

Toward a categorical foundation of functional reactive programming

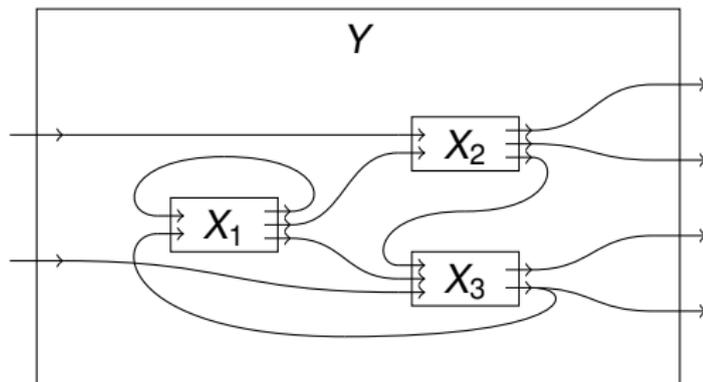
David I. Spivak

`dspivak@math.mit.edu`
Mathematics Department
Massachusetts Institute of Technology

Presented on 2014/02/19

My goal: a visual, formal language for processes

- I want to be able to draw pictures like this:

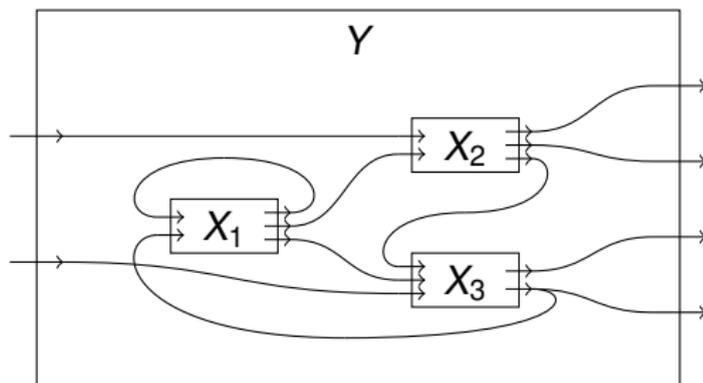


such that, if one fills in each box X_i with a machine, it results in a new machine for Y .

- And I want it all to work as expected.

What does all this mean?

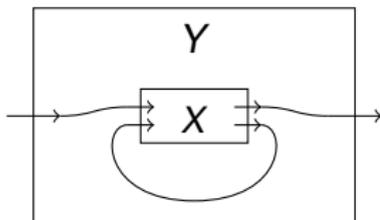
- But what *is* a picture like this?



- And what kind of machines have this fill-me-in property?
- And what expectations should we have about all this?

Plan of this talk

- I will show that wiring diagrams (WDs)



form a symmetric monoidal category (or SMC), denoted \mathbf{W} .

- I will show that there is an algebra $\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Set}$ of propagators.
- I will explain SMCs and their algebras as we go along.
- Time permitting, I'll talk about adding special symbols to the language.

Relation to Functional reactive programming

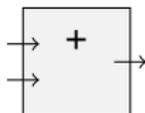
- I began this work to understand the semantics of wiring diagrams.
 - Much of this talk reflects joint work with Dylan Rupel,
 - as well as some preliminary work with Nat Stapleton.
- Disclaimers:
 - I'm new to FRP, Incremental Computation, and Haskell arrows.
 - I see them only through the lens of the above category theory, not PL.
- I hired David Darais to implement these ideas.
 - Darais told me about the relation to FRP, IC, and Haskell arrows.
 - Hopefully, he can clarify if there's a question as to the relationship.

A bit about the notation $A \rightarrow B$

- What does the notation $A \rightarrow B$ mean in this talk?
 - Type theorists write this to denote what I'd call $\text{Hom}(A, B)$ or B^A .
 - When I write $A \rightarrow B$, I'm indicating *an inhabitant* of $\text{Hom}(A, B)$.
 - I might denote this inhabitant $f: A \rightarrow B$, like a type theorist,
 - I might denote it like this: $A \xrightarrow{f} B$,
 - or if the inhabitant is clear from context I can write it $A \rightarrow B$.
- What does $A \rightarrow B \rightarrow C$ mean in this talk?
 - Type theorists write this to denote what I'd call $\text{Hom}(A, \text{Hom}(B, C))$.
 - However, when I write $A \rightarrow B \rightarrow C$, I mean that:
 - I've produced an inhabitant of $\text{Hom}(A, B)$,
 - I've produced an inhabitant of $\text{Hom}(B, C)$, and
 - I'm showing you their composite as an inhabitant of $\text{Hom}(A, C)$.

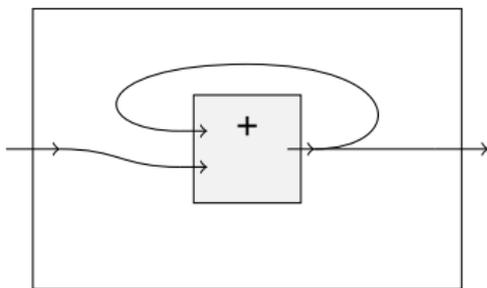
First example: a running total

- Consider the machine



which takes two integers and reports their sum.

- Installing it into the following wiring diagram

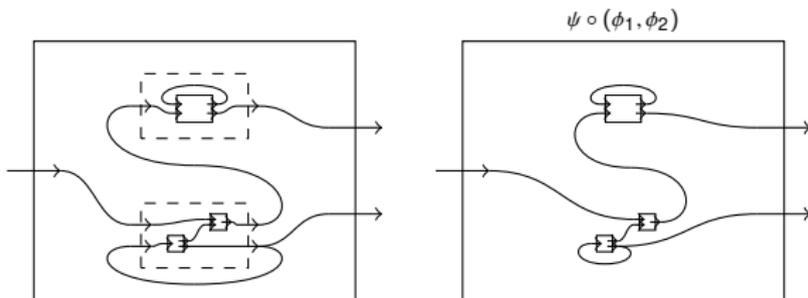
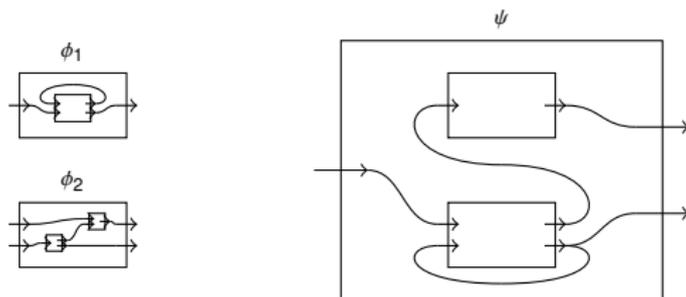


constructs a new machine for the outer box.

- The constructed machine reports a running total of its inputs.
- It carries the previous sum on the internal wire as state.

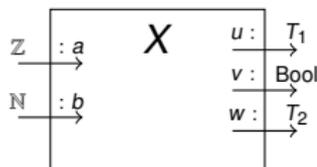
Box-manipulation language vs. box-content language

- We want to distinguish between the architecture and the contents.
 - Analogy 1: In databases, we have a schema and an instance.
 - Analogy 2: In a dependent type $\Sigma_{x:X} Q(x)$, think of X as architecture and $Q : X \rightarrow \mathbf{Type}$ as the contents.
 - Today's wording: syntax and semantics.
- A monoidal category \mathbf{W} will define the syntax of box-manipulation.
- A monoidal functor $\mathcal{P} : \mathbf{W} \rightarrow \mathbf{Set}$ will model box-contents.
 - Part of the beauty: there are many monoidal functors $\mathbf{W} \rightarrow \mathbf{Set}$.
 - In this talk we'll focus on \mathcal{P} , discrete state propagators.
 - One can ask whether continuous propagators also model this \mathbf{W} syntax.

The picture of \mathbf{W} 

Wires and boxes

- Wires carry a defined set of values.
 - A *wire* $w \in \mathbf{Set}_*$ is a pointed set $w = (T, t_0)$, where $t_0 \in T$.
 - A *finite set of wires* is a pair (I, τ) , where $I = \{i_1, \dots, i_n\}$ is a finite set, and $\tau: I \rightarrow \mathbf{Set}_*$ is a function.
 - We write **TFS** (“typed finite sets”) to denote the collection of (I, τ) ’s.
- Boxes have input wires and output wires.
 - A *box* X consists of a pair $X := (\text{inp}(X), \text{out}(X))$
 - $\text{inp}(X) \in \mathbf{TFS}$ is called the *set of input wires to* X , and
 - $\text{out}(X) \in \mathbf{TFS}$ is called the *set of output wires to* X .
 - Another term for box might be *interface*.
- Example: Box $X = (\{a : \mathbb{Z}, b : \mathbb{N}\}, \{u : T_1, v : \text{Bool}, w : T_2\})$

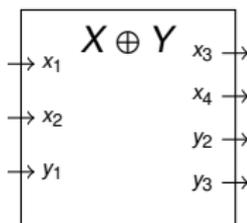


Tensor product of boxes

- Given boxes $X = (\text{inp}(X), \text{out}(X))$ and $Y = (\text{inp}(Y), \text{out}(Y))$,



we can stack them on top of each other and call that a box



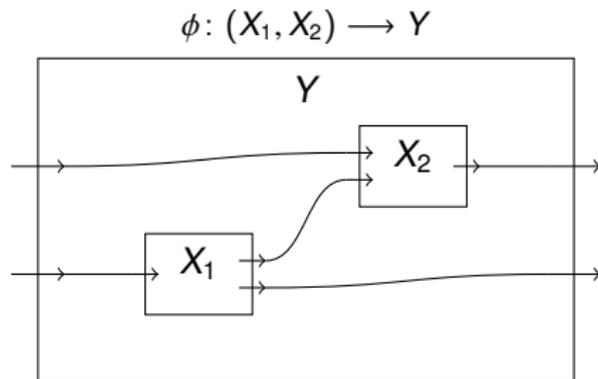
- Define the *tensor product of X and Y*, denoted $X \oplus Y$, by

$$X \oplus Y := (\text{inp}(X) + \text{inp}(Y), \text{out}(X) + \text{out}(Y)).$$
- We define the *inert box* to be $\square := (\emptyset, \emptyset)$. It is a \oplus -unit:

$$X \oplus \square \cong X \cong \square \oplus X.$$

Wiring diagrams, operad flavor: Many boxes inside

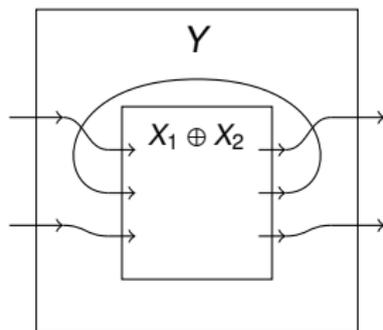
- Operads are many-inside, one-outside.
 - More precisely, morphisms in an operad have many domain objects.
 - For example $\phi: (X_1, X_2, \dots, X_n) \rightarrow Y$.
- These make for nicer, more intuitive pictures.
- If desired, one can restrict to the sub-operad of *loop-free WDs*.
 - Loop-free being a smaller syntax, it is more easily modeled.
 - For example, spreadsheets (incremental computation?).



Wiring diagrams, monoidal flavor: One box inside

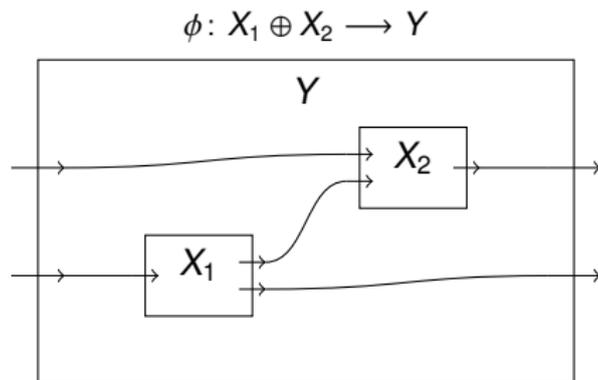
- Monoidal categories are more like regular old categories.
 - Morphisms in a monoidal category have one domain object.
 - But there's a tensor operation that serves an operad-like purpose.
 - We can have $\phi: X_1 \oplus X_2 \oplus \cdots \oplus X_n \rightarrow Y$.
- Advantages to using monoidal categories:
 - The mathematics works out cleaner for wiring diagrams.
 - More people know about monoidal categories.
- Disadvantage: the pictures can be ugly and unintuitive.
 - Here's the monoidal version of the picture from the previous slide.

$$\phi: X_1 \oplus X_2 \longrightarrow Y$$



Today's compromise: monoidal maths, operadic pictures

- In our case (with loops allowed), these two notions are equivalent.
- So we'll go with the pretty option in both cases:
 - Pretty math: symmetric monoidal categories (SMCs)
 - Pretty pictures: operads.
- We'll write $\phi: X_1 \oplus X_2 \rightarrow Y$ and allow ourselves to draw the diagram below.



Where are we now?

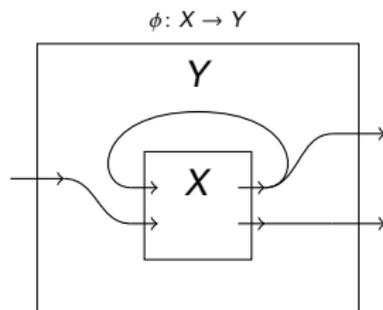
- We're on our way to defining a symmetric monoidal category \mathbf{W} .
 - I'll tell you the definition of SMC's soon.
 - For now just bear with me.
- An object $X \in \text{Ob}(\mathbf{W})$ is called a *box*.
 - Recall a box is a pair $X = (\text{inp}(X), \text{out}(X))$ of typed finite sets.
 - The coincidence of the term "object" with OOP is not bad.
 - We are trying to formalize encapsulation.
- Boxes can be tensored together by stacking them.

$$X \oplus Y = \left(\text{inp}(X) + \text{inp}(Y), \text{out}(X) + \text{out}(Y) \right)$$

- Morphisms in \mathbf{W} are wiring diagrams.
 - I showed pictures of the monoidal version and the operadic version.
 - Hopefully these pictures make intuitive sense.
 - But I haven't told you what WDs are *mathematically*.

Thinking about wiring diagrams

- Let $X = (\text{inp}(X), \text{out}(X))$ and $Y = (\text{inp}(Y), \text{out}(Y))$ be boxes.
- What is a wiring diagram?



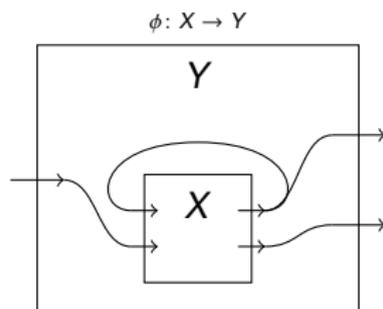
- Think of ϕ as an economy, in which every demand needs a supply.
 - The inputs of X are supplied either by inputs of Y or by internal wires.
 - Both the internal wires and the outputs of Y are sourced by X -outputs.
 - A wiring diagram expresses these relationships in terms of functions.

Mathematical formulation of wiring diagrams

Definition

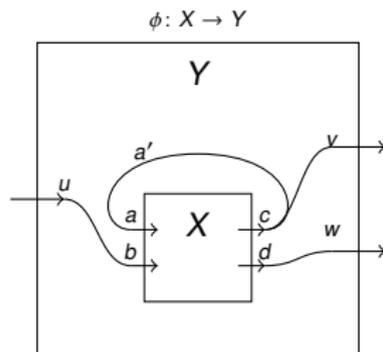
Let $X = (\text{inp}(X), \text{out}(X))$ and $Y = (\text{inp}(Y), \text{out}(Y))$ be boxes. A *wiring diagram* $\phi: X \rightarrow Y$ consists of:

- a typed finite set $\text{int}(\phi)$, called the set of *internal wires*,
- a typed function $\phi^{in}: \text{inp}(X) \rightarrow \text{int}(\phi) + \text{inp}(Y)$, and
- a typed function $\phi^{out}: \text{int}(\phi) + \text{out}(Y) \rightarrow \text{out}(X)$.



Example of a wiring diagram ($\text{int}(\phi), \phi^{in}, \phi^{out}$)

- Let X be the box with $\text{inp}(X) = \{a, b\}$ and $\text{out}(X) = \{c, d\}$.
- Let Y be the box with $\text{inp}(Y) = \{u\}$ and $\text{out}(Y) = \{v, w\}$.
- Here's a WD with internal wires $\text{int}(\phi) = \{a'\}$:



- Here's the function $\phi^{in}: \text{inp}(X) \rightarrow \text{inp}(Y) + \text{int}(\phi)$:

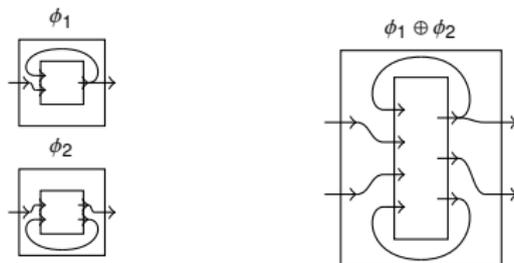
$$b \mapsto u \quad \text{and} \quad a \mapsto a'.$$

- Here's the function $\phi^{out}: \text{int}(\phi) + \text{out}(Y) \rightarrow \text{out}(X)$:

$$a' \mapsto c, \quad v \mapsto c, \quad \text{and} \quad w \mapsto d.$$

Tensor product of wiring diagrams

- Suppose given two wiring diagrams, $\phi_1: X_1 \rightarrow Y_1$ and $\phi_2: X_2 \rightarrow Y_2$.
 - Say $\phi_1 = (\text{int}(\phi_1), \phi_1^{\text{in}}, \phi_1^{\text{out}})$ and $\phi_2 = (\text{int}(\phi_2), \phi_2^{\text{in}}, \phi_2^{\text{out}})$
- To tensor morphisms, we stack them.



- As with boxes, tensor is achieved by summation across the board:

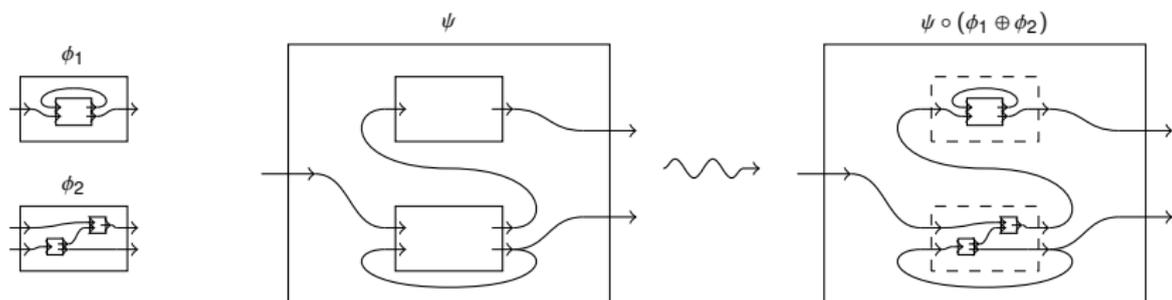
$$\text{int}(\phi_1 \oplus \phi_2) = \text{int}(\phi_1) + \text{int}(\phi_2),$$

$$(\phi_1 \oplus \phi_2)^{\text{in}} = \phi_1^{\text{in}} + \phi_2^{\text{in}},$$

$$(\phi_1 \oplus \phi_2)^{\text{out}} = \phi_1^{\text{out}} + \phi_2^{\text{out}},$$

Composing wiring diagrams

- We want to be able to plug wiring diagrams into wiring diagrams.



- Quiz: what are the internal wires of $\psi \circ (\phi_1 \oplus \phi_2)$?

Composing wiring diagrams, $X \xrightarrow{\phi} Y \xrightarrow{\psi} Z$

- Recall that each wiring diagram, say ϕ , consists of
 - a typed finite set of internal wires $\text{int}(\phi)$,
 - a typed function $\phi^{\text{in}}: \text{inp}(X) \rightarrow \text{int}(\phi) + \text{inp}(Y)$, and
 - a typed function $\phi^{\text{out}}: \text{int}(\phi) + \text{out}(Y) \rightarrow \text{out}(X)$.
- The internal wires of $\psi \circ \phi$ are $\text{int}(\psi \circ \phi) := \text{int}(\phi) + \text{int}(\psi)$.
- The function $(\psi \circ \phi)^{\text{in}}: \text{inp}(X) \rightarrow \text{int}(\psi \circ \phi) + \text{inp}(Z)$ is given by

$$\text{inp}(X) \xrightarrow{\phi^{\text{in}}} \text{int}(\phi) + \text{inp}(Y) \xrightarrow{\text{int}(\phi) + \psi^{\text{in}}} \text{int}(\phi) + \text{int}(\psi) + \text{inp}(Z).$$
- The function $(\psi \circ \phi)^{\text{out}}: \text{int}(\psi \circ \phi) + \text{out}(Z) \rightarrow \text{out}(X)$ is given by

$$\text{int}(\phi) + \text{int}(\psi) + \text{out}(Z) \xrightarrow{\text{int}(\phi) + \psi^{\text{out}}} \text{int}(\phi) + \text{out}(Y) \xrightarrow{\phi^{\text{out}}} \text{out}(X).$$

\mathbf{W} is a symmetric monoidal category

- Let's recap what we know about \mathbf{W} .
- First of all, \mathbf{W} is a category:
 - We defined an object of \mathbf{W} to be a box (a pair of typed finite sets).
 - We defined a morphism $\phi: X \rightarrow Y$ in \mathbf{W} to be a wiring diagram,

$$(\text{int}(\phi), \phi^{in}, \phi^{out}).$$

- On the last slide we showed the composition formula for $\psi \circ \phi$.
 - The identity (having $\text{int}(\text{id}_X) = \emptyset$) is straightforward.
 - Proving the associativity law is straightforward too.
 - So we indeed have a category.
- Add a tensor product to that, and we have an SMC.
- The tensor product needs to satisfy some laws:
 - For example, we need $X \oplus Y \cong Y \oplus X$.
 - Another example: $\square \oplus X \cong X \cong X \oplus \square$.
 - But these are all straightforward, because we're just working with finite sets and their sums.

What is a symmetric monoidal category

- A symmetric monoidal category consists of
 - a category \mathcal{M} ,
 - a functor $\otimes: \mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$, called the *tensor*,
 - an object $I \in \text{Ob}(\mathcal{M})$ called the *unit*,
 - as well as various *coherence isomorphisms* and commutative diagrams that ensure that everything works as expected, e.g.
 - $X \otimes I \cong X \cong IX$,
 - $(X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$, etc.
- Your favorite: **Type** with Cartesian \times , and unit type 1.
- Another: **Set** with disjoint union $+$, and unit set \emptyset .
- Another: **Vect** $_{\mathbb{R}}$ with tensor product \otimes , and unit vector space \mathbb{R} .
- Another: **W** with stacking tensor \oplus , and inert box \square .

Quick aside: how is \otimes different than \times ?

- Some people want to know how \otimes is different than Cartesian product.
- Note that $(\mathbf{Set}, \times, 1)$ is an SMC, so we must be saying SMCs are more general, i.e. that \times is more constrained than arbitrary \otimes .
 - The additional constraint on \times is that you can project,

$$A \longleftarrow A \times B \longrightarrow B.$$

- Note that $(\mathbf{Set}, +, 0)$ is an SMC, but there is no canonical map $A + B \rightarrow B$.

So... what to plug into these boxes?

- We have this syntax of boxes; what are we going to do with it?
 - “Let’s put stuff into them!”
 - “What can we put into them?”
 - “Whatever we want, as long as we understand stacking and wiring.”
- A W -algebra is a lax monoidal functor

$$F: \mathbf{W} \rightarrow \mathbf{Set}.$$

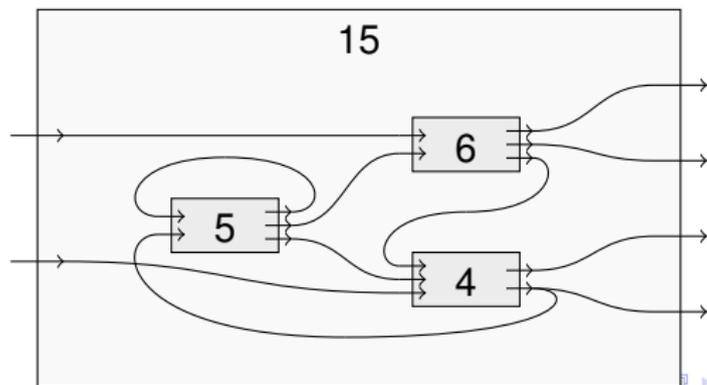
- To choose a W -algebra F is to choose semantics for the box syntax.

What is a lax monoidal functor $F: \mathbf{W} \rightarrow \mathbf{Set}$?

- Suppose we want to choose semantics F for this box syntax.
- We get to choose what we allow ourselves to put into the boxes.
 - For a box $X \in \text{Ob}(\mathbf{W})$ we get to choose a set $F(X)$.
 - Once we've done so, we'll call $f \in F(X)$ an F -fill for box X .
- We get to say how to stack F -fills.
 - Given boxes X, Y and F -fills $f \in F(X)$ and $g \in F(Y)$,
 - we need to give an F -fill for their tensor, $\sigma(f, g) \in F(X \oplus Y)$.
- We get to say how a wiring diagram $\phi: X \rightarrow Y$ sends fills for X to fills for Y .
- Once we do that, we will have specified:
 - a function $\text{Ob}(F): \text{Ob}(\mathbf{W}) \rightarrow \text{Ob}(\mathbf{Set})$,
 - a function $\sigma_{X,Y}: F(X) \times F(Y) \rightarrow F(X \oplus Y)$, and
 - a function $\text{Hom}_F: \text{Hom}_{\mathbf{W}}(X, Y) \rightarrow \text{Hom}_{\mathbf{Set}}(F(X), F(Y))$.
- For our choices to constitute a \mathbf{W} -algebra, various laws must hold.

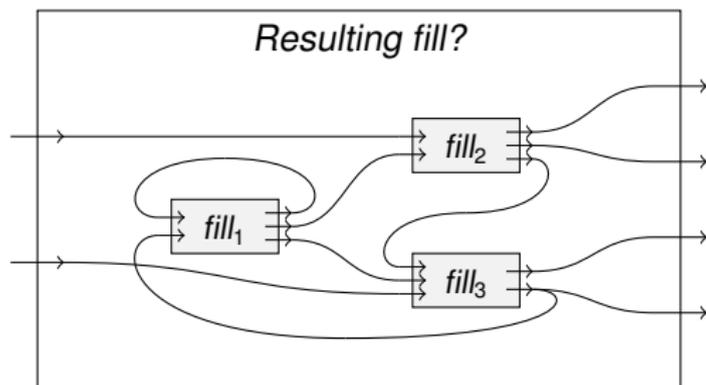
Some stupid W -algebras

- Let $\mathcal{M} = (M, \star, e)$ be any commutative monoid.
 - For example the natural numbers, with addition, $(\mathbb{N}, +, 0)$,
 - or the integers, with multiplication, $(\mathbb{Z}, *, 1)$,
 - or the subsets of some set, with union, $(\mathbb{P}(\{0, 1, \dots, 9\}), \cup, \emptyset)$.
- Then there is an algebra $F: \mathbf{W} \rightarrow \mathbf{Set}$ that assigns
 - $F(X) := M$,
 - $\sigma := \star: M \times M \rightarrow M$, and
 - $\text{Hom}_F(\phi) := \text{id}_M$.
- For example, with $M = (\mathbb{N}, +, 0)$, we have



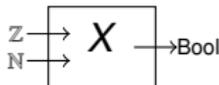
More interesting algebras $W \rightarrow \text{Set}$

- The previous algebras didn't take advantage of the wiring structure.
- We will focus on propagators, taking input-streams to output-streams.
- Variations include:
 - asking the propagators to be continuous or differentiable.
 - continuous-time machines, etc.
- In each case, just say what to put into boxes and how stacking and wiring are to work.

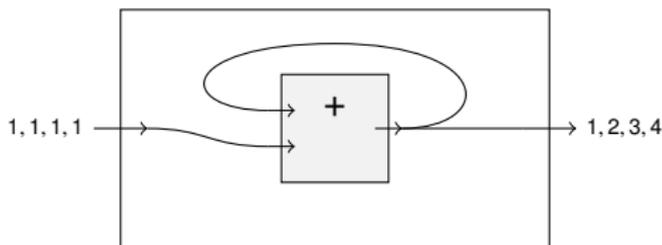


A questionable algebra

- One idea might be to put into each box the set of functions of the specified type.
 - That is, suppose X is the box below.
 - Define $\mathcal{F}(X) = \text{Hom}(\mathbb{Z} \times \mathbb{N}, T1 \times \text{Bool} \times T2)$, the set of functions.



- But then how do wiring diagrams operate on functions?
- Recall the running total.
 - It is made out of a pure function, but the result is not functional.
 - The same input in two successive moments returns different outputs.



State propagators

Definition

Let A and B be sets. An (A, B) -propagator consists of

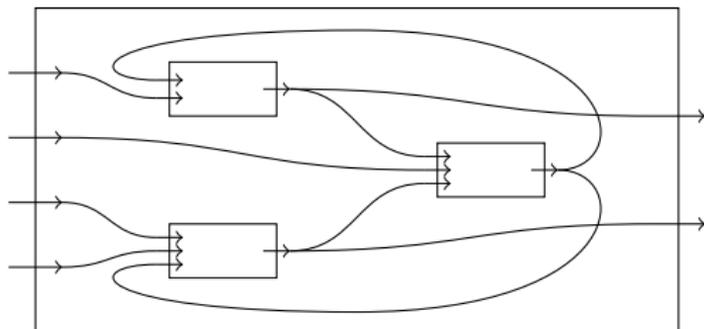
1. a set S , called the *state-set*,
2. a function $f: S \times A \rightarrow S \times B$, called the *propagation function*.

An (A, B) -propagator is called *initialized* if we have chosen

3. an element $s_0 \in S$, called the *initial state*.

We call a propagator (S, f) *simple* if its state-set has one element, $|S| = 1$.

Motivation for state propagators



- My motivation: how does the brain work?
 - The architecture of the brain is of neurons with dendrites (inputs) and axons (outputs)
 - How does this architecture form a mind, i.e. something that can think?
 - What about learning, habituation, sensitization?
- The propagator model may also have applications to FRP, IC, etc, because it was designed with computation in mind.

Aside: Initialized propagators act on lists

- Let (S, s_0, f) be an initialized (A, B) -propagator, where $s_0 \in S$.
- For convenience, swap the propagation function's outputs:

$$f: S \times A \longrightarrow B \times S.$$

- For $n \in \mathbb{N}$, we define $f_n: A^n \rightarrow B^n \times S$, as follows:
 - define $f_0 = s_0$, the initial state, and
 - define $f_{n+1}: A^{n+1} \rightarrow B^{n+1} \times S$ to be the composite

$$A^n \times A \xrightarrow{f_n \times A} B^n \times S \times A \xrightarrow{B^n \times f} B^n \times B \times S$$

- Project each $f_n: A^n \rightarrow B^n \times S$ and then sum the results to obtain

$$\text{LP}(S, s_0, f): \text{List}(A) \longrightarrow \text{List}(B),$$

called the *list propagator associated to* (S, s_0, f) .

Fill box X with the set of \overline{X} -propagators

- Quick aside on dependent products: notation and contravariance.
 - Given a typed finite set (I, τ) we denote the dependent product by

$$\overline{(I, \tau)} := \prod_{i \in I} \tau(i).$$

- This is contravariant: given a typed function $p: (I, \tau) \rightarrow (I', \tau')$ we get

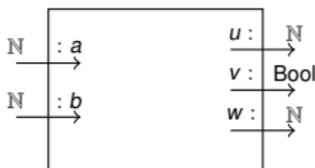
$$\overline{p}: \overline{(I', \tau')} \rightarrow \overline{(I, \tau)}.$$

- Recall that a box $X = (\text{inp}(X), \text{out}(X))$ is a pair of typed finite sets.
 - For example, if $\text{inp}(X) = \{a : \mathbb{Z}, b : \text{Bool}\}$, then $\overline{\text{inp}(X)} = \mathbb{Z} \times \text{Bool}$.
 - Define $\overline{X} := (\overline{\text{inp}(X)}, \overline{\text{out}(X)})$.
- So an \overline{X} -propagator includes a state-set S and a propagation function

$$f: S \times \overline{\text{inp}(X)} \longrightarrow S \times \overline{\text{out}(X)}.$$

$\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Set}$ on objects

- On boxes $X \in \mathbf{Ob}(\mathbf{W})$, define $\mathcal{P}(X)$ to be the set of \overline{X} -propagators,
- For example, let $X = (\{a : \mathbb{N}, b : \mathbb{N}\}, \{u : \mathbb{N}, v : \mathbf{Bool}, w : \mathbb{N}\})$,



- Choosing an initialized \overline{X} -propagator means:
 - choosing a state set S , an initial state $s_0 \in S$, and a function,

$$f: S \times (\mathbb{N} \times \mathbb{N}) \rightarrow S \times (\mathbb{N} \times \mathbf{Bool} \times \mathbb{N}).$$

- For example, let's choose $S = \mathbb{N} \times \mathbb{N}$, with $s_0 = (0, 0)$, and

$$f((s_1, s_2), a, b) = ((s_1 + a, s_2 + b), (s_1, s_1 \stackrel{?}{=} s_2, s_2)).$$

- This returns running totals of a and b , as well as whether they're equal.

Stacking propagators

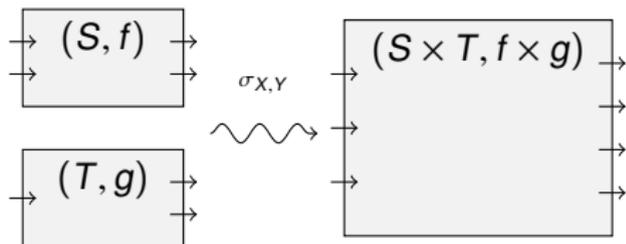
- Recall an \overline{X} -propagator consists of a set S and a function

$$f: S \times \overline{\text{inp}(X)} \rightarrow S \times \overline{\text{out}(X)}.$$

- For any two boxes $X, Y \in \text{Ob}(\mathbf{W})$, we need a stacking function

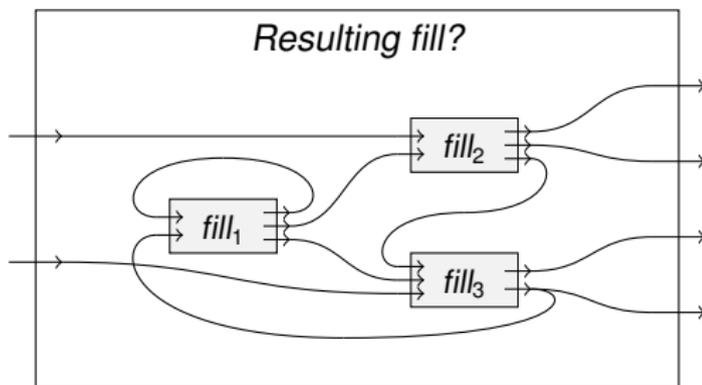
$$\sigma_{X,Y}: \mathcal{P}(X) \times \mathcal{P}(Y) \rightarrow \mathcal{P}(X \oplus Y).$$

- Given an \overline{X} -propagator (S, f) and a \overline{Y} -propagator (T, g) , we need a $\overline{X \oplus Y}$ -propagator.
- We use $\sigma_{X,Y}((S, f), (T, g)) := (S \times T, f \times g)$.



Wiring propagators

- We've decided how $\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Set}$ works on boxes $X \in \text{Ob}(\mathbf{W})$;
- We've decided how \mathcal{P} works with stacking.
- Now we need to decide how \mathcal{P} works with wiring diagrams.



- Afterwards we need to check that the composition formula holds.

$\mathcal{P}(\phi): \mathcal{P}(X) \longrightarrow \mathcal{P}(Y)$

- We begin with boxes X and Y , and a wiring diagram $\phi: X \rightarrow Y$.
- Recall that each wiring diagram, say ϕ , consists of
 - a typed finite set of internal wires $\text{int}(\phi)$,
 - a typed function $\phi^{\text{in}}: \text{inp}(X) \rightarrow \text{int}(\phi) + \text{inp}(Y)$, and
 - a typed function $\phi^{\text{out}}: \text{int}(\phi) + \text{out}(Y) \rightarrow \text{out}(X)$.
- Recall the contravariance of dependent products, e.g.

$$\overline{\phi^{\text{in}}}: \overline{\text{int}(\phi)} \times \overline{\text{inp}(Y)} \longrightarrow \overline{\text{inp}(X)}.$$

- Suppose given an \overline{X} -propagator $(S, f) \in \mathcal{P}(X)$, where

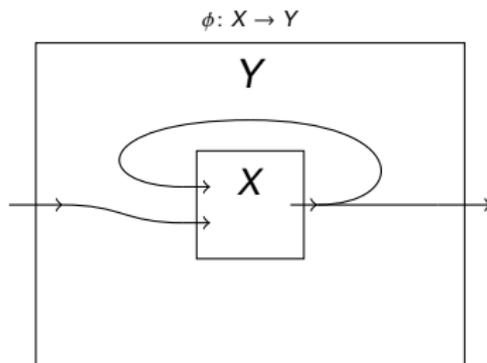
$$f: S \times \overline{\text{inp}(X)} \longrightarrow S \times \overline{\text{out}(X)}.$$

- We need to define a \overline{Y} -propagator $(T, g) = \mathcal{P}(\phi)(S, f) \in \mathcal{P}(Y)$.
 - For the new state-set, use the product, $T := S \times \overline{\text{int}(\phi)}$.
 - For the new propagation function, use the composite,

$$S \times \overline{\text{int}(\phi)} \times \overline{\text{inp}(Y)} \xrightarrow{S \times \overline{\phi^{\text{in}}}} S \times \overline{\text{inp}(X)} \xrightarrow{f} S \times \overline{\text{out}(X)} \xrightarrow{S \times \overline{\phi^{\text{out}}}} S \times \overline{\text{int}(\phi)} \times \overline{\text{out}(Y)}.$$

Example wiring diagram $\phi: X \rightarrow Y$

- Let all wires carry the pointed type $(\mathbb{N}, 0)$.
- Note that there is one internal wire, so $\overline{\text{int}(\phi)} = \mathbb{N}$.



- Consider the \bar{X} -propagator $(\{*\}, +)$, where $+: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is sum.
- Then $\mathcal{P}(\phi)(\{*\}, +) = (\mathbb{N}, f)$ has propagation function given by

$$f(s, y) = (s + y, s + y)$$

- As a list propagator, it reports the running total as advertised,
 $(y_1, \dots, y_n) \mapsto \sum_{i=1}^n y_i$.

Checking \mathcal{P} on the composition $X \xrightarrow{\phi} Y \xrightarrow{\psi} Z$

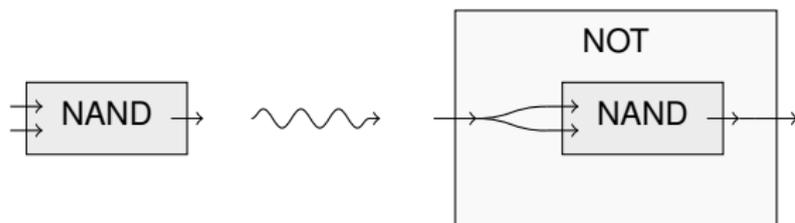
- We have defined $\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Set}$ on objects, morphisms, and stacking.
- We must check that it works well with composition.
- The computation is very straightforward:

$$\begin{array}{ccccc}
 S \times \overline{\text{int}(\phi)} \times \overline{\text{int}(\psi)} \times \overline{\text{inp}(Z)} & \xrightarrow{S \times \overline{\text{int}(\phi)} \times \overline{\psi}^{\text{in}}} & S \times \overline{\text{int}(\phi)} \times \overline{\text{inp}(Y)} & \xrightarrow{S \times \overline{\phi}^{\text{in}}} & S \times \overline{\text{inp}(X)} \\
 \downarrow \mathcal{P}(\psi \circ \phi)(f) & & \downarrow \mathcal{P}(\phi)(f) & & \downarrow f \\
 S \times \overline{\text{int}(\phi)} \times \overline{\text{int}(\psi)} \times \overline{\text{out}(Z)} & \xleftarrow{S \times \overline{\text{int}(\phi)} \times \overline{\psi}^{\text{out}}} & S \times \overline{\text{int}(\phi)} \times \overline{\text{out}(Y)} & \xleftarrow{S \times \overline{\phi}^{\text{out}}} & S \times \overline{\text{out}(X)}
 \end{array}$$

- I show you this not because it's hard, but because it's easy.
 - We worked hard to make this as simple as possible.
 - Our goal was to have something people would want to use!

The subalgebra generated by NANDs?

- Each transistor on a chip acts as a NAND gate, a simple propagator.



- From here we can get NOT gates, then AND gates, and all logic gates.
- Then n -bit adders, multiplication circuits, etc.
- Consider the box $T := (\{a, b : Bool\}, \{c : Bool\}) \in \text{Ob}(\mathbf{W})$.
 - Begin with the *free algebra on T* , denoted $Fr(T) : \mathbf{W} \rightarrow \mathbf{Set}$.
 - It is the algebra that sends X to $\sum_{n \in \mathbb{N}} \text{Hom}_{\mathbf{W}}(T^{\oplus n}, X)$.
 - Now, there's a unique map $Fr(T) \rightarrow \mathcal{P}$, sending $T \mapsto \text{NAND}$.
 - Its image defines *the algebra of propagators generated by NAND*.
- Question: How does it compare to the computable functions?

Morphisms of propagators

- Let A and B be sets.
- Suppose we have two (A, B) -propagators, (S, f) and (T, g) .
- A *morphism of propagators from (S, f) to (T, g)* consists of:
 - a function $\rho: S \rightarrow T$,
 - such that the following diagram commutes:

$$\begin{array}{ccc}
 S \times A & \xrightarrow{f} & S \times B \\
 \rho \times A \downarrow & & \downarrow \rho \times B \\
 T \times A & \xrightarrow{g} & T \times B
 \end{array}$$

- If we're working with initialized propagators, we require $\rho(s_0) = t_0$.
- We want brains/manufacturers to reduce the complexity of their problem.

Connected propagators act the same on lists

- Let (S, s_0, f) be an initialized (A, B) -propagator.
 - Recall: for each $n \in \mathbb{N}$, it induces a function $A^n \rightarrow B^n$, and
 - their sum is a function $\text{LP}(S, s_0, f): \text{List}(A) \rightarrow \text{List}(B)$.
- Suppose given a morphism $\rho: (S, s_0, f) \rightarrow (T, t_0, g)$ of propagators.
- In this case it is easy to show that $\text{LP}(S, s_0, f) = \text{LP}(T, t_0, g)$.
- So if two propagators are connected, they act the same on lists.
 - We write $(S, s_0, f) \sim (T, t_0, g)$ if they are connected by a zigzag.
 - (Aside: zigzags are chains like this, $P_0 \leftarrow P_1 \rightarrow P_2 \leftarrow \cdots \rightarrow P_n$.)
 - The relation \sim is an equivalence relation on \overline{X} -propagators.

List(A) can always serve as state-set

- Let (S, s_0, f) be an initialized (A, B) -propagator.
 - For each $n \in \mathbb{N}$, it induces a function $f_n: A^n \rightarrow S \times B^n$.
 - For convenience, we give names to its first and last projections,

$$\sigma_n: A^n \rightarrow S \quad \text{and} \quad \omega_{n+1}: A^{n+1} \rightarrow B.$$

- We'll find an equivalent propagator with state-set $\text{List}(A)$.
 - Let $T = \text{List}(A)$ and let $t_0 = []$ be the empty list.
 - We need a propagation function $\widehat{f}: T \times A \rightarrow T \times B$.
 - It's sufficient to provide $\widehat{f}_n: A^n \times A \rightarrow A^{n+1} \times B$ for every $n \in \mathbb{N}$.
 - Use the top row in the diagram below.

$$\begin{array}{ccc}
 A^n \times A & \xlongequal{\quad} & A^{n+1} \xrightarrow{(A^{n+1}, \omega_{n+1})} A^{n+1} \times B \\
 \sigma_n \times A \downarrow & & \downarrow \sigma_{n+1} \times B \\
 S \times A & \xrightarrow{\quad f \quad} & S \times B
 \end{array}$$

- The rest of the diagram shows the morphism $(T, t_0, \widehat{f}) \rightarrow (S, s_0, f)$.

State reduction

- For any (A, B) -propagator (S, s_0, f) we found a morphism

$$\rho: (\text{List}(A), [], \widehat{f}) \longrightarrow (S, s_0, f).$$

- In fact ρ is unique.
- The image of ρ is some (S', s_0, f) having a subset of states $S' \subseteq S$.
- S' is the set of *reachable states*, those that obtain on some list of input.
- We can also quotient by an equivalence relation on states.
 - Declare two states equivalent if they act the same on any input list.
 - We have $\text{LP}(S, -, f): S \longrightarrow \text{List}(B)^{\text{List}(A)}$.
 - Let \widetilde{S} be its image, so we have $q: S \twoheadrightarrow \widetilde{S} \subseteq \text{List}(B)^{\text{List}(A)}$.
 - So \widetilde{S} is the quotient of S by the equivalence relation.
 - It is easy to show that \widetilde{S} is the state-set for an equivalent propagator.

$$\begin{array}{ccc} S \times A & \xrightarrow{f} & S \times B \\ q \times A \downarrow & & \downarrow q \times B \\ \widetilde{S} \times A & \xrightarrow{\widetilde{f}} & \widetilde{S} \times B \end{array}$$

Algorithmic state reduction

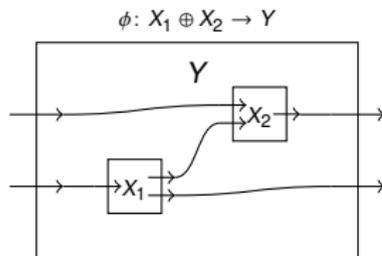
- Given an (A, B) -propagator, we want the smallest equivalent one.
 - If (S, s_0, f) is such that every state is reachable, use $(\widetilde{S}, s_0, \widetilde{f})$.
 - In this case, and if A and S are finite, Hopcroft's algorithm finds the smallest equivalent propagator $(\widetilde{S}, s_0, \widetilde{f})$ in $O(|S||A|/\log|S|)$ time.
 - If some states are not reachable, use $(\widetilde{\text{List}(A)}, [], \widetilde{f})$.
- Call this the *minimal reduction* of (S, s_0, f) .
- It is a normal form for propagators.

State reduction and wiring diagrams

- Back to the main theme, we had $\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Set}$.
- But in fact it can be extended to a monoidal functor $\mathcal{P}: \mathbf{W} \rightarrow \mathbf{Cat}$.
 - For each $X \in \text{Ob}(\mathbf{W})$ we now have a category $\mathcal{P}(X)$ of propagators.
 - For stacking boxes, there's a functor $\mathcal{P}(X) \times \mathcal{P}(Y) \rightarrow \mathcal{P}(X \oplus Y)$.
 - For each WD $\phi: X \rightarrow Y$ there's a functor $\mathcal{P}(\phi): \mathcal{P}(X) \rightarrow \mathcal{P}(Y)$.
 - And these all work together as required.
- This means that reducing commutes with wiring.
 - Given a morphism $\phi: X \rightarrow Y$ and a propagator $P \in \mathcal{P}(X)$,
 - you can reduce $P \twoheadrightarrow P'$ then apply $\mathcal{P}(\phi)$,
 - and the result is a reduction, $\mathcal{P}(\phi)(P) \twoheadrightarrow \mathcal{P}(\phi)(P')$.
- Note that this does not say much about minimal reduction.

Minimal reduction for wiring diagrams?

- Suppose given a wiring diagram $\phi: X_1 \oplus X_2 \rightarrow Y$.



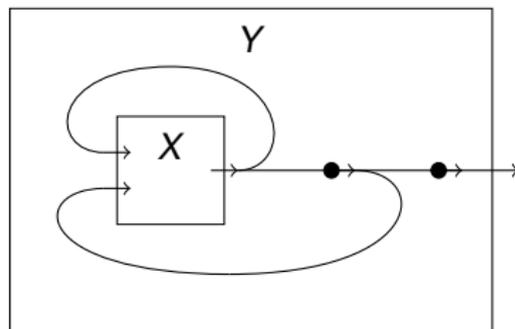
- And suppose given propagators $P_1 \in \mathcal{P}(X_1)$ and $P_2 \in \mathcal{P}(X_2)$.
- You can find minimal reductions of each, but it's a waste of time.
 - If P_1 only spits out one value, it effectively reduces P_2 's input set.
- Question: find an efficient algorithm to minimize $\mathcal{P}(\phi)(P_1, P_2)$?
 - Can you take advantage of the knowledge that it has this form?
 - Perhaps use machine learning to “atrophy” unused expressivity in the wires?

How's the time?

Shall we change gears a bit, or skip to the end?

Wiring diagrams as a visual language

- One major feature of wiring diagrams is to engage the human visual system.
 - Operadic pictures are a visual language for building instructions.
 - The category \mathbf{W} purely syntactic.
- We can build predefined functions into \mathbf{W} .
 - For example, delay propagators might be denoted by nodes \bullet .

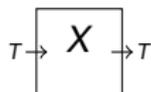


- Or propagators that bundle four wires into a bus might be denoted by

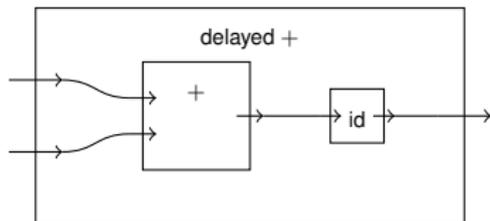


Aside: timing in a wiring diagram

- The formulas are written above; here we interpret them in terms of timing.
 - Wires move data instantaneously.
 - Each propagator takes one “clock-cycle” to process data.
- Consider a box X with one input wire and one output wire, $\text{inp}(X) = \{T\} = \text{out}(X)$ of the same type, T .

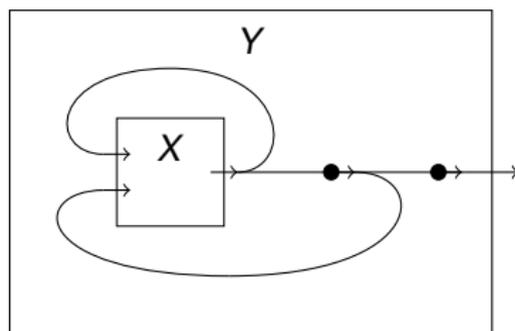


- We define the *delay propagator of type T* to be the simple propagator with propagation function $\text{id}_T: T \rightarrow T$.



Baking in special propagators

- What does it mean to bake the delay node \bullet , etc., into **W**?
- We want the following to count as a wiring diagram $\phi: X \rightarrow Y$.



- That is, we name special boxes for which we have chosen interpretations.
- What's the math?

The math for baking in special symbols, part 1

- We need to choose special symbols in \mathbf{W} and propagators for them.
 - Fix a SMC, \mathcal{S} , objects of which are called *special symbols*.
 - Fix a strong monoidal functor $\iota: \mathcal{S} \rightarrow \mathbf{W}$.
 - For each symbol $s \in \text{Ob}(\mathcal{S})$ choose an element $m_s \in \mathcal{P}(\iota(s))$.

$$\begin{array}{ccc}
 \mathcal{S} & \xrightarrow{\iota} & \mathbf{W} \\
 \downarrow ! & m_s \nearrow & \downarrow \varphi \\
 \bullet & \xrightarrow{\{1\}} & \mathbf{Set}
 \end{array}$$

- Now define a new SMC, denoted $\mathbf{W}[S]$ as follows:
 - It has the same objects as \mathbf{W} , but morphisms are defined as:

$$\text{Hom}_{\mathbf{W}[S]}(X, Y) = \sum_{s \in \text{Ob}(\mathcal{S})} \text{Hom}_{\mathbf{W}}(X \oplus \iota(s), Y).$$

- Given $X \oplus \iota(s) \rightarrow Y$ and $Y \oplus \iota(t) \rightarrow Z$,
- we can compose to $X \oplus \iota(s \oplus t) \rightarrow Z$, because ι is strong.
- Stacking \oplus in $\mathbf{W}[S]$ is also achieved by the strong-ness of ι .

The math for baking in special symbols, part 2

- We have constructed a SMC denoted $\mathbf{W}[\mathcal{S}]$ out of our setup,

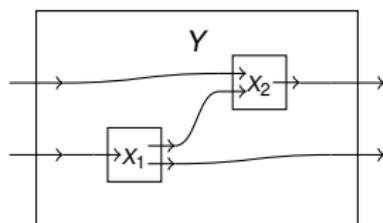
$$\begin{array}{ccc}
 \mathcal{S} & \xrightarrow{\iota} & \mathbf{W} \\
 \downarrow ! & \nearrow m & \downarrow \mathcal{P} \\
 \bullet & \xrightarrow{\{1\}} & \mathbf{Set}
 \end{array}$$

- Note we haven't used m yet, we've only used ι up to now.
- We need an algebra $\mathcal{P}[\mathcal{S}]: \mathbf{W}[\mathcal{S}] \rightarrow \mathbf{Set}$.
 - Have it act the same on boxes as \mathcal{P} does: $\mathcal{P}[\mathcal{S}](X) := \mathcal{P}(X)$.
 - A morphism $\phi: X \rightarrow Y$ in $\mathbf{W}[\mathcal{S}]$ is a morphism $\phi: X \oplus \iota(s) \rightarrow Y$ in \mathbf{W} .
 - We need to assign a function $\mathcal{P}[\mathcal{S}](\phi): \mathcal{P}(X) \rightarrow \mathcal{P}(Y)$.
 - Use the following composite:

$$\mathcal{P}(X) \cong \mathcal{P}(X) \times \{1\} \xrightarrow{\mathcal{P}(X) \times m} \mathcal{P}(X) \times \mathcal{P}(\iota(s)) \xrightarrow{\cong} \mathcal{P}(X \oplus \iota(s)) \xrightarrow{\mathcal{P}(\phi)} \mathcal{P}(Y).$$

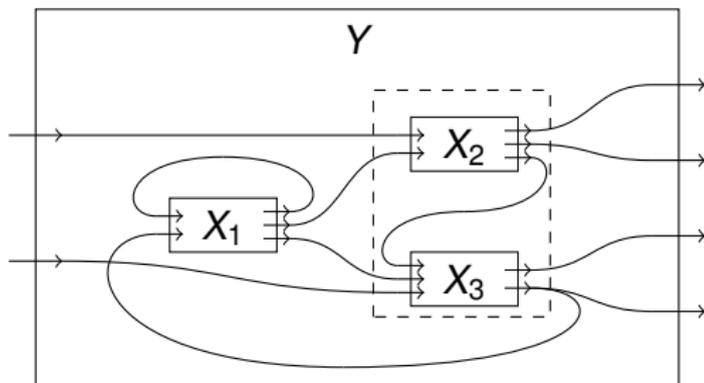
Work to be done (please feel free to help!)

- I'd like wiring diagrams to be implemented.
 - As mentioned in the introduction, David Darais plans to do so.
 - I'd like an open source WD-programming language, based in NANDs.
- I'd like an operadic model of reinforcement learning.
 - Each box could represent a population of learning agents.
 - Suppose given an operadic wiring diagram $\phi: (X_1, \dots, X_n) \rightarrow Y$.
 - Can ϕ distribute global rewards among the X_i in such a way that:
 - if each X_i optimizes its behavior according to this sub-reward,
 - the result is that Y has optimized according to the global reward?
- Similar vein: operadic verification and validation?



Summary

- We can draw pictures like this:



- Such a picture represents a morphism $\phi: X_1 \oplus X_2 \oplus X_3 \rightarrow Y$ in a symmetric monoidal category called \mathbf{W} .
- We can fill each interior box of ϕ with a machine, and thus derive a machine for the exterior box.
- We can abstract away the details of any part by enclosing it.
- The requisite formulas are straightforward and written out here in full.

Thanks!

Thanks for inviting me!

