Recurrent Neural Networks: A Categorical Perspective

Shin-ya Katsumata and David Sprunger*
National Institute of Informatics, Tokyo

University of Calgary
10 September 2019

Outline

Motivations (general to specific)

- Re-examining the foundations of machine/deep learning

Motivations (general to specific)

- Re-examining the foundations of machine/deep learning
- Understanding recurrent neural networks and their training

Motivations (general to specific)

- Re-examining the foundations of machine/deep learning
- Understanding recurrent neural networks and their training
- Striking up collaboration

Machine learning, as a field

- Tremendous business value
  - Netflix's 2009 Kaggle competition for $1M
  - McKinsey Global expects to add $3.5T annually by 2020

Machine learning, as a field

- Tremendous business value
  - Netflix's 2009 Kaggle competition for $1M
  - McKinsey Global expects to add $3.5T annually by 2020
- Tremendous academic volume
  - NeurIPS 2019 has ∼1400 accepted papers
  - ICML 2019 has ∼800 accepted papers
  - Ilya Sutskever (PhD 2012) has ∼123,000 citations

Machine learning, as a field

- Tremendous business value
  - Netflix's 2009 Kaggle competition for $1M
  - McKinsey Global expects to add $3.5T annually by 2020
- Tremendous academic volume
  - NeurIPS 2019 has ∼1400 accepted papers
  - ICML 2019 has ∼800 accepted papers
  - Ilya Sutskever (PhD 2012) has ∼123,000 citations
- My opinion
  - Opportunities to improve understanding of mathematical foundations are plentiful

Machine learning, as a field

- Tremendous business value
  - Netflix's 2009 Kaggle competition for $1M
  - McKinsey Global expects to add $3.5T annually by 2020
- Tremendous academic volume
  - NeurIPS 2019 has $\sim$1400 accepted papers
  - ICML 2019 has $\sim$800 accepted papers
  - Ilya Sutskever (PhD 2012) has $\sim$123,000 citations
- My opinion
  - Opportunities to improve understanding of mathematical foundations are plentiful
  - Experimental results can be better tied to theory

Examples of confusion

A recent answer on the data science StackExchange: "gradient descent . . . relies on local information only. . . .

So if the gradient magnitude is high, you are fairly certain that you are far from a good solution . . . and if the magnitude is small, then you are closer to a good solution . . . "

Examples of confusion

A recent answer on the data science StackExchange: "gradient descent . . . relies on local information only. . . .

So if the gradient magnitude is high, you are fairly certain that you are far from a good solution . . . and if the magnitude is small, then you are closer to a good solution . . ."

Some common ideas about LSTMs: "Remembering information for long periods of time is practically their default behavior!" (Chris Olah's introduction)

This makes training difficult. It is common practice to create an "unrolled" version of the network; this finite approximation of the RNN is then trained. (Tensorflow documentation)

Research question

Is there a robust, theoretical justification for the adaptation of training methods for stateless (functional) networks to stateful (programmatic) networks?

Research question

Is there a robust, theoretical justification for the adaptation of training methods for stateless (functional) networks to stateful (programmatic) networks?

Main challenge: incorporating both statefulness and differentiation in a single framework.

Research question

Is there a robust, theoretical justification for the adaptation of training methods for stateless (functional) networks to stateful (programmatic) networks?

Main challenge: incorporating both statefulness and differentiation in a single framework.

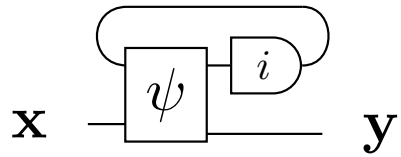
This talk: mostly on the treatment of state and the techniques we use (!)

Outline

Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

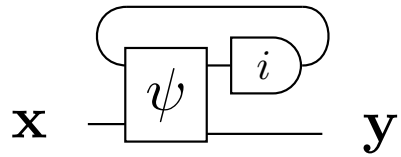
$\mathbf{\Psi}$:



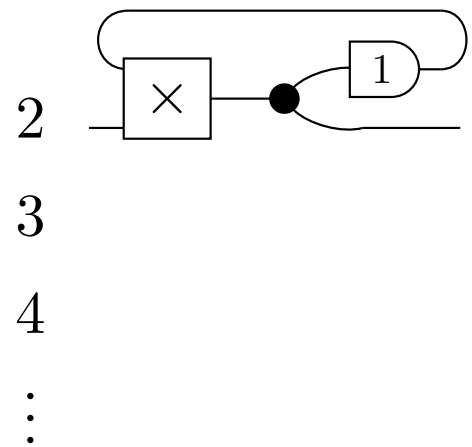
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



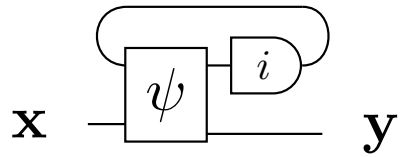
Operationally, these work as you expect:



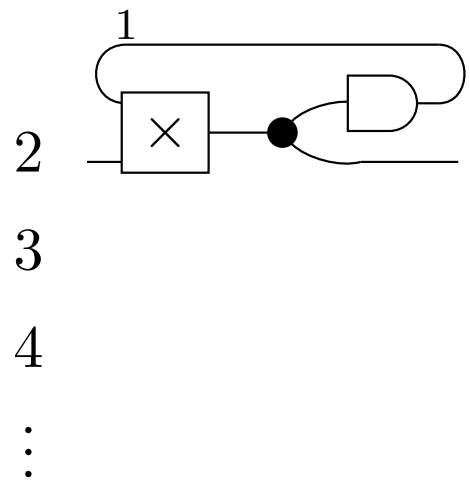
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



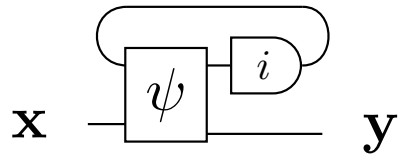
Operationally, these work as you expect:



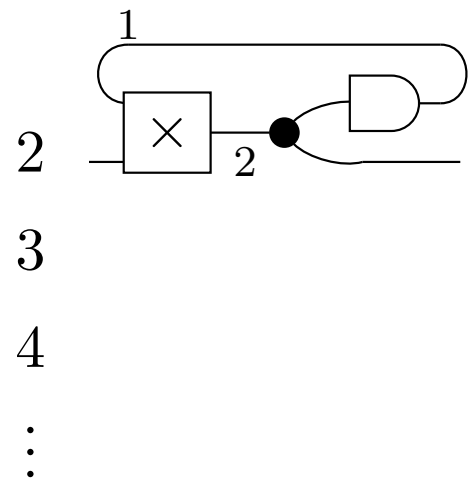
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



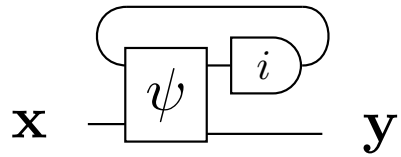
Operationally, these work as you expect:



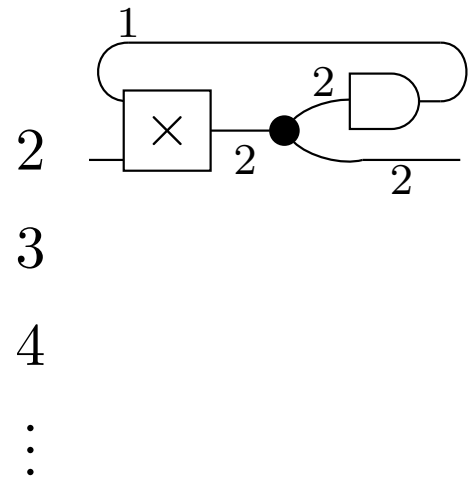
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



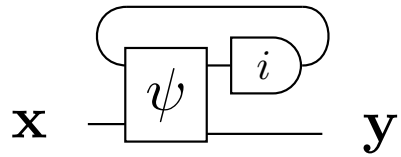
Operationally, these work as you expect:



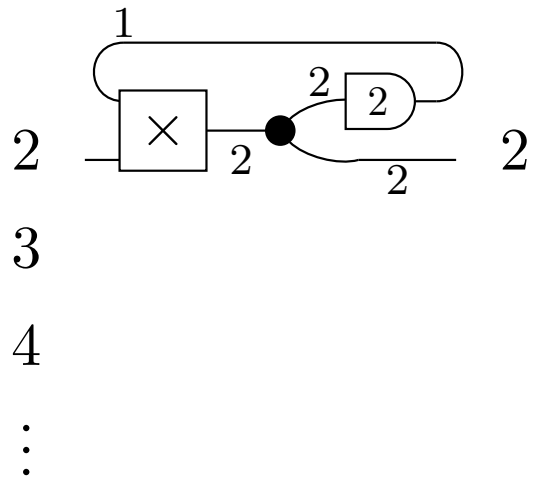
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



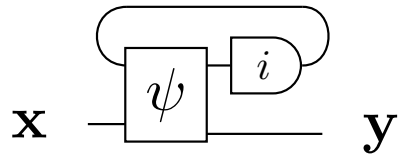
Operationally, these work as you expect:



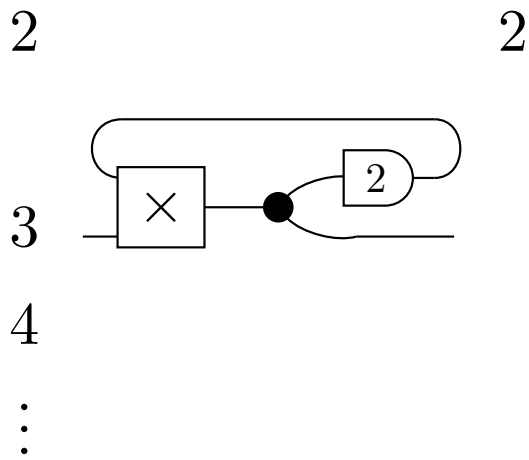
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



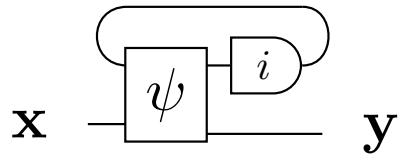
Operationally, these work as you expect:



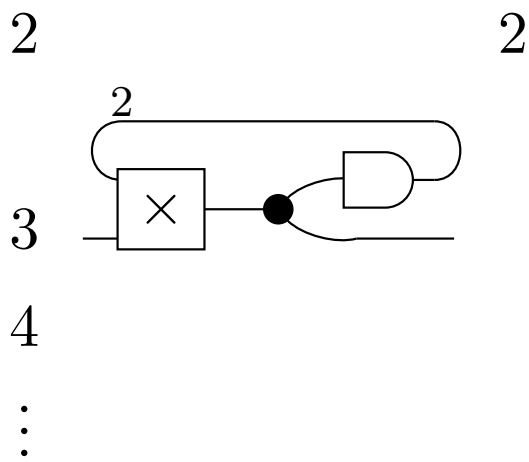
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



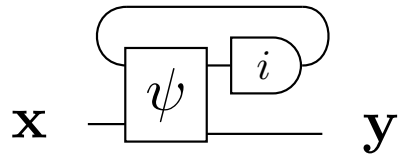
Operationally, these work as you expect:



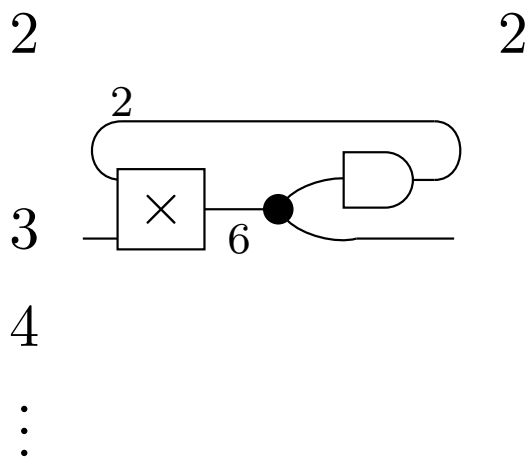
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



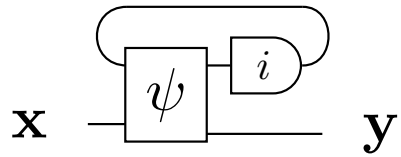
Operationally, these work as you expect:



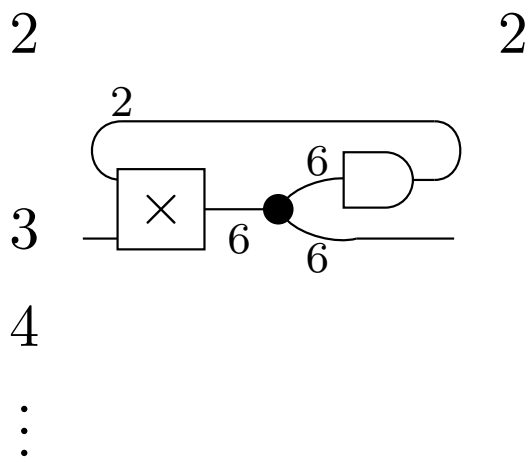
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



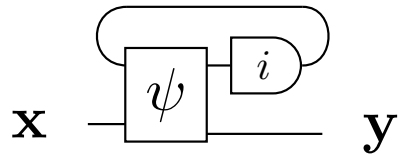
Operationally, these work as you expect:



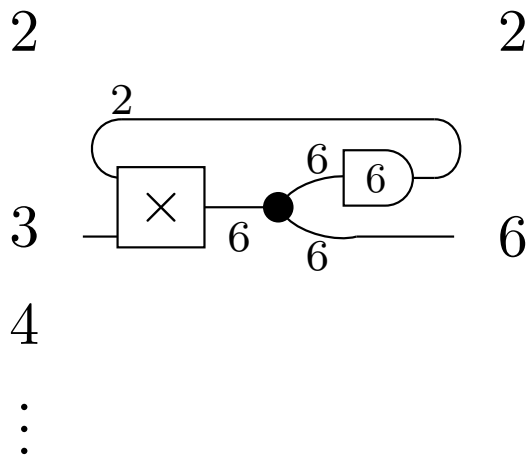
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:



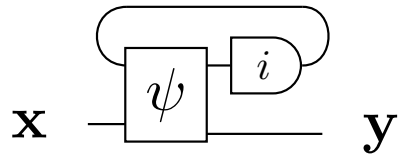
Operationally, these work as you expect:



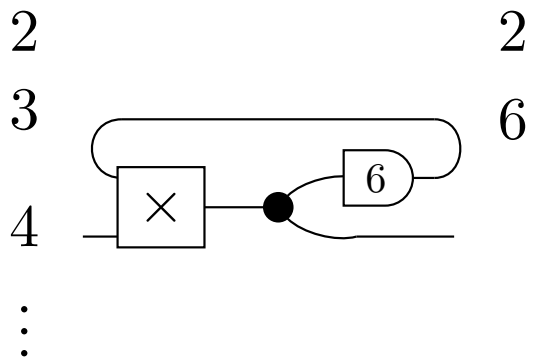
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$:

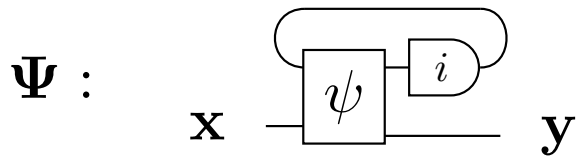


Operationally, these work as you expect:

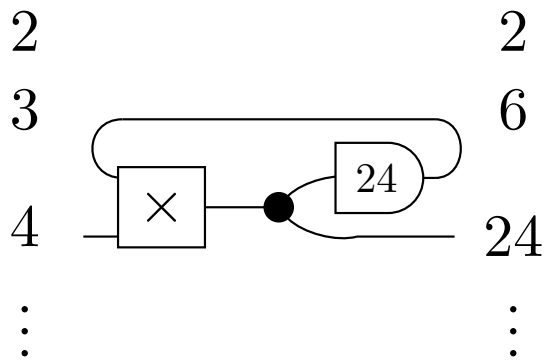


Recurrent neural networks

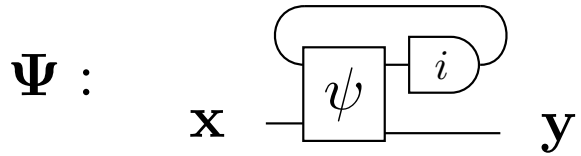
Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:



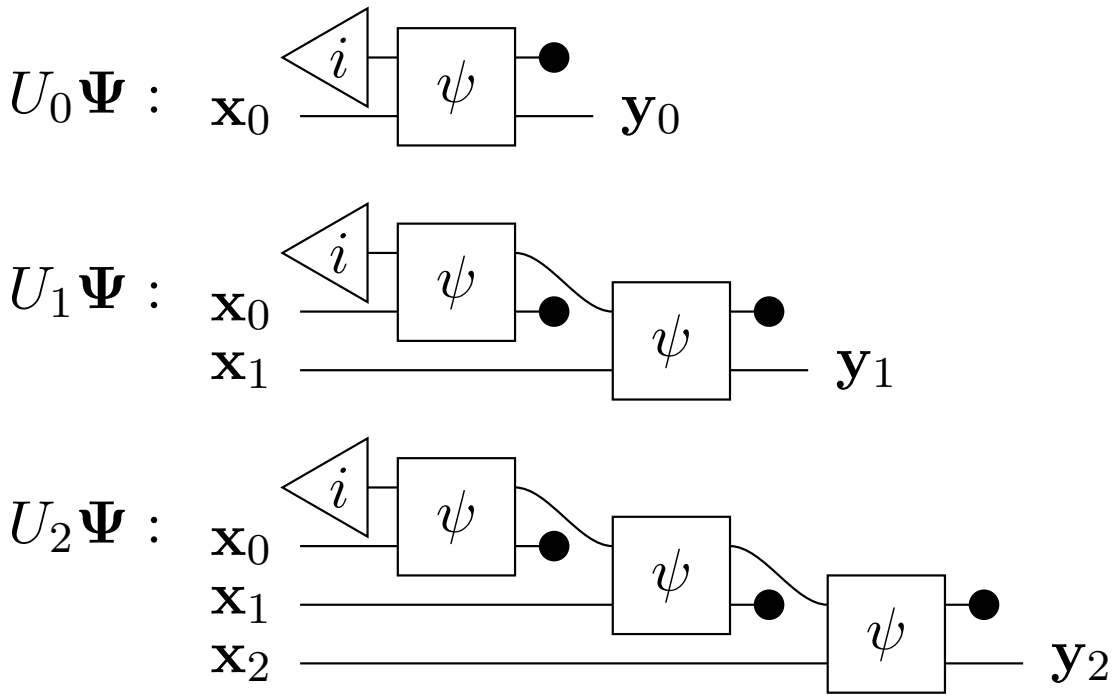
Operationally, these work as you expect:



Unrolling semantics

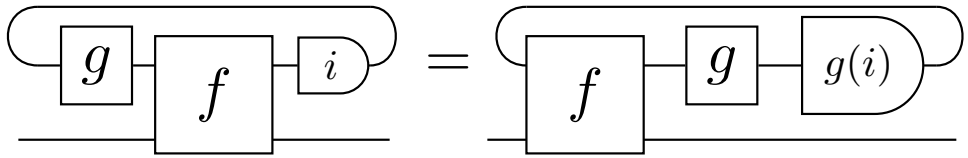


A common semantics of RNNs uses the *unrollings* of the network:



A (desired) consequence of these semantics

From the unrolling semantics, we can infer that these two RNNs should be equal:

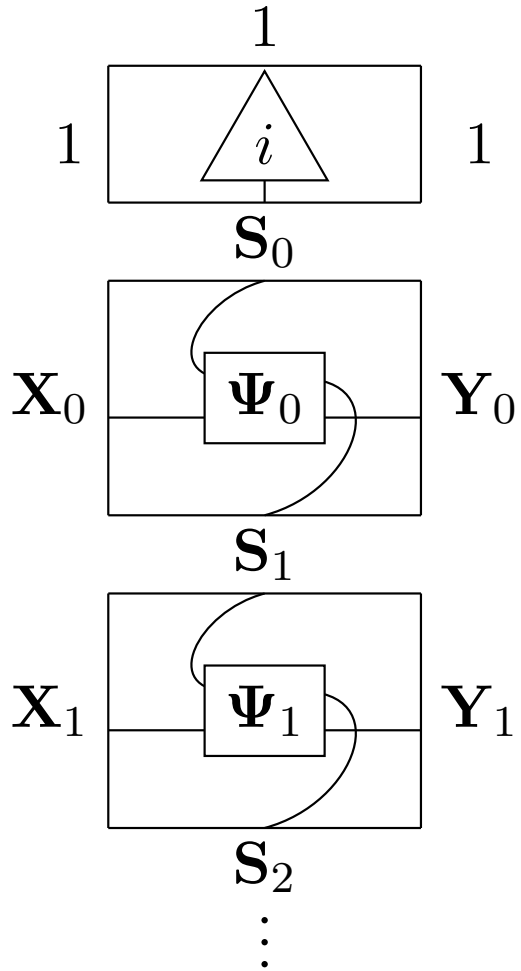


Outline

Stateful **computations**

Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category, whose morphisms we think of as stateless functions. A stateful sequence computation looks like this:

$$\mathbf{\Psi}_i \in \mathbb{C}(\mathbf{S}_i \times \mathbf{X}_i, \mathbf{S}_{i+1} \times \mathbf{Y}_i)$$

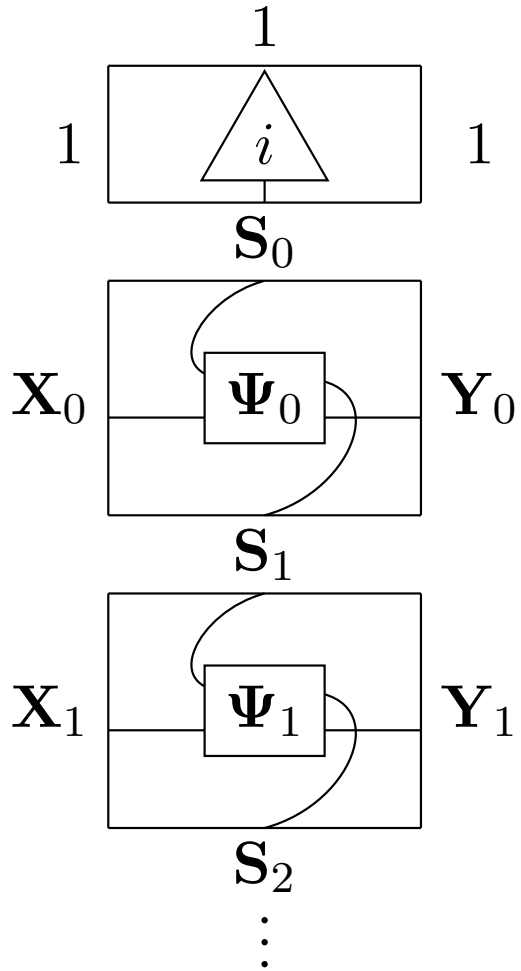


Stateful **computations**

Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category, whose morphisms we think of as stateless functions. A stateful sequence computation looks like this:

$$\mathbf{\Psi}_i \in \mathbb{C}(\mathbf{S}_i \times \mathbf{X}_i, \mathbf{S}_{i+1} \times \mathbf{Y}_i)$$

This is a sequence of 2-cells in a double category based on $\mathbb{C}$, with a restriction on the first 2-cell.

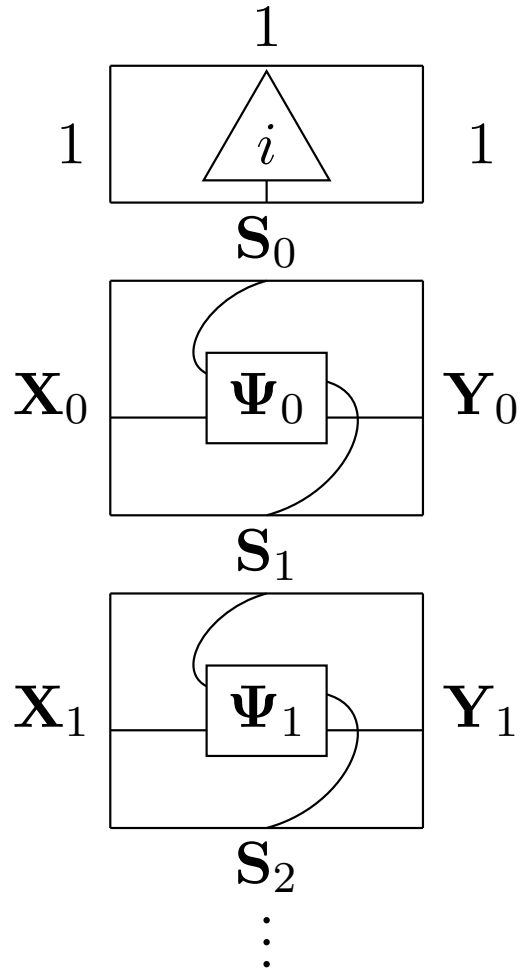


Stateful **computations**

Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category, whose morphisms we think of as stateless functions. A stateful sequence computation looks like this:

$$\mathbf{\Psi}_i \in \mathbb{C}(\mathbf{S}_i \times \mathbf{X}_i, \mathbf{S}_{i+1} \times \mathbf{Y}_i)$$

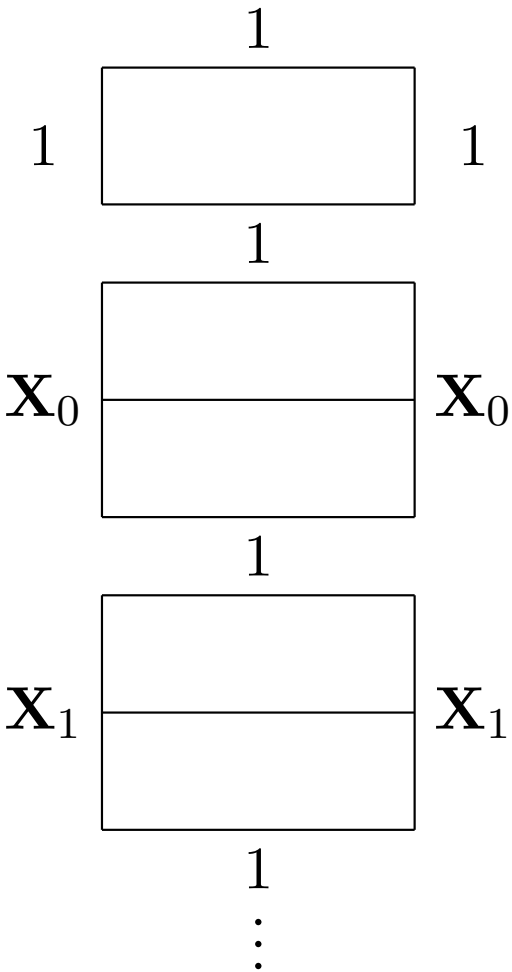
This is a sequence of 2-cells in a double category based on $\mathbb{C}$, with a restriction on the first 2-cell.

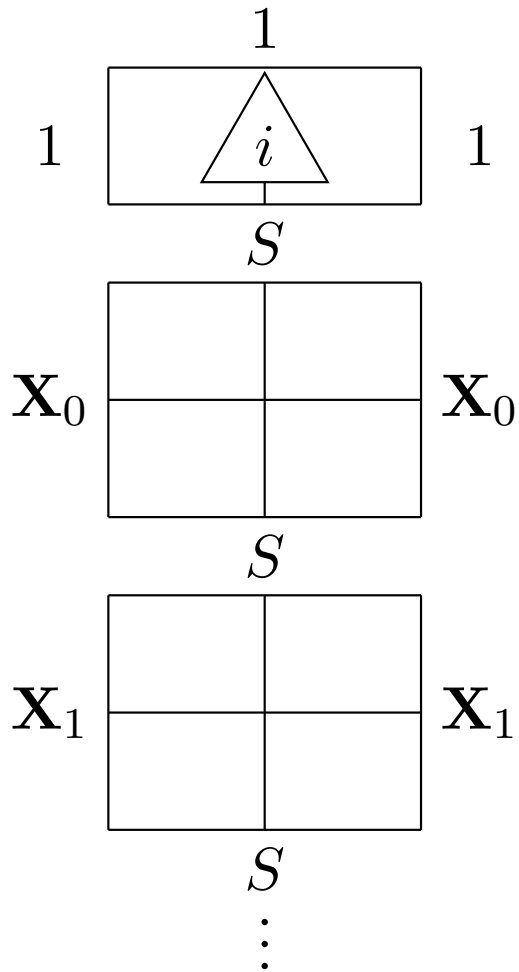


At time i, the computation executes function $\mathbf{\Psi}_i$, which takes a *value* of type $\mathbf{X}_i$ from the environment and a *state* of type $\mathbf{S}_i$ prepared by the previous step and returns a value of type $\mathbf{Y}_i$ and a state of type $\mathbf{S}_{i+1}$.

Stateful **functions**

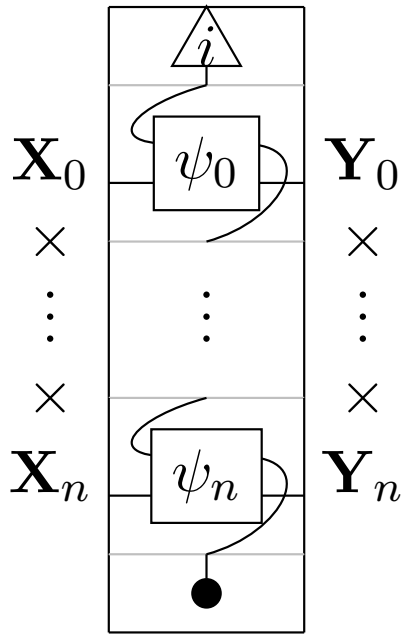
Two computation sequences might have different state spaces and still compute the same function. For example:





Stateful **functions**

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



Definition

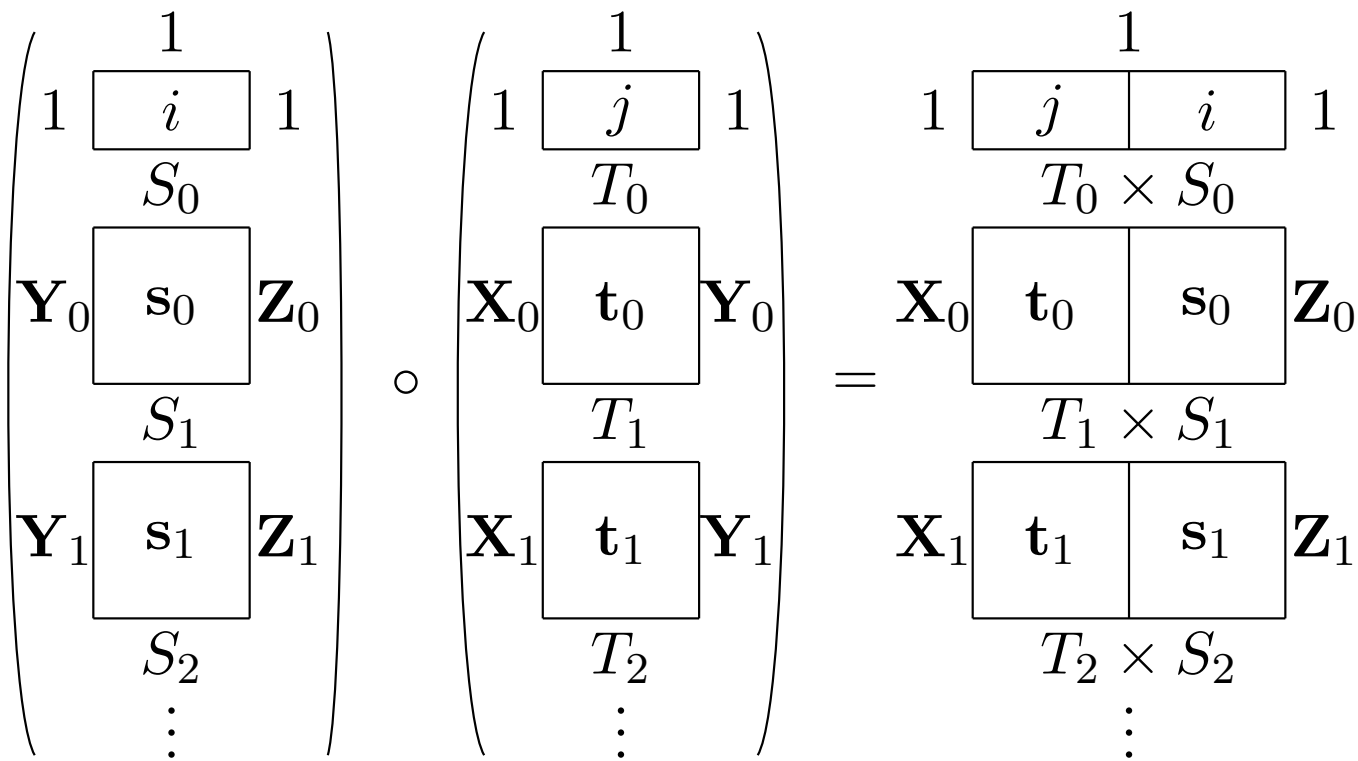
Two computation sequences are *extensionally equivalent* means they have the same nth truncation for all $n \in \mathbb{N}$. A *stateful (sequence) function* is an extensional equivalence class of computation sequences.

Stateful **functions**

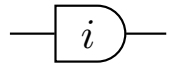
Definition

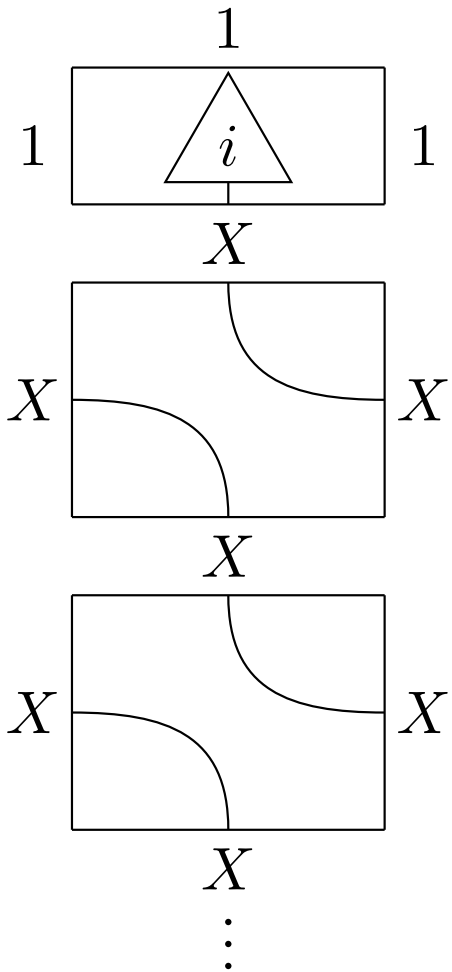
If $\mathbb{C}$ is a strict Cartesian category, then its *stateful sequence extension* is a category $\mathrm{St}(\mathbb{C})$ where

- objects are infinite sequences of objects in $\mathbb{C}$ and
- morphisms are stateful functions $\mathbf{\Psi} : \mathbf{X} \to \mathbf{Y}$.

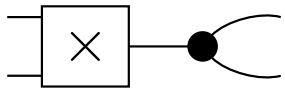


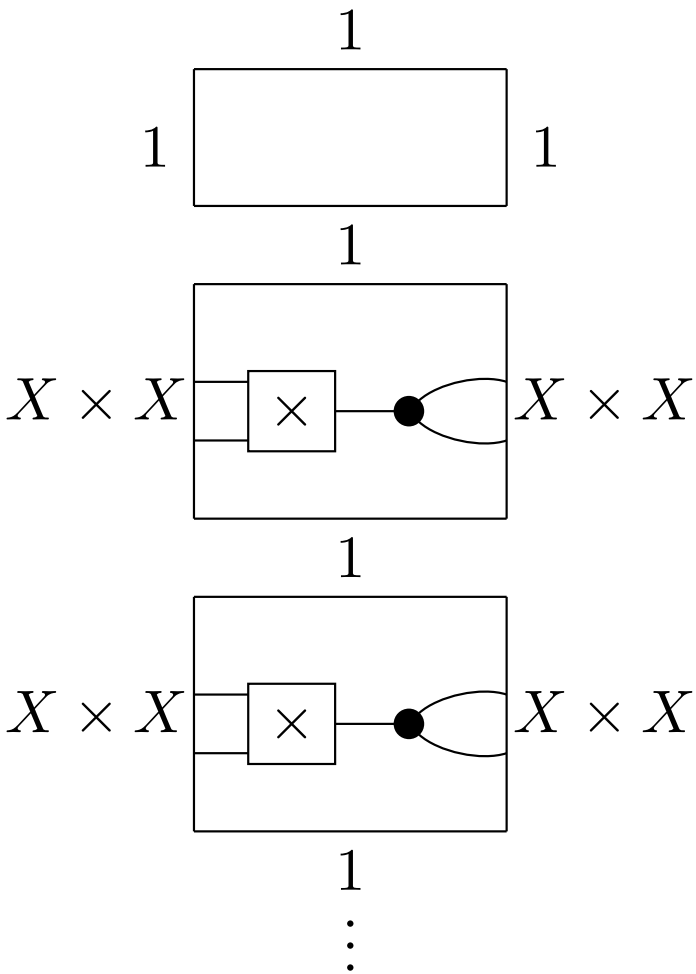
Example computation sequences

Here is  as a computation sequence:

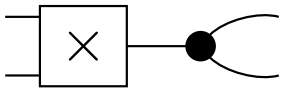


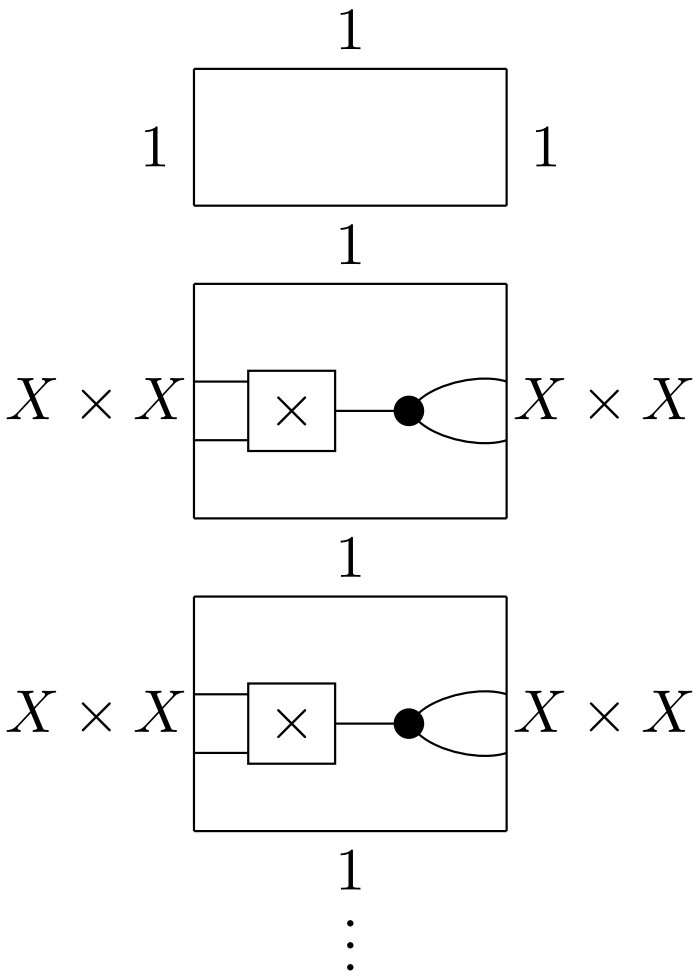
Example computation sequences

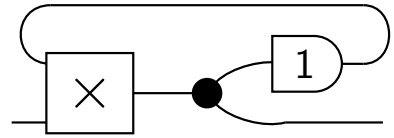
Here is  as a computation sequence:

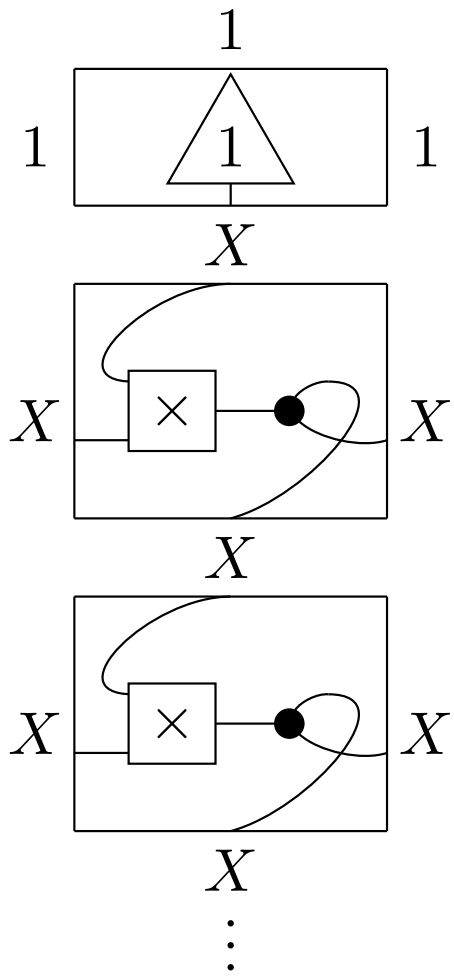


Example computation sequences

Here is  as a computation sequence:



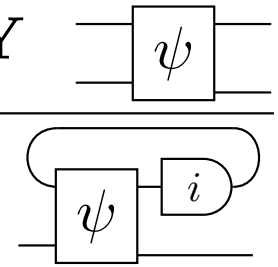
Here is  as a computation sequence:



Delayed trace

This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*. It satisfies several properties of an ordinary trace:

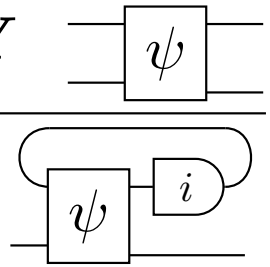
$$\frac{\psi : \mathbf{S} \times \mathbf{X} \to \mathbf{S} \times \mathbf{Y}}{dtr_i^{\mathbf{S}}(\psi) : \mathbf{X} \to \mathbf{Y}}$$



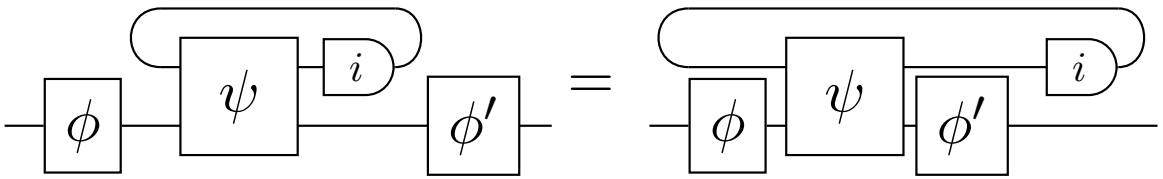
Delayed trace

This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*. It satisfies several properties of an ordinary trace:

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



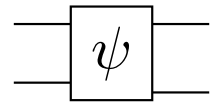
S/T naturality:



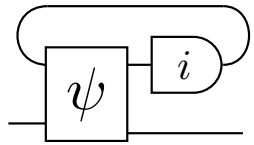
Delayed trace

This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*. It satisfies several properties of an ordinary trace:

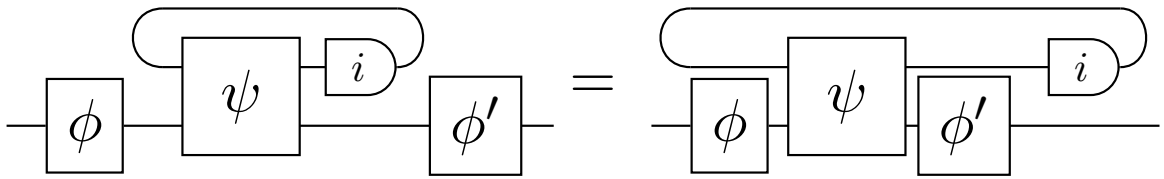
$$\psi : \mathbf{S} \times \mathbf{X} \to \mathbf{S} \times \mathbf{Y}$$



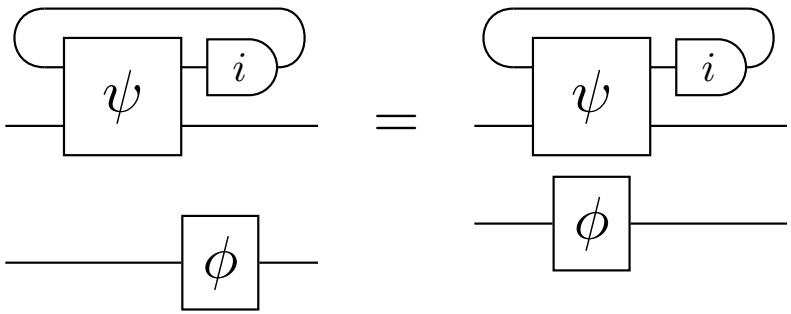
$$dtr_i^{\mathbf{S}}(\psi) : \mathbf{X} \to \mathbf{Y}$$



S/T naturality:



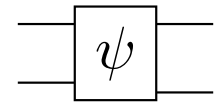
Superposition:

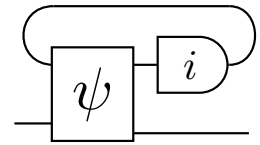


Delayed trace

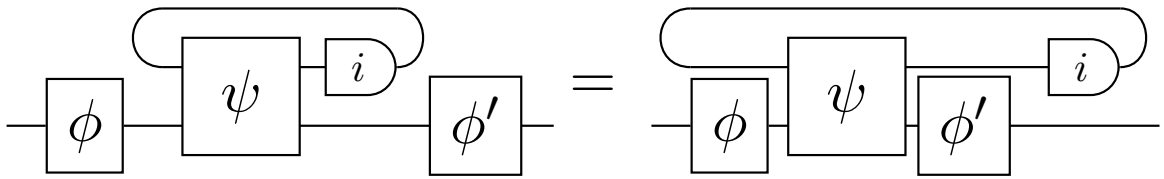
This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*. It satisfies several properties of an ordinary trace:

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

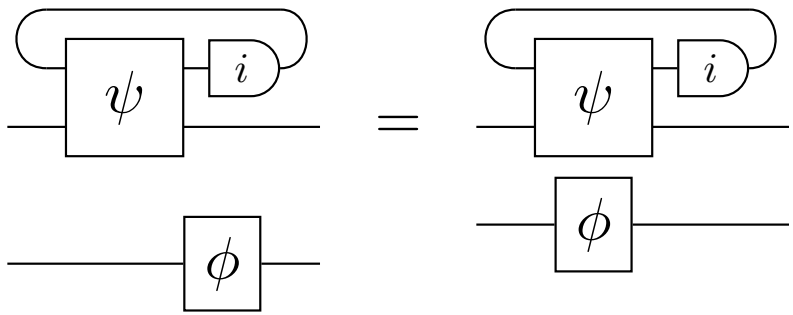




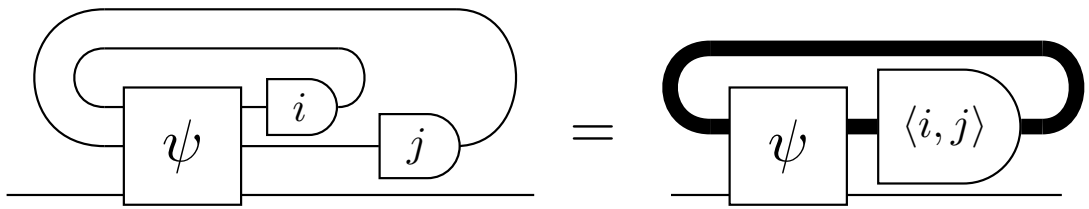
S/T naturality:



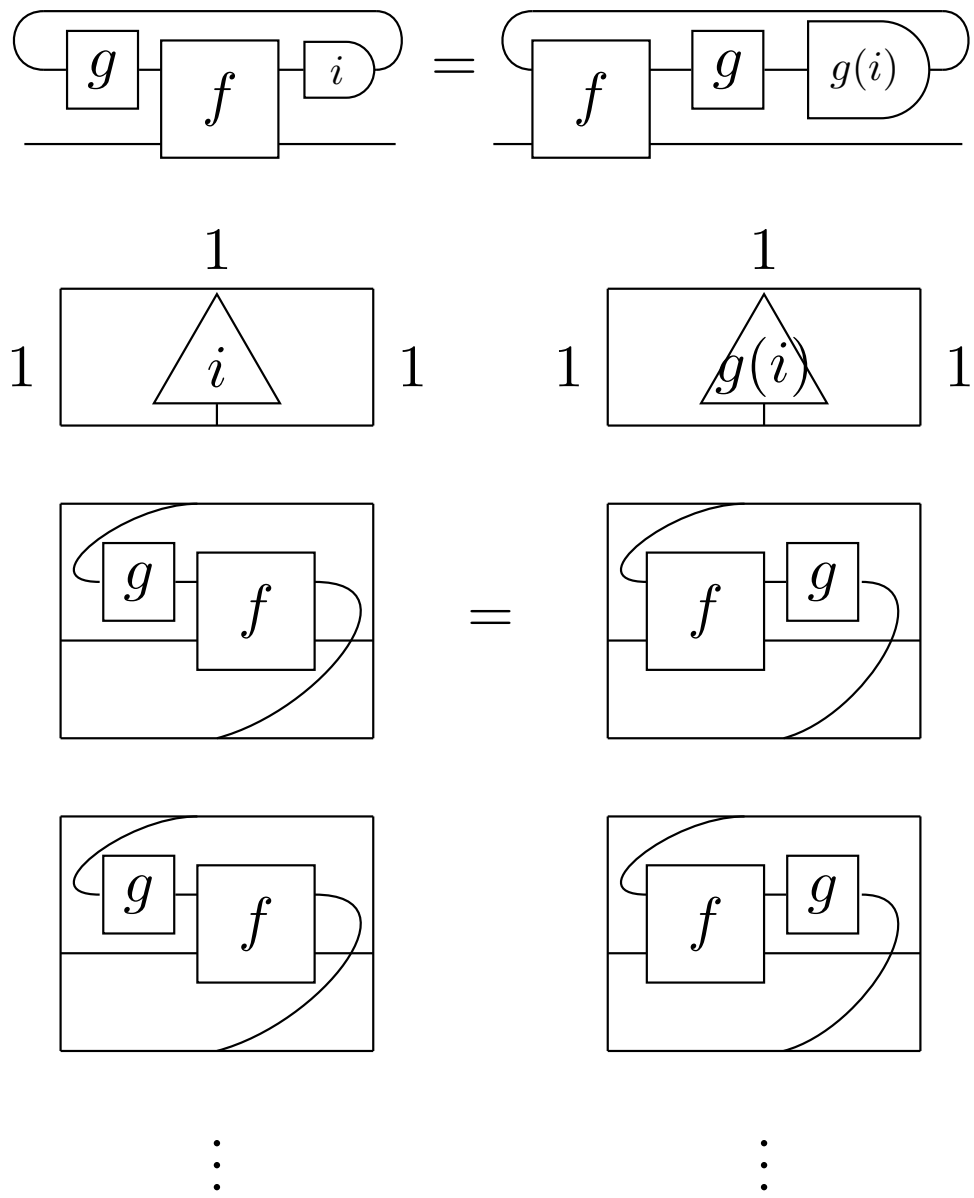
Superposition:



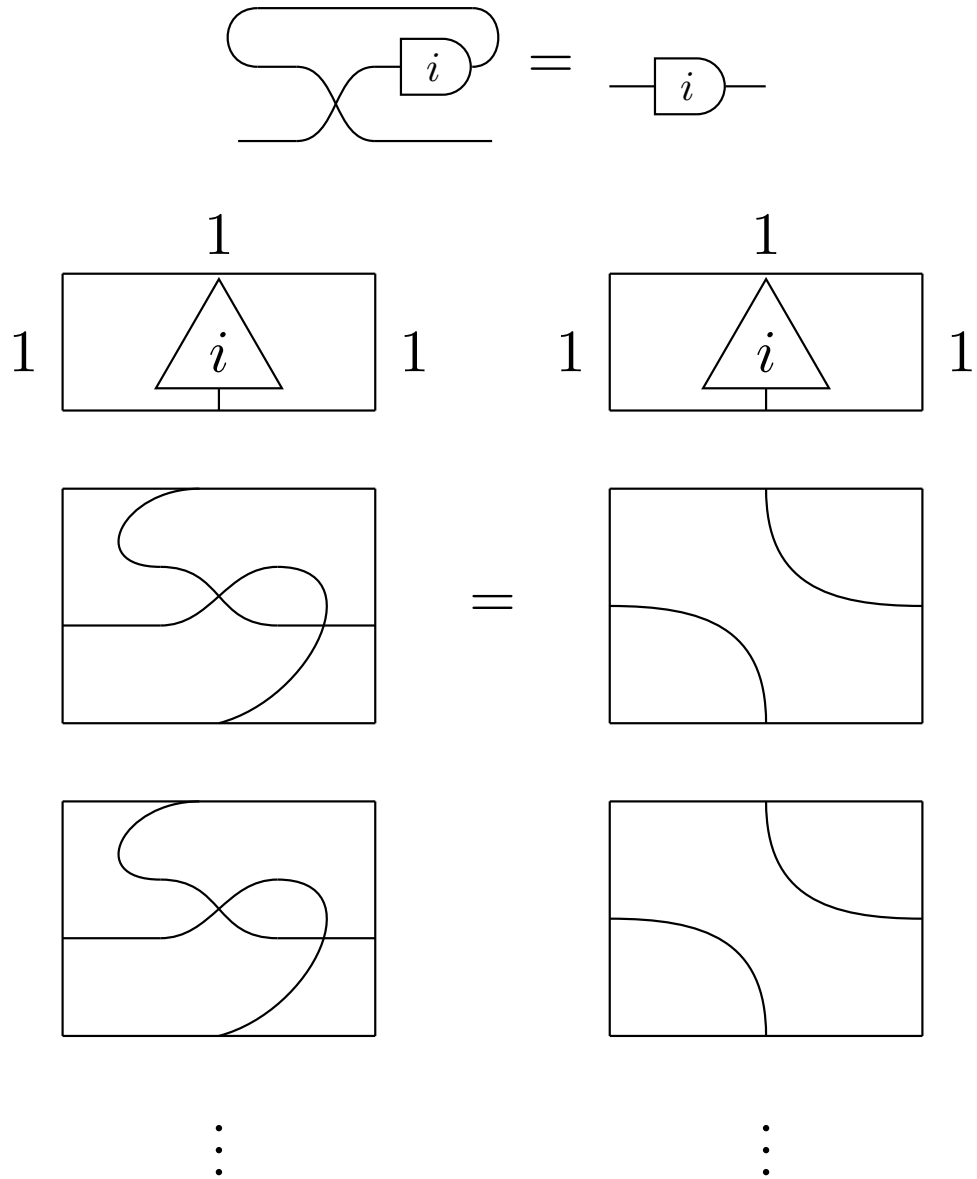
Vanishing $\times$:



~~Dinaturality~~ → retiming



~~Yanking~~ → delay

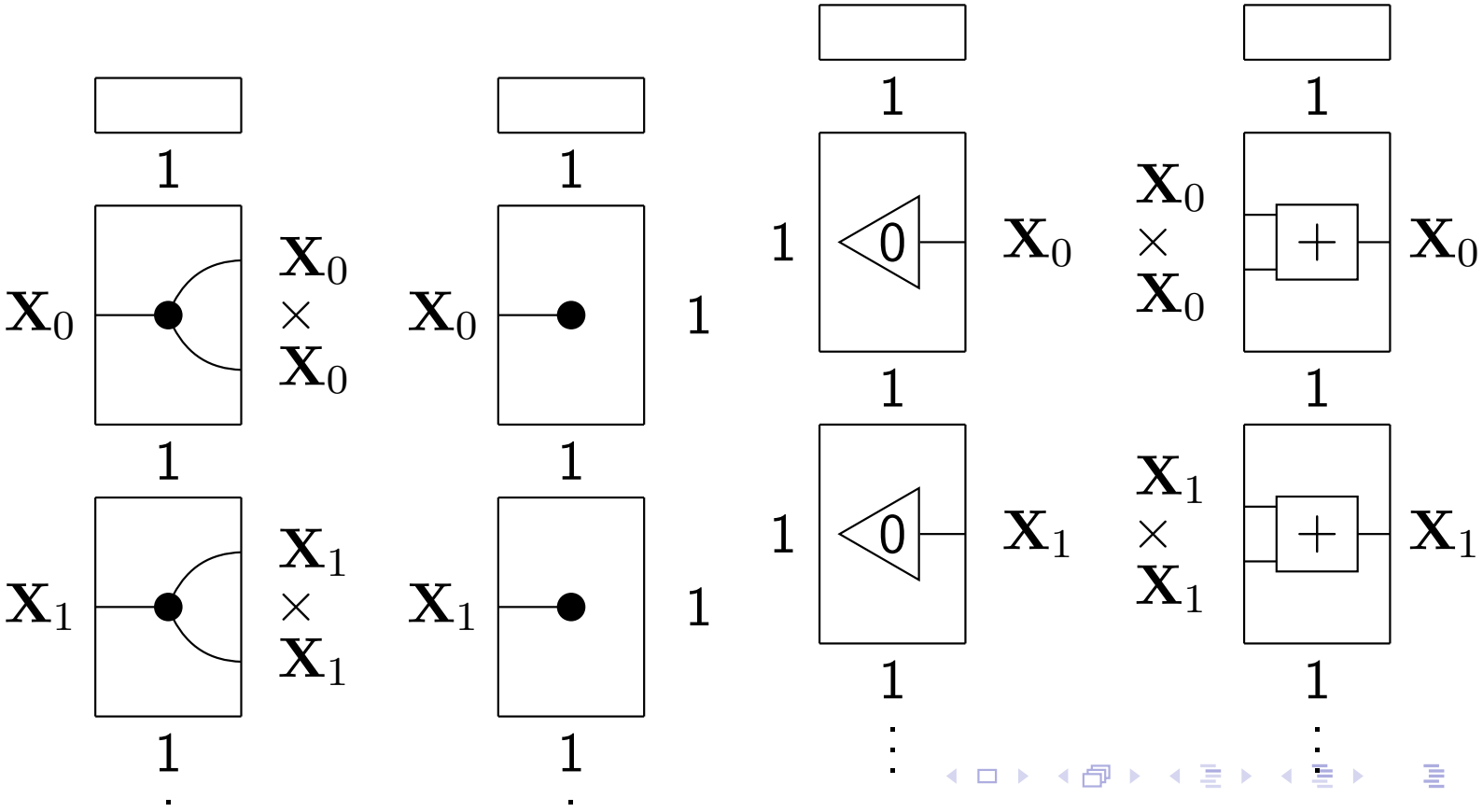


$\mathrm{St}(-)$ preserves a lot of structure

Theorem

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

The first two are easy:



Sanity checks

Proposition: $\mathbb{C}$ embeds in $\mathrm{St}(\mathbb{C})$

There is a finite-product preserving functor $H : \mathbb{C} \to \mathrm{St}(\mathbb{C})$. It takes A to $(A, A, A, \ldots)$ and f to the stateless sequence executing f at each step.

Sanity checks

Proposition: $\mathbb{C}$ embeds in $\mathrm{St}(\mathbb{C})$

There is a finite-product preserving functor $H : \mathbb{C} \to \mathrm{St}(\mathbb{C})$. It takes A to $(A, A, A, \ldots)$ and f to the stateless sequence executing f at each step.

Proposition: Normal forms for $\mathrm{St}(\mathbb{C})$

Every $\Psi \in \mathrm{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$ can be written as $\Psi = dtr_i^{\mathbf{S}}(\psi)$ where ψ has a stateless representative and $i : 1 \to S$.

Sanity checks

Proposition: $\mathbb{C}$ embeds in $\mathrm{St}(\mathbb{C})$

There is a finite-product preserving functor $H : \mathbb{C} \to \mathrm{St}(\mathbb{C})$. It takes A to $(A, A, A, \ldots)$ and f to the stateless sequence executing f at each step.

Proposition: Normal forms for $\mathrm{St}(\mathbb{C})$

Every $\Psi \in \mathrm{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$ can be written as $\Psi = dtr_i^{\mathbf{S}}(\psi)$ where ψ has a stateless representative and $i : 1 \to S$.

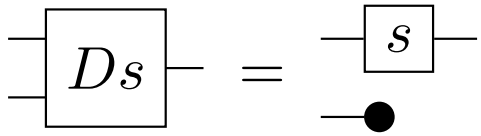
Adding state to computations is common:

1. Bicategorically—Katis, Sabadini, & Walters '97
2. Digital circuits—Ghica & Jung, '16
3. Signal flow graphs—Bonchi, Sobociński, & Zanasi, '14

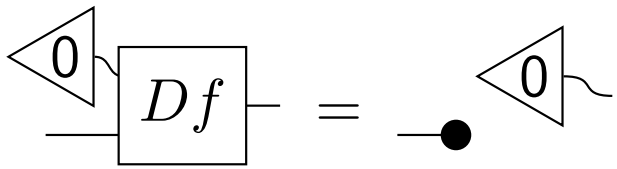
Outline

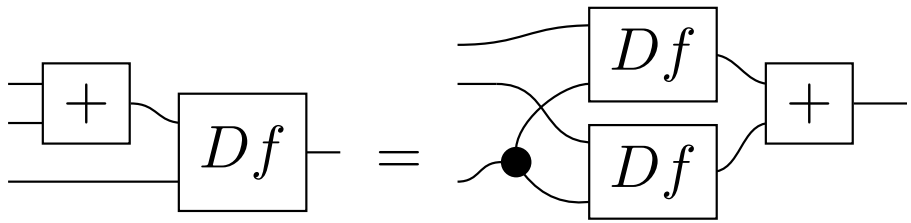
Cartesian differential categories [Blute, Cockett, Seely '09]

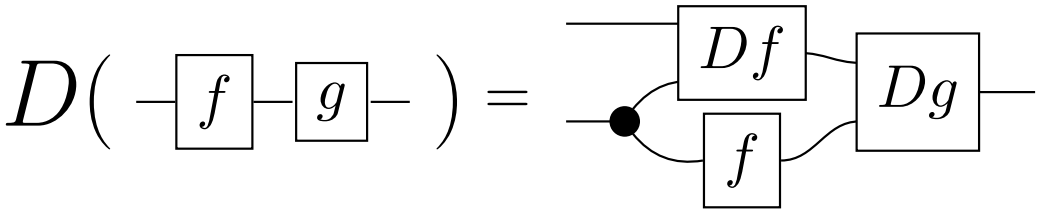
A *Cartesian differential category* has a differential operation on morphisms sending $f : X \to Y$ to $Df : X \times X \to Y$, satisfying seven axioms:

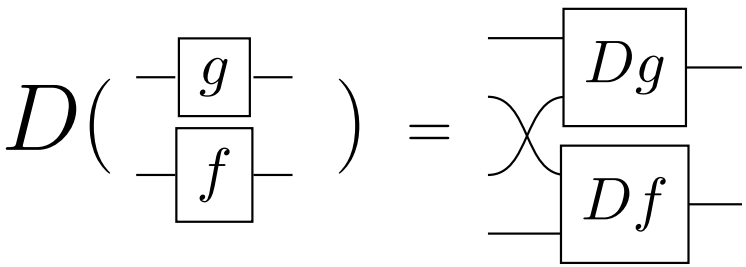
CD1. 

for $s \in \{\mathrm{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

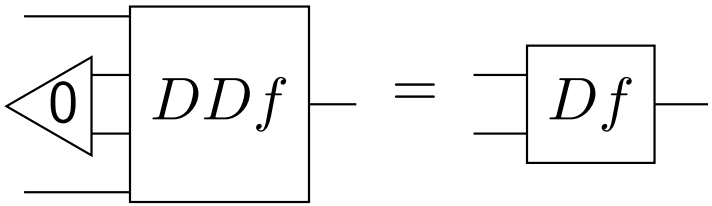
CD2. 

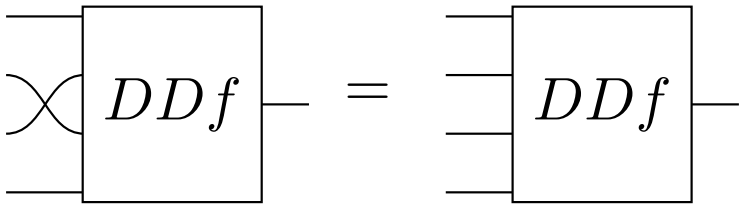
CD3. 

CD4. 

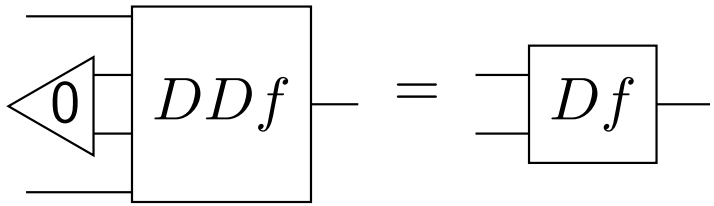
CD5. 

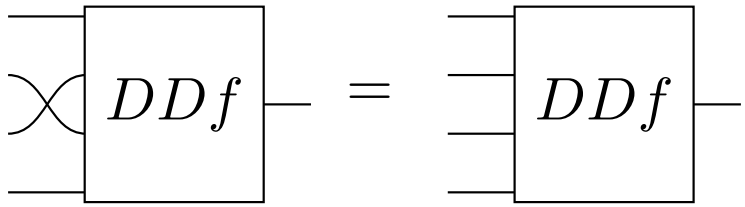
Cartesian differential category axioms, continued

CD6. 

CD7. 

Cartesian differential category axioms, continued

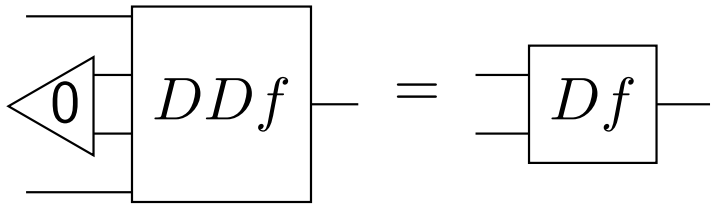
CD6. 

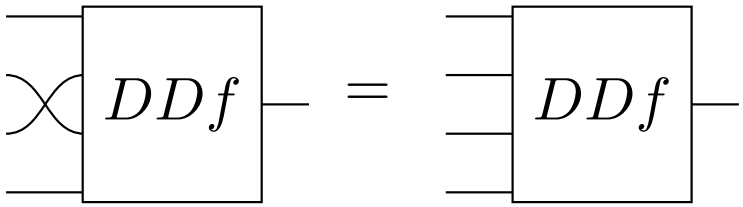
CD7. 

Example

Objects of the category $\mathbf{Euc}_\infty$ are $\mathbb{R}^n$ for $n \in \mathbb{N}$, maps are smooth maps between them. $\mathbf{Euc}_\infty$ is a Cartesian differential category with the (curried) Jacobian sending $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

Cartesian differential category axioms, continued

CD6. 

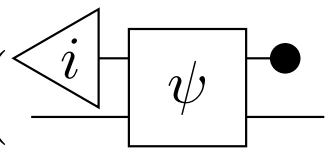
CD7. 

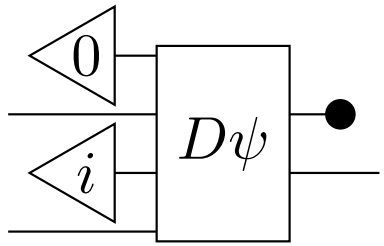
Example

Objects of the category $\mathbf{Euc}_\infty$ are $\mathbb{R}^n$ for $n \in \mathbb{N}$, maps are smooth maps between them. $\mathbf{Euc}_\infty$ is a Cartesian differential category with the (curried) Jacobian sending $f : \mathbb{R}^n \to \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

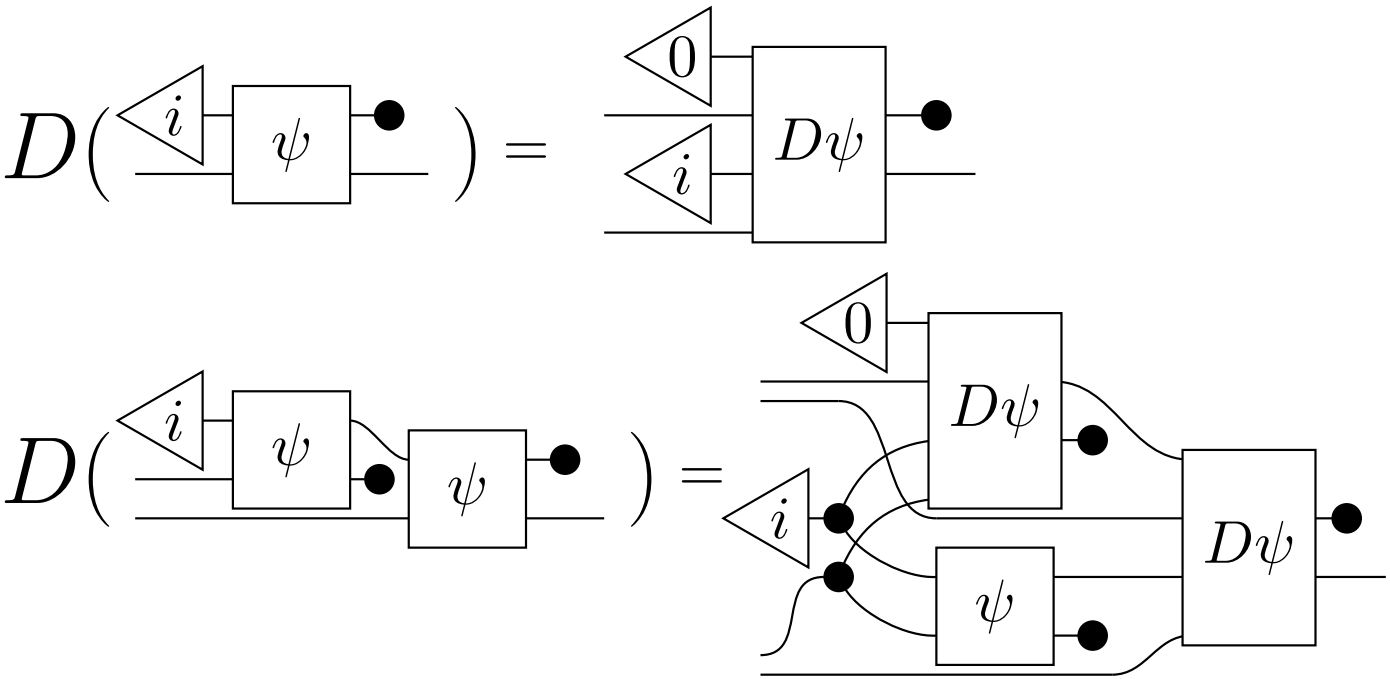
What's the right rule for delayed trace?

Differentiating the unrollings of a simple RNN

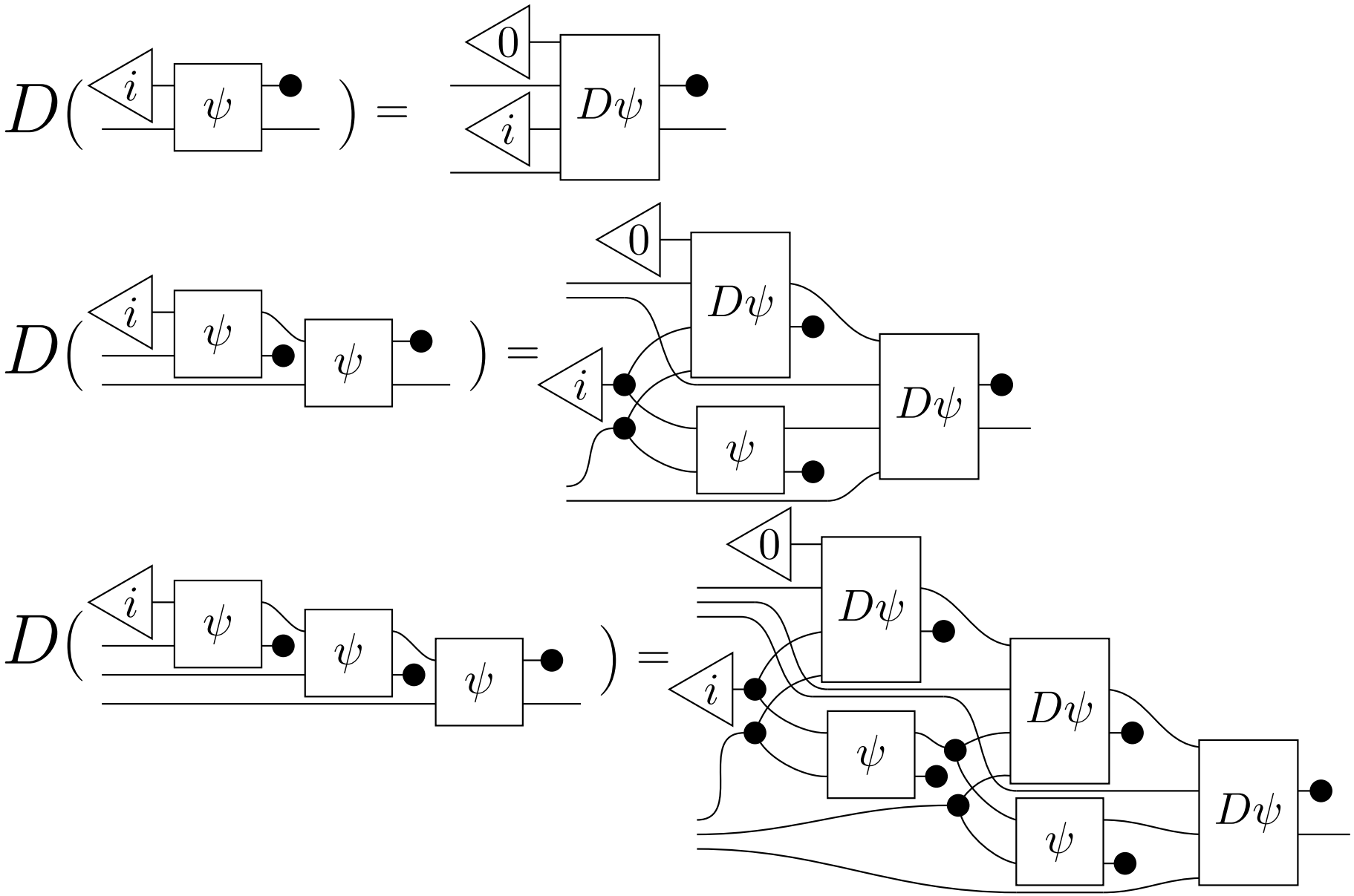
$D($ 

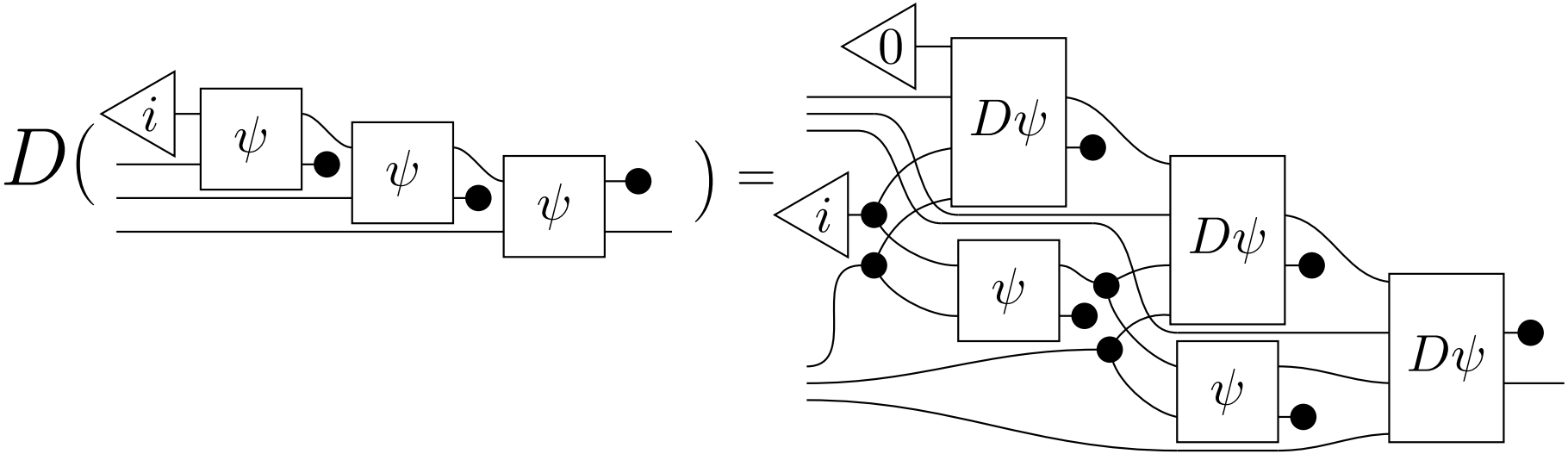
$) =$ 

Differentiating the unrollings of a simple RNN

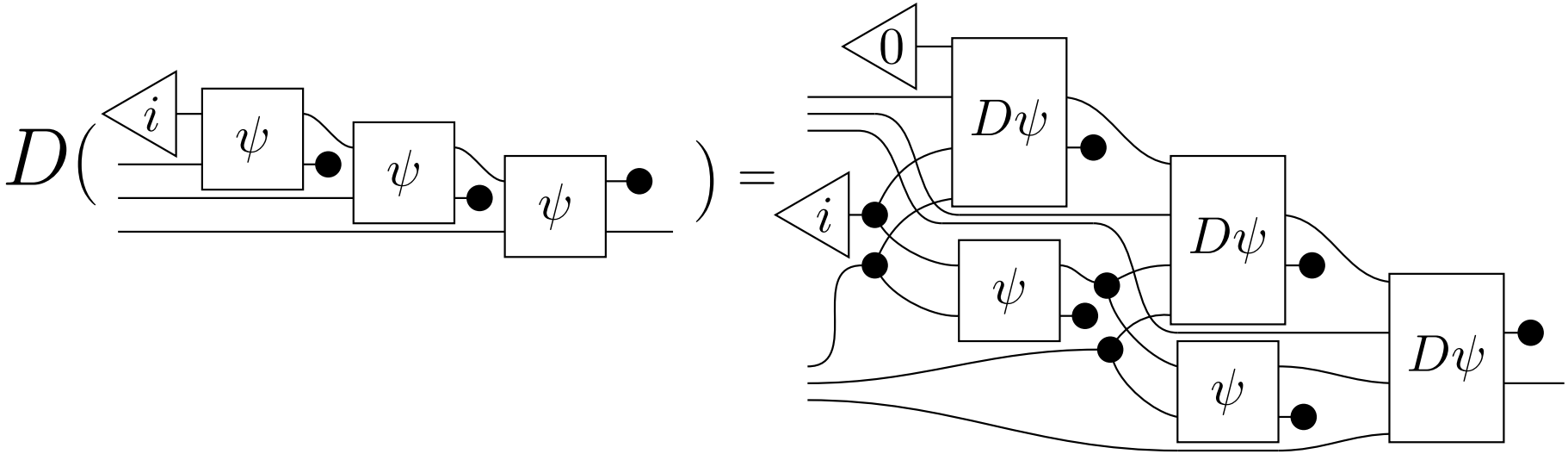


Differentiating the unrollings of a simple RNN

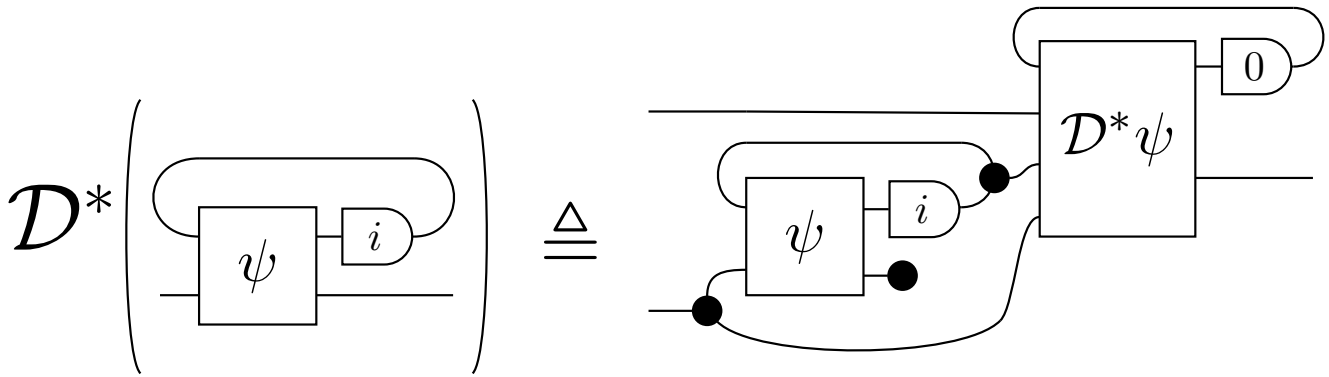




This suggests a hypothesis:

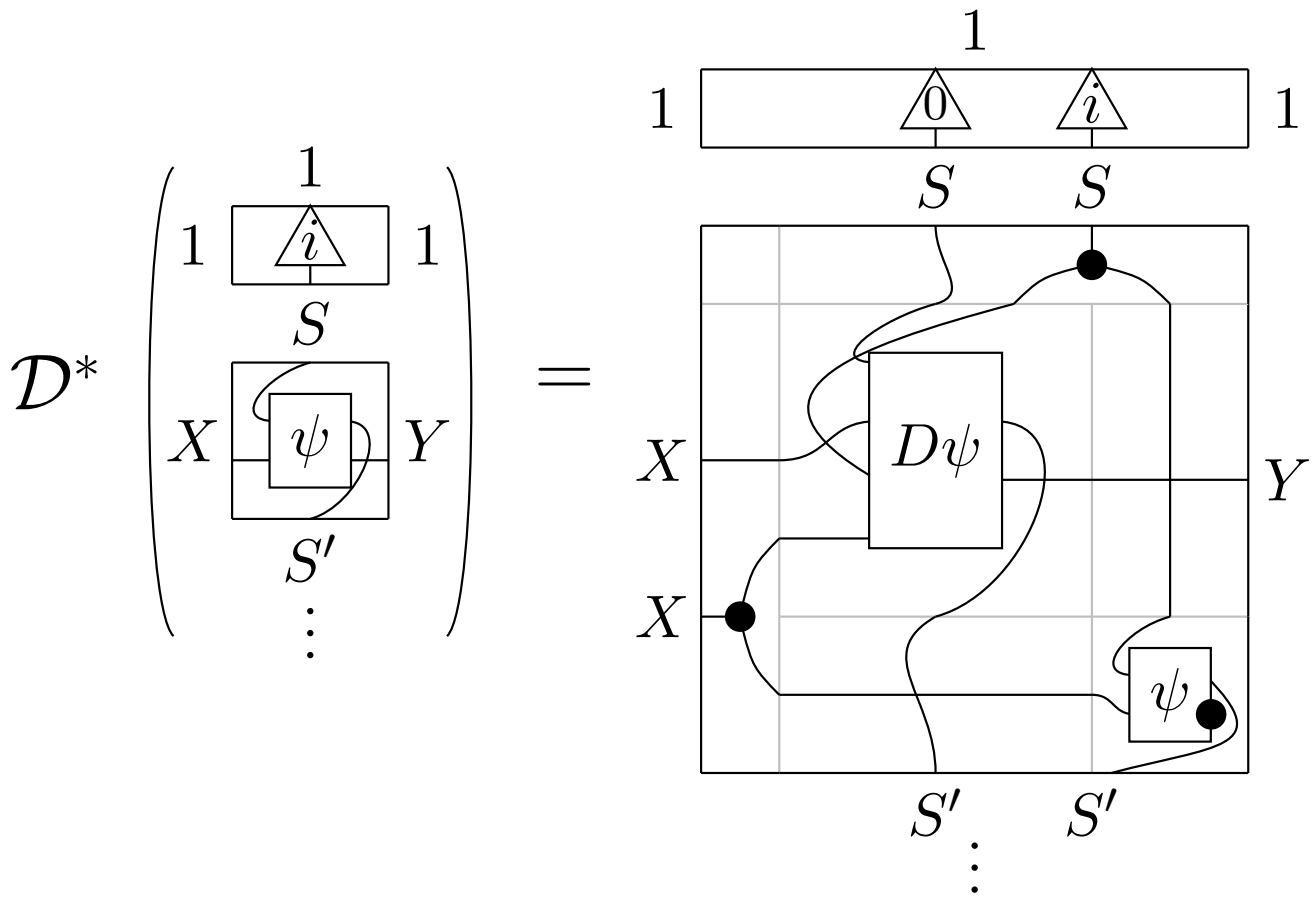


This suggests a hypothesis:



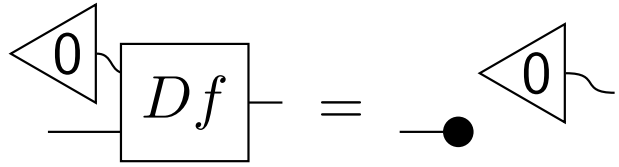
Differentiation for stateful functions

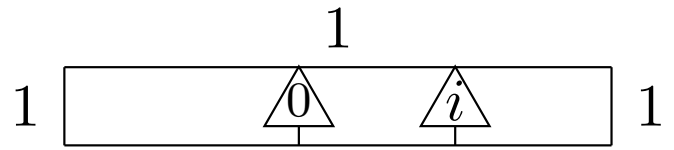
Let $\mathbb{C}$ be Cartesian differential with differential operator D. The following is a Cartesian differential operator on $\mathrm{St}(\mathbb{C})$:

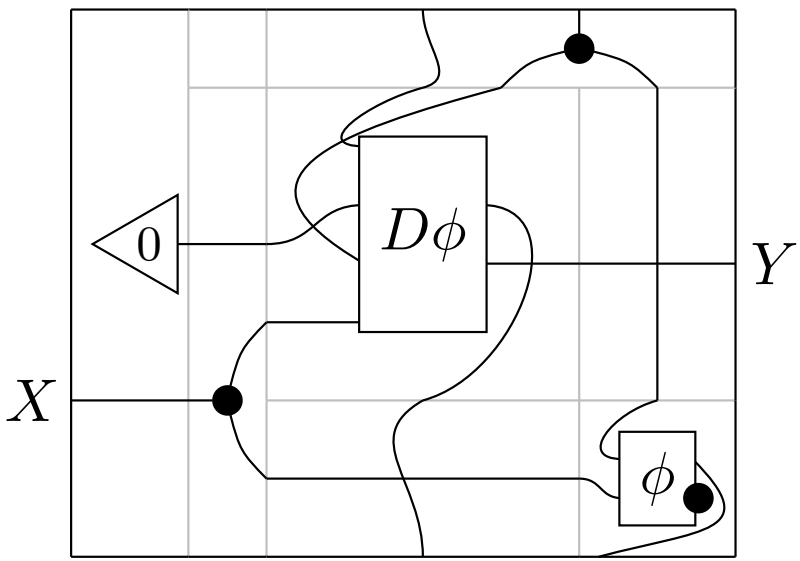


$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD2:



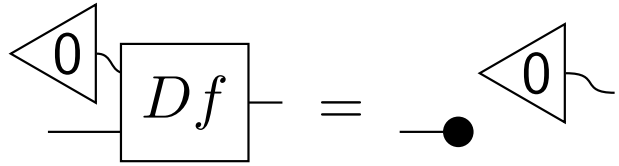


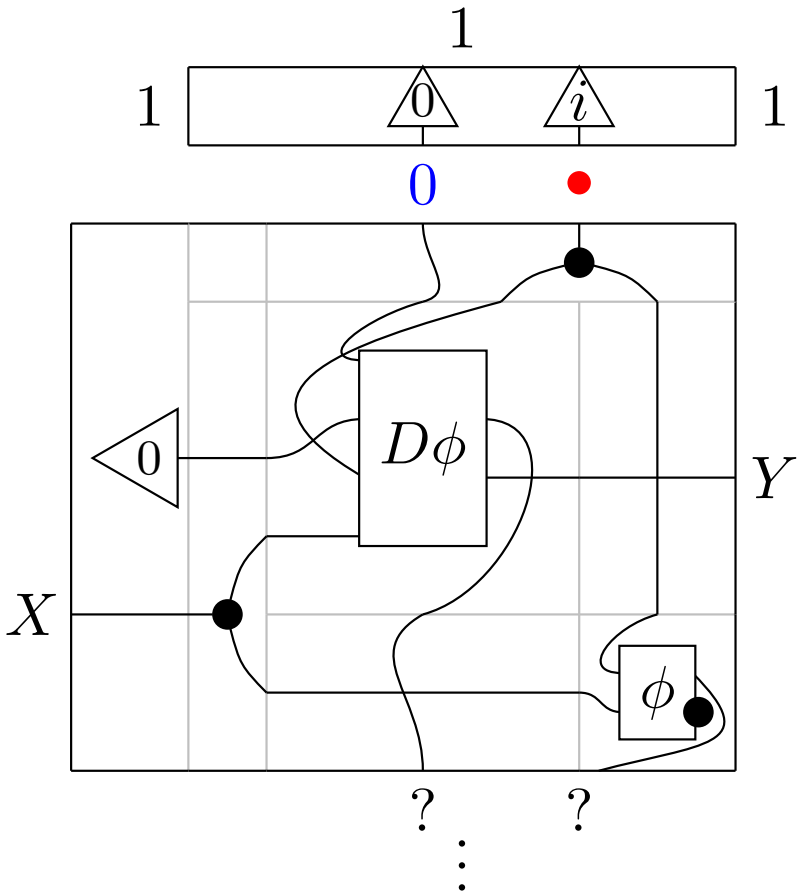


$\vdots$

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

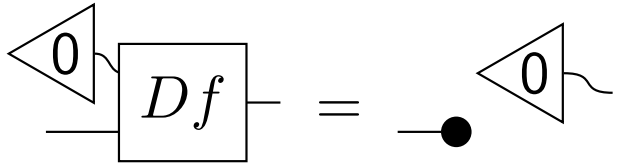
Proof idea. For CD2:

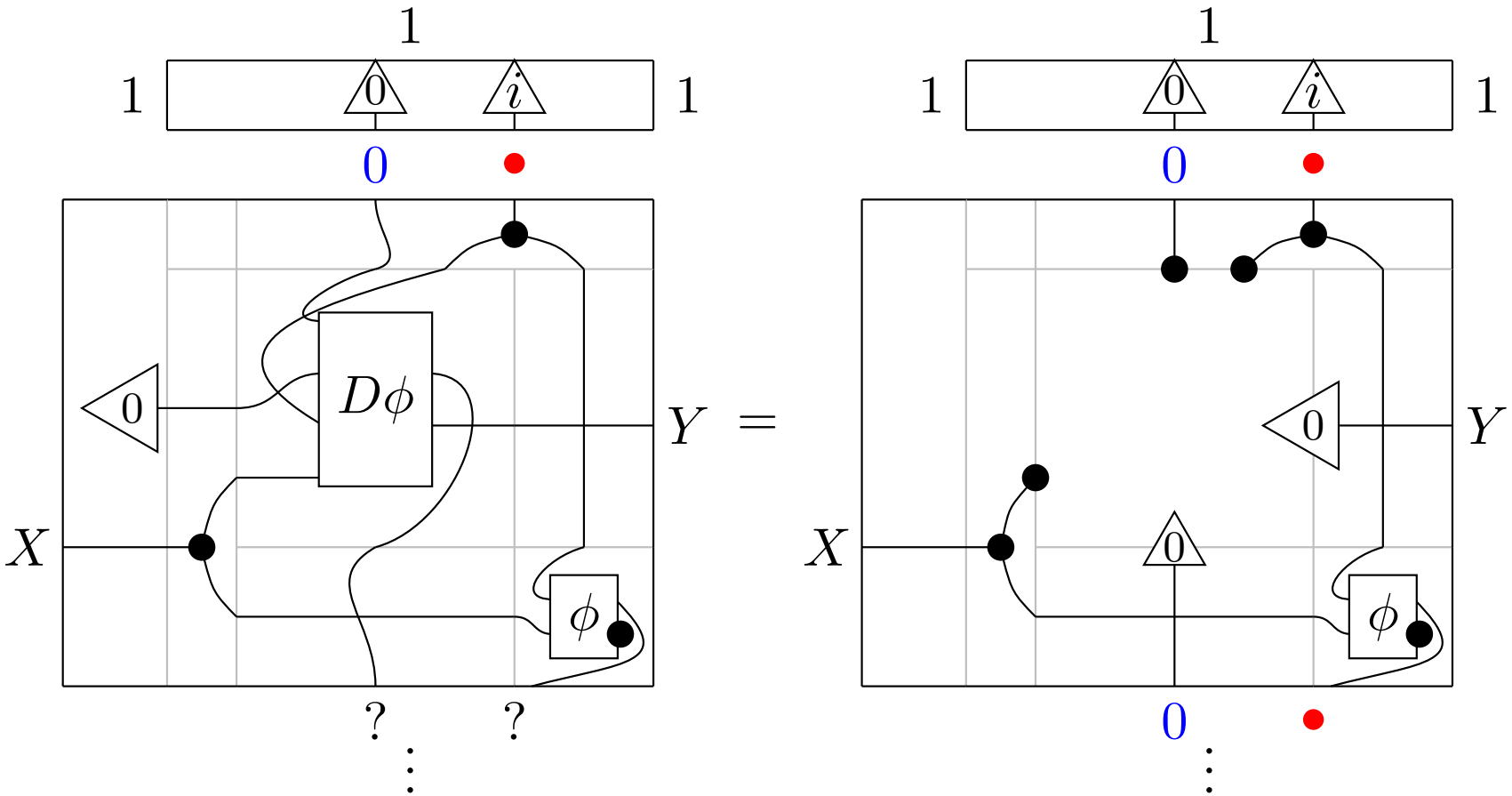




$\mathcal{D}^*$ is a Cartesian differential operator

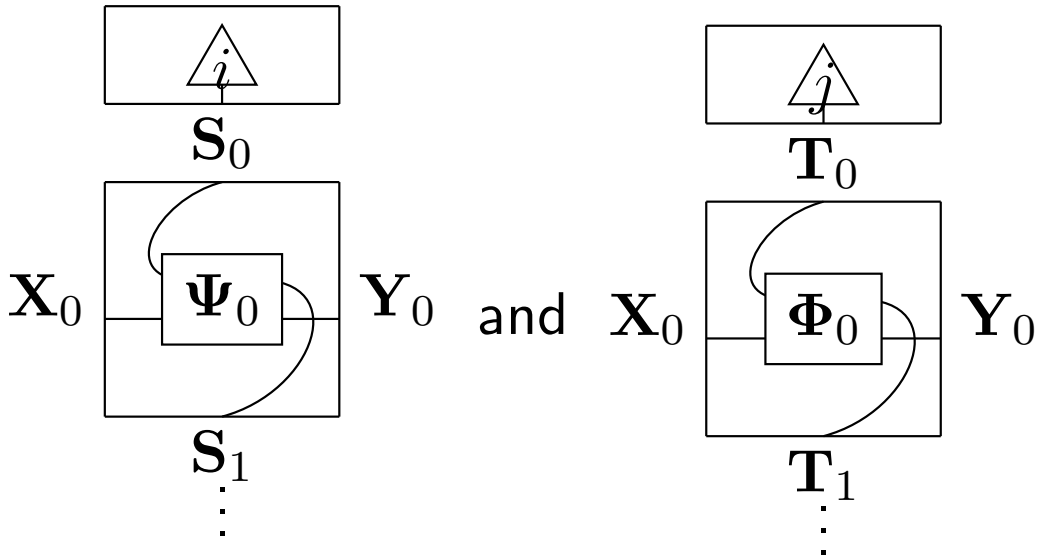
Proof idea. For CD2:



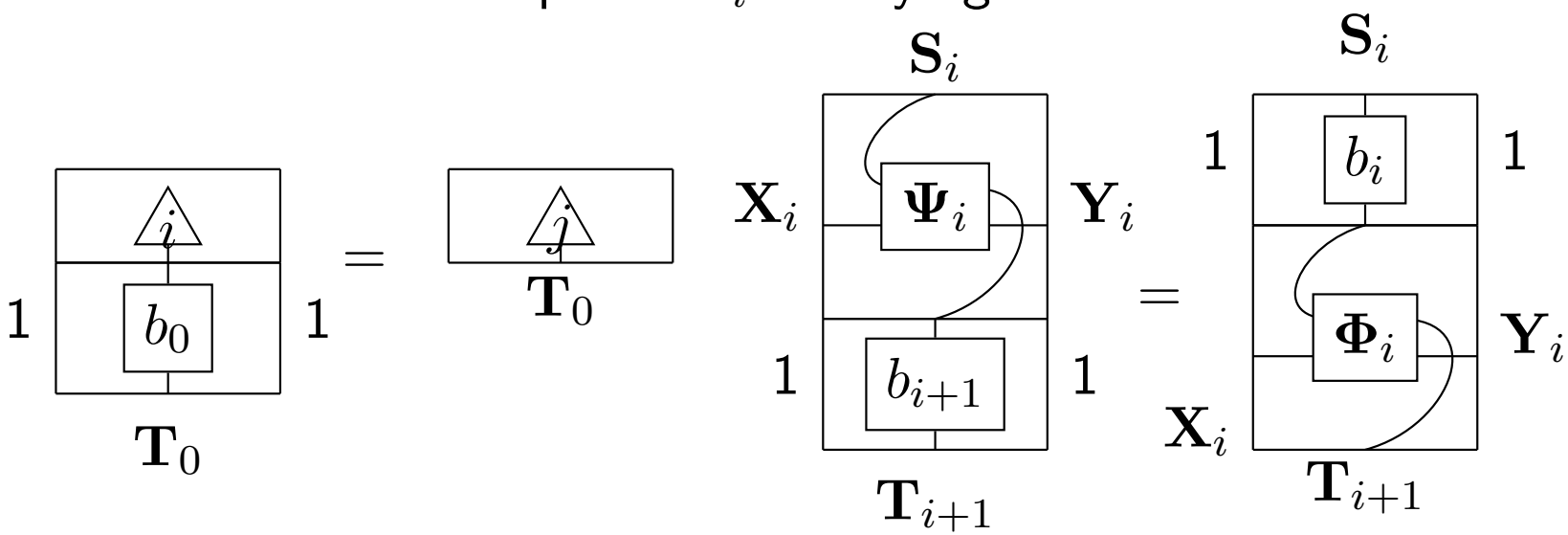


The (illustrated) shim lemma

In order to show the extensional equivalence of

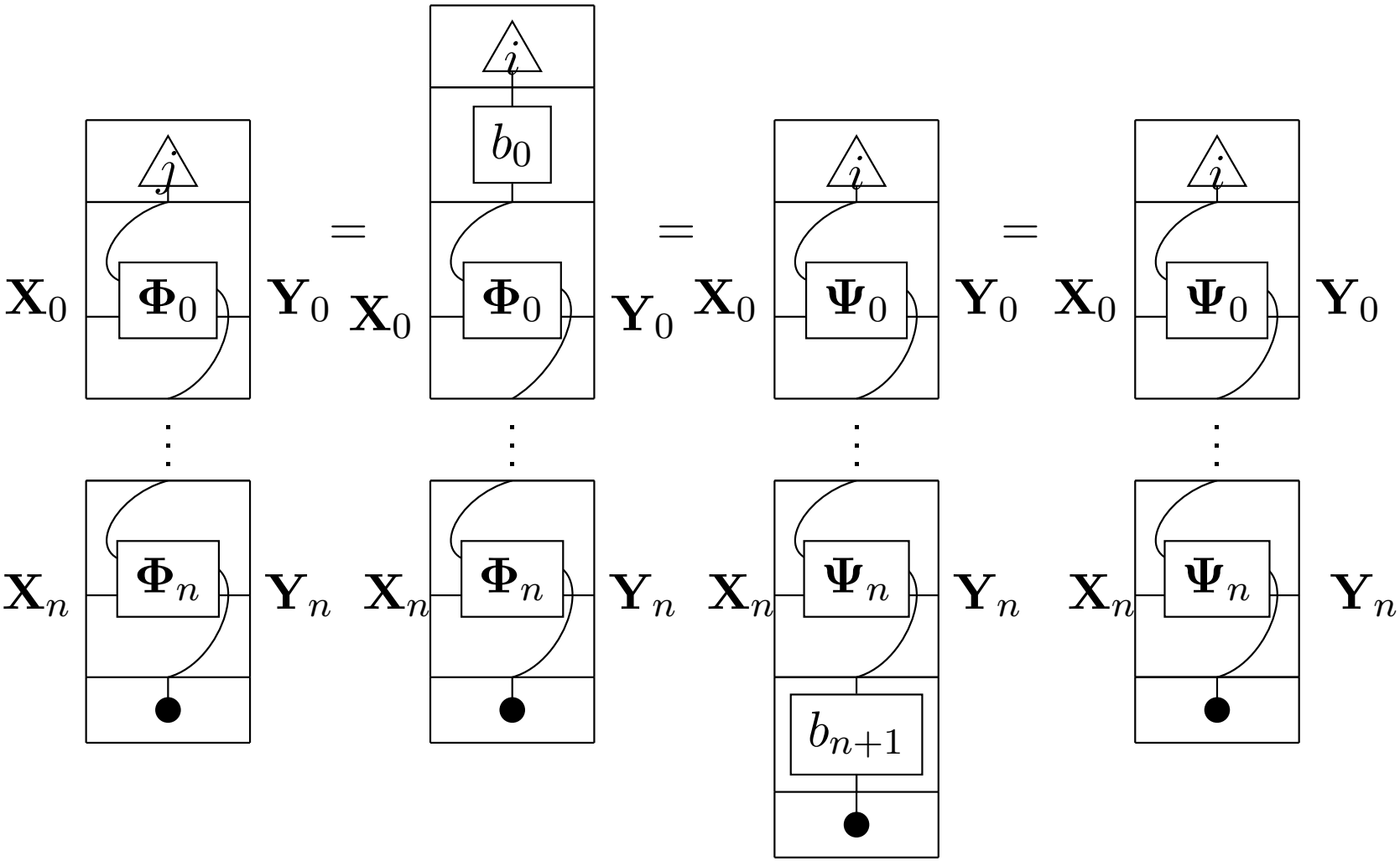


it suffices to find a sequence b_i satisfying:



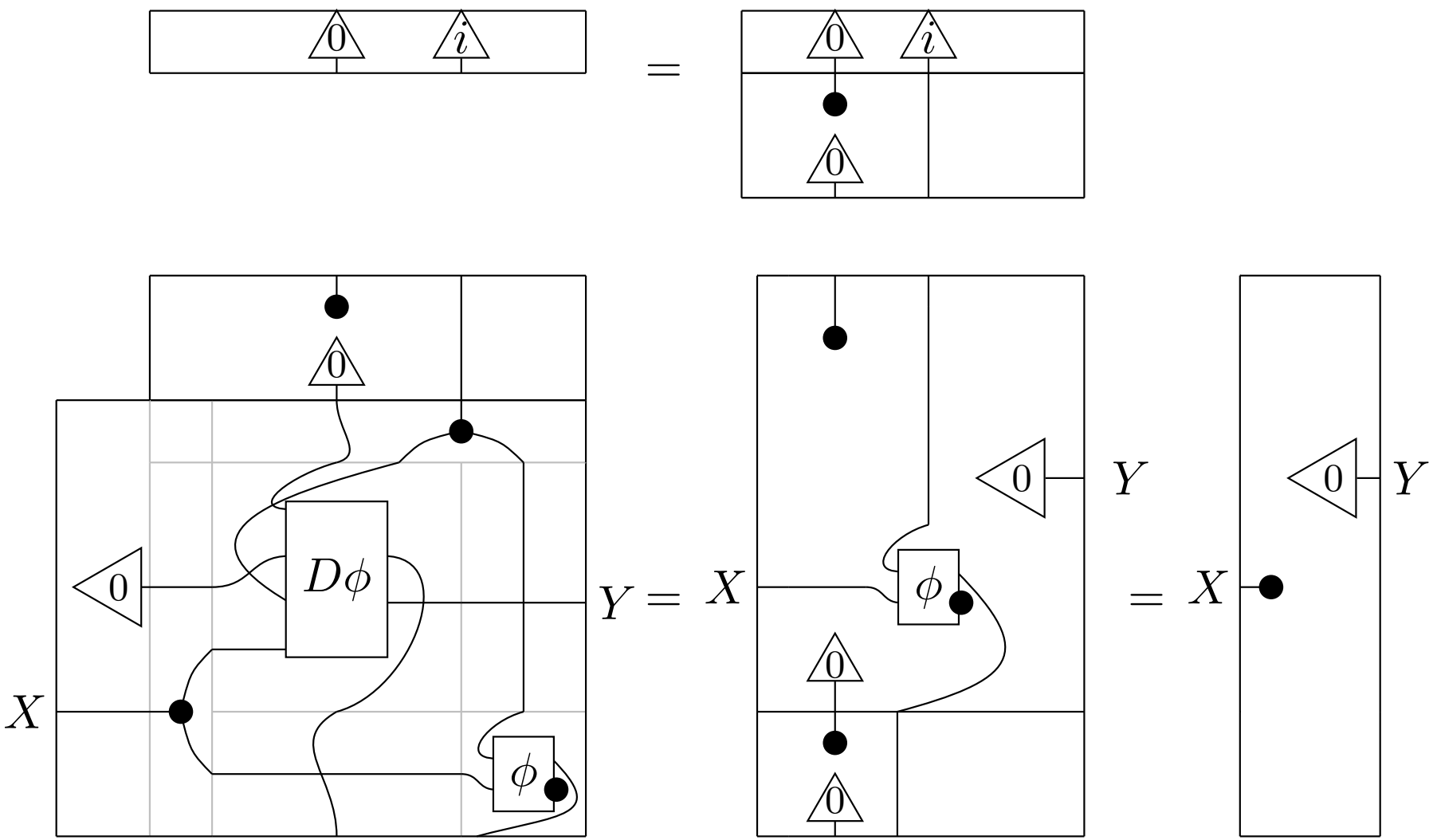
Proof of shim lemma

For all $n \in \omega$,

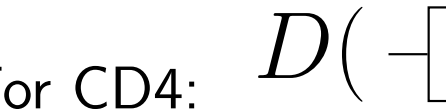
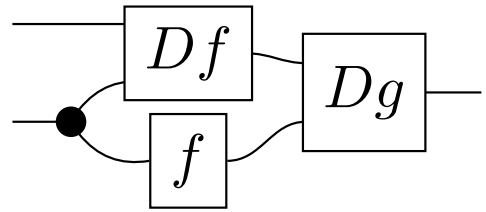


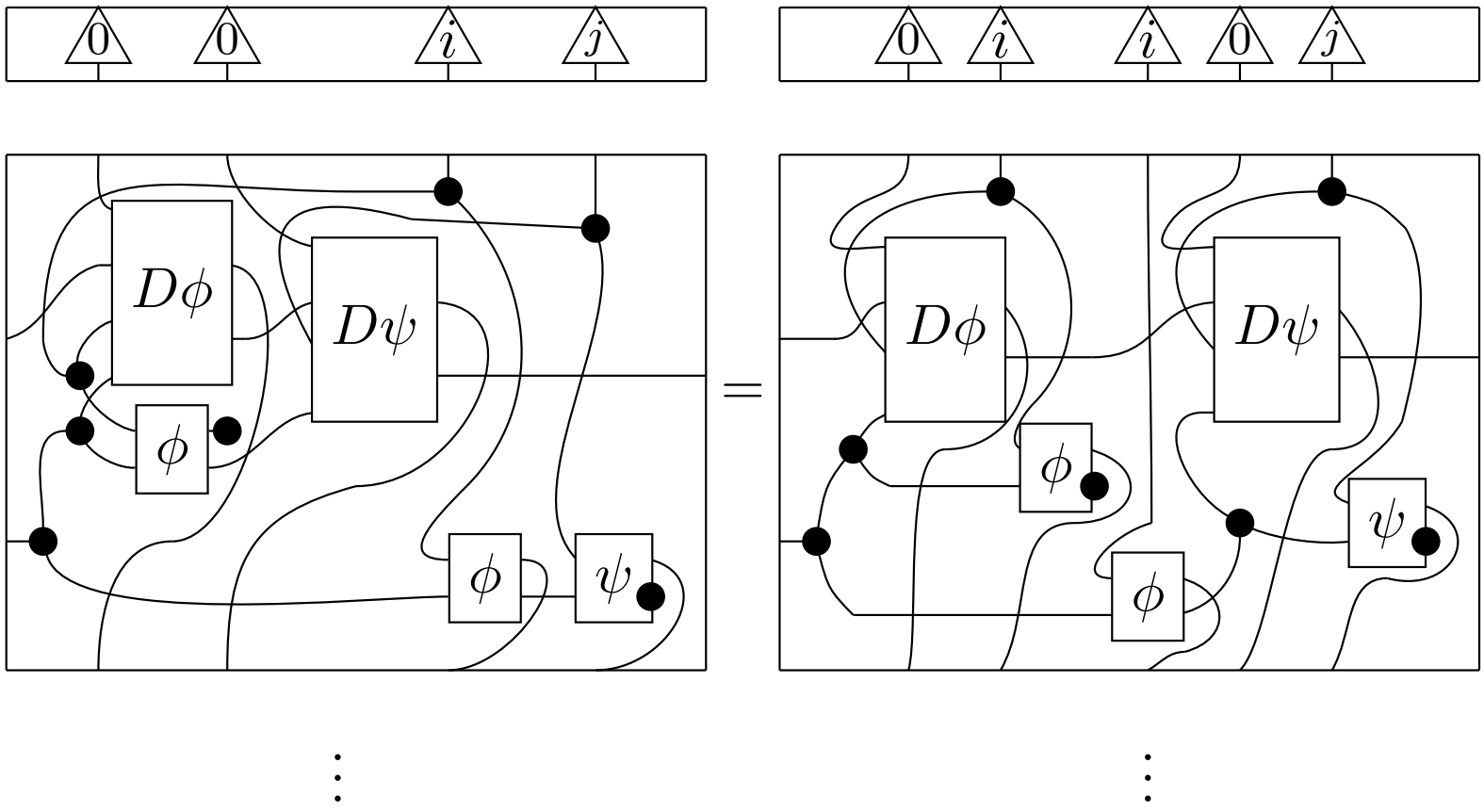
$\mathcal{D}^*$ is a Cartesian differential operator

Proof. For CD2:

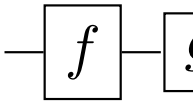
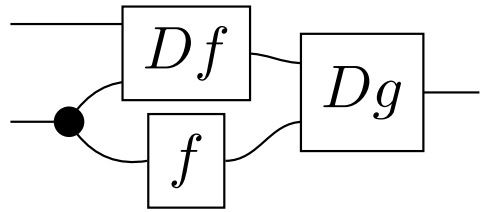


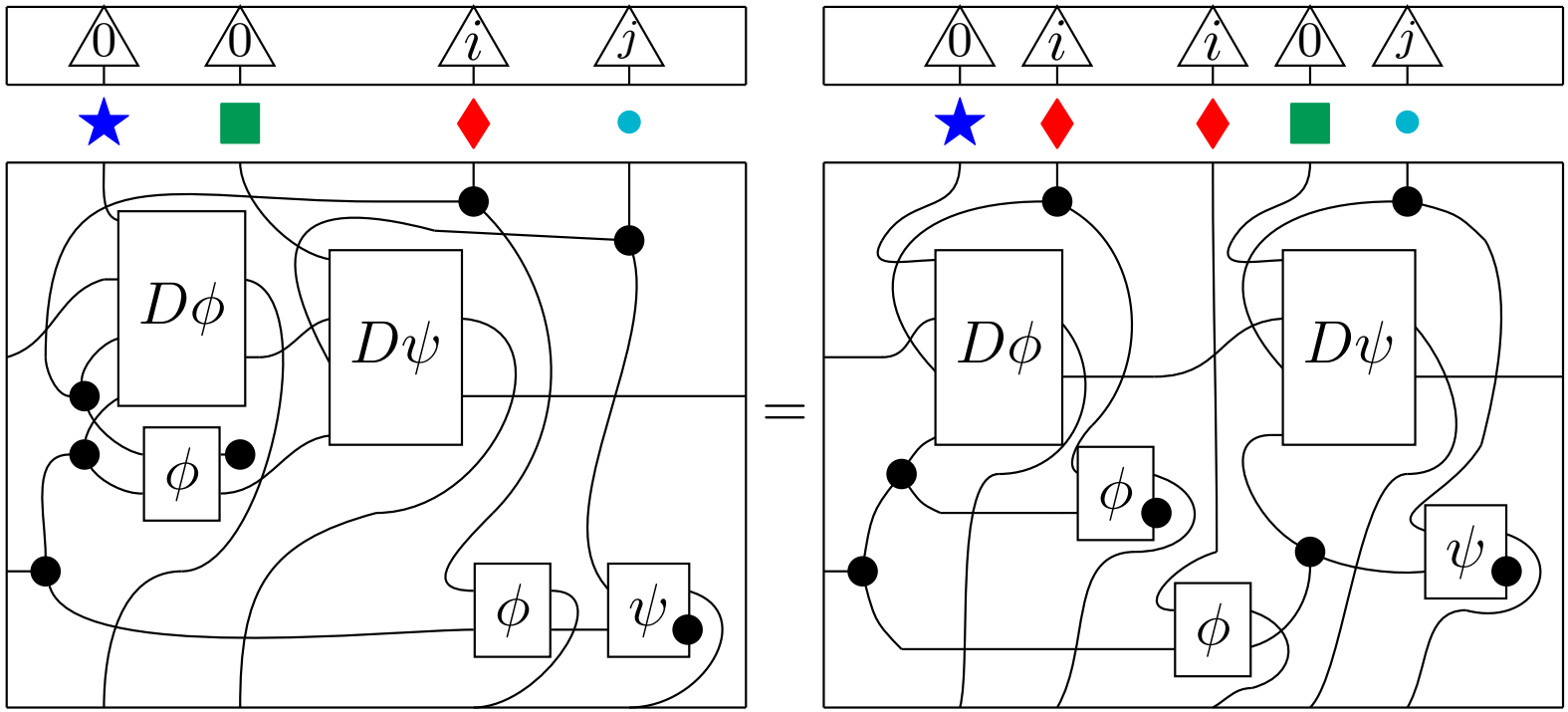
$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD4: $D($  $) =$ 

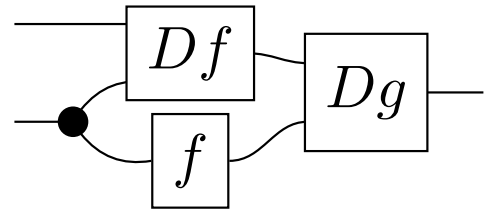


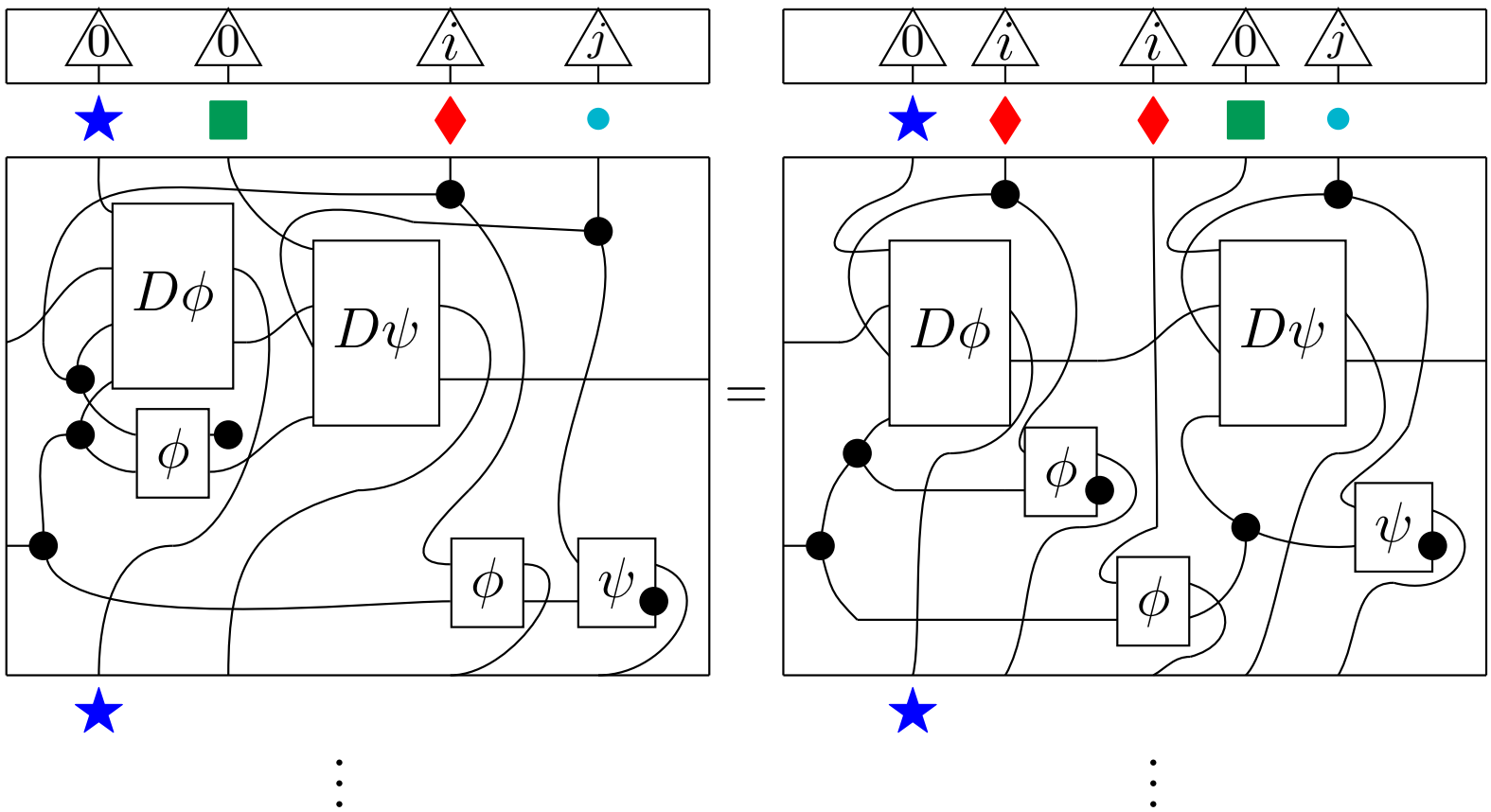
$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD4: $D($  $) =$ 

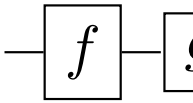
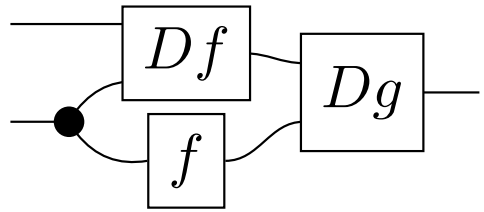


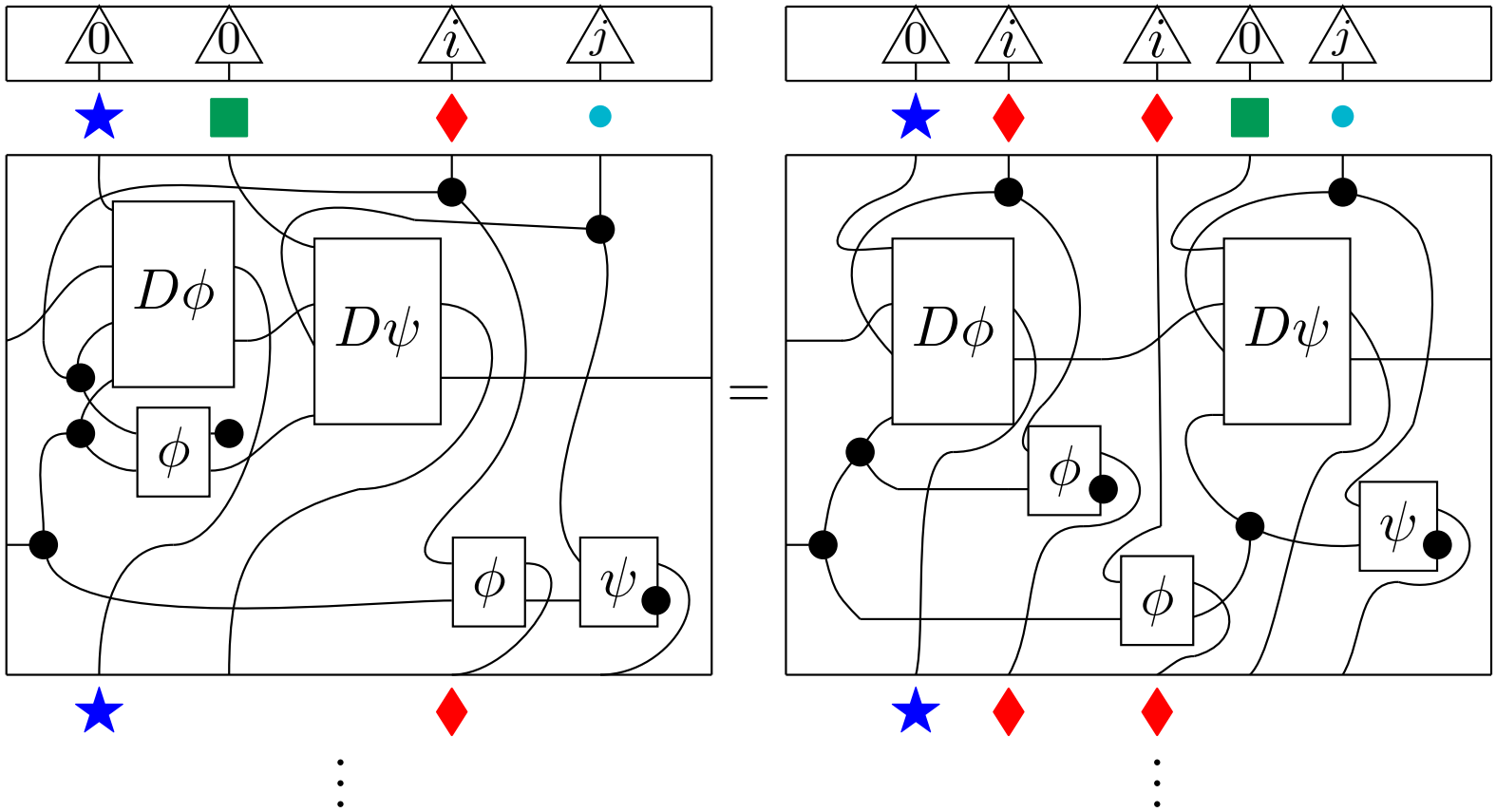
$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD4: $D(\ -[f]-[g]-\ ) =$ 

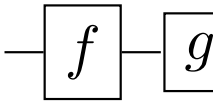
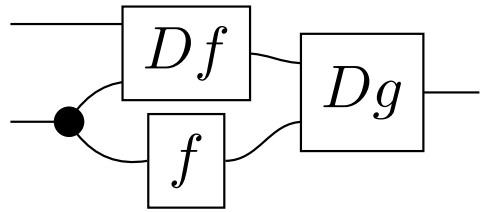


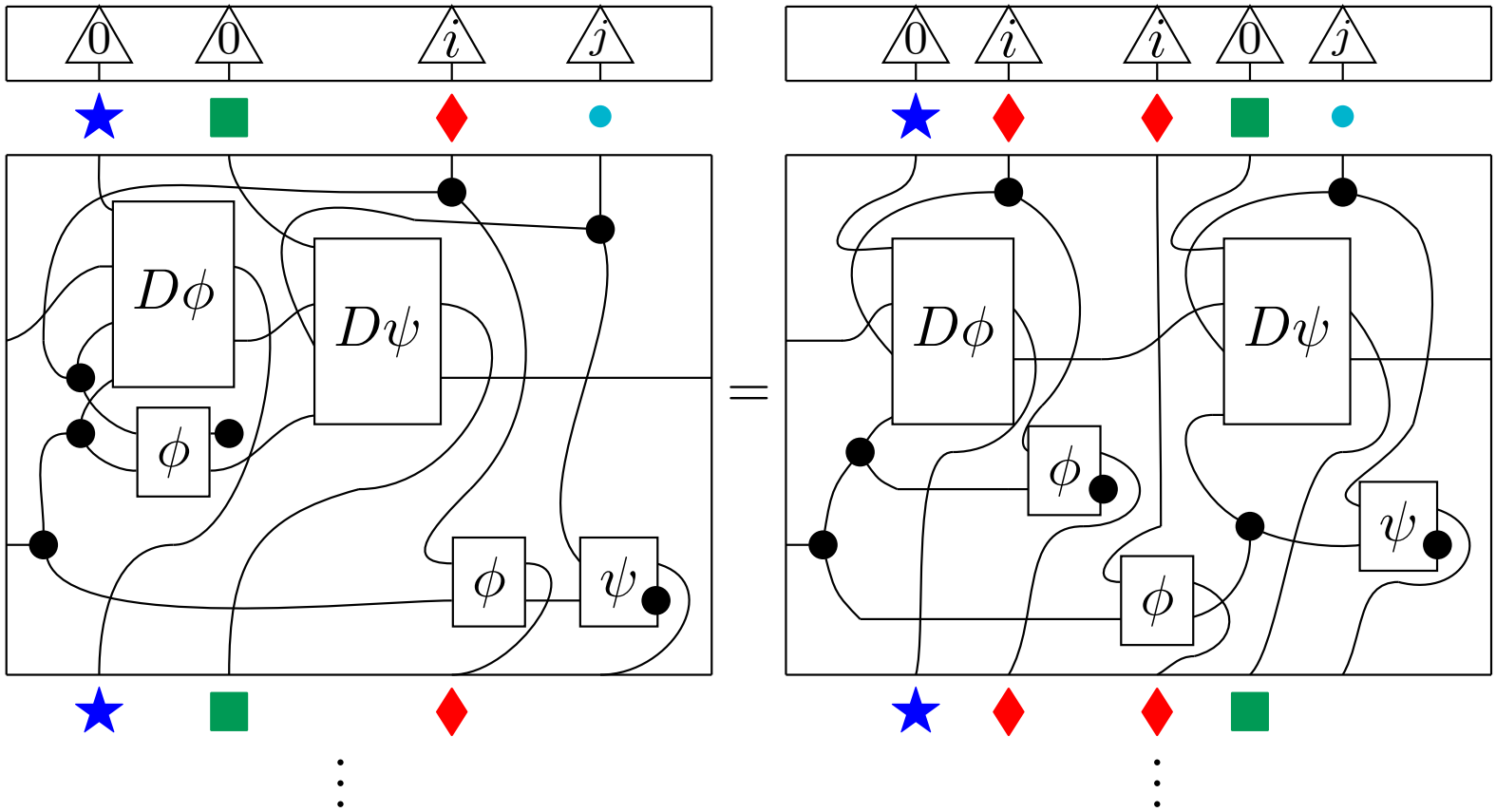
$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD4: $D($  $) =$ 

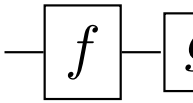
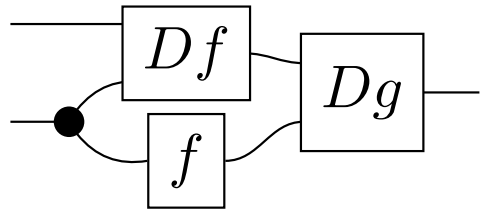


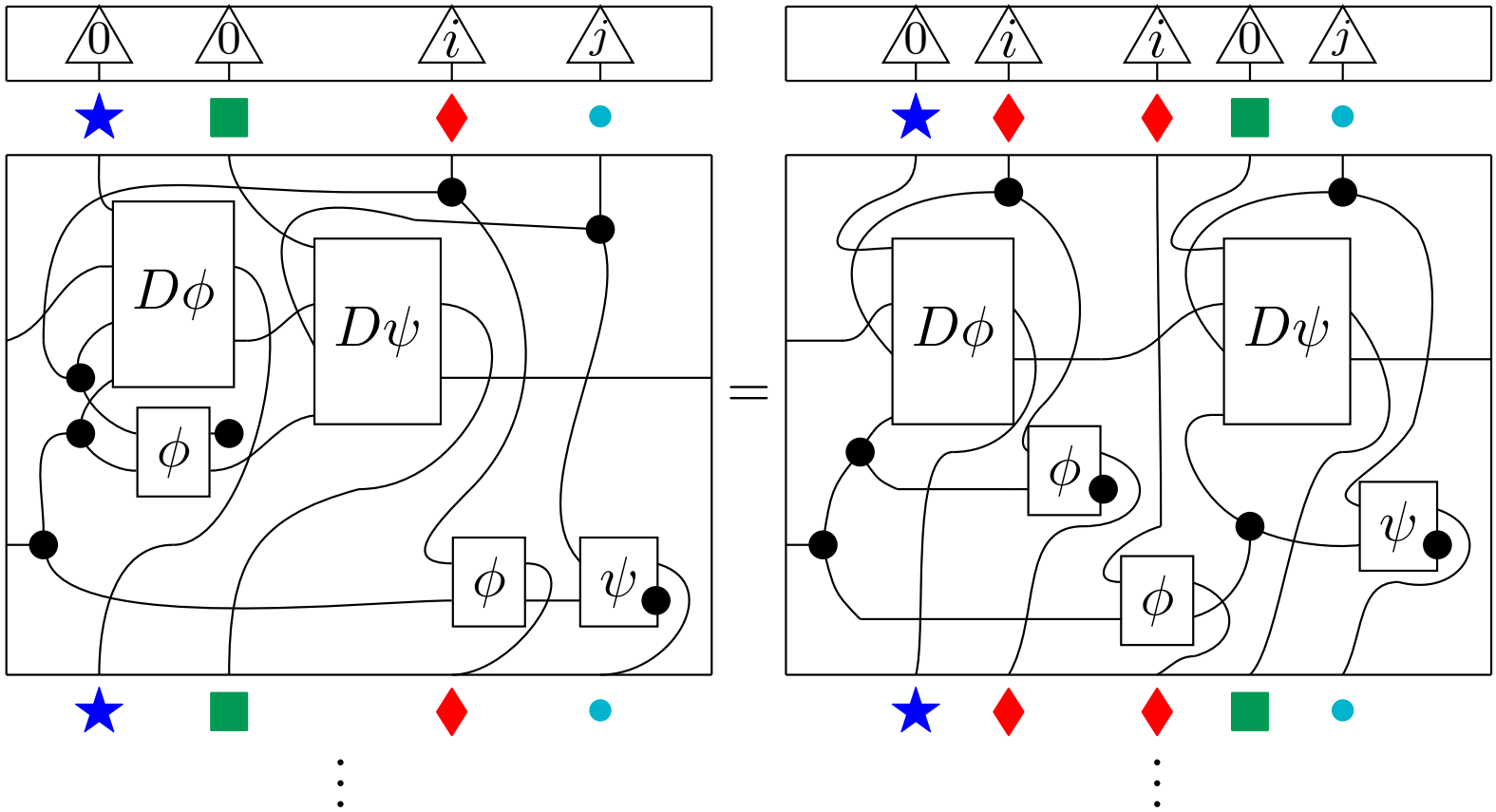
$\mathcal{D}^*$ is a Cartesian differential operator

Proof idea. For CD4: $D($  $) =$ 



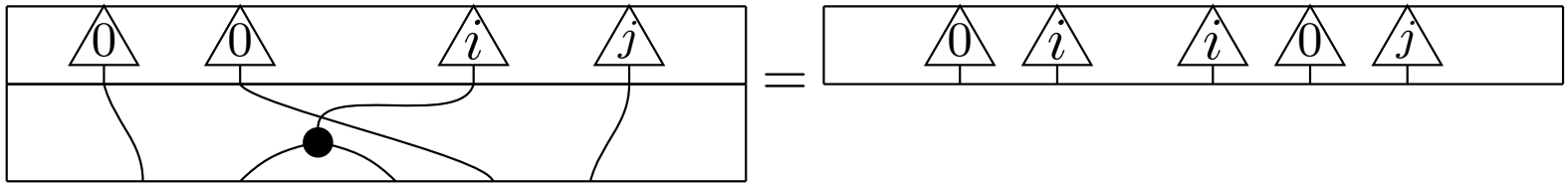
$\mathcal{D}^*$ is a Cartesian differential operator

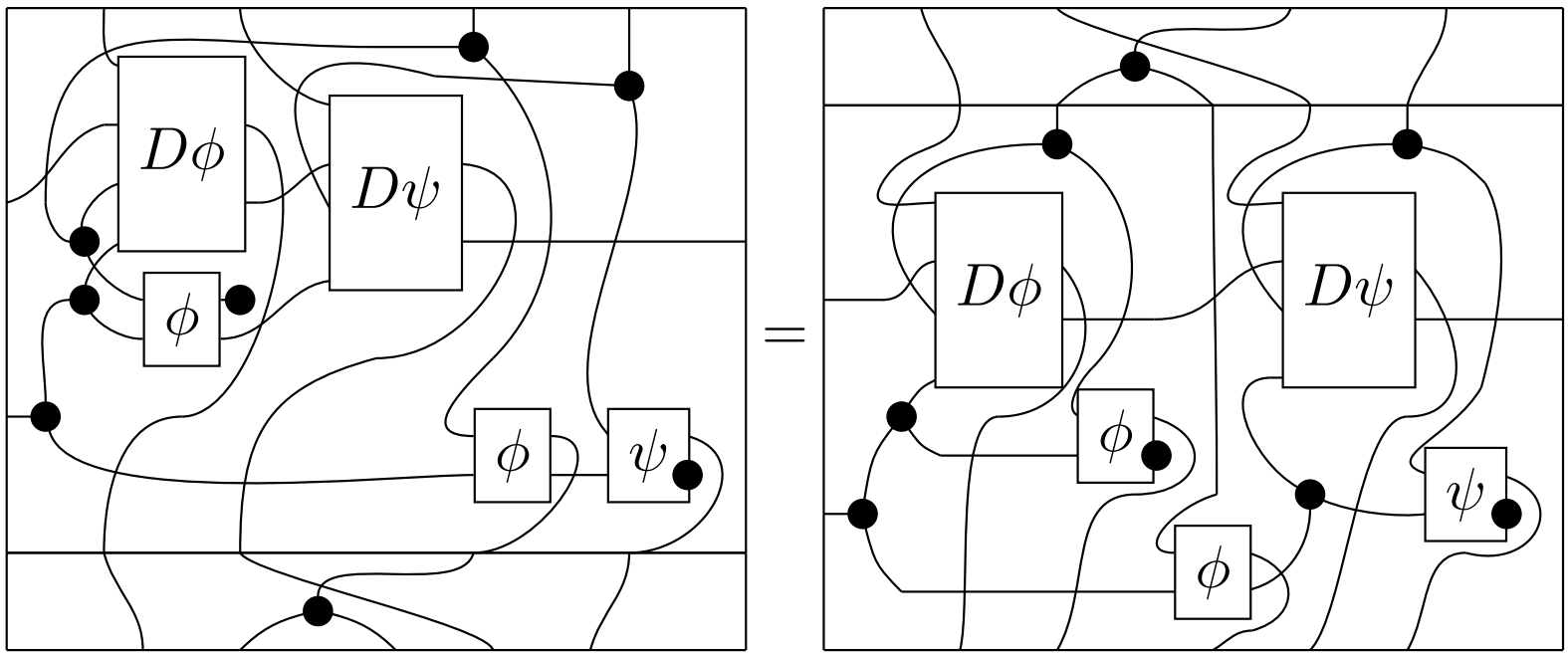
Proof idea. For CD4: $D($  $) =$ 



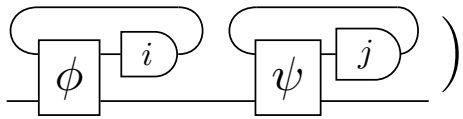
$\mathcal{D}^*$ is a Cartesian differential operator

Proof. For CD4:



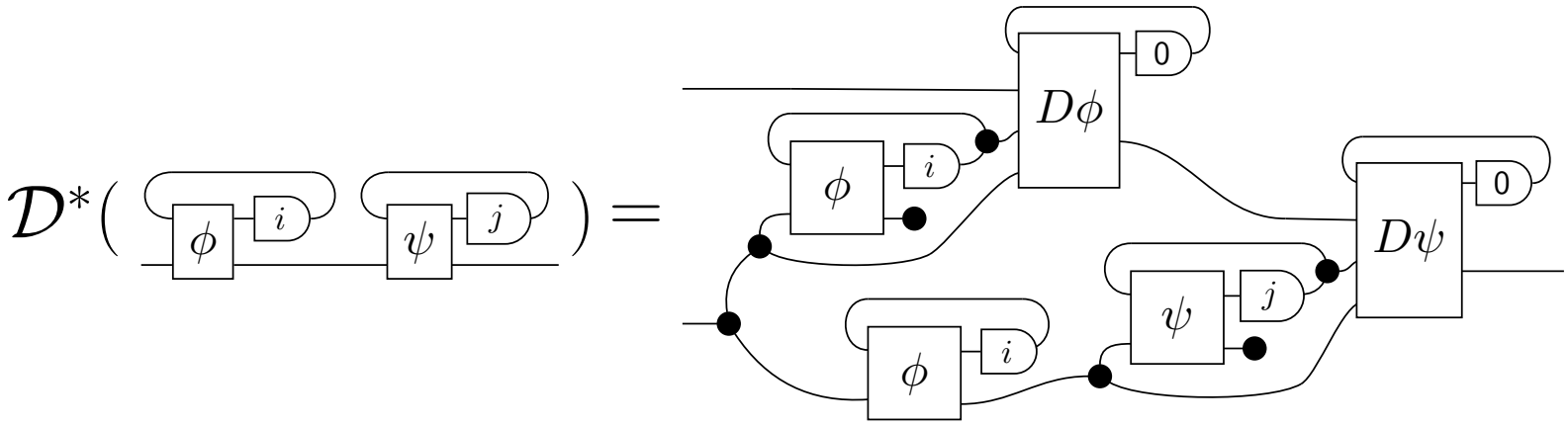


Stateful chain rule

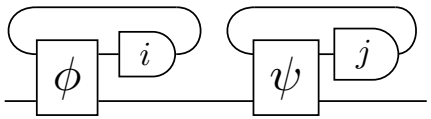
$\mathcal{D}^*($ 

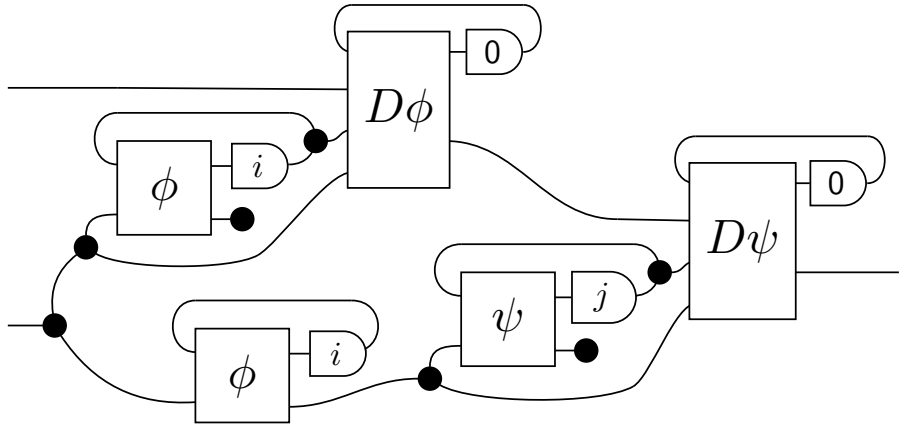
$)$

Stateful chain rule

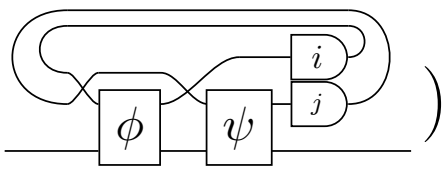


Stateful chain rule

$\mathcal{D}^*($ 

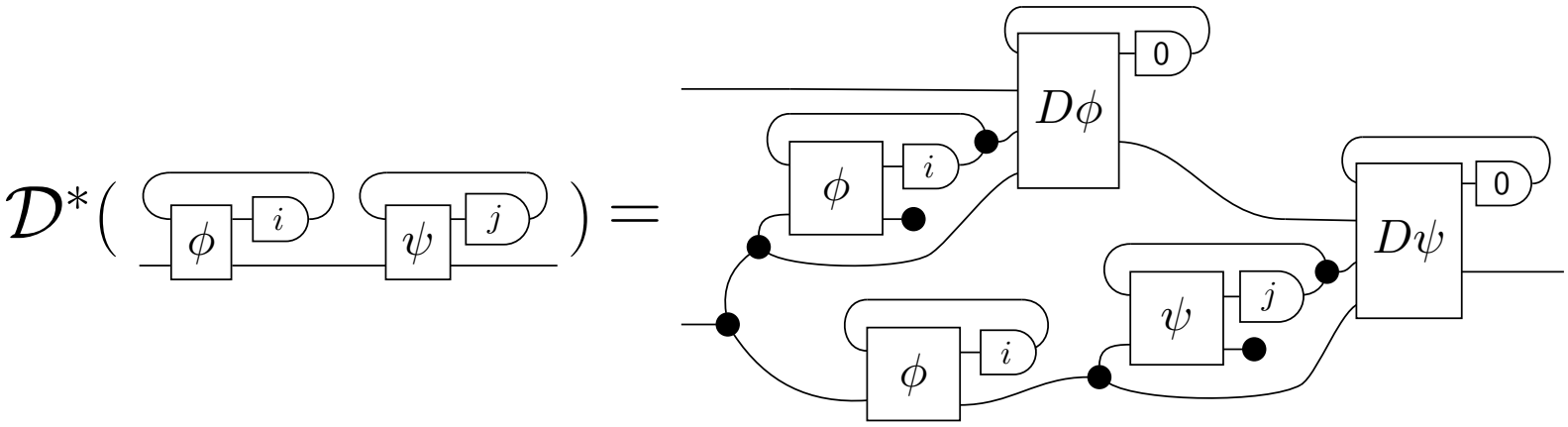
$) =$ 

$\|$

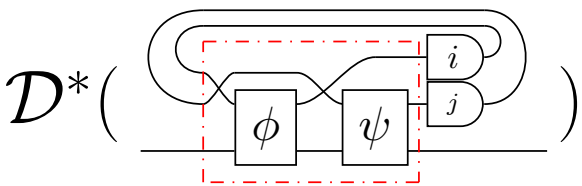
$\mathcal{D}^*($ 

$)$

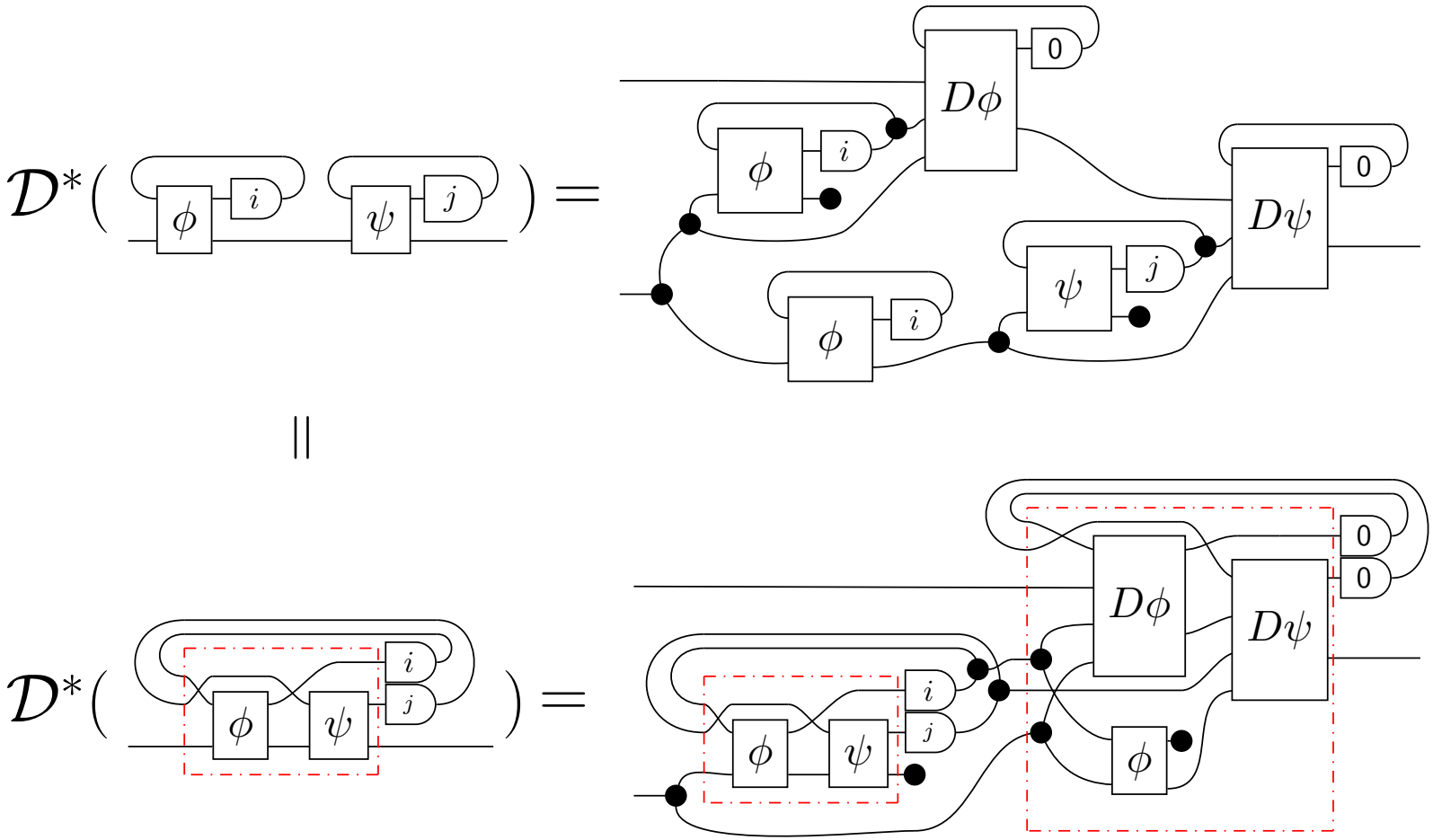
Stateful chain rule



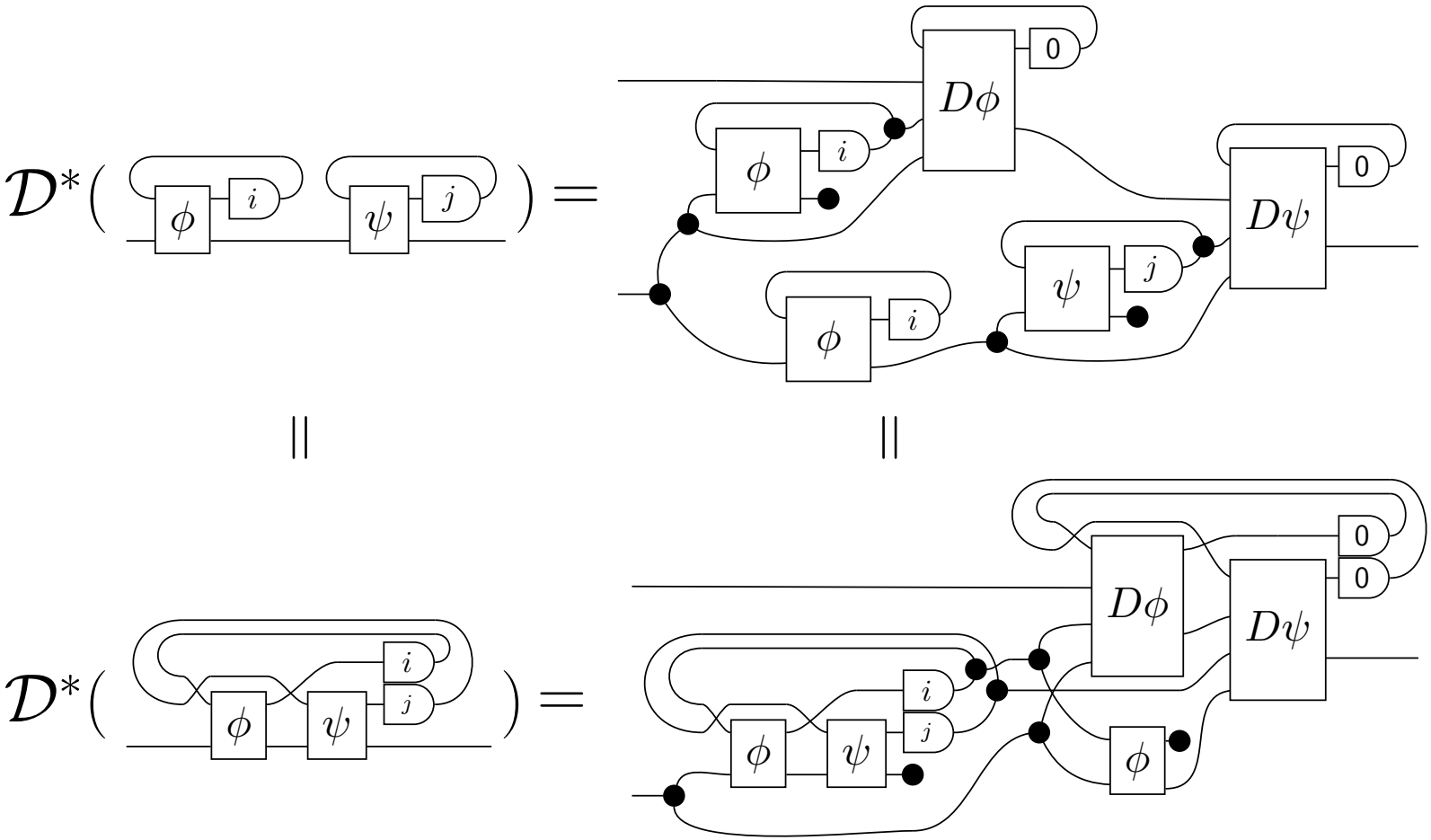
$\|$



Stateful chain rule



Stateful chain rule



Outline

A recent observation

Since the paper was published, we noticed that $\mathrm{St}(\mathbb{C})$ is a full subcategory of the functor category $[\omega^{op}, \mathbb{C}]$, namely consisting of diagrams of the form:

$$X_0 \xleftarrow[\pi_0]{} X_0 \times X_1 \xleftarrow[\pi_0]{} X_0 \times X_1 \times X_2 \xleftarrow[\pi_0]{} \dots$$

Together with natural transformations ϕ

$$\begin{array}{ccccccc} X_0 & \xleftarrow[\pi_0]{} & X_0 \times X_1 & \xleftarrow[\pi_0]{} & X_0 \times X_1 \times X_2 & \xleftarrow[\pi_0]{} & \dots \\ \downarrow{\scriptstyle \phi_0} & & \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & \\ Y_0 & \xleftarrow[\pi_0]{} & Y_0 \times Y_1 & \xleftarrow[\pi_0]{} & Y_0 \times Y_1 \times Y_2 & \xleftarrow[\pi_0]{} & \dots \end{array}$$

A recent observation

Since the paper was published, we noticed that $\mathrm{St}(\mathbb{C})$ is a full subcategory of the functor category $[\omega^{op}, \mathbb{C}]$, namely consisting of diagrams of the form:

$$X_0 \xleftarrow[\pi_0]{} X_0 \times X_1 \xleftarrow[\pi_0]{} X_0 \times X_1 \times X_2 \xleftarrow[\pi_0]{} \dots$$

Together with natural transformations ϕ

$$\begin{array}{ccccccc} X_0 & \xleftarrow[\pi_0]{} & X_0 \times X_1 & \xleftarrow[\pi_0]{} & X_0 \times X_1 \times X_2 & \xleftarrow[\pi_0]{} & \dots \\ \downarrow{\scriptstyle \phi_0} & & \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & \\ Y_0 & \xleftarrow[\pi_0]{} & Y_0 \times Y_1 & \xleftarrow[\pi_0]{} & Y_0 \times Y_1 \times Y_2 & \xleftarrow[\pi_0]{} & \dots \end{array}$$

Perhaps we can use this trick to find differential categories for more exotic network types (e.g. *recursive* neural networks)?

Future directions

Several obstacles prevent us from applying these ideas in practice right away:

1. Need non-smooth, partly differentiable, or partial functions
2. Need a transpose for computational efficiency

Future directions

Several obstacles prevent us from applying these ideas in practice right away:

1. Need non-smooth, partly differentiable, or partial functions
2. Need a transpose for computational efficiency

There are some questions related to the theory that we would like to understand better:

1. Can we add more advanced differential category structure (restriction, join, tangent, reverse, . . .)?
2. Categorical properties of $\mathrm{St}(-)$?
3. Bisimulations and extensional equality?
4. Basic results for delayed trace categories?
5. Other data shapes (trees, distributions, . . .)?

Summary

1. $\mathrm{St}(-)$ adds a delayed trace to a category.
2. $\mathrm{St}(-)$ preserves Cartesian differential category structure.
3. This notion of differentiation is connected to RNNs.
4. Cartesian differential categories are a useful tool for organizing unusual derivatives.
5. Machine learning needs compositional thinkers.

Thanks!

References


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


F. Bonchi, P. Sobociński, and F. Zanasi.
A categorical semantics of signal flow graphs.
In *CONCUR 2014*, 2014.


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


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

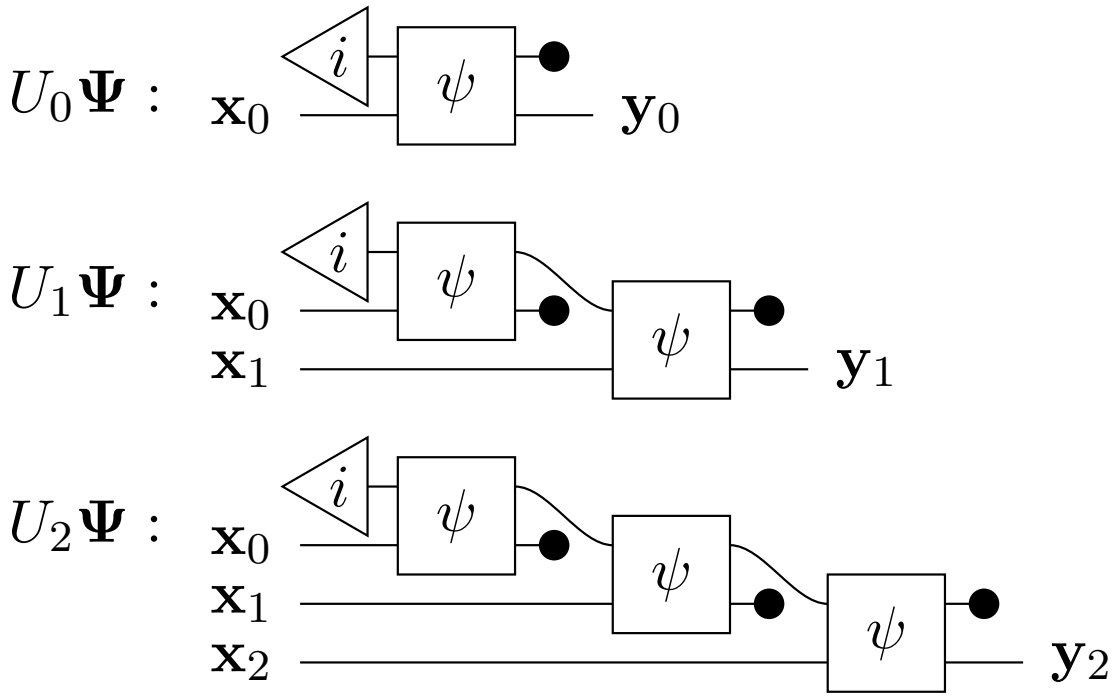

D.R. Ghica and A. Jung.
Categorical semantics of digital circuits.
FMCAD '16, Austin, TX, 2016.


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


David Sprunger and Shin-ya Katsumata.
Differentiable causal computations via delayed trace.
CoRR, abs/1903.01093, 2019.

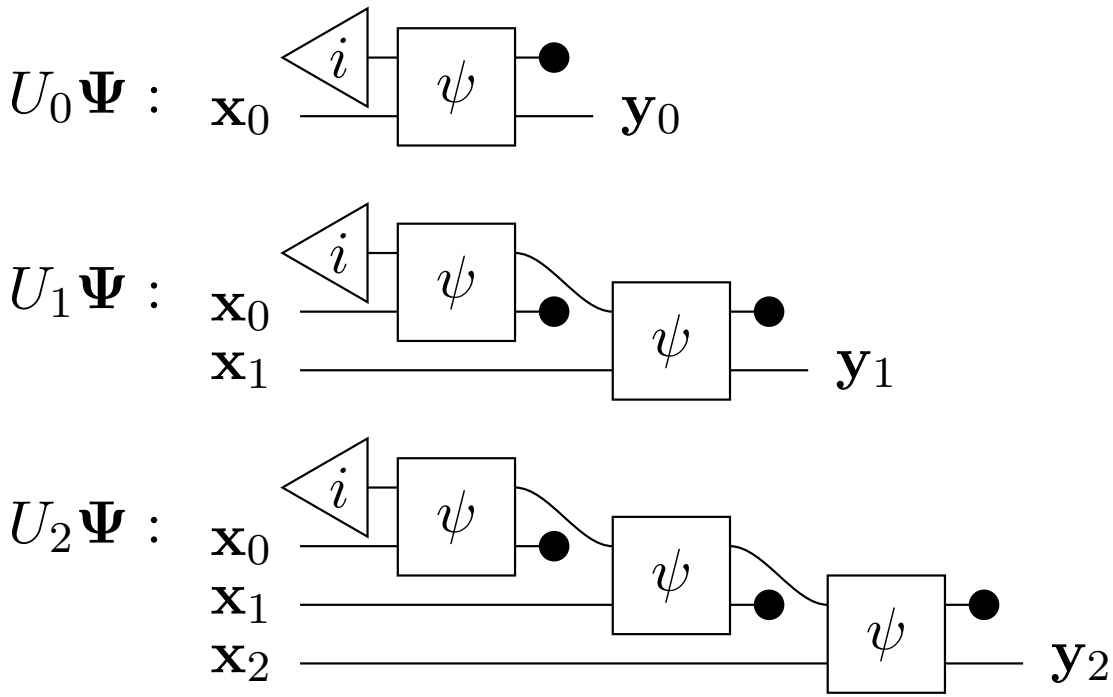
Unrollings and backpropagation through time (BPTT)

RNN training uses its *unrollings*:



Unrollings and backpropagation through time (BPTT)

RNN training uses its *unrollings*:



Backpropagation through time (BPTT, Werbos '90): Whenever the derivative of $\mathbf{\Psi}$ is needed at an input of length $k+1$, the derivative of $U_k\mathbf{\Psi}$ is used instead.

BPTT with category theory

BPTT generates good hints, but it leaves some questions:

1. $U_k(\mathbf{\Psi} \circ \mathbf{\Phi}) \neq U_k \mathbf{\Psi} \circ U_k \mathbf{\Phi}$. Did we lose the chain rule? What properties of derivatives hold for BPTT?

BPTT with category theory

BPTT generates good hints, but it leaves some questions:

1. $U_k(\mathbf{\Psi} \circ \mathbf{\Phi}) \neq U_k\mathbf{\Psi} \circ U_k\mathbf{\Phi}$. Did we lose the chain rule? What properties of derivatives hold for BPTT?
2. $U_k\mathbf{\Psi}$ and $U_{k+1}\mathbf{\Psi}$ have a lot in common, so their derivatives should as well. Is there a more compact representation for the derivative of $\mathbf{\Psi}$ than a sequence of functions?

BPTT with category theory

BPTT generates good hints, but it leaves some questions:

1. $U_k(\mathbf{\Psi} \circ \mathbf{\Phi}) \neq U_k\mathbf{\Psi} \circ U_k\mathbf{\Phi}$. Did we lose the chain rule? What properties of derivatives hold for BPTT?
2. $U_k\mathbf{\Psi}$ and $U_{k+1}\mathbf{\Psi}$ have a lot in common, so their derivatives should as well. Is there a more compact representation for the derivative of $\mathbf{\Psi}$ than a sequence of functions?

Theorem: categorical BPTT

$DU_k(\Psi) = U_k(\mathcal{D}^*\Psi) \circ z_k$ for all $k \in \mathbb{N}$ and $\Psi \in \mathrm{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$.
($z_k : (\prod X_i) \times (\prod Y_i) \to \prod (X_i \times Y_i)$ is a zipping isomorphism.)

Upshot: All the properties of derivatives in Cartesian differential categories hold for backpropagation through time!