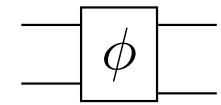


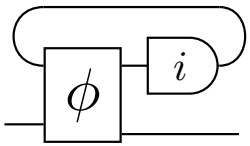
Outline

Delayed trace

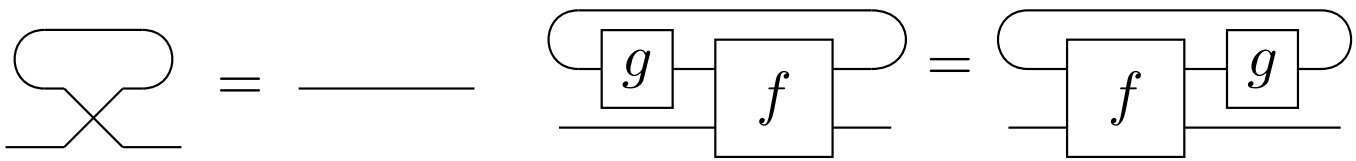
This loop-with-delay-gate is a trace-like operation.

$$\frac{\phi : S \times X \to S \times Y}{dtr_i^S(\phi) : X \to Y}$$

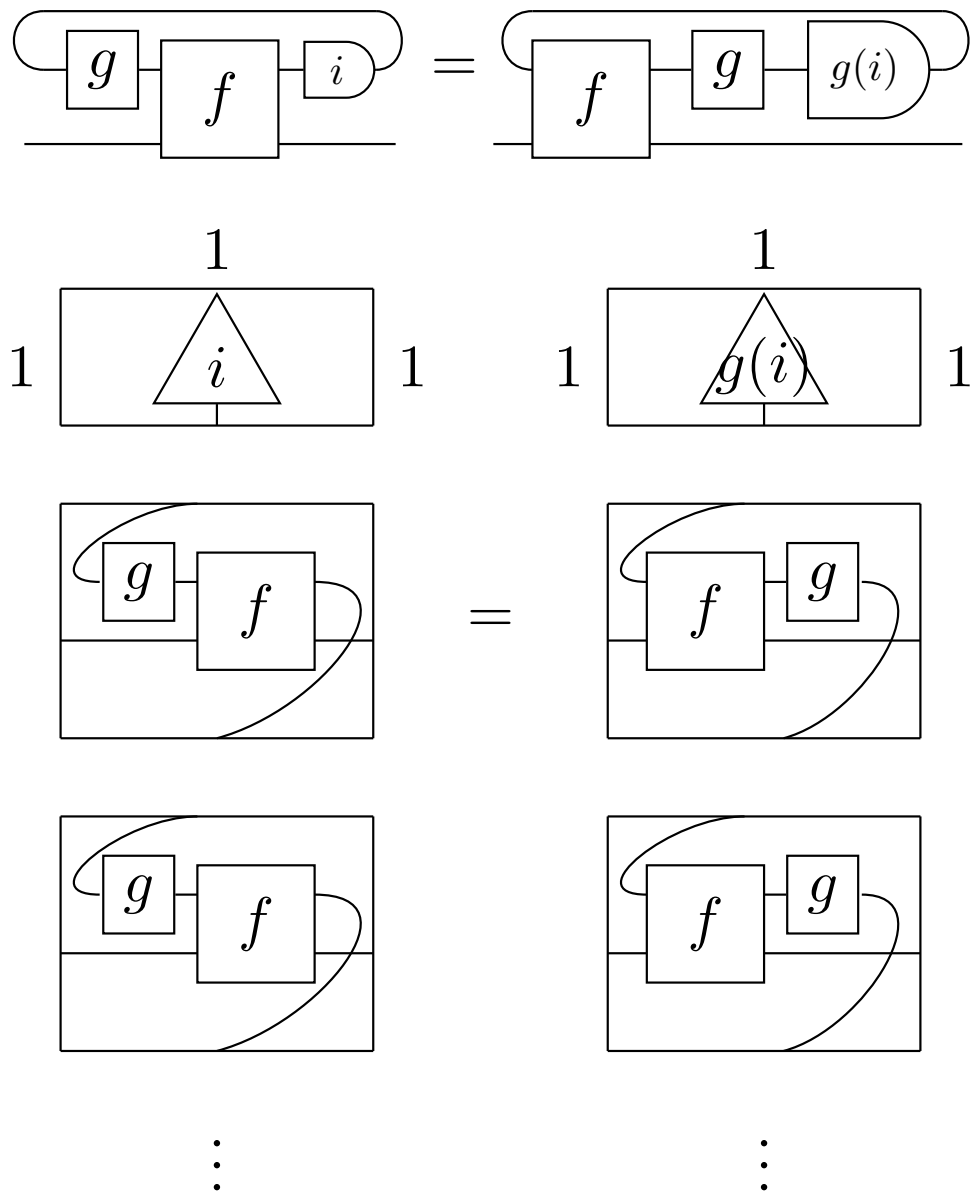




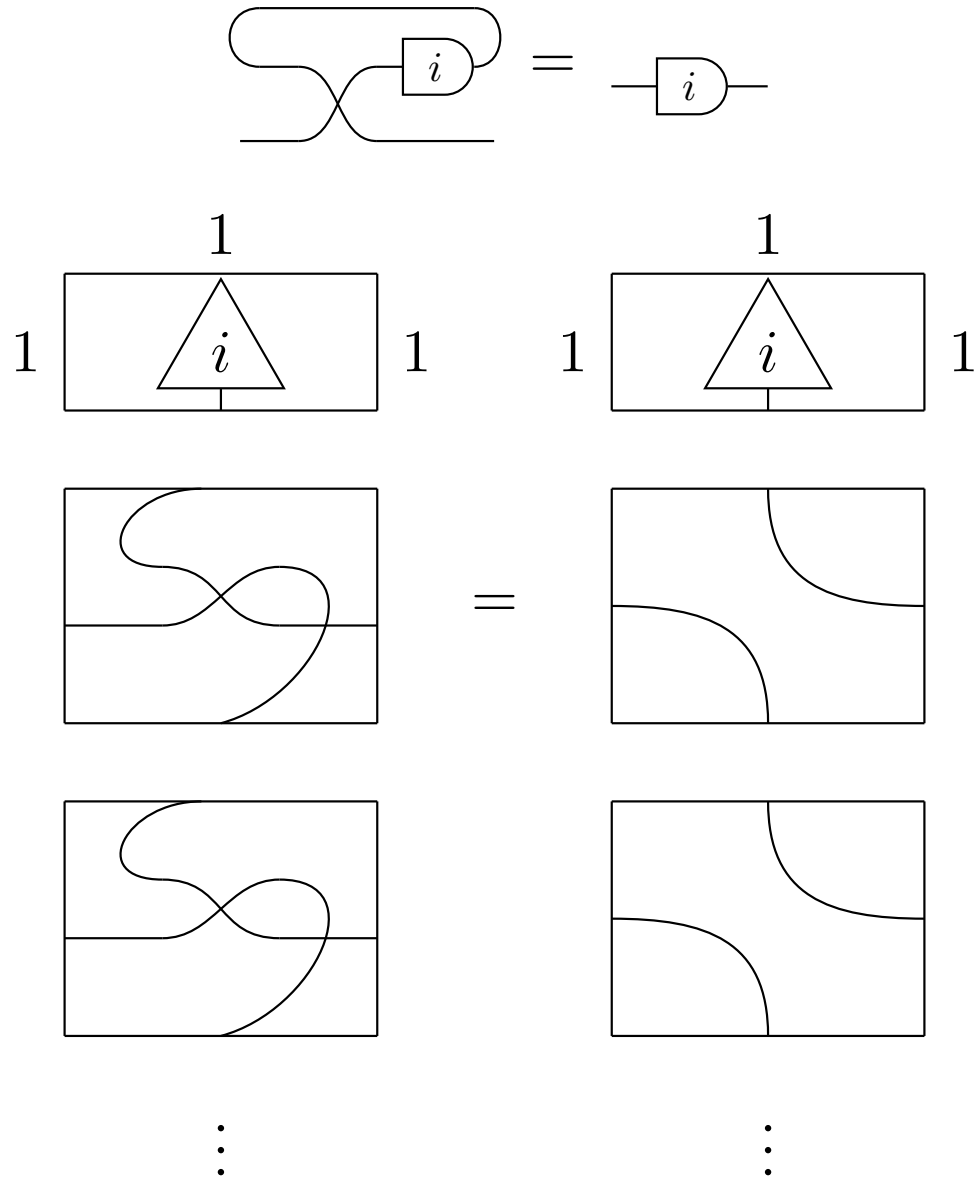
It satisfies **most** of the the trace axioms (source & target naturality, vanishing 1 and $\times$, and superposing) but misses two: yanking and dinaturality. For regular trace, those are



~~Dinaturality~~ → retiming



~~Yanking~~ → delay



Advantages of delayed trace

Often, people formalizing circuits in category theory do these:

1. (freely) add registers to the category,
2. add trace to the category,
3. restrict to a subcategory such that all traces are taken on positions guarded by a register.

Advantages of delayed trace

Often, people formalizing circuits in category theory do these:

1. (freely) add registers to the category,
2. add trace to the category,
3. restrict to a subcategory such that all traces are taken on positions guarded by a register.

We suggest instead to:

1. add a delayed trace to the category,
2. recover registers by delay-tracing symmetry

This ensures all loops are guarded (without a syntactic restriction!) and requires less structure.

Advantages of delayed trace

Often, people formalizing circuits in category theory do these:

1. (freely) add registers to the category,
2. add trace to the category,
3. restrict to a subcategory such that all traces are taken on positions guarded by a register.

We suggest instead to:

1. add a delayed trace to the category,
2. recover registers by delay-tracing symmetry

This ensures all loops are guarded (without a syntactic restriction!) and requires less structure.

We are also interested in looking at delayed traces abstractly...

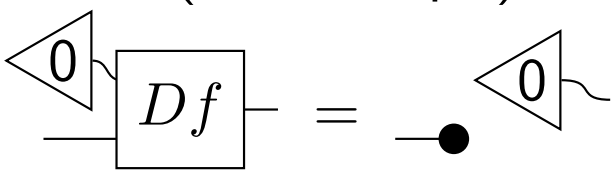
Outline

Cartesian differential categories [Blute, Cockett, Seely '09]

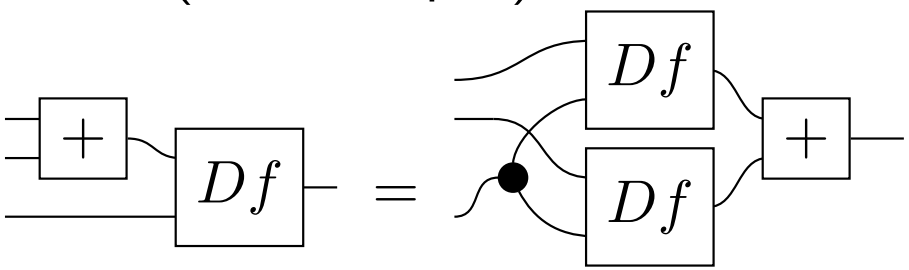
A *Cartesian differential category* has a differential operation on morphisms sending $f : X \to Y$ to $Df : X \times X \to Y$, satisfying:

CD1. $Ds = s \times !_{\mathrm{dom}(s)}$ for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

CD2. (linear map I)

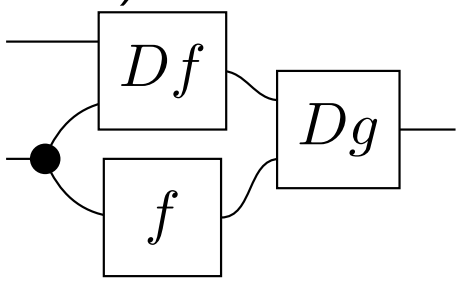


CD3. (linear map II)



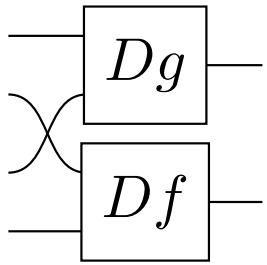
CD4. (chain rule)

$D(g \circ f) =$

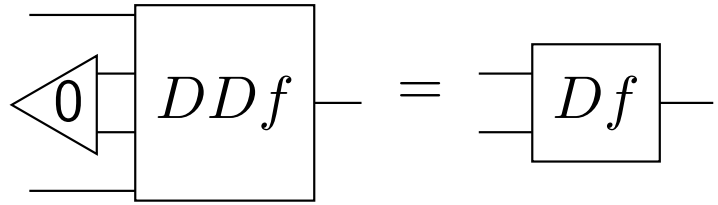


CD5. (parallel rule)

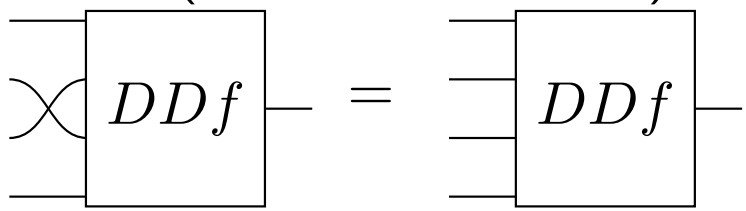
$D(g \times f) =$



CD6. (linearity of 2^{nd} deriv.)



CD7. (Schwartz theorem)



$\mathbf{Euc}_\infty$ is Cartesian differential

The differential operator on Euclidean spaces and smooth maps sends $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

$\mathbf{Euc}_\infty$ is Cartesian differential

The differential operator on Euclidean spaces and smooth maps sends $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

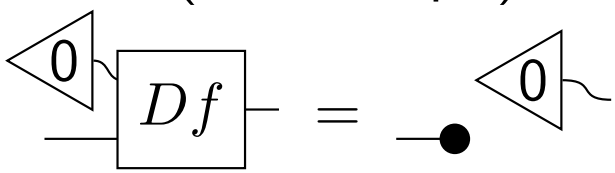
CD1. $Ds(\Delta x, x) = s(\Delta x)$ for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

$\mathbf{Euc}_\infty$ is Cartesian differential

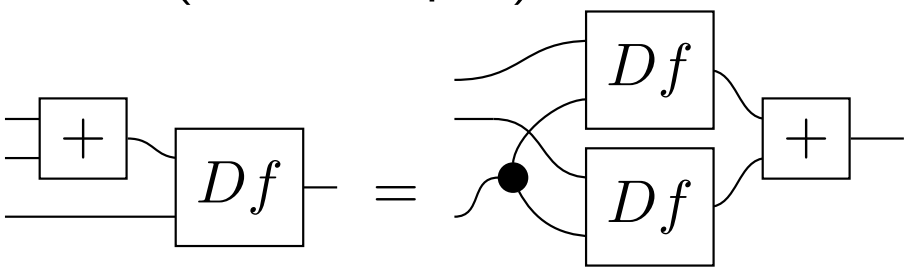
The differential operator on Euclidean spaces and smooth maps sends $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

CD1. $Ds(\Delta x, x) = s(\Delta x)$ for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

CD2. (linear map I)



CD3. (linear map II)

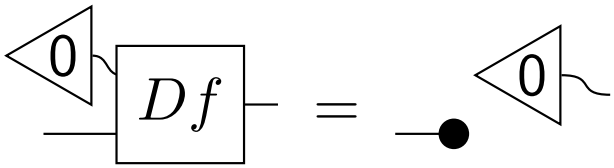


$\mathbf{Euc}_\infty$ is Cartesian differential

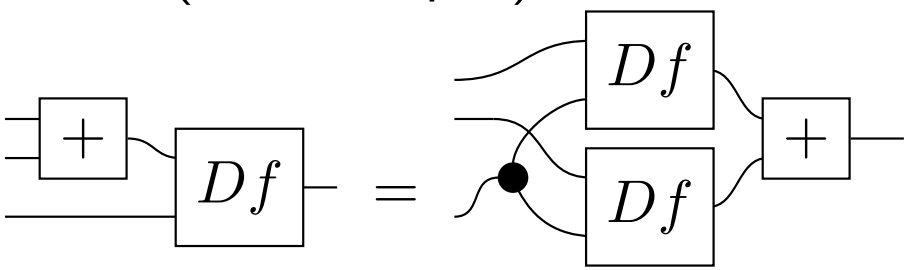
The differential operator on Euclidean spaces and smooth maps sends $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

CD1. $Ds(\Delta x, x) = s(\Delta x)$ for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

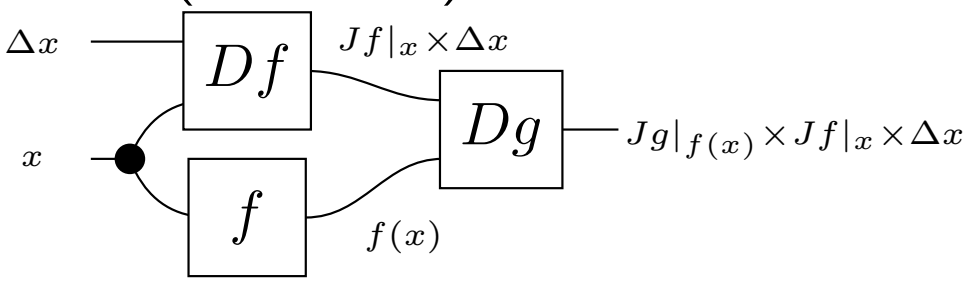
CD2. (linear map I)



CD3. (linear map II)



CD4. (chain rule)

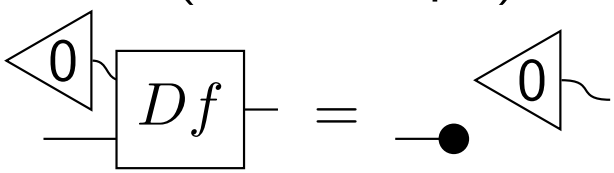


$\mathbf{Euc}_\infty$ is Cartesian differential

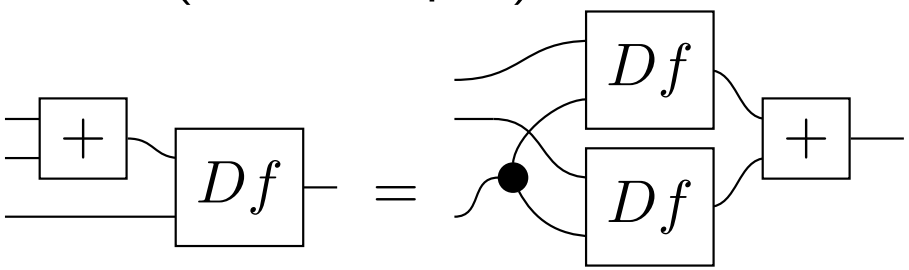
The differential operator on Euclidean spaces and smooth maps sends $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

CD1. $Ds(\Delta x, x) = s(\Delta x)$ for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

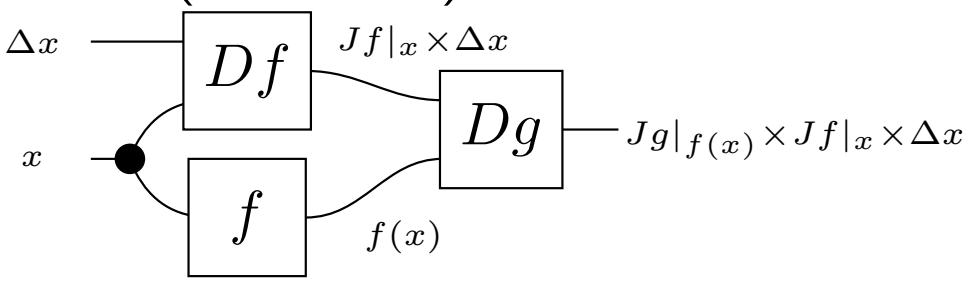
CD2. (linear map I)



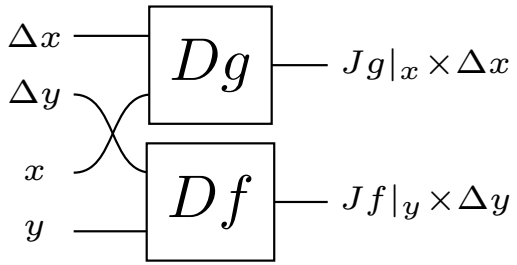
CD3. (linear map II)



CD4. (chain rule)



CD5. (parallel rule)



Second derivative axioms in $\mathbf{Euc}_\infty$

For $f : \mathbb{R} \to \mathbb{R}$, $Df(a, b) = f'(b) \cdot a$.

Second derivative axioms in $\mathbf{Euc}_\infty$

For $f : \mathbb{R} \to \mathbb{R}$, $Df(a, b) = f'(b) \cdot a$.

$$\begin{aligned} DDf(\Delta a, \Delta b, a, b) &= \frac{\partial}{\partial a}(f'(b) \cdot a) \cdot \Delta a + \frac{\partial}{\partial b}(f'(b) \cdot a) \cdot \Delta b \\ &= f'(b) \cdot \Delta a + f''(b) \cdot a \cdot \Delta b \end{aligned}$$

Second derivative axioms in $\mathbf{Euc}_\infty$

For $f : \mathbb{R} \to \mathbb{R}$, $Df(a, b) = f'(b) \cdot a$.

$$\begin{aligned} DDf(\Delta a, \Delta b, a, b) &= \frac{\partial}{\partial a}(f'(b) \cdot a) \cdot \Delta a + \frac{\partial}{\partial b}(f'(b) \cdot a) \cdot \Delta b \\ &= f'(b) \cdot \Delta a + f''(b) \cdot a \cdot \Delta b \end{aligned}$$

Note two things:

6. If $a = \Delta b = 0$, the expression is $f'(b) \cdot \Delta a = Df(\Delta a, b)$.
7. The expression is the same if a and Δb are swapped.

Second derivative axioms in $\mathbf{Euc}_\infty$

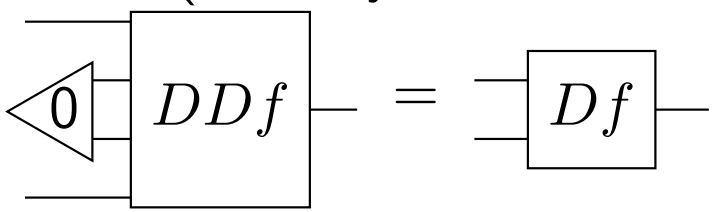
For $f : \mathbb{R} \to \mathbb{R}$, $Df(a, b) = f'(b) \cdot a$.

$$\begin{aligned} DDf(\Delta a, \Delta b, a, b) &= \frac{\partial}{\partial a}(f'(b) \cdot a) \cdot \Delta a + \frac{\partial}{\partial b}(f'(b) \cdot a) \cdot \Delta b \\ &= f'(b) \cdot \Delta a + f''(b) \cdot a \cdot \Delta b \end{aligned}$$

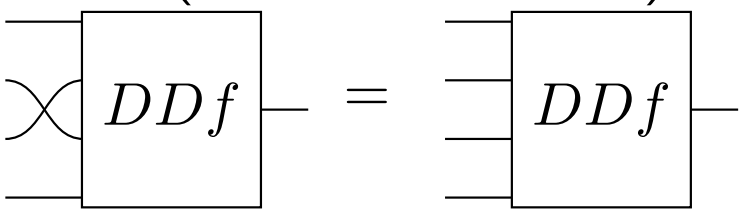
Note two things:

6. If $a = \Delta b = 0$, the expression is $f'(b) \cdot \Delta a = Df(\Delta a, b)$.
7. The expression is the same if a and Δb are swapped.

CD6. (linearity of 2^{nd} deriv.)



CD7. (Schwartz theorem)

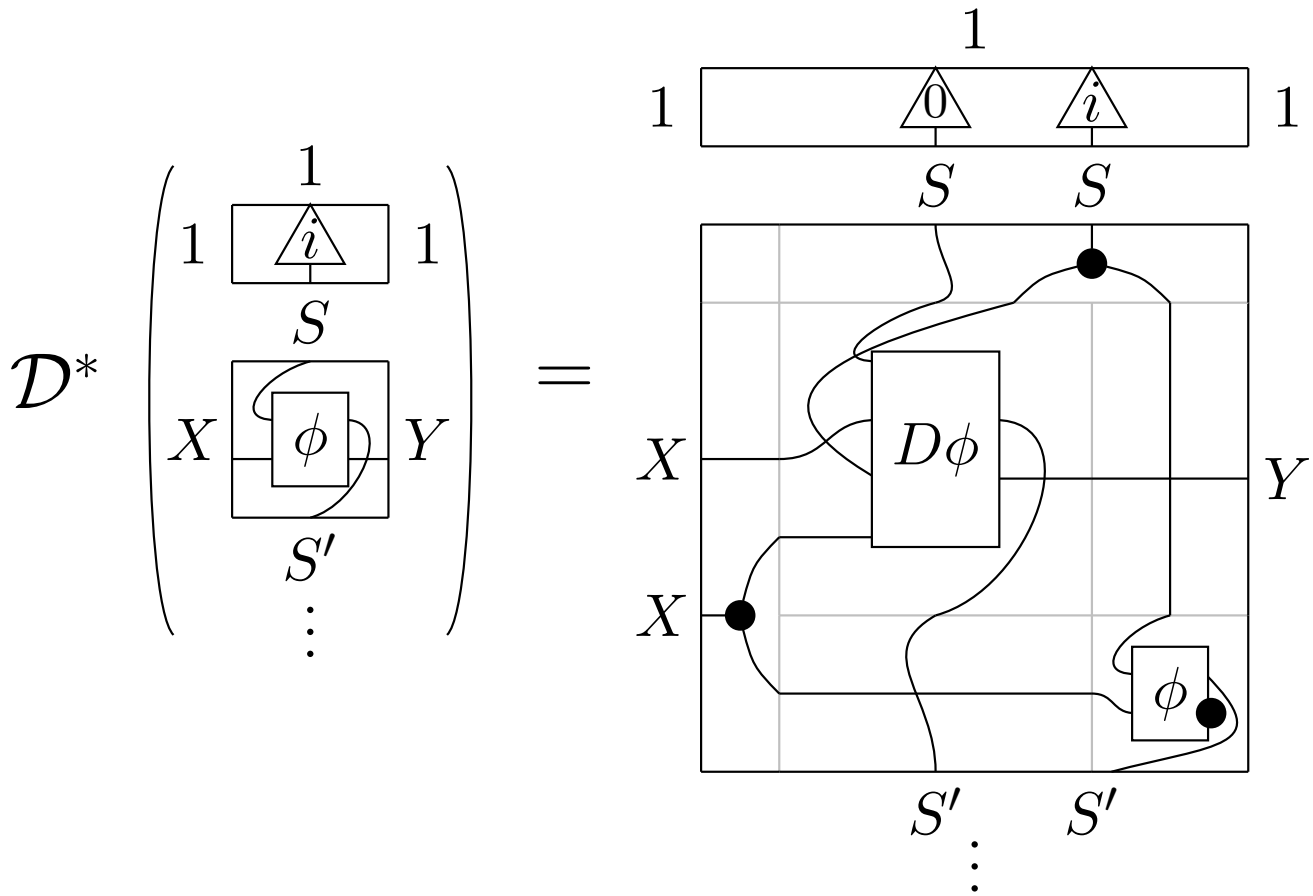


Key contribution: differential operator lifts

Theorem

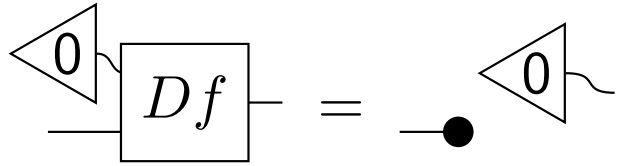
If $\mathbb{C}$ is a Cartesian differential category, so is $\mathrm{St}(\mathbb{C})$.

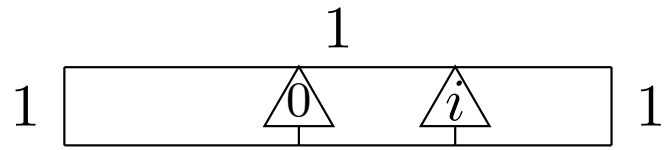
The Cartesian differential operator on $\mathrm{St}(\mathbb{C})$ is defined as follows:

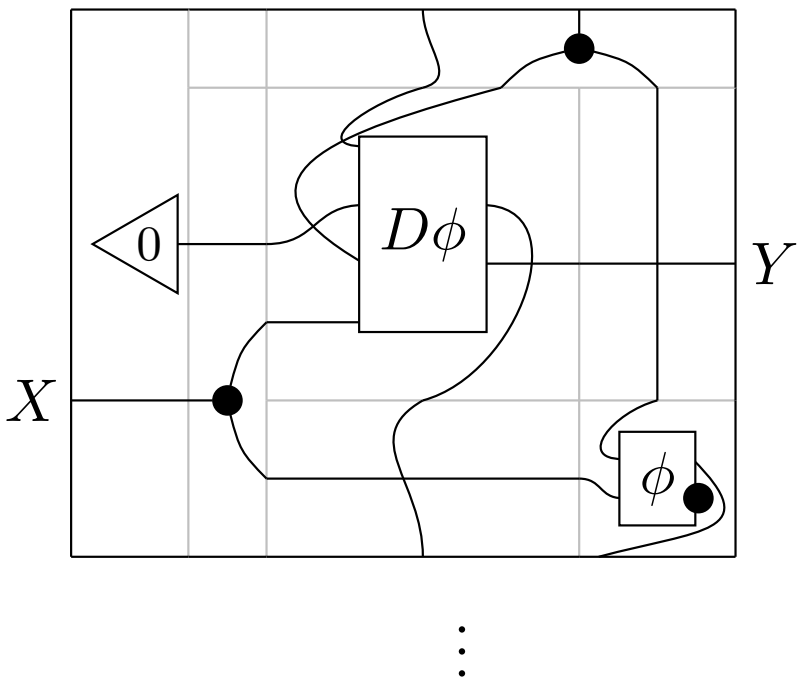


Key contribution: differential operator lifts

Proof idea. For CD2:

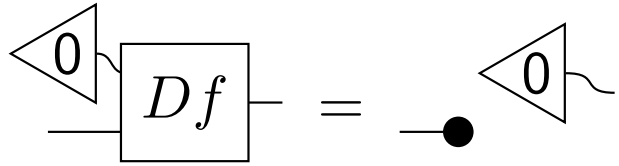


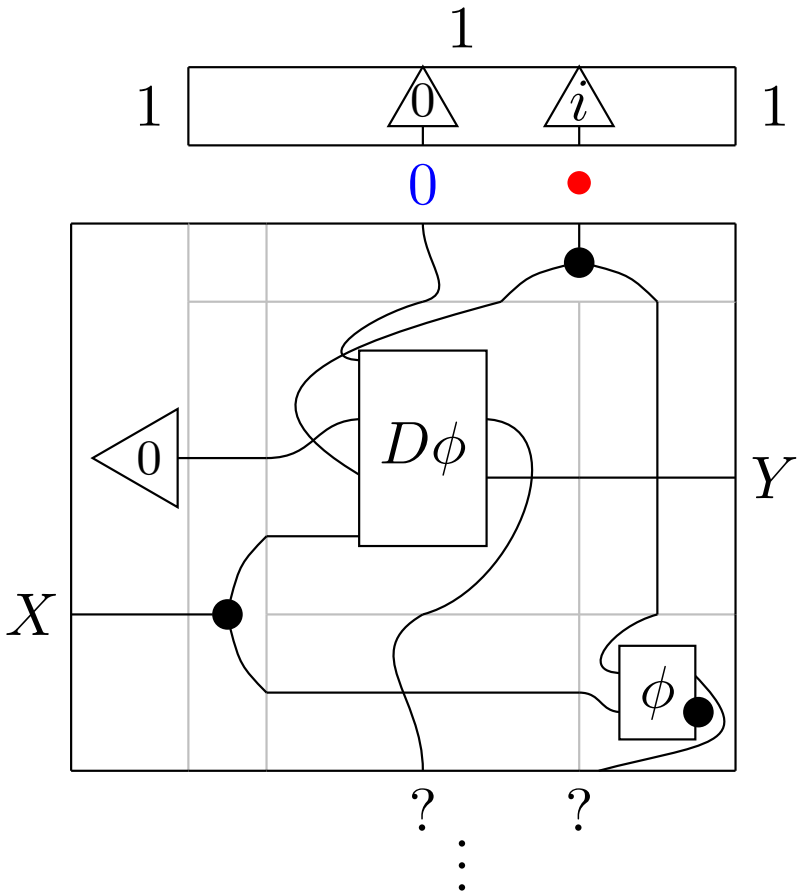




Key contribution: differential operator lifts

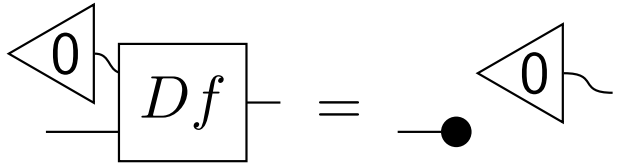
Proof idea. For CD2:

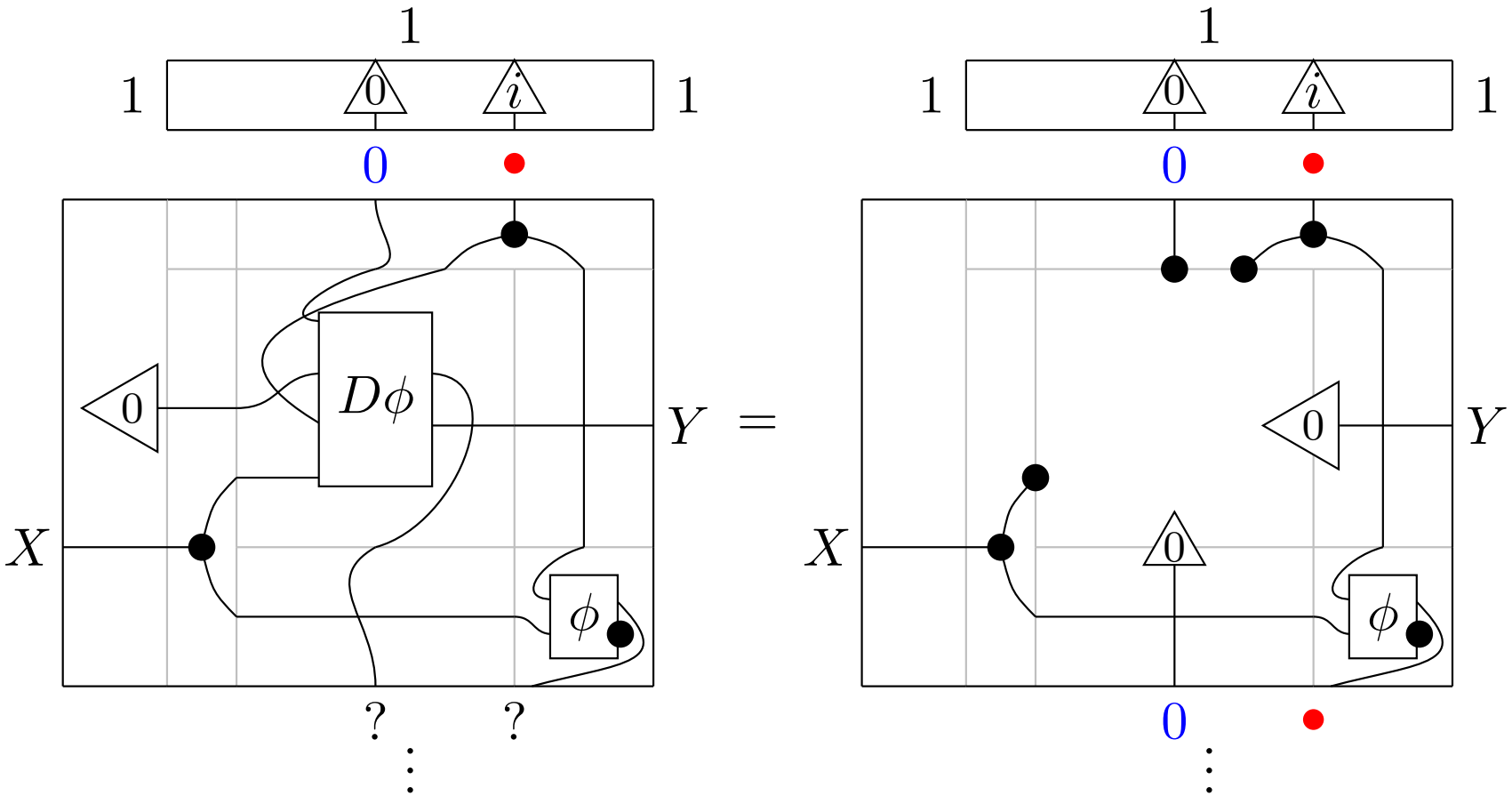




Key contribution: differential operator lifts

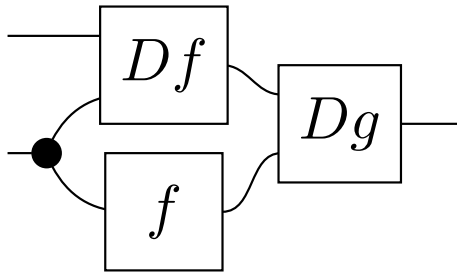
Proof idea. For CD2:

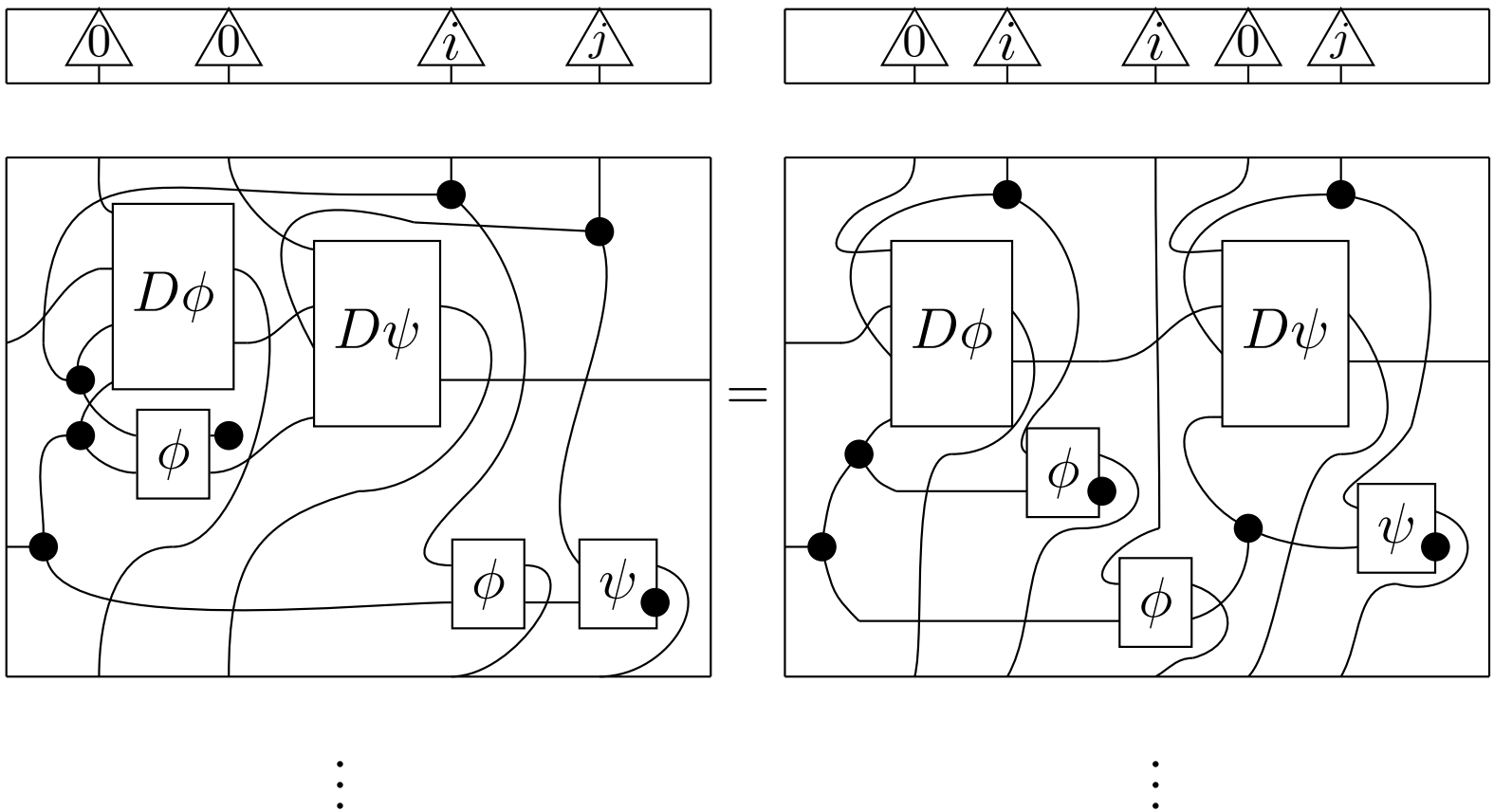




Key contribution: differential operator lifts

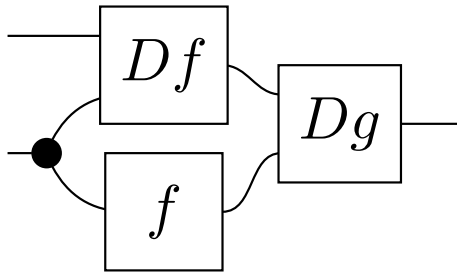
Proof idea. For CD4: $D(g \circ f) =$

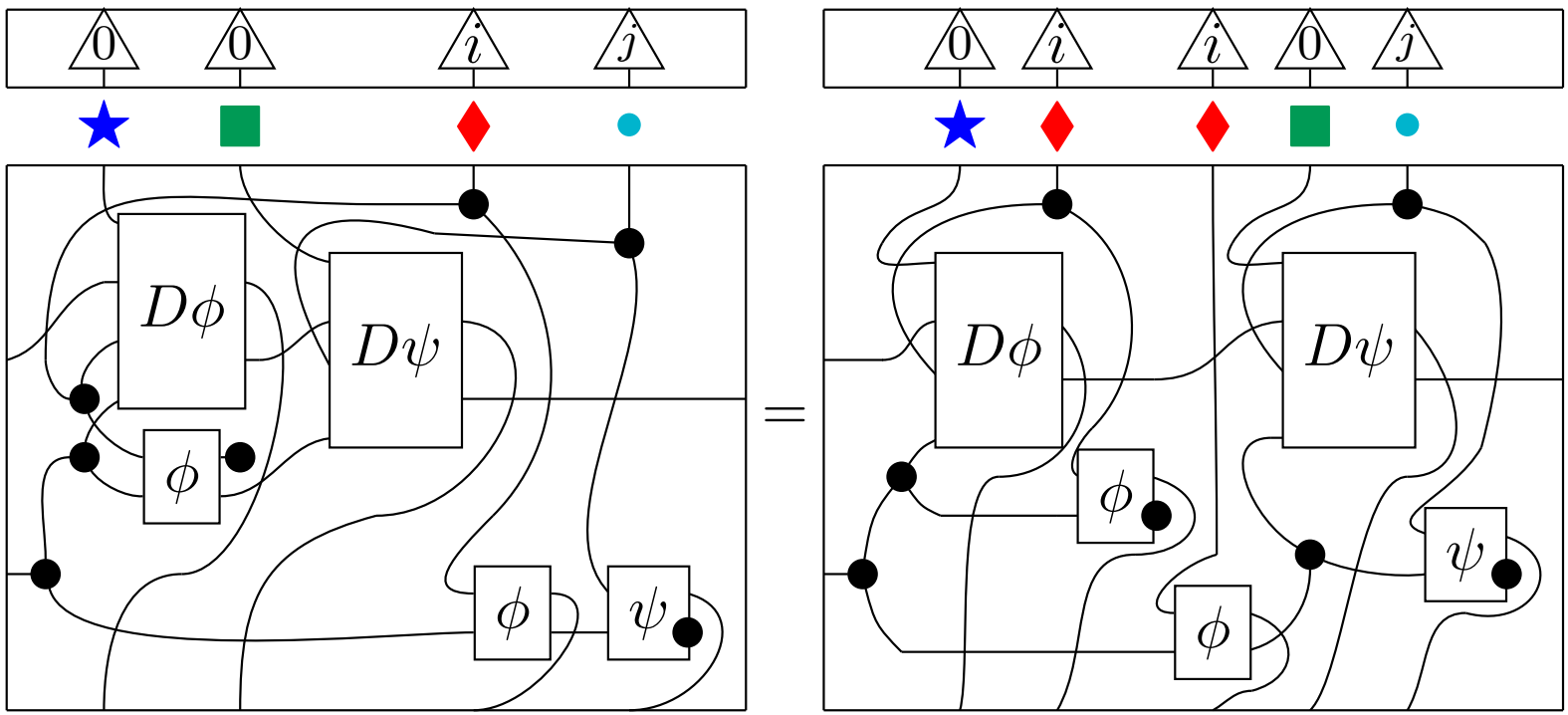




Key contribution: differential operator lifts

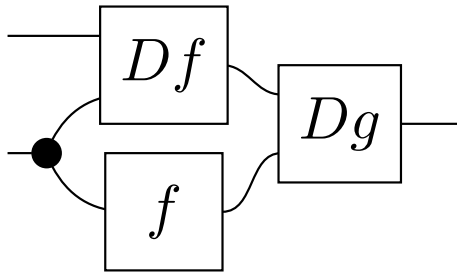
Proof idea. For CD4: $D(g \circ f) =$

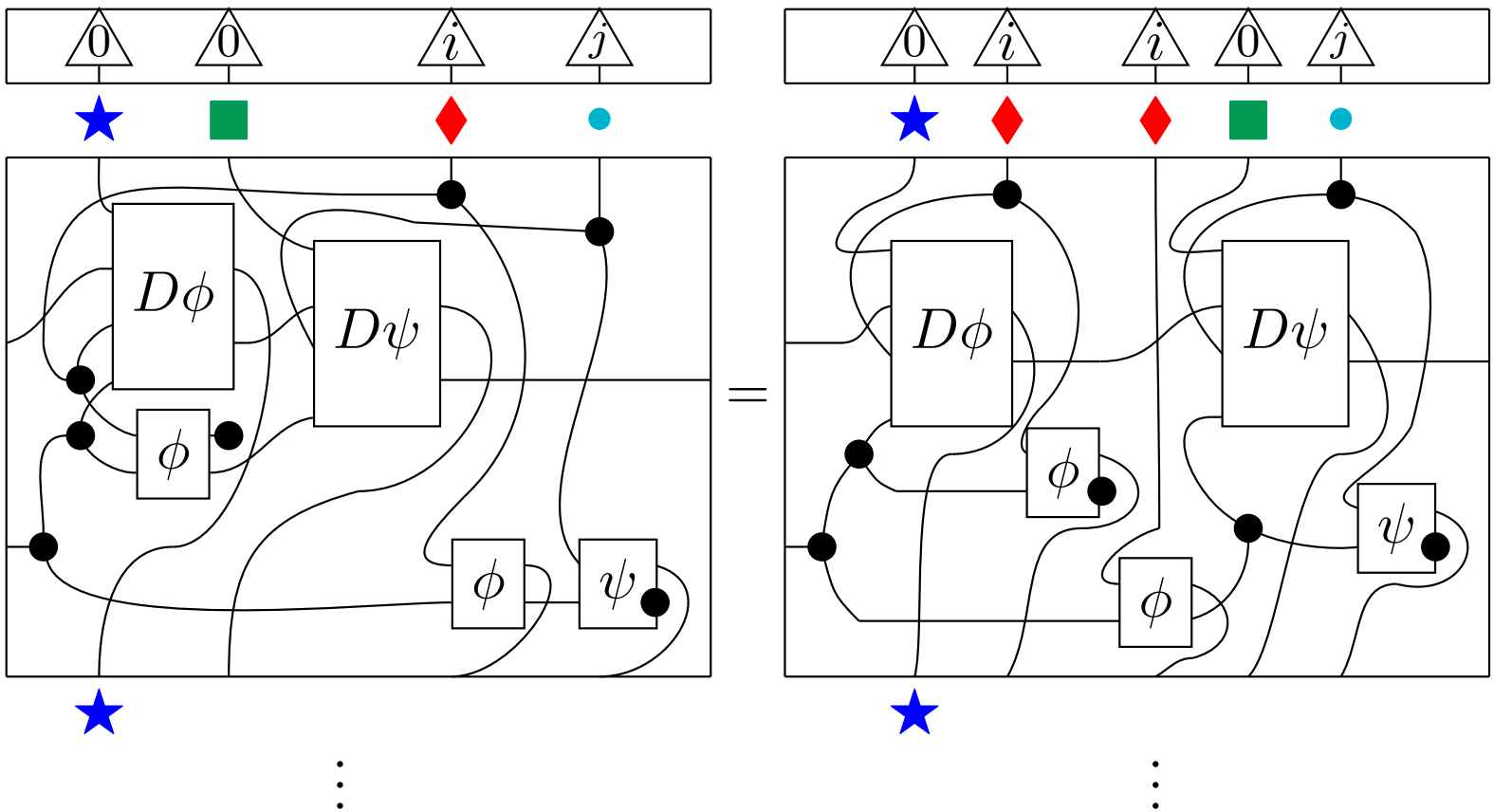




Key contribution: differential operator lifts

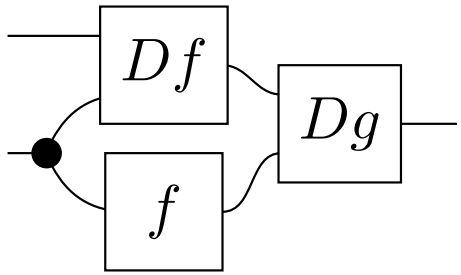
Proof idea. For CD4: $D(g \circ f) =$

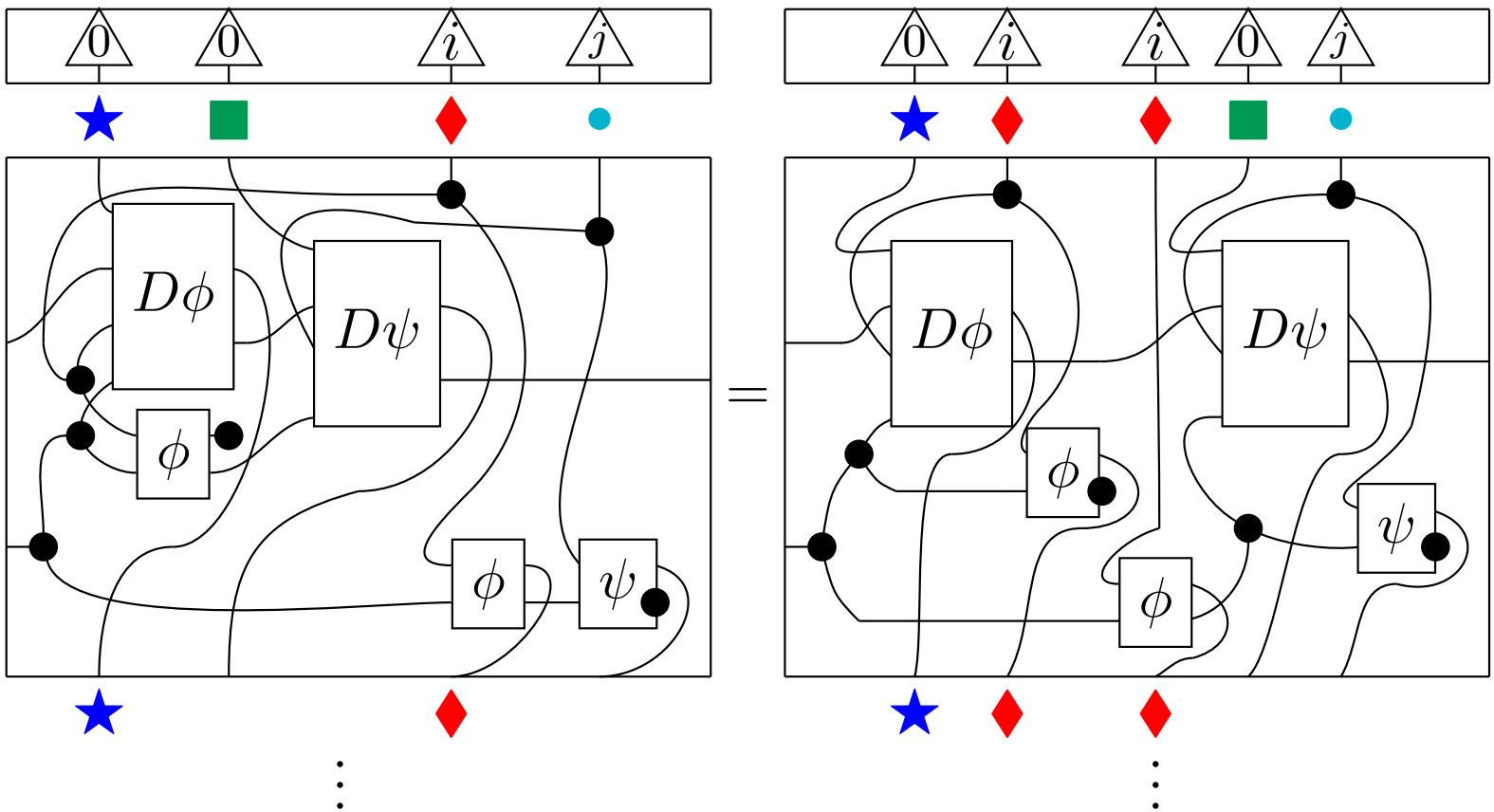




Key contribution: differential operator lifts

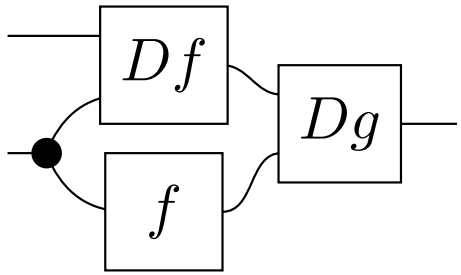
Proof idea. For CD4: $D(g \circ f) =$

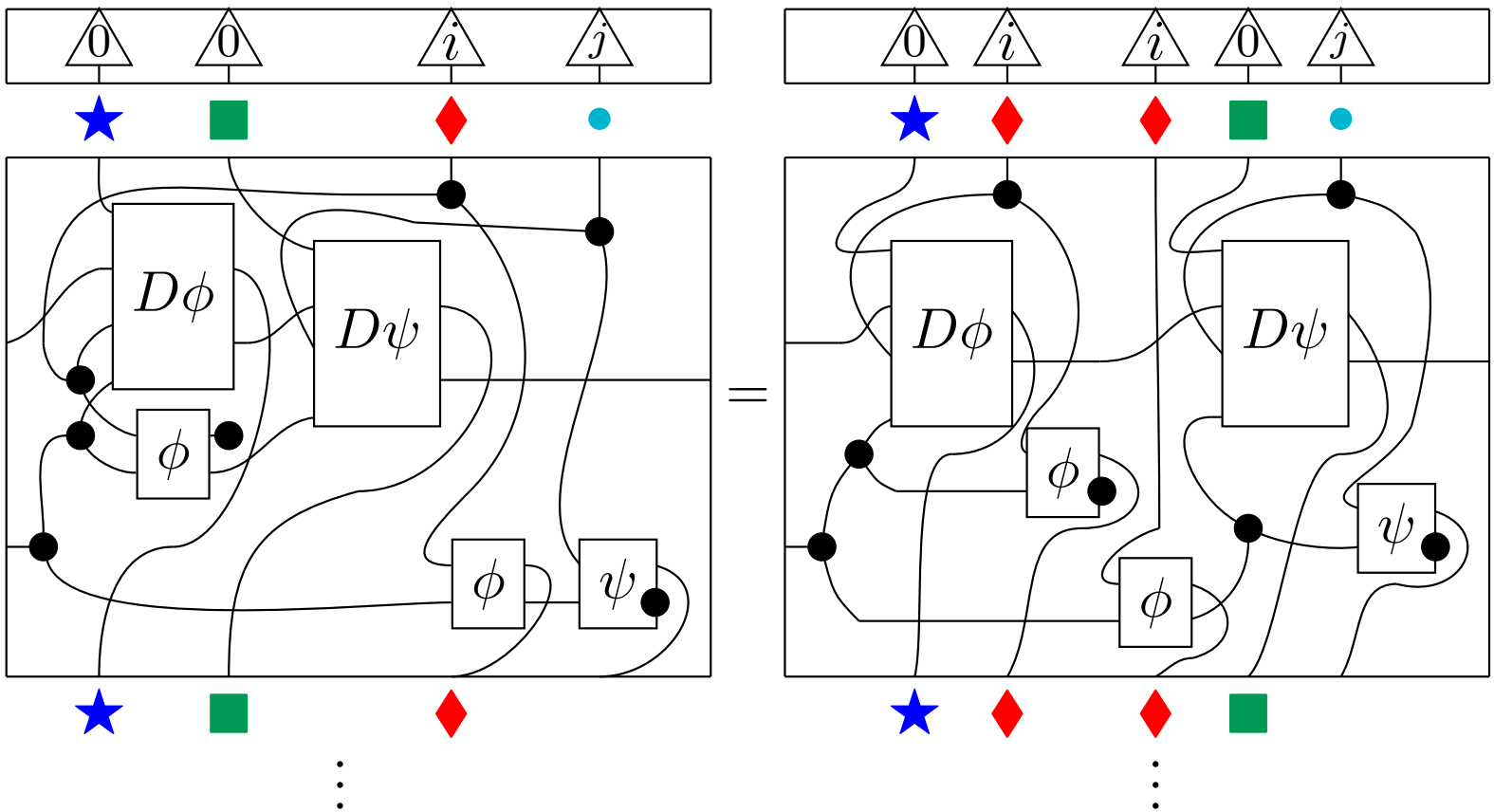




Key contribution: differential operator lifts

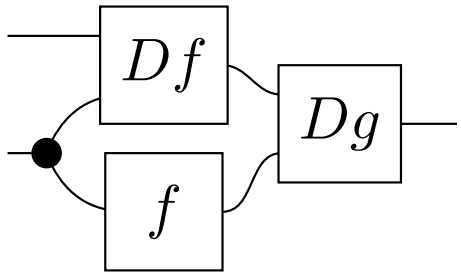
Proof idea. For CD4: $D(g \circ f) =$

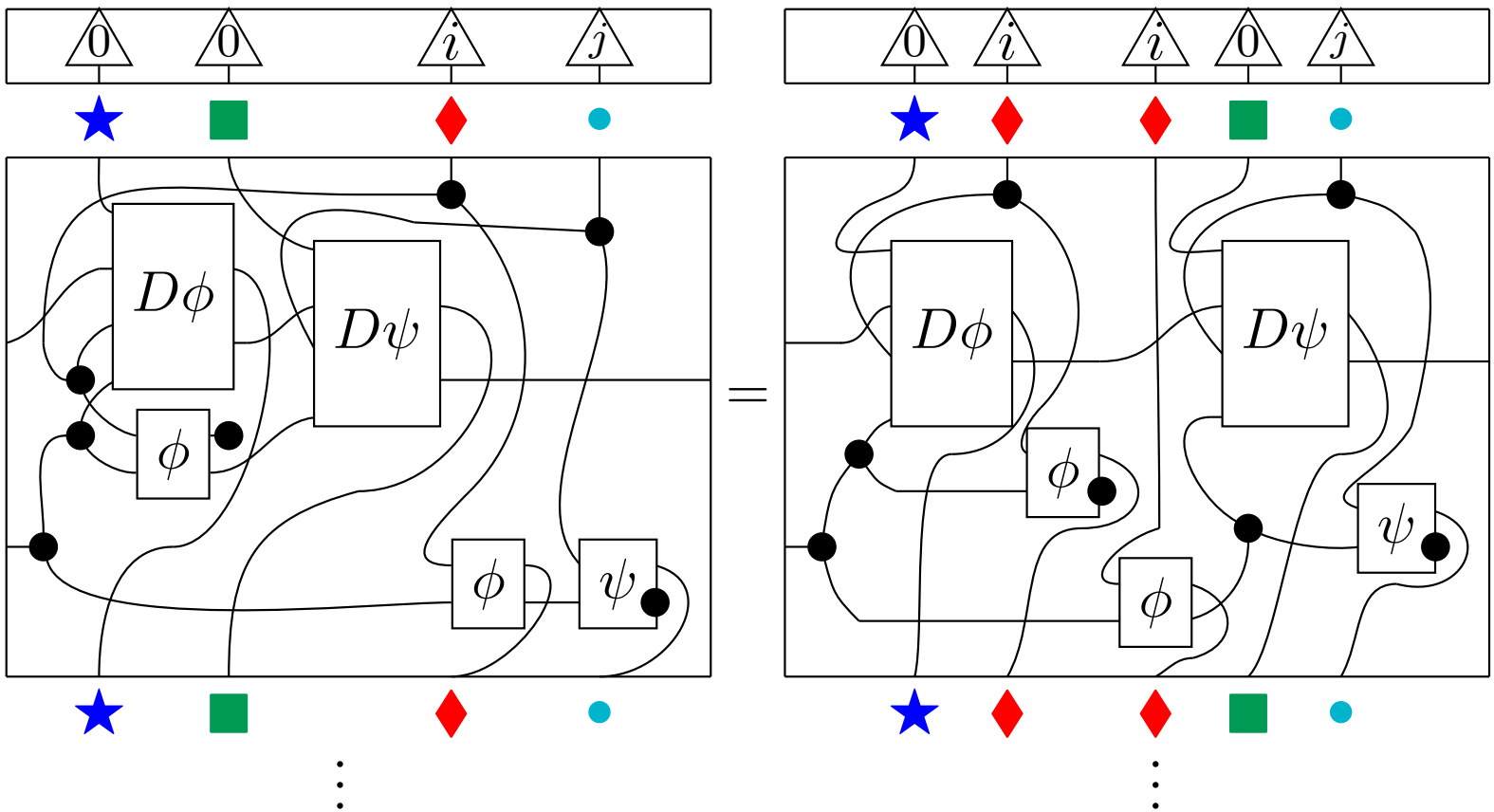




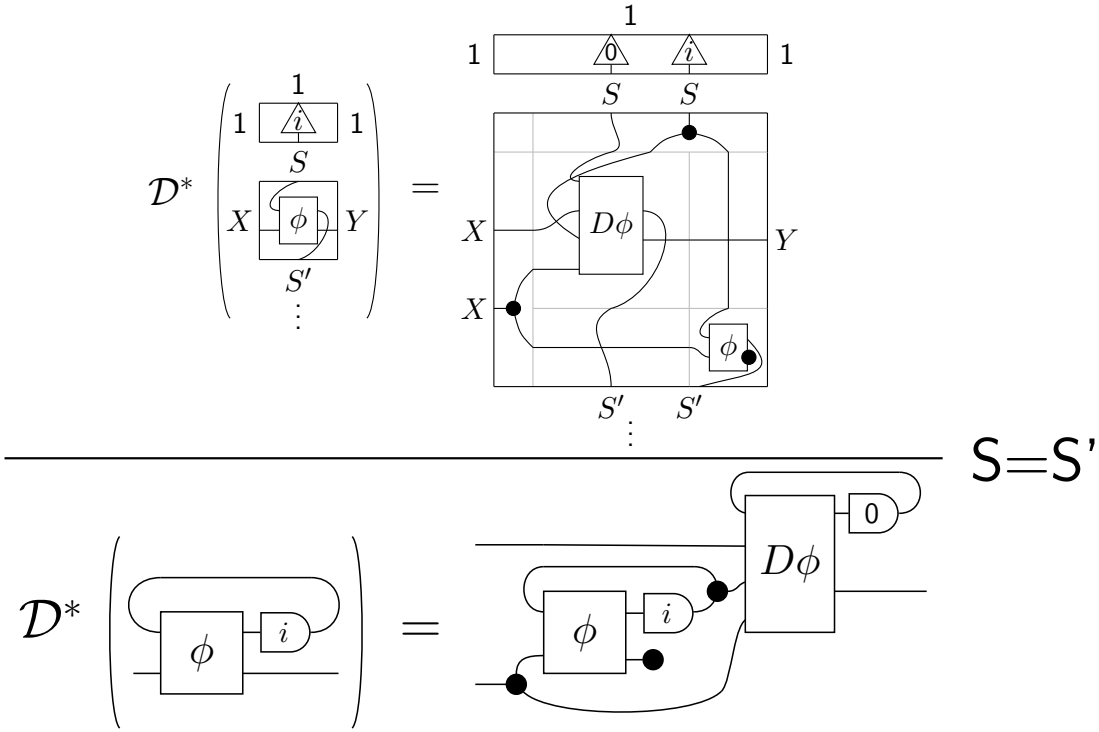
Key contribution: differential operator lifts

Proof idea. For CD4: $D(g \circ f) =$





Differentiating circuits



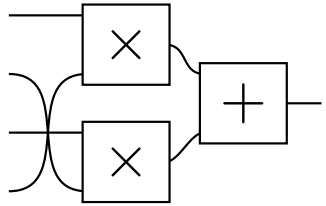
Theorem ($\mathcal{D}^*$ matches BPTT)

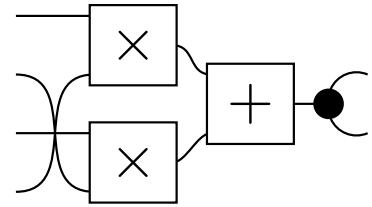
The unrolling of $\mathcal{D}^*((i, [\phi]))$ is the component-wise application of D to the unrolling of $(i, [\phi])$.

What if $\phi = \Delta_{\mathbb{R}} \circ \times$?

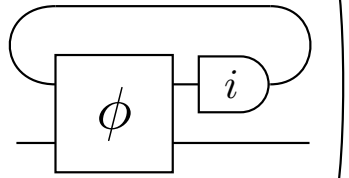
$$D\left(\times\right) = \quad , \text{ so } \quad D\left(\times\right) =$$

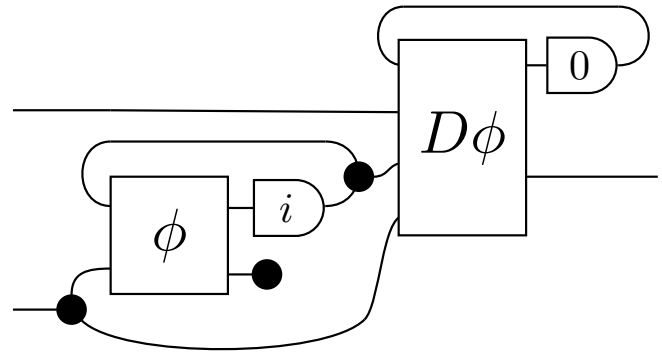
What if $\phi = \Delta_{\mathbb{R}} \circ \times$?

$D\left(\times\right) =$ 

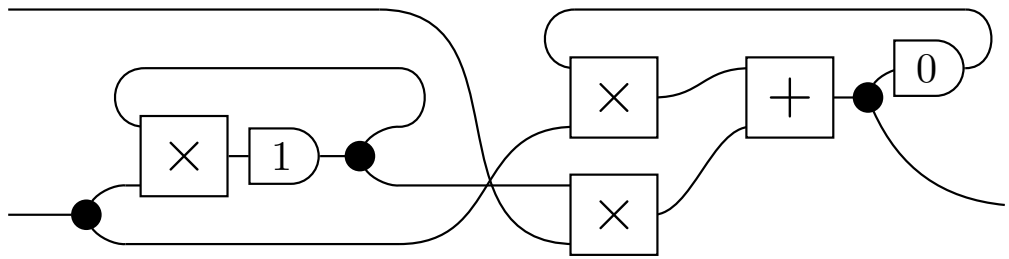
, so $D\left(\times \subset\right) =$ 

Then

$\mathcal{D}^*\left(\right.$ 

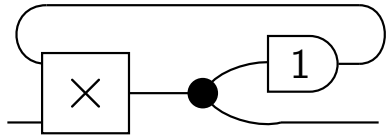
$\left.\right) =$ 

becomes

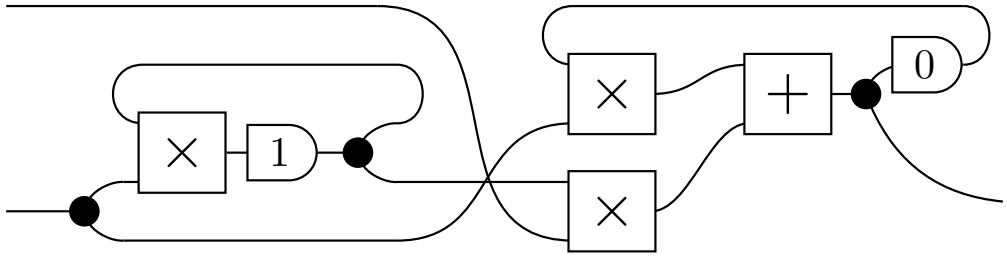


Outline

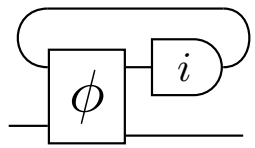
Goals of this talk, again

A. The derivative of 

is this monster



B. There is a general rule for derivatives of stateful functions

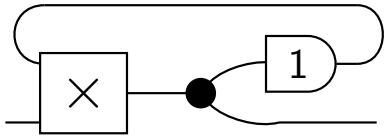


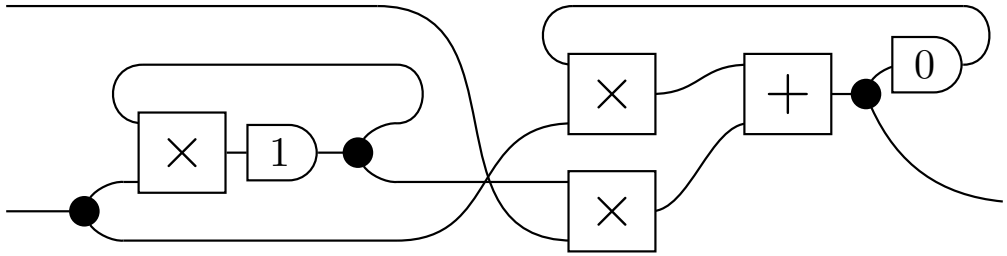
C. We understand many properties of this rule.

⋮

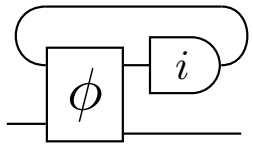
Z. ~~Profit??~~ Get paper??

Goals of this talk, again

A. The derivative of  is this monster



B. There is a general rule for derivatives of stateful functions



C. We understand many properties of this rule.

⋮

Z. ~~Profit??~~ ~~Get paper??~~ Write papers??

Future questions (roughly mathematical to practical)

1. Categorical properties of $\mathrm{St}(-)$?
2. $\mathcal{D}^*$ and derivatives in sequence spaces?
3. Bisimulations and extensional equality?
4. Basic results for delayed trace categories?

Future questions (roughly mathematical to practical)

1. Categorical properties of $\mathrm{St}(-)$?
2. $\mathcal{D}^*$ and derivatives in sequence spaces?
3. Bisimulations and extensional equality?
4. Basic results for delayed trace categories?
5. Axiomatization of stateful function diagrams?
6. * Explicit Jacobians using closed structure?
7. Probabilistic/nondeterministic causal functions?
8. * Partial functions with differential restriction categories?

Future questions (roughly mathematical to practical)

1. Categorical properties of $\mathrm{St}(-)$?
2. $\mathcal{D}^*$ and derivatives in sequence spaces?
3. Bisimulations and extensional equality?
4. Basic results for delayed trace categories?
5. Axiomatization of stateful function diagrams?
6. * Explicit Jacobians using closed structure?
7. Probabilistic/nondeterministic causal functions?
8. * Partial functions with differential restriction categories?
9. Iterations I: RNN parameters in $\mathrm{St}(\mathrm{St}(\mathbb{C}))$?
10. Iterations II: RNN hyperparameters in $\mathrm{St}(\mathrm{St}(\mathrm{St}(\mathbb{C})))$?

Future questions (roughly mathematical to practical)

1. Categorical properties of $\mathrm{St}(-)$?
2. $\mathcal{D}^*$ and derivatives in sequence spaces?
3. Bisimulations and extensional equality?
4. Basic results for delayed trace categories?
5. Axiomatization of stateful function diagrams?
6. * Explicit Jacobians using closed structure?
7. Probabilistic/nondeterministic causal functions?
8. * Partial functions with differential restriction categories?
9. Iterations I: RNN parameters in $\mathrm{St}(\mathrm{St}(\mathbb{C}))$?
10. Iterations II: RNN hyperparameters in $\mathrm{St}(\mathrm{St}(\mathrm{St}(\mathbb{C})))$?
11. * Measuring complexity with string diagrams?
12. What happens when $\phi \in \{LSTM, GRU, \ldots\}$?
13. Implementation/plugin to neural net library?

Thanks!