STRING DIAGRAMS FOR TEXT

VINCENT WANG-MAŚCIANICA

ST. CATHERINE'S COLLEGE
THE UNIVERSITY OF OXFORD
DEPARTMENT OF COMPUTER SCIENCE

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
2023

Contents

(Acknowledgements will go in a margin note here.)

Novel contributions:

- Section REF is a pedestrian introduction to weak n-category theory (via `homotopy.io`, underpinned by the theory of associative n-categories) from the perspective of generalising familiar string-rewrite systems to higher dimensions. The chief development of this section is a demonstration that context-free grammars and tree-adjoining grammars may be formalised in the n-categorical setting.
- Section REF spells out a generative grammar for text using an n-categorical signature as a rewrite system, which additionally provides a unified framework from which the Text Circuit Theorem first proved in CITE is recovered.
- Section REF introduces the category **ContRel** of continuous relations. I detail the relationships (or lack thereof) of **ContRel** to its cousins **Top** and **Rel**. Though **ContRel** is constructed naïvely, its definition and an exposition of its expressivity from the monoidal perspective appears to be novel.
- Section REF string-diagrammatically characterises set-indexed collections of disjoint open subsets of spaces in **ContRel** as *sticky spiders* – special frobenius algebras that satisfy certain interaction relations with an idempotent. The diagrammatic outcome is that reasoning with such set-indexed collections remains as graphically intuitive as with spiders.
- Section REF – with the aid of a theorem by Friedman [Fri05] – string-diagrammatically characterises a vocabulary of linguistic topological relations in **ContRel** such as simple connectedness, touching, insideness, and rigid motion.
- Section REF argues for the centrality of explaining communication as a criterion for formal approaches to syntax, and explores the relationship between productive and parsing grammars as organised by a monoidal cofunctor. A diagrammatic treatment of monoidal cofunctor boxes is introduced for this purpose.
- REF is a standalone introduction to the mathematical setup of Montague's *Universal Grammar*, aimed at modern algebraists. An outcome of this section is the placement of text circuits as a natural mathematical development in the broadly conceived programme of Montague Semantics.
- Appendix REF constructs a subcategory of sticky spiders in **ContRel** on the unit square that behaves as a model of **FinRel** equipped with a *Turing object* – a single object in a category with respect to which all other objects and morphisms between them may be encoded. This is of particular relevance to the linguistic phenomenon of *entification* – that essentially all grammatical categories of words and even phrases have noun-equivalents, e.g. `to run` $\simeq$ `running`, `Bob drinks` $\simeq$ `the fact that Bob drinks`. The presence of a Turing object in a symmetric monoidal category additionally provides a semantic model for higher-order processes of generic input type, and for *lasso diagrams*.
- In Section REF , by parsing text as circuits (Section REF) and using monoidal cofunctor boxes (Section REF) to interpret those circuits in **ContRel** as iconic representations equipped with a vocabulary of lingustic-topological concepts (Section REF), I compute some metaphors by hand.

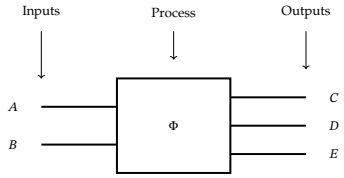
0
Context and synopsis

There are potentially practical and theoretical benefits to a fresh mathematical take on basic linguistics. String diagrams are formal, intuitive, expressive, fun, and pretty. I review the relevant research context.

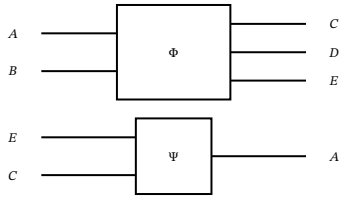
0.1 Process Theories

This section seeks to introduce process theories via string diagrams. The margin material will provide the formal mathematics of string diagrams from the bottum-up. The main body develops process theories via string diagrams by example, through which we develop towards a model of linguistic spatial relations – words like "to the left of" and "between" – which are a common ground of competence we all possess. Here we only focus on geometric relations between points in two dimensional Euclidean space equipped with the usual notions of metric and distance, providing adequate foundations to follow []talkspace , in which I demonstrate how text circuits can be obtained from sentences and how such text circuits interpreted in the category of sets and relations **Rel** provides a semantics for such sentences. This motivates the question of how to express the (arguably more primitive []piaget) linguistic topological concepts – such as "touching" and "inside" – the mathematics of which will be in Chapter **??**; the reader may skip straight to that chapter after this section if they are uninterested in syntax. We close this section with a brief note on how process theories relate to mathematical foundations and computer science.

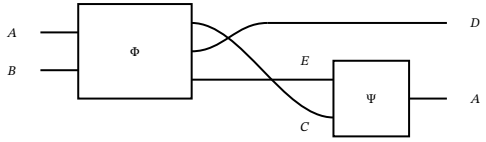
A *process* is something that transforms some number of input system types to some number of output system types. We depict systems as wires, labelled with their type, and processes as boxes. Unless otherwise specified, we read processes from left to right.



Processes may compose in parallel, depicted as placing boxes next to each other.



Processes may compose sequentially, depicted as connecting wires of the same type.



There are a lot of definitions to get started, but as with programming languages, preloading the work makes it easier to scale. This is the mathematical source code of string diagrams, which is only necessary if we need to show that something new is a symmetric monoidal category or if we are tinkering deeply, so it can be skipped for now. The important takeaway is that string diagrams are syntax, with morphisms in symmetric monoidal categories as semantics.

Definition 0.1.1 (Category). A *category* $\mathcal{C}$ consists of the following data

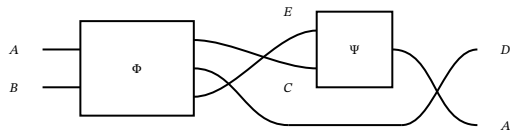
- A collection $\mathrm{Ob}(\mathcal{C})$ of *objects*
- For every pair of objects $A, B \in \mathrm{Ob}(\mathcal{C})$, a set $\mathcal{C}(A, B)$ of *morphisms* from a to b.
- Every object $a \in \mathrm{Ob}(\mathcal{C})$ has a specified morphism 1_a in $\mathcal{C}(a, a)$ called *the identity morphism* on a.
- Every triple of objects $A, B, C \in \mathrm{Ob}(\mathcal{C})$, and every pair of morphisms $f \in \mathcal{C}(A, B)$ and $g \in \mathcal{C}(b, c)$ has a specified morphism $(f; g) \in \mathcal{C}(a, c)$ called *the composite* of f and g.

This data has to satisfy two coherence conditions, which are:

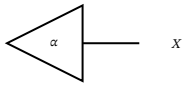
Unitality: For any morphism $f : a \to b, 1_a; f = f = f; 1_b$

Associativity: For any four objects A, B, C, D and three morphisms $f : A \to B, g : B \to C, h : C \to D$, $(f; g); h = f; (g; h)$

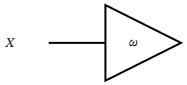
In these diagrams only input-output connectivity matters: so we may twist wires and slide boxes along wires to obtain different diagrams that still refer to the same process. So the diagram below is equal to the diagram above.



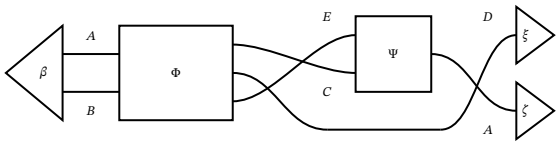
Some processes have no inputs; we call these *states*.



Some processes have no outputs; we call these *tests*.



A process with no inputs and no outputs is a *number*; the number tells us the outcome of applying tests to a composite of states modified by processes.



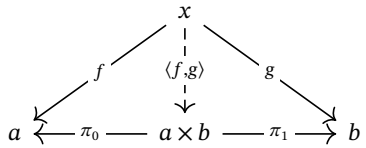
A process theory is given by the following data:

- A collection of systems
- A collection of processes along with their input and output systems
- A methodology to compose systems and processes sequentially and in parallel, and a specification of the unit of parallel composition.
- A collection of equations between composite processes

Example 0.1.8 (Linear maps with direct sum)**.** Systems are finite-dimensional vector spaces over $\mathbb{R}$. Processes are linear maps, expressed as matrices with entries in $\mathbb{R}$.

Sequential composition is matrix multiplication. Parallel composition of systems is the direct sum of vector

Definition 0.1.2 (Categorical Product)**.** In a category $\mathcal{C}$, given two objects $a, b \in \mathrm{Ob}(\mathcal{C})$, the *product* $A \times B$, if it exists, is an object with projection morphisms $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$ such that for any object $x \in \mathrm{Ob}(\mathcal{C})$ and any pair of morphisms $f : X \to A$ and $g : x \to b$, there exists a unique morphism $f \times g : X \to A \times B$ such that $f = (f \times g); \pi_0$ and $g = (f \times g); \pi_1$. This is a mouthful which is easier expressed as a commuting diagram as below. The dashed arrow indicates uniqueness. $A \times B$ is a product when every path through the diagram following the arrows between two objects is an equality.



The idea behind the definition of product is simple: instead of explicitly constructing the cartesian product of sets from within, let's say *a product is as a product does*. For objects, the cartesian product of sets $A \times B$ is a set of pairs, and we may destruct those pairs by extracting or projecting out the first and second elements, hence the projection maps π_0, π_1. Another thing we would like to do with pairs is construct them; whenever we have some A-data and B-data, we can pair them in such a way that construction followed by destruction is lossless and doesn't add anything. In category-theoretic terms, we select 'arbitrary' A- and B-data by arrows $f : X \to A$ and $g : X \to B$, and we declare that $f \times g : X \to A \times B$ is the unique way to select corresponding tuples in $A \times B$. This design-pattern of "for all such-and-such there exists a unique such-and-such" is an instance of a so-called *universal property*, the purpose of which is to establish isomorphism between operationally equivalent implementations.

To understand what this style of definition gives us, let's revisit Kuratowski's and Wiener's definitions of cartesian product, which are, respectively:

$$A \overset{K}{\times} B := \left\{ \{\{a\}, \{a, b\}\} \mid a \in A\ ,\ b \in B \right\}$$

$$A \overset{W}{\times} B := \left\{ \{\{a, \varnothing\}, b\} \mid a \in A\ ,\ b \in B \right\}$$

Keeping overset-labels and using maplet notation, the associated projections are:

$$\overset{K}{\pi}_0 := \{\{a\}, \{a, b\}\} \mapsto a$$

$$\overset{K}{\pi}_1 := \{\{a\}, \{a, b\}\} \mapsto b$$

$$\overset{W}{\pi}_0 := \{\{a, \varnothing\}, b\} \mapsto a$$

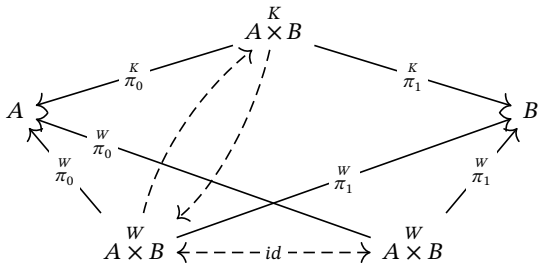
$$\overset{W}{\pi}_1 := \{\{a, \varnothing\}, b\} \mapsto b$$

And maps f, g into A and B are tupled by the following:

$$f \overset{K}{\times} g := x \mapsto \{\{f(x)\}, \{f(x), g(x)\}\}$$

$$f \overset{W}{\times} g := x \mapsto \{\{f(x), \varnothing\}, g(x)\}$$

Both satisfy the commutative diagram defining the product. Something neat happens when we pick $A \overset{K}{\times} B$ to be the arbitrary X for the product definition of $A \overset{W}{\times} B$ and vice versa. We get to mash the commuting diagrams together:



spaces $\oplus$. The parallel composition of matrices $\mathbf{A}, \mathbf{B}$ is the block-diagonal matrix

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}$$

The unit of parallel composition is the singleton 0-dimensional vector space. States are row vectors. Tests are column vectors. The numbers are $\mathbb{R}$. Usually the monoidal product is written with the symbol $\otimes$, which clashes with notation for the hadamard product for linear maps, while the process theory we have just described takes the direct sum $\oplus$ to be the monoidal product.

Example 0.1.9 (Sets and functions with cartesian product)**.** Systems are sets A, B. Processes are functions between sets $f : A \to B$. Sequential composition is function composition. Parallel composition of systems is the cartesian product of sets: the set of ordered pairs of two sets.

$$A \otimes B = A \times B := \{(a, b) \mid a \in A, b \in B\}$$

The parallel composition $f \otimes g : A \times C \to B \times D$ of functions $f : A \to B$ and $g : C \to D$ is defined:

$$f \otimes g := (a, c) \mapsto (f(a), g(c))$$

The unit of parallel composition is the singleton set $\{\star\}$. There are many singletons, but this presents no problem for the later formal definition because they are all equivalent up to unique isomorphism. States of a set A correspond to elements $a \in A$ – we forgo the usual categorical definition of points from the terminal object in favour of generalised points from the monoidal perspective. Every system A has only one test $a \mapsto \star$; this is since the singleton is terminal in **Set**. So there is only one number.

Example 0.1.10 (Sets and relations with cartesian product)**.** Systems are sets A, B. Processes are relations between sets $\Phi \subseteq A \times B$, which we may write in either direction $\Phi^* : A \nrightarrow B$ or $\Phi_* : B \nrightarrow A$. Relations between sets are equivalently matrices with entries from the boolean semiring. Relation composition is matrix multiplication with the boolean semiring. Φ^*, Φ_* are the transposes of one another. Sequential composition is relation composition:

$$A \overset{\Phi}{\nrightarrow} B \overset{\Psi}{\nrightarrow} C := \{(a, c) \mid a \in A,\ c \in C,\ \exists b_{\in B} : (a, b) \in \Phi \wedge (b, c) \in \Psi\}$$

Parallel composition of systems is the cartesian product of sets. The parallel composition of relations $A \otimes C \overset{\Phi \otimes \Psi}{\nrightarrow} B \otimes D$ of relations $A \overset{\Phi}{\nrightarrow} B$ and $C \overset{\Psi}{\nrightarrow} D$ is defined:

$$\Phi \otimes \Psi := \{\big((a, c), (b, d)\big) \mid (a, b) \in \Phi \wedge (c, d) \in \Psi\}$$

The unit of parallel composition is the singleton. States and tests of a set A are subsets of A.

0.1.1 What does it mean to copy and delete?

Now we discuss how we might define the properties and behaviour of processes by positing equations between diagrams. Let's begin simply with two intuitive processes *copy* and *delete*:

Example 0.1.11 (Linear maps)**.** Consider a vector space $\mathbf{V}$, which we assume includes a choice of basis. The copy map is the rectangular matrix made of two identity matrices:

$$\Delta_{\mathbf{V}} : \mathbf{V} \to \mathbf{V} \oplus \mathbf{V} := \begin{bmatrix} \mathbf{1}_{\mathbf{V}} & \mathbf{1}_{\mathbf{V}} \end{bmatrix}$$

The delete map is the column vector of 1s:

$$\epsilon_{\mathbf{V}} : \mathbf{V} \to \mathbf{0} := \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Example 0.1.12 (Sets and functions)**.** Consider a set A. The copy function is defined:

$$\Delta_A : A \to A \times A := a \mapsto (a, a)$$

The delete function is defined:

$$\epsilon_A : A \to \{\star\} := a \mapsto \star$$

Example 0.1.13 (Sets and relations)**.** Consider a set A. The copy relation is defined:

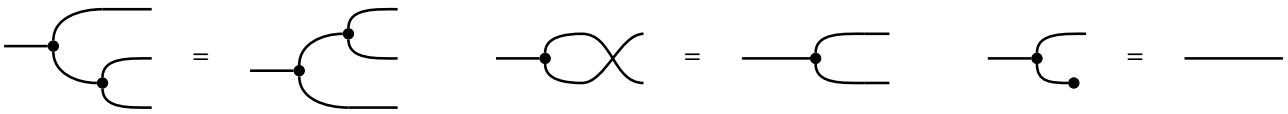
$$\Delta_A : A \nrightarrow A \times A := \{\big(a, (a, a)\big) \mid a \in A\}$$

The delete relation is defined:

$$\epsilon_A : A \nrightarrow \{\star\} := \{(a, \star) \mid a \in A\}$$

We may verify that, no matter the concrete interpretation of the diagram in terms of linear maps, functions or relations, the following equations characterise a cocommutative comonoid internal to a monoidal category.

coassociativity cocommutativity counitality



The two unique arrows between $\overset{K}{\times}$ and $\overset{W}{\times}$ are format-conversions, and we know by definition that the unique arrow that performs format conversion from $\overset{W}{\times}$ to itself in the bottom face is the identity. In maplet notation, the conversion from $A \overset{K}{\times} B \to A \overset{W}{\times} B$ is $\{\{a\}, \{a, b\}\} \mapsto \{\{a, \varnothing\}, b\}$, and similarly for the other direction. Because these conversions are uniquely determined arrows, their composite is also uniquely determined, and we know their composite is equal to the identity. So, the nontrivial conversions witness an *isomorphism* between $A \overset{K}{\times} B$ and $A \overset{W}{\times} W$; a pair of maps $X \to Y$ and $Y \to X$ such that their loop-composites equal identities. This in a nutshell is the category-theoretic approach to overcoming the bureaucracy of syntax: use universal properties (or whatever else) encode your intents and purposes, establish isomorphisms, and then treat isomorphic things as "the same for all intents and purposes". The idea of treating isomorphic objects as the same is ingrained in category theory, so isomorphism notation $\simeq$ is often just written as equality $=$; going forward we will use equality notation unless there are good reasons to remember that we only have isomorphisms.

Definition 0.1.3 (Functor)**.** A functor $F : \mathcal{C} \to \mathcal{D}$ (read: with domain a category $\mathcal{C}$ and codomain a category $\mathcal{D}$) consists of two suitably related functions. An object function $F_0 : \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{D})$ and a morphism function (equivalently viewed as a family of functions indexed by pairs of objects of $\mathcal{C}$) $F_1(X,Y) : \mathcal{C}(X,Y) \to \mathcal{D}(F_0X, F_0Y)$. F_1 must map identities to identities – i.e., be such that for all $X \in \mathcal{C}$, $F_1(1_X) = 1_{F_0X}$ – and F_1 must map composites to composites – i.e., for all $X, Y, Z \in \mathrm{Ob}(\mathcal{C})$ and all $f : X \to Y$ and $g : Y \to Z$, $F_1(f;g) = F_1f;F_1g$.

Functors in short map categories to categories, preserving the structure of identities and composition. They are the essence of "structure preserving transformation". Insofar as semantics is the science of finding structure-preserving transformations that tell us when syntactic things are equal, functors are just that. They are incredibly useful and mysterious and worth internalising in a way I am not adept enough to impress by example in this margin. For us, for now, they are just stepping stones to define transformations *between functors*.

Definition 0.1.4 (Natural Transformation)**.** A natural transformation $\theta : F \to G$ for (co)domain-aligned functors $F, G : \mathcal{C} \to \mathcal{D}$ is a family of morphisms in $\mathcal{D}$ indexed by objects $X \in \mathcal{C}$ such that for all $f : X \to Y$ in $\mathcal{C}$, the following commuting diagram holds in $\mathcal{D}$:

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ \downarrow{\scriptstyle \theta_X} & & \downarrow{\scriptstyle \theta_Y} \\ GX & \xrightarrow{Gf} & GY \end{array}$$

It is worth pausing here to think about how one might characterise the process of copying in words; it is challenging to do so for such an intuitive process. The diagrammatic equations, when translated into prose, provide an answer.

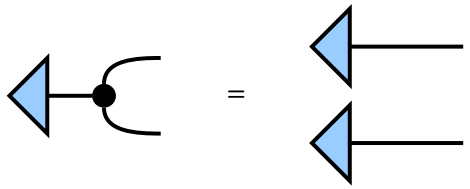
Coassociativity: says there is no difference between copying copies.

Cocommutativity: says there is no difference between the outputs of a copy process.

Counitality: says that if a copy is made and one of the copies is deleted, the remaining copy is the same as the original.

Insofar as we think this is an acceptable characterisation of copying, rather than specify concretely what a copy and delete does for each system X we encounter, we can instead posit that so long as we have processes $\Delta_X : X \otimes X \to X$ and $\epsilon_X : X \to I$ that obey all the equational constraints above, Δ_X and ϵ_X are as good as a copy and delete.

Example 0.1.14 (Not all states are copyable)**.** Call a state *copyable* when it satisfies the following diagrammatic equation:

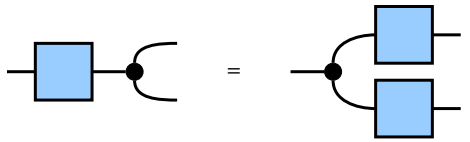


In the process theory of sets and functions, all states are copyable. Not all states are copyable in the process theories of sets and relations and of linear maps. For example, consider the two element set $\mathbb{B} := \{0,1\}$, and let $\top : \{\star\} \nrightarrow \mathbb{B} := \{(\star,0),(\star,1)\} \simeq \{0,1\}$. Consider the composite of $\top$ with the copy relation:

$$\top;\Delta_{\mathbb{B}} := \{(\star,(0,0)),(\star,(1,1))\} \simeq \{(0,0),(1,1)\}$$

This is a perfectly correlated bipartite state, and it is not equal to $\{0,1\}\times\{0,1\}$, so $\top$ is not copyable.

Remark 0.1.15. The copyability of states is a special case of a more general form of interaction with the copy relation:



A cyan map that satisfies this equation is said to be a homomorphism with respect to the commutative comonoid. In the process theory of relations, those relations that satisfy this equation are precisely the functions; in other words, this diagrammatic equation expresses *determinism*.

Here is an unexpected consequence. Suppose we insist that *to copy* in principle also implies the ability to copy *anything* – arbitrary states. From Example 0.1.14 and Remark 0.1.15, we know that this demand is incompatible with certain process theories. In particular, this demand would constrain a process theory of sets and relations to a subtheory of sets and functions. The moral here is that process theories are flexible enough to meet ontological needs. A classical computer scientist who works with perfectly copyable data and processes might demand universal copying along with the commutative comonoid equations, whereas a quantum physicist who wishes to distinguish between copyable classical data and non-copyable quantum data might taxonomise copy and delete as a special case of a more generic quasi-copy and quasi-delete that is only a commutative comonoid. In fact, quantum physicists *do* do this; see Dodo: CITE .

0.1.2 What is an update?

In the previous section we have seen how we can start with concrete examples of copying in distinct process theories, and obtain a generic characterisation of copying by finding diagrammatic equations copying satisfies in each concrete case. In this section, we show how to go in the opposite direction: we start by positing diagrammatic equations that characterise the operational behaviour of a particular process – such as *updating* – and it will turn out that any concrete process that satisfies the equational constraints we set out will *by our own definition* be an update.

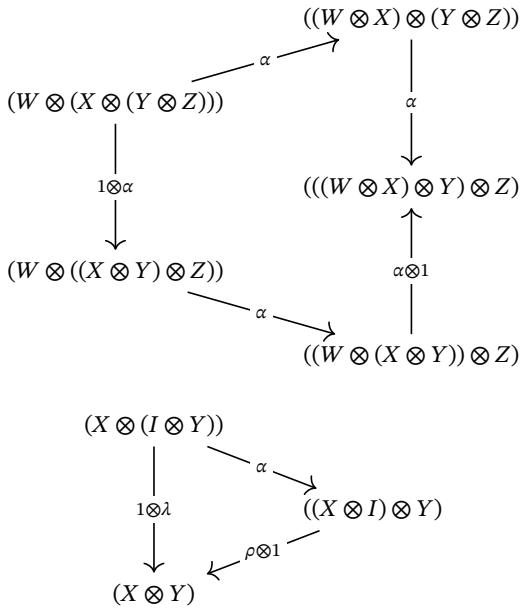
Perhaps the most familiar setting for an update is a database. In a database, an entry often takes the form of pairs of fields and values. For example, where a database contains information about employees, a typical entry might look like:

```
< NAME:Jono Doe, AGE:69, JOB:CONTENT CREATOR, SALARY:$420, ... >
```

There are all kinds of reasons one might wish to update the value of a field: Jono might legally change their name, a year might pass and Jono's age must be incremented, Jono might be promoted or demoted or get a raise and so on. It was the concern of database theorists to formalise and axiomatise the notion of updating the value of a field – *independently of the specific programming language implementation of a database* – so that they had reasoning tools to ensure program correctness CITE . The problem is reducible to axiomatising a *rewrite*: we can think of updating a value as first calculating the new value, then *putting* the new value in place of the old. Since often the new value depends in some way on the old value, we also need a procedure to *get* the current value. That was a flash-prehistory of *bidirectional transformations* CITE . Following the monoidal generalisation of lenses in CITE , a rewrite as we have described above is specified by system diagrammatic equations in the margin, each of which we introduce in prose.

Definition 0.1.5 (Monoidal Category)**.** A monoidal category consists of a category $\mathcal{C}$, a functor $\otimes : \mathcal{C} \times \mathcal{C} \to C$, a monoidal unit object $I \in \mathrm{Ob}(\mathcal{C})$, and the following natural isomorphisms – i.e. natural transformations with inverses, where multiple bar notation indicates variable object argument positions: an associator $\alpha : ((- \otimes =) \otimes \equiv) \mapsto (- \otimes (= \otimes \equiv))$, a right unitor $\rho - \otimes I \mapsto -$, and a left unitor $\lambda I \otimes - \mapsto -$). These natural isomorphisms must in addition satisfy certain *coherence* diagrams, to be displayed shortly.

Theorem 0.1.6 (Coherence for monoidal categories)**.** The following pentagon and triangle diagrams are conditions in the definition of a monoidal category. When they hold, all composites of associators and unitors (and their inverses) are isomorphisms CITE .



Definition 0.1.7 (Symmetric Monoidal Category)**.** A symmetric monoidal category is a monoidal category with an additional natural isomorphism

$$\theta : - \otimes = \mapsto = \otimes -$$

Which satisfies the following pair of hexagons CITE .

$$\begin{array}{ccc}
(X \otimes (Y \otimes Z)) & \xrightarrow{\theta} & (Z \otimes (X \otimes Y)) \\
\downarrow{\scriptstyle \alpha^{-1}} & & \downarrow{\scriptstyle \alpha} \\
(X \otimes (Z \otimes Y)) & & ((Z \otimes X) \otimes Y) \\
\downarrow{\scriptstyle 1 \otimes \theta} & & \downarrow{\scriptstyle \theta^{-1}} \\
(X \otimes (Z \otimes Y)) & \xrightarrow[\alpha]{} & ((X \otimes Z) \otimes Y)
\end{array}$$

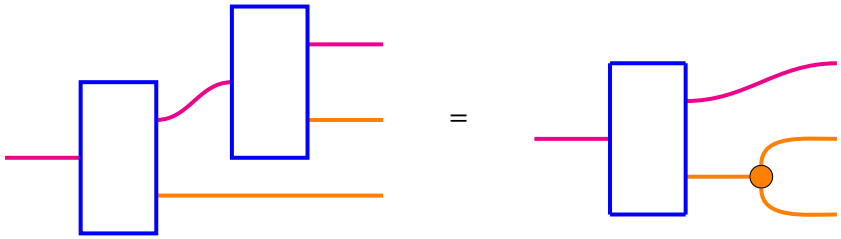
$$\begin{array}{ccc}
(X \otimes (Y \otimes Z)) & \xrightarrow{\theta} & ((Y \otimes Z) \otimes X) \\
\downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \alpha^{-1}} \\
((X \otimes Y) \otimes Z) & & (Y \otimes (Z \otimes X)) \\
\downarrow{\scriptstyle \theta^{-1}} & & \downarrow{\scriptstyle 1 \otimes \theta} \\
((Y \otimes X) \otimes Z) & \xrightarrow[\alpha^{-1}]{} & (Z \otimes (X \otimes Y))
\end{array}$$

PutPut: Putting in one value and then a second is the same as deleting the first value and just putting in the second.

GetPut: Getting a value from a field and putting it back in is the same as not doing anything.

PutGet: Putting in a value and getting a value from the field is the same as first copying the value, putting in one copy and keeping the second.

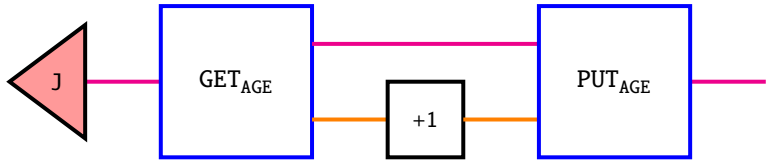
GetGet: Getting a value from a field twice is the same as getting the value once and copying it.



These diagrammatic equations do two things. First, they completely specify what it means to get and put values in a field in an implementation independent manner; it doesn't matter whether database entries are encoded as bitstrings, qubits, slips or paper or anything else, what matters is the interaction of get and put. Second, the diagrammatic equations give us the right to call our processes *get* and *put* in the first place: we define what it means to get and put by outlining the mutual interactions of get, put, copy, and delete. These two points are worth condensing and rephrasing:

A *kind of process* is determined by patterns of interaction with other kinds of processes.

Now we can diagrammatically depict the process of updating Jono's age, by getting Jono's age value from their entry, incrementing it by 1, and putting it back in.



But what are the *things* that the processes operate on?

This is a common objection from philosophers who want their ontologies tidy. The claim roughly goes that you can't really reason about processes without knowing the underlying objects that participate, and since set theory is the only way we know how to spell out objects intensionally in this way, we should stick to sets. In simpler terms, if we're drawing (black)-boxes in our diagrams, how will we know what they do to the elements of the underlying sets?

The short answer is that – perhaps surprisingly – reasoning process-theoretically is mathematically equivalent to reasoning about sets and elements for all practical purposes; it is as if whatever is going on *out there* is indifferent to whether we describe using a language of only nouns or only verbs.

In the case of set theory (the practical kind, not the one with crazy infinities), let's suppose that instead of encoding functions as sets, we treat functions as primitive, so that we have a process theory where wires are labelled with sets, and functions are process boxes that we draw. The problem we face now is that it is not immediately clear how we would access the elements of any set using only the diagrammatic language. The solution is the observation that the elements $\{x \mid X\}$ of a set X are in bijective correspondence with the functions from a singleton into X: $\{f(\star) \mapsto x \mid \{\star\} \xrightarrow{f} X$. In prose, for any element x in a set X, we can find a function that behaves as a pointer to that element $\{\star\} \to X$. So the states we have been drawing, when interpreted in the category of sets and function, are precisely elements of the sets that label their output wires.

Coherence is about getting rid of syntactic bureaucracy. Addition for example is a commutative monoid, which satisfies equations that let us forget about bracketings, zeroes, and the ordering of summation.

$$(x + (y + z)) = ((x + y) + z)$$

$$x + 0 = x = 0 + x$$

$$x + y = y + x$$

Now the situation is that we have replaced the associativity equation with associator natural transformations, unit equations with unitors (and commutation with natural isomorphisms θ that are depicted as twisting wires in string diagrams for symmetric monoidal categories.)

$$((X \otimes Y) \otimes Z) \xrightarrow{\alpha_{XYZ}} (X \otimes (Y \otimes Z))$$

$$(X \otimes I) \xrightarrow{\rho_X} X \xrightarrow{\lambda_X} (I \otimes X)$$

Recalling that we're happy with isomorphism in place of equality, coherence conditions ask that every possible composite of these structural operations is an isomorphism. In terms of the graphical language for monoidal categories, this means that string diagrams up-to-(processive)-planar-isotopy (and connectivity of wires in the case of symmetric monoidal categories) represent equivalent-up-to-isomorphism morphisms in an appropriate monoidal category. The reader is referred to selinger, joyal 1 and 2 for details.

But if they're expressively the same, what's the point?

The following rebuttal draws on Harold Abelson's introductory lecture to computer science CITE (in which string diagrams appear to introduce programs without being explicitly named as such). There is a distinction between declarative and imperative knowledge. Declarative knowledge is *knowing-that*, for example, 6 is the square root of 36, which we might write $6 = \sqrt{36}$. Imperative knowledge is *knowing-how*, for example, to obtain the square root of a positive number, for instance, by Heron's iterative method: to obtain the square root of Y, make a guess X, and take the average of X and $\frac{Y}{X}$ until your guess is good enough.

Computer science concerns imperative knowledge. An obstacle to the study of imperative knowledge is complexity, which computer scientists manage by black-box abstraction – suppressing irrelevant details, so that for instance once a square root procedure is defined, the reasoner outside the system does not need to know whether the procedure inside is an iterative method by Heron or Newton, only that it works and has certain properties. These black-boxes can be then composed into larger processes and procedures within human cognitive load.

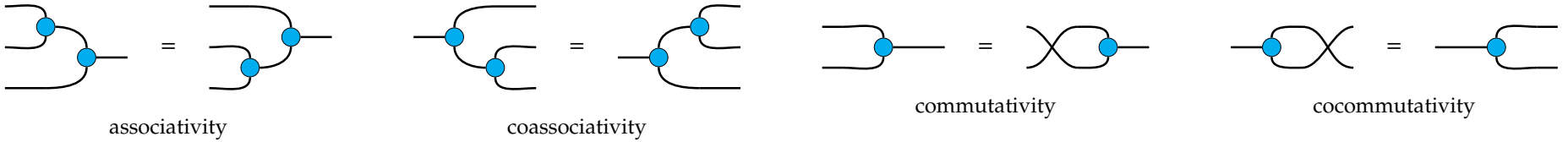
Abstraction also yields generality. For example, in the case of addition, it is not only numbers we may care to add, but perhaps vectors, or the waveforms of signals. So there is an abstract notion of addition which we concretely instantiate for different domains that share a common interface; we may decide for example that all binary operations that are commutative monoids are valid candidates for what it means to be an addition operation.

In this light, string diagrams are a natural metalanguage for the study of imperative knowledge; string diagrams in fact independently evolved within computer science from flowcharts describing processes. Process theories, which are equations or logical sentences about processes, allow us to reason declaratively about imperative knowledge. Moreover, string diagrams as syntactic objects can be interpreted in various concrete settings, so that the same diagram serves as the common interface for a process like addition, with compliant implementation details for each particular domain spelled out separately.

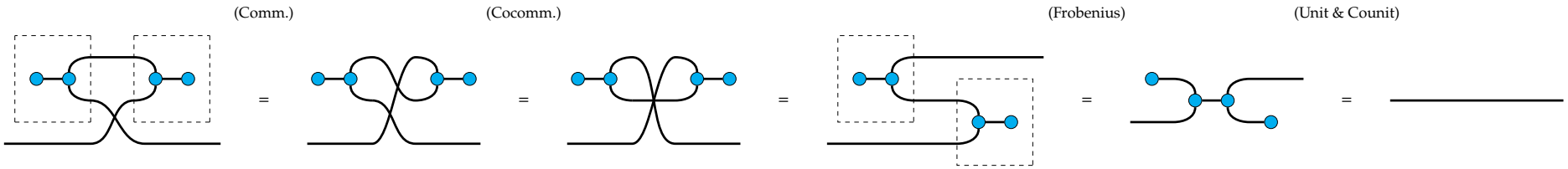
Now we define the one structure we really care about Spiders – or special frobenius algebras CITE – will be the master structure we work with.

Definition 0.1.16 (Spiders)**.** I will describe the behaviour of a spider as a PROP CITE , which as far as we care is just a way to list out processes and equations that their composites satisfy in a symmetric monoidal category. We say that an object is *equipped* with a spider when it has the following processes:

That satisfy the following equations:

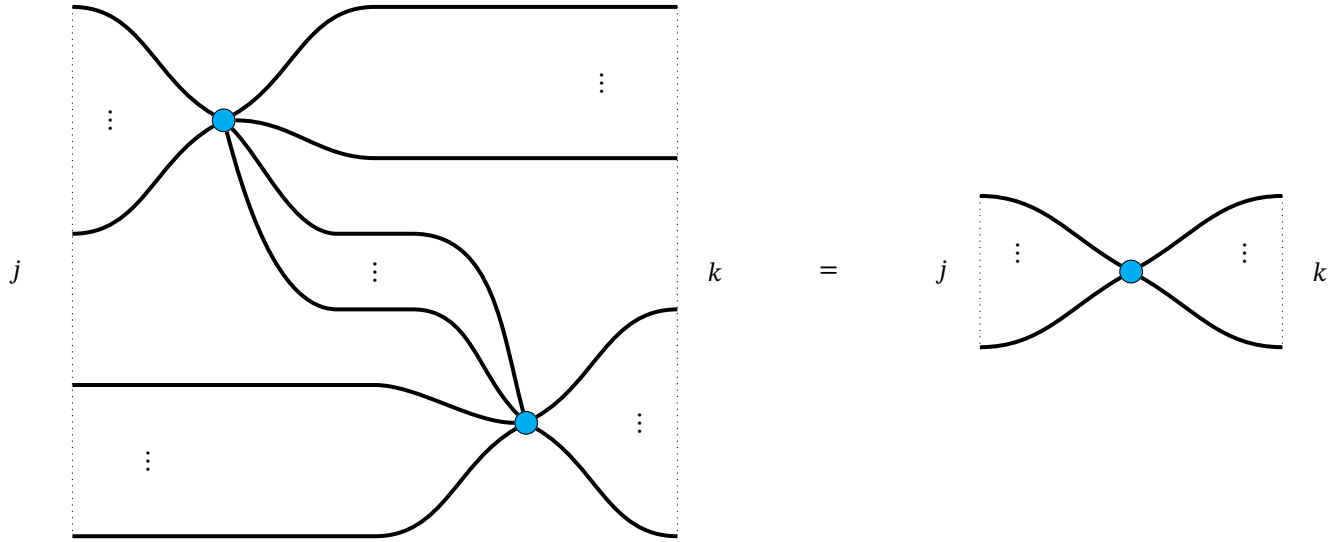


Spiders imply strong compact closure; self-dual cups and caps may be viewed as composite spiders. These are the cups which we use in pregroup diagrams to come. The equations for cups and caps are derived from spider rules, taking advantage of equality up to connectivity of wires.



Terminology 0.1.17. We call a category in which every object has a spider a *hypergraph category* CITE .

Remark 0.1.18 (Spiders are easy)**.** All connected configurations of spiders with the same number of inputs and outputs are equal. Intuitively, whereas wires connect end-to-end in string diagrams, spiders give us *multiwires* that we may freely split and connect.



Remark 0.1.19 (Only connectivity matters)**.** In hypergraph categories, by recovery of strong compact closure, we may freely reason without a convention for reading direction of processes. By the multiwire property, all diagrams with the same connectivity are equal. Hence, *only connectivity matters*.

0.2 *Previously, on DisCoCat*

DisCoCat is a research programme in applied mathematical linguistics that is **Dis**tributional, **Co**mpositional and **Cat**egorical. In this section I will recount a selective development of DisCoCat as relevant for this thesis.

0.2.1 *Lambek's Linguistics*

Jim Lambek was a jovial man who always carried a wad of twenties. can't do better than Moortgat's history and exposition of typelogical grammar in CITE , so I will borrow Moortgat's phrasing and summarise Lambek's role in the story. Typelogical grammar originated in two seminal papers by Lambek in 1958 and 1961 CITE , where Lambek sought "to obtain an effective rule (or algorithm) for distinguishing sentences from non-sentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages [. . .]". The method is to assign grammatical categories – parts of speech such as nouns and verbs – logical formulae. Whether a sentence is grammatical or not is obtained from deduction using these logical formulae in a Gentzen-style sequent proof.

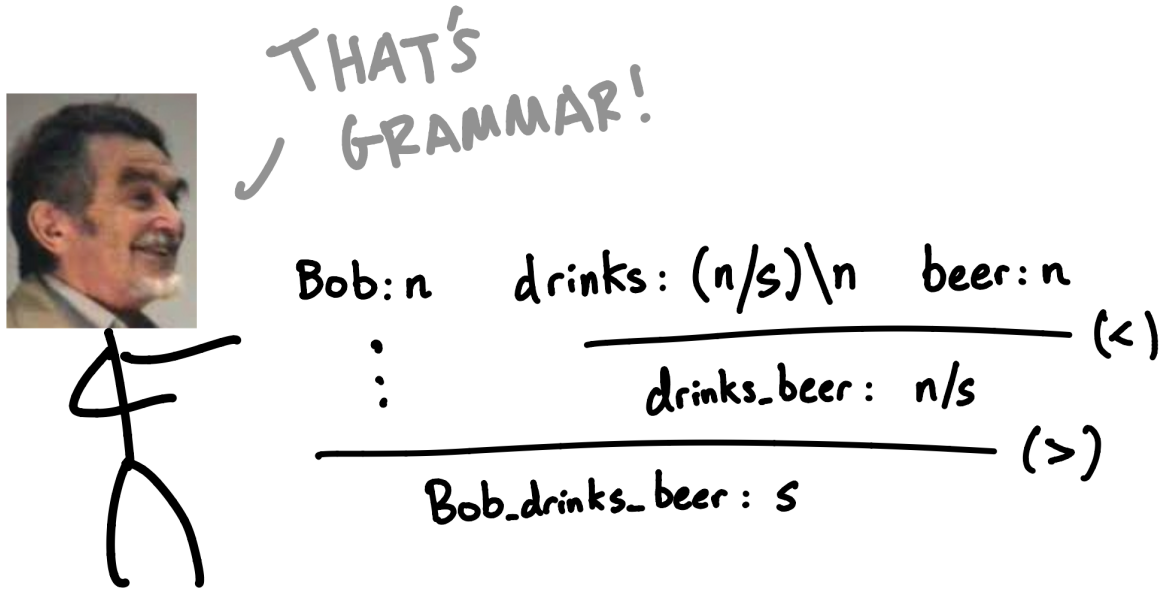


Figure 1: In English, we may consider a noun to have type n, and a transitive verb $(n/s) \setminus n$, to yield a well-formedness proof of `Bob drinks beer`. The type formation rules for such a grammar are intuitive. Apart from a stock of basic types $\mathbb{B}$ that contains special final types to indicate sentences, we have two type formation operators $(-/=)$ and $(-\setminus=)$, which along with their elimination rules establish a requirement that grammatical categories require other grammatical categories to their left or right. This is the essence of Lambek's calculi **NL** and **L**. CCGs keep the same minimal type-formations, but include extra sequent rules such as type-raising and cross-composition.

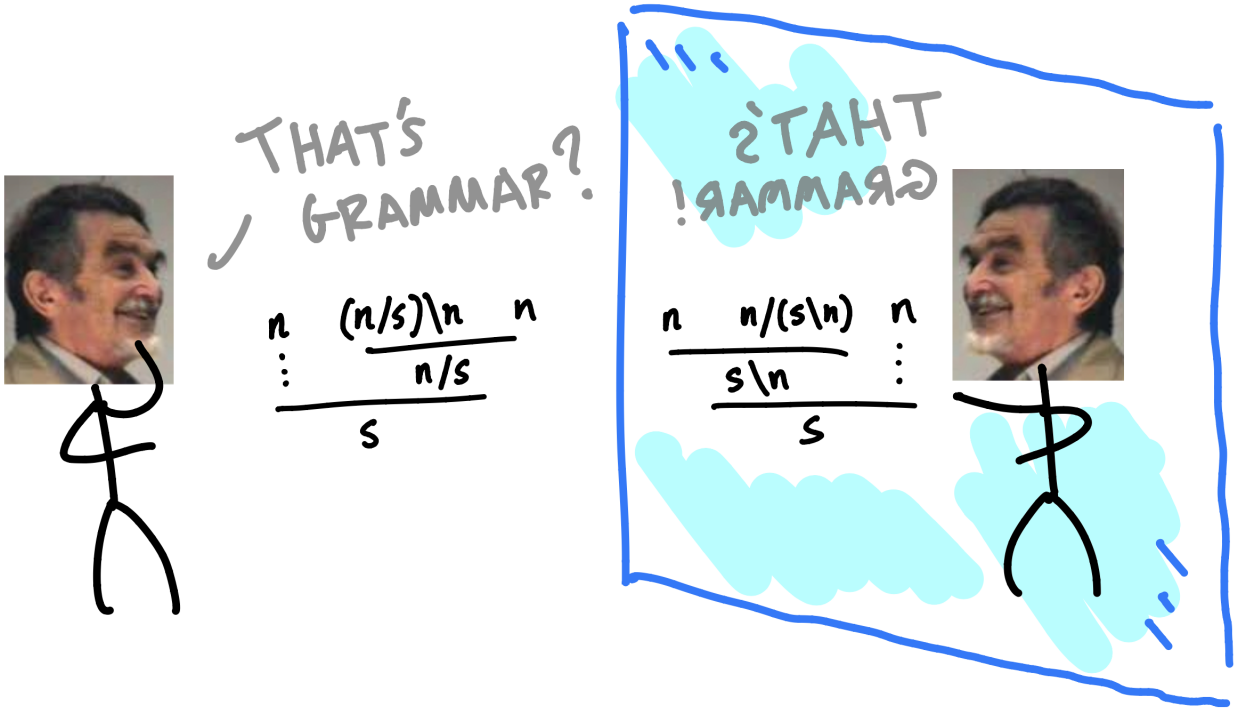


Figure 2: We can notice an asymmetry in the above formulation when we examine the transitive verb type $(n/s) \setminus n$ again; it asks first for a noun to the right, and then a noun to the left. We could just as well have asked for the nouns in the other order with the typing $(n/s) \setminus n$ and obtained all of the same proofs.

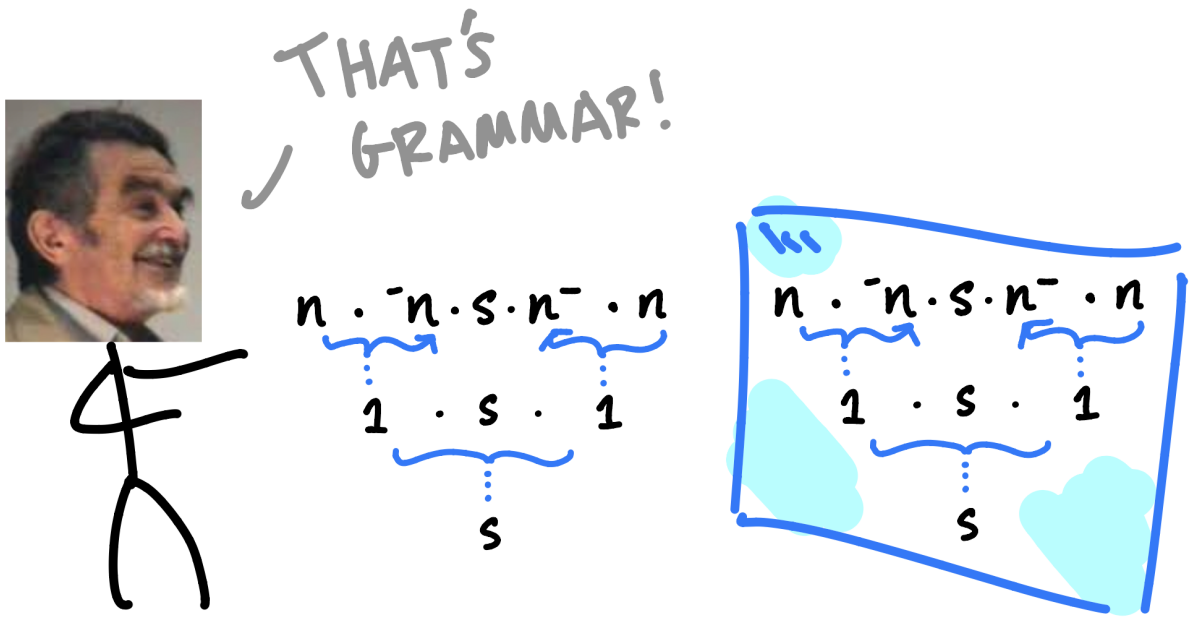


Figure 3: To eliminate this asymmetry, Lambek devised pregroup grammars. Whereas a group is a monoid with inverses up to left- and right-multiplication, a pregroup weakens the requirement for inverses so that all elements have distinct left- and right- inverses, denoted x^{-1} and ${}^{-1}x$ respectively. Eliminating or introducing inverses is a non-identity relation on elements of the pregroup, so we have axioms of the form e.g. $x \cdot^{-1} x \to 1 \to^{1} x \cdot x$. In this formulation, denoting the multiplication with a dot, both $(n/s) \setminus n$ and $(n/s) \setminus n$ become ${}^{-1}n \cdot s \cdot n^{-1}$, which just wants a noun to the left and a noun to the right in whatever order to eliminate the flanking inverses to reveal the embedded sentence type. Now we can obtain the same proof of correctness as a series of algebraic reductions.

$$
\begin{aligned}
& n \cdot ({}^{-1}n \cdot s \cdot n^{-1}) \cdot n \\
\to \quad & (n \cdot^{-1} n) \cdot s \cdot (n^{-1} \cdot n) \\
\to \quad & 1 \cdot s \cdot 1 \\
\to \quad & s
\end{aligned}
$$

0.2.2 *Coecke's Composition*

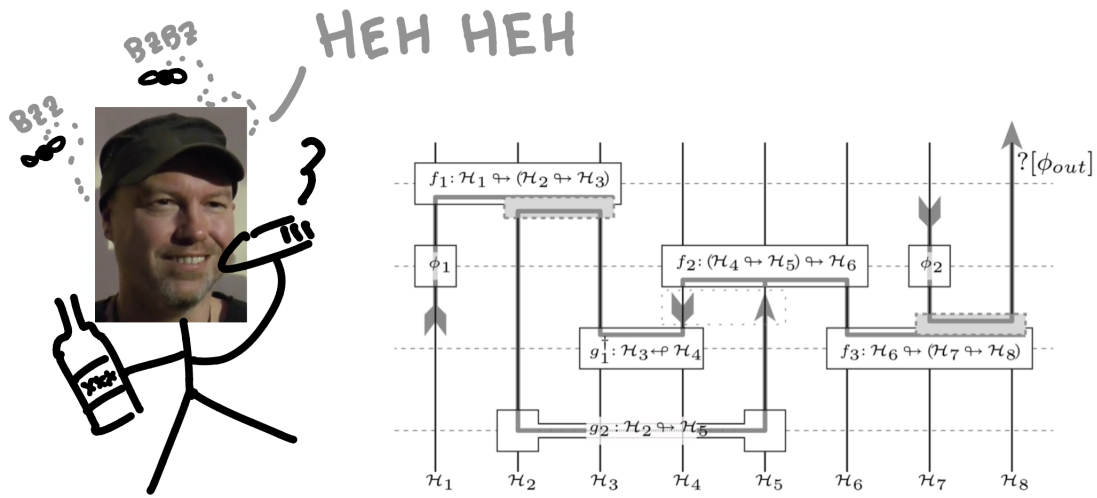


Figure 4: Meanwhile, an underground grunge vagabond moonlighting as a quantum physicist moonlighting as a computer scientist was causing a shortage of cigars and whiskey in a small English town. He noticed a funny thing about the composition of multiple non-destructive measurements of a quantum system, which was that information could be carried, or flow, between them. So he wrote a paper invitationtoquantum , which contained informal diagrams that looked like this.

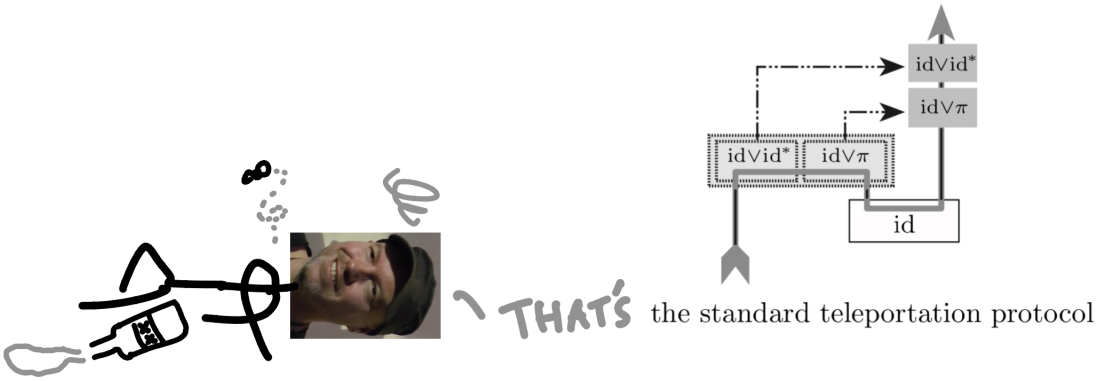


Figure 5: There were two impressive things about these diagrams. First, the effects such as transparencies for text boxes and curved serifs for angled arrows give a modern feel, but they were done manually in macdraw, the diagrammatic equivalent of sticks and stones. Second, though the diagrams were informal, they provided a way to visualise and reason about entanglement that was impossible by staring at the equivalent matrix formulation of the same composite operator. The most important diagram for our story was this one, which captures the information flow of quantum teleportation.

0.2.3 *Categorical quantum mechanics*

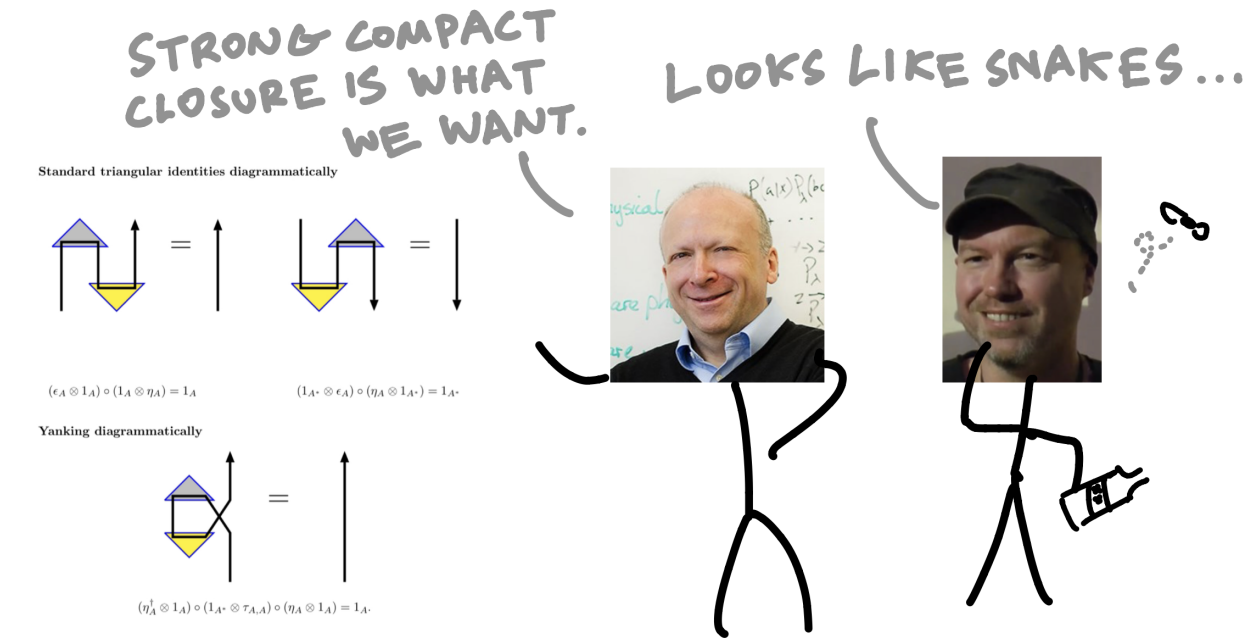


Figure 6: Category theorists and physicists such as Abramsky and Baez were excited about these diagrams, which looked like string diagrams waiting to be made formal. The graphical cups and caps in the important diagram were determined to correspond to a special form of symmetric monoidal closed category called strong compact closed.

Figure 7: Diagrammatically, reasoning in a strongly compact closed category amounts to ignoring the usual requirement of processiveness and forgetting the distinction between inputs and outputs, so that "future" outputs could curl back and be "past" inputs. This formulation also gave insight into the structure of quantum mechanics. For example, the process-state duality of strong compact closure manifested as the Choi–Jamiołkowski isomorphism.

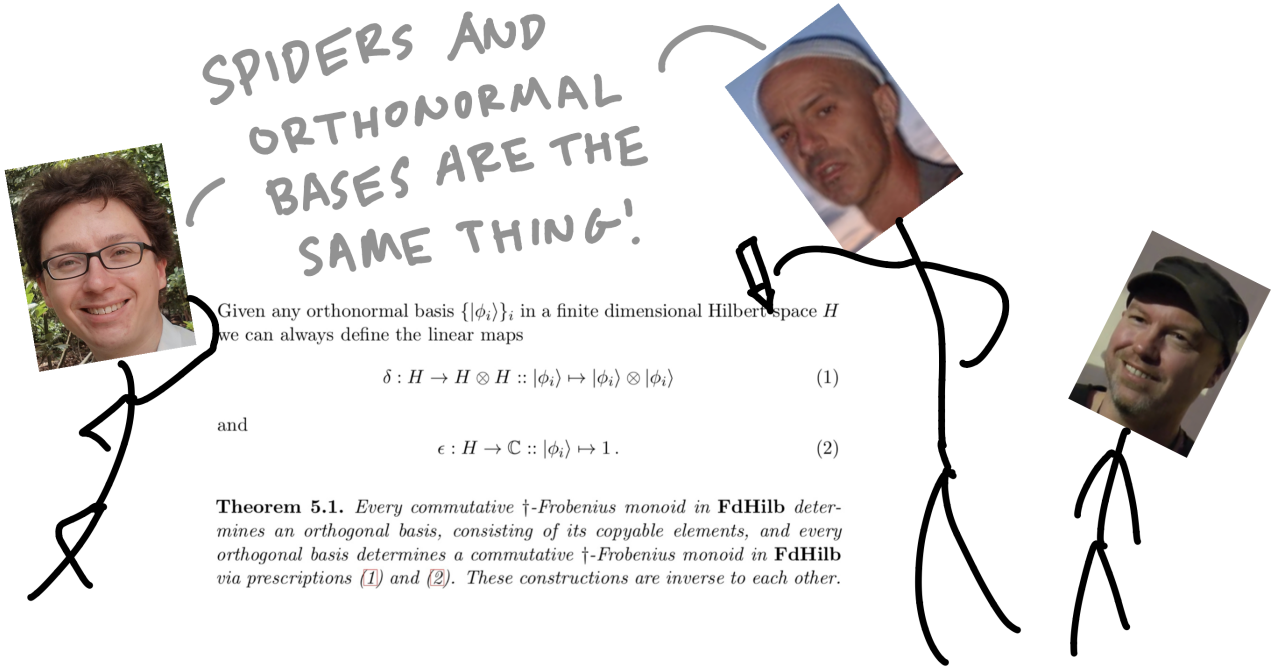


Figure 8: However, dealing with superpositions necessitated using summation operators within diagrams, which is cumbersome to write especially when dealing with even theoretically simple Bell states. An elegant diagrammatic simplification arose with the observation that special-†-frobenius algebras classicquantumstruct , or spiders, correspond to choices of orthonormal bases novelchar in **FdHilb**, the ambient setting of finite-dimensional hilbert spaces. Not only did this remove the need for summation operators, it also revealed that strong compact closure was a derived, rather than fundamental structure, since spiders induce compact closed structure.

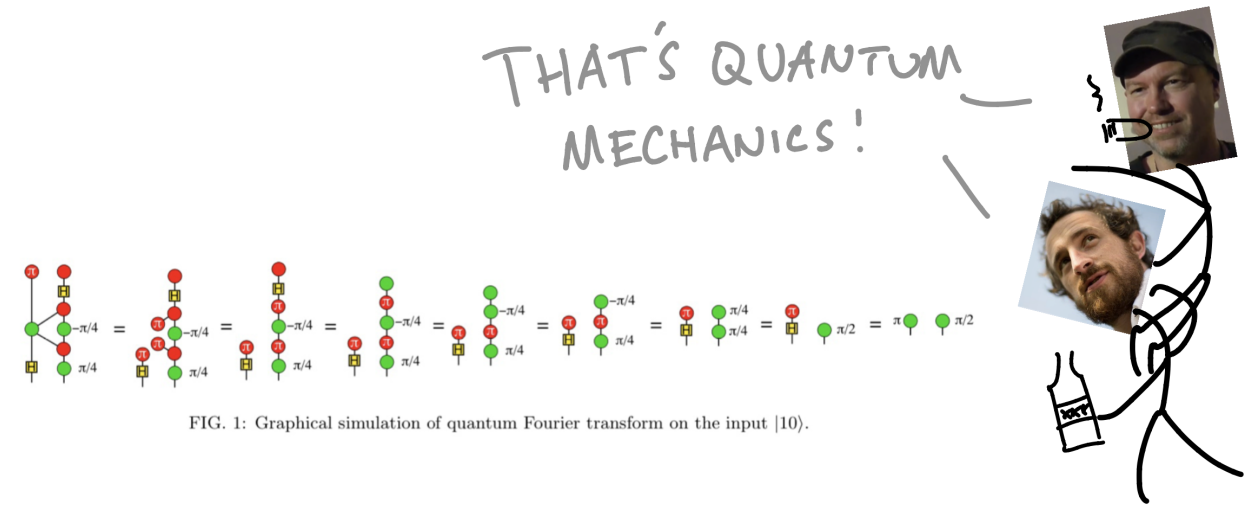


Figure 9: And so the stage was set for a purely diagrammatic treatment of ZX quantum mechanics. The story of ZX diverges away from our interest, so I will summarise what happened afterwards. In no particular order, the development of ZX went on to accommodate a third axis of measurement to yield a ZXW calculus CITE , the systems were proven to be complete CITES , there are at the time of writing two expository books CITES , and ZX-variants are becoming an industry standard for quantum circuit specification and rewriting CITE .

0.2.4 *Enter computational linguistics*

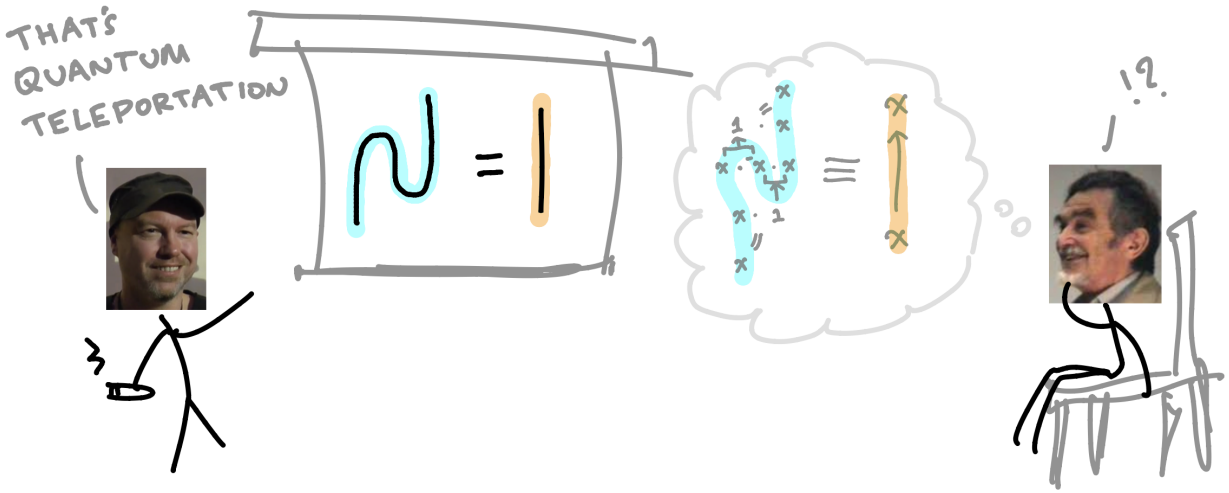


Figure 10: Somewhere in Canada at the turn of the millenium, Bob met Jim, who saw something familiar about the diagram for quantum teleportation. The snake equation for compact closure looked a lot like the categorified version of introducing and eliminating pregroup types.

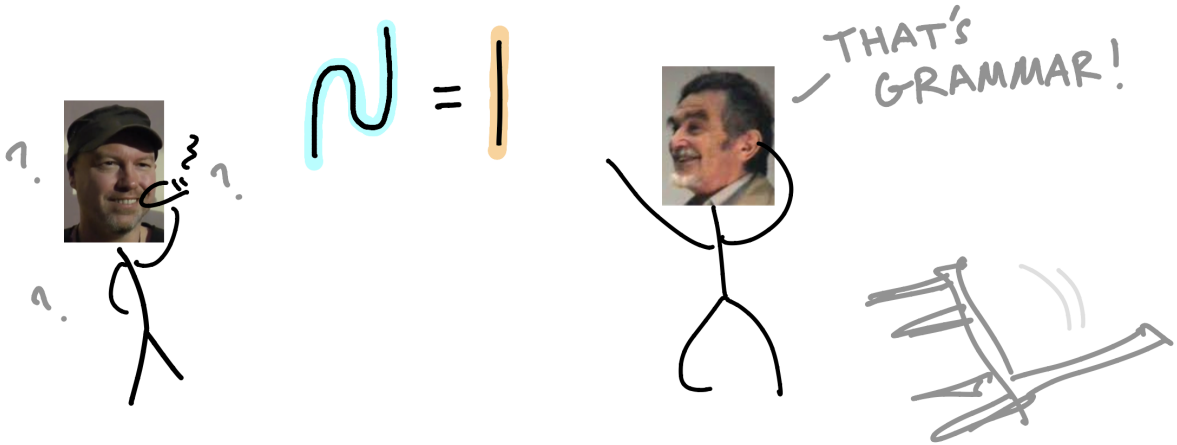


Figure 11: Bob and Jim's meeting put the adjectives *compositional* and *categorical* on the same table, but the cake wasn't ready. Two more actors Steve and Mehrnoosh were required to introduce *distributional*, which refers to Firth's maxim CITE "you shall know a word by the company it keeps". In its modern incarnation, this refers generally to vector-based semantics for words, where it is desirable but not necessarily so (as in the case of generic latent space embeddings by an autoencoder) that proximity of vectors models semantic closeness.

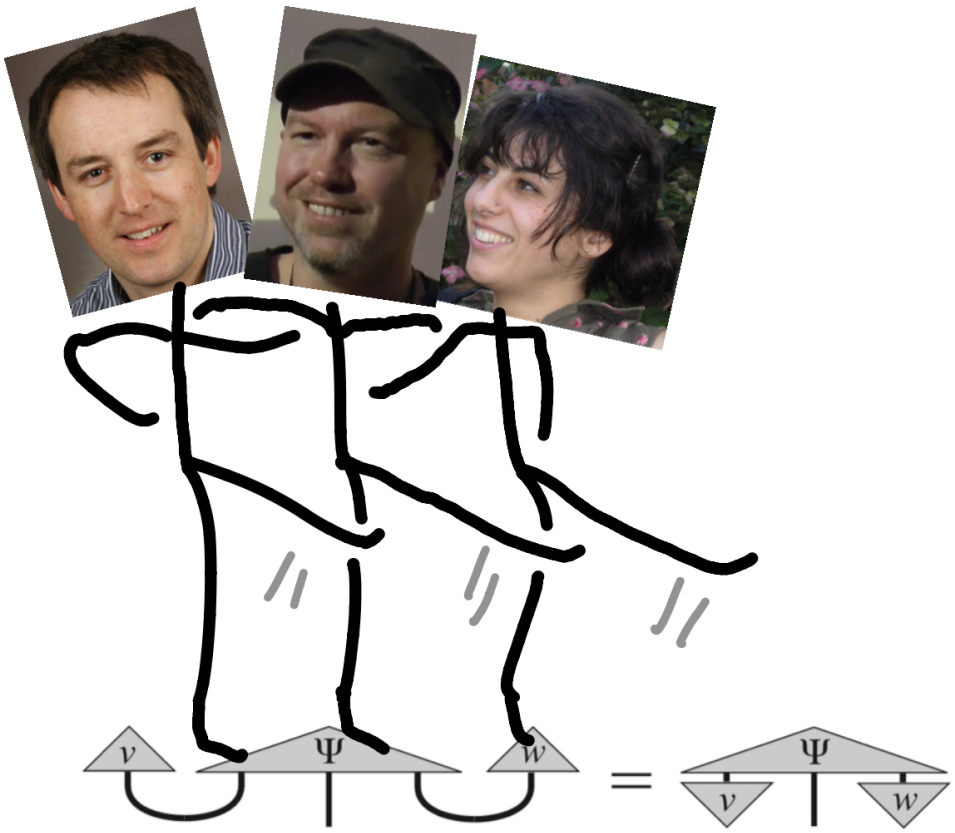


Figure 12: Steve Clark was a professor in the computer science department at Oxford, and he was wondering how to compose vector-based semantic representations. Steve asked Bob, who realised suddenly what Jim was talking about. Mediated by the linguistic expertise of Mehrnoosh who was a postdoctoral researcher in Oxford at the time, pregroup diagrams were born. The basic types *n* and *s* are assigned finite-dimensional vector spaces, concatenation of types the kronecker product $\otimes$, and by the isomorphism of dual spaces in finite dimensions there is no need to keep track of the left- and right- inverse data. Words become vectors, and pregroup reductions become bell-states, or bell-measurements, depending on whether one reads top-down or bottom-up. There was simply no other game in town for an approach to computational linguistics that combined linguistic compositionality with distributional representations.

where the reversed triangles are now the corresponding Dirac-bra's, or in vector space terms, the corresponding functionals in the dual space. This simplifies the expression that we need to compute to:

$$(\langle \overrightarrow{v} | \otimes 1_S \otimes \langle \overrightarrow{v} | |) | \overrightarrow{\Psi} \rangle$$

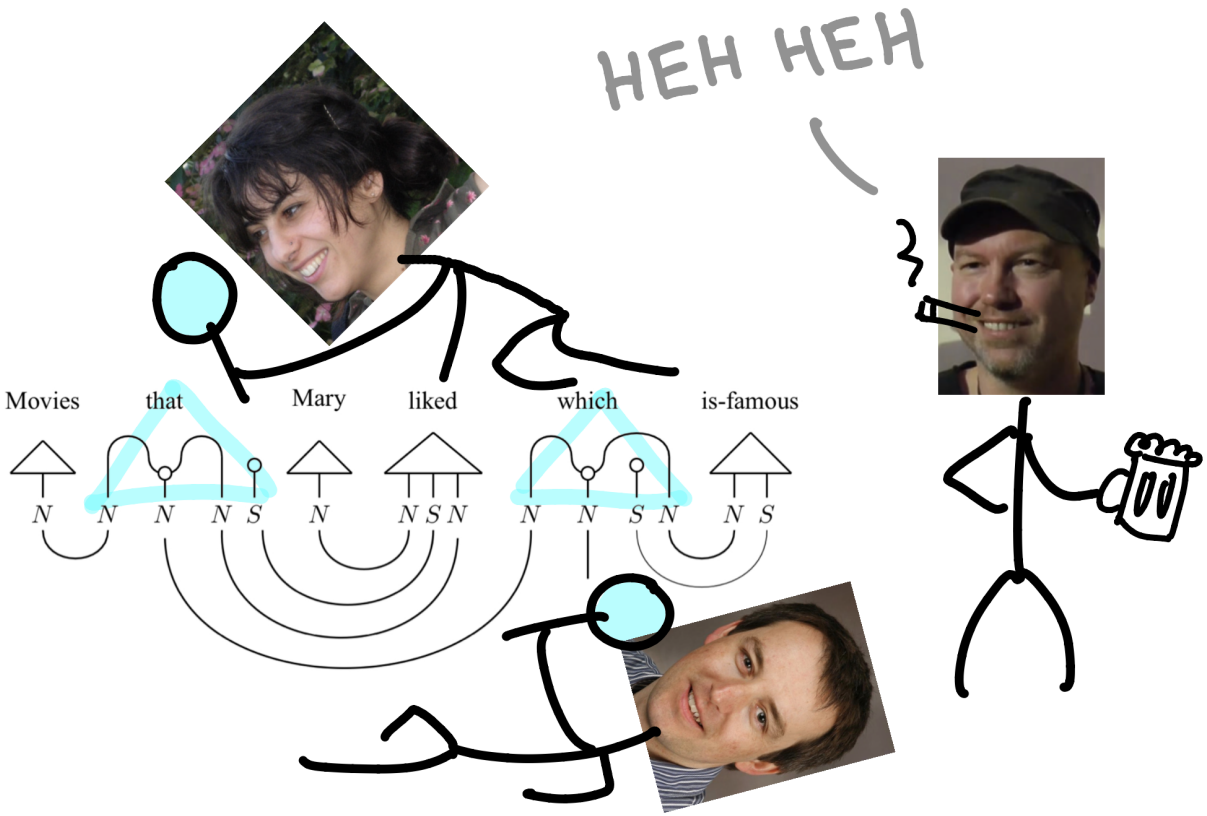


Figure 13: In *the frobenius anatomy of relative pronouns*CITE , the trio realised that spiders could play the role of relative pronouns, which was genuinely novel linguistics. If one follows the noun-wire of "movies", one sees that by declaring the relative pronoun to be a vector made up of a particular bunch of spiders-as-multiwires, "movies" is copied to be related to the "liked" word, copied again by "which" to be related to the "is-famous" word, and a third time to act as the noun in the whole noun-phrase. This discovery clarified a value proposition: insights from quantum theory could be applied in the linguistic setting, and linguistics offered a novel use-case for quantum computers. For example, density matrices were used to model semantic ambiguity CITE , and natural language experiments were performed on real quantum computeres CITE .

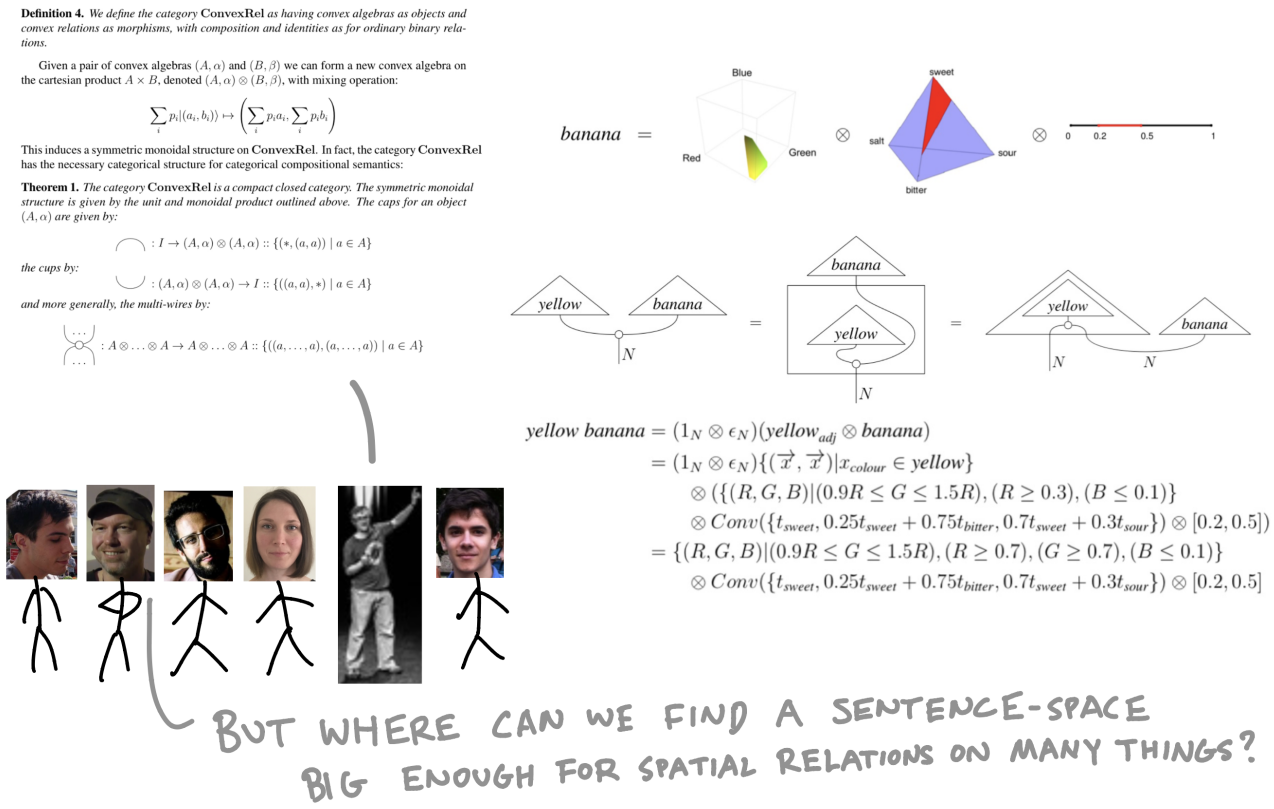


Figure 14: Keeping the structure of the diagrams but seeking set-relational rather than vector-based semantics, a bridge was made between linguistics and cognitive science in *interacting conceptual spaces* ICITE . Briefly, Gärdenfors posits that spatial representations of concepts mediate raw sense data and symbolic representations – e.g. red is a region in colourspace – and moreover that concepts ought to be spatially convex – e.g. mixing any two shades of red still gives red. This paper created a new point in the value proposition: that new mathematics would arise from investigating the linguistic-quantum bridge, e.g. generalised relations CITE . Although labelled as if it is the first in a series, the paper never saw a sequel by the same title, blocked by an apparently simple but actually tricky theoretical problem. The problem is that while this convex-relational story worked for conceptual adjectives modifying a single noun such for "sweet yellow bananas", there was difficulty in extending the story to work for multiple objects interacting in the same space, as in "cup on table in room". It couldn't be worked out what structure a sentence-wire in **ConvexRel** ought to have in order to accommodate (in principle) arbitrarily many objects and spatial relations between them.

DisCoCat then diverges from the story I want to tell. In no particular order, QNLP was done on an actual quantum computer CITE , some software packages were written CITE , and some art was made CITE .

0.2.5 *I killed DisCoCat, and I would do it again.*

Figure 15: It is a common evolutionary step in linguistics that theories 'break the sentential barrier', moving from sentence-restricted to text- or discourse-level analysis CITE . The same thing happened with DisCoCirc, due to a combination of practical constraints and theoretical ambition. On the practical side, wide tensors were (and remain) prohibitively expensive to simulate classically and actual quantum computers did not (and still do not) have many qubits, hence in practice pregroup diagrams were reduced to thinner and deeper circuits, often with the help of an additional simplifying assumption that sentence wires were pairs of noun wires in the illustrated form on the left. Theoretically, seeking dynamic epistemic logic, Bob had an epiphanous hangover (really) where he envisioned that these "Cartesian verbs" could be used in service of compositional text meanings, and he called this idea DisCoCirc CITE .

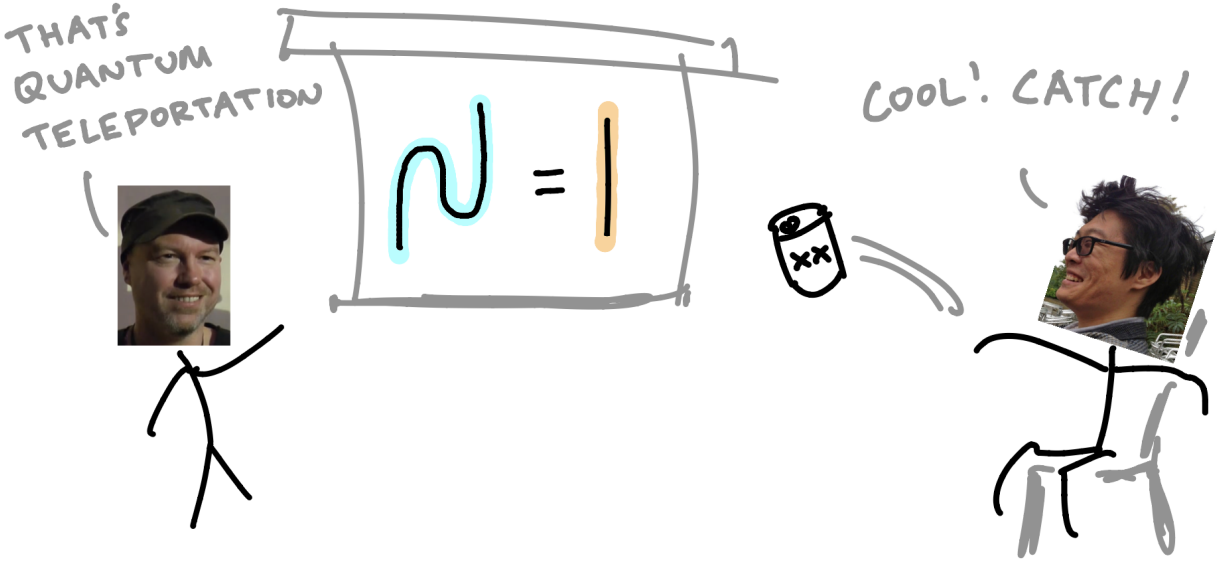


Figure 16: I met Bob in my master's in 2019, where he taught the picturing quantum processes course. When quantum teleportation was explained in half a minute by a diagram, I decided to pursue a DPhil in diagrammatic mathematics. In the last lecture, I threw Bob a cider, after which he seemed to like me. I did not know he was an alcoholic.

I was shanghaied into thinking about diagrams for language. I was deeply dissatisfied with the content from the standpoint my own intellectual integrity. Firstly, there seemed to me an unspoken claim that the presence of cups in pregroup diagrams (which implied a noncartesian and hence large tensor product) made it necessary to use quantum computers to effectively compute pregroup diagrams. I just could not believe that my brain required quantum computation to understand language. This implicit claim of kinship between quantum and linguistics was further entrenched by the analysis of the relative pronoun in terms of frobenius algebras, since spiders in $\mathbf{Vect}^{\otimes}$ were the *sine qua non* of categorical quantum mechanics. The best steelman for spiders I have is that frobenius algebras (which are central to bicategories of relations CITE) just happen to be a ubiquitous mathematical structure that are well-suited to express the mathematics of connections, both in language and in quantum.

Second, representing the content of a sentence as a vector in a sentence-vector-space did not sit well with me, since this move meant that the only meaningful thing one could do with two sentences was take their inner-product as a measure of similarity. Moreover, I had the theoretical concern that language is in principle indefinitely productive, so one could construct a sentence that marshalled indefinitely many nouns, and at some point for any finite vector space s one would run out of room to encode relationships, or else they would be cramped together in a way that did not suit intuitions about the freedom of constructing meanings using language. I always believed in the existence of a simple, practical, and intuitive categorical, compositional, and distributional semantics; I just didn't believe that the role of quantum – however helpful or interesting – was *necessary*.

My first unsatisfactory attempt was in my Master's thesis CITE . It had been known for a while that a free autonomous category construction by Delpeuch CITE could potentially eliminate some of the cups in pregroup diagrams, yielding what amounted to a method to transform a pregroup diagram into a monoidal string diagram in the shape of a context-free grammar tree. This trick had the limitation that freely adding directed cups and caps to a string diagrammatic signature did not turn a symmetric monoidal category into a (weakly) compact closed one, rather just into a monoidal category where the original wires had braidings, but all the new left and right dual wires did not; this presented difficulties in accounting for iterated duals for higher-order modifiers such as adverbs in grammatical types, and had nothing to say about spiders. I tried to generalise this trick to 'freely' adding arbitrary diagrammatic gadgets to string diagrams, but my assessor Samson pointed out that it was nontrivial to determine whether such constructions were faithful. In retrospect the free autonomous completion of a parameterised CITE markov category CITE is in the ballpark of dequantumfying pregroup diagrams, but I didn't learn about them until later, and that still wouldn't have addressed the issues that come with only having a sentence-wire.

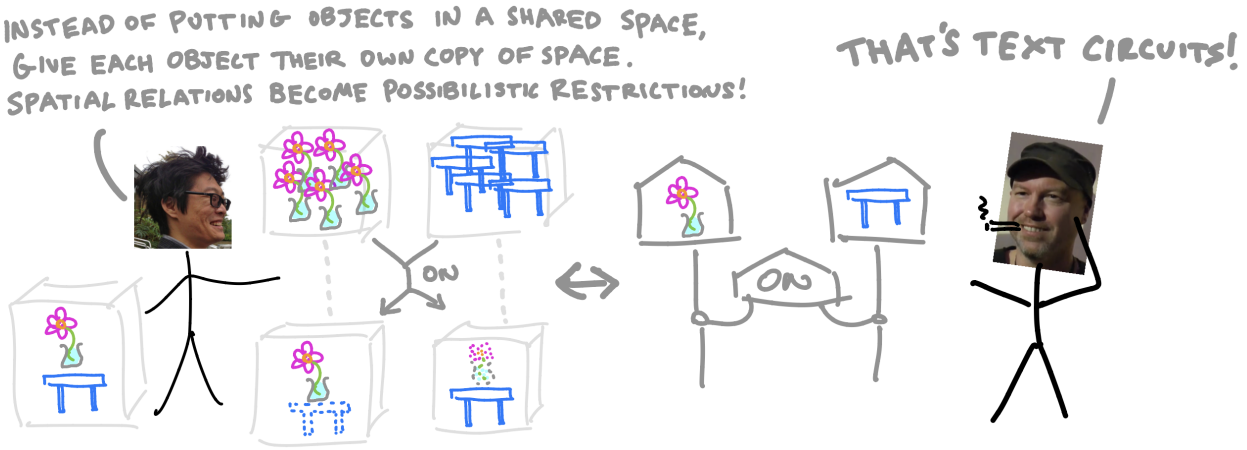


Figure 17: Then COVID happened. During the first lockdown, I visited Bob's garden under technically legal circumstances, and I suggested a solution to the longstanding problem of representing linguistic spatial relationships. My theoretical concern was the culprit: the initial attempts at the problem failed because the approach was to find a single sentence object s in which one could paste the data of arbitrarily many distinct spatial entities. The simple solution was a change in perspective.

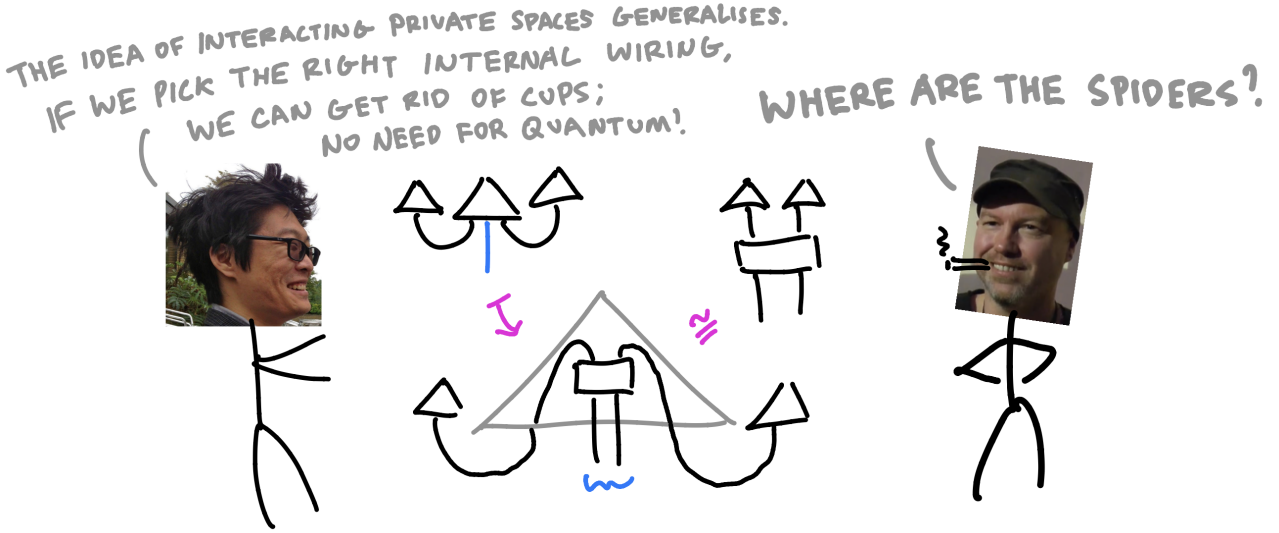


Figure 18: That this move of splitting up the sentence-wire into a sentence-dependent collection of wires was sufficient to solve what had appeared to be a difficult problem prompted some re-examination of foundations. The free autonomisation trick in conjunction with sentence-wire-as-tensored-nouns seemed promising, but it became clear that right way to drown a DisCoCat thoroughly was to explain and eliminate the spiders.

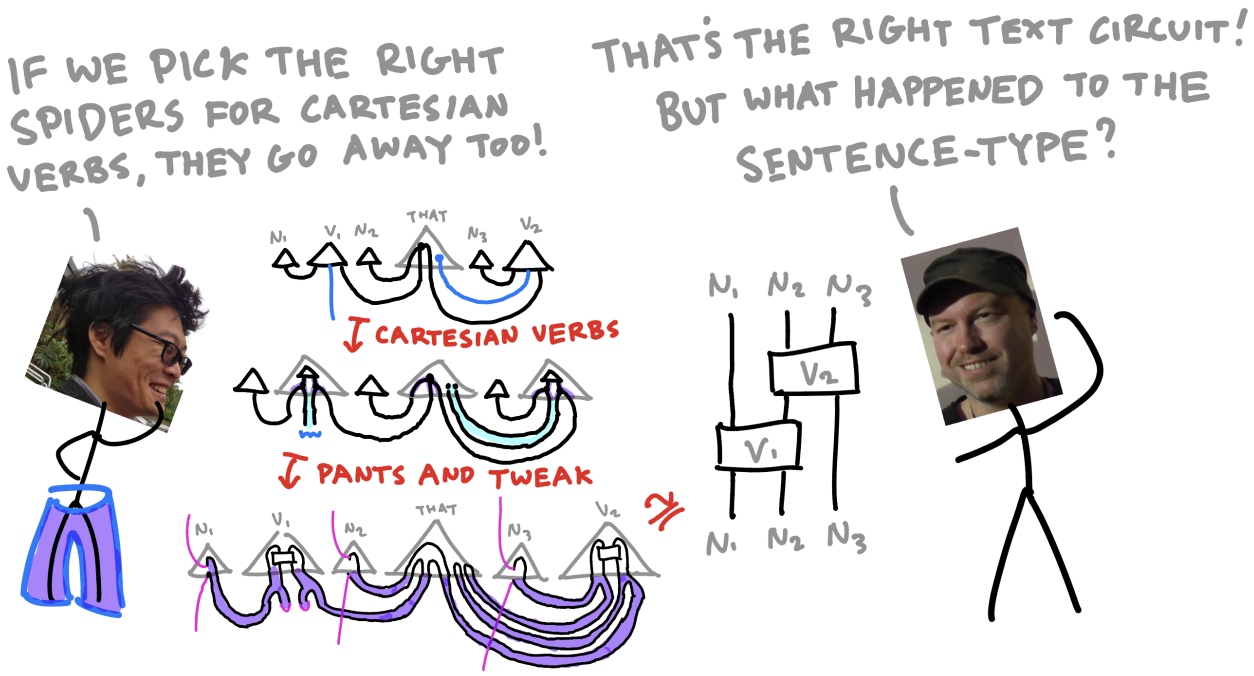
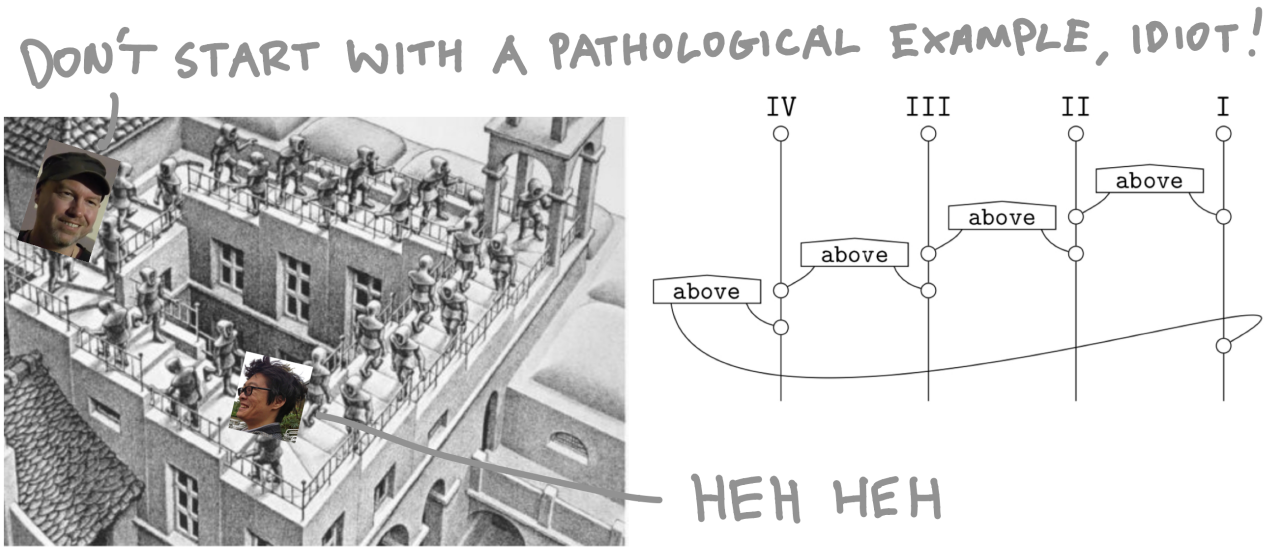


Figure 19: I then discovered that by interpreting spiders as the well-known "pair of pants" algebra in a compact closed monoidal setting allowed for a procedure in which the final form was purely symmetric monoidal – the absence of cups and caps meant that there was no practical necessity to interpret diagrams on quantum computers: *any* computer would suffice. The role of spiders for relative pronouns was illuminated in the presence of splitting the sentence wire: the pair-of-pants are the algebra of morphism composition, and splitting the sentence wire into a collection of nouns allowed relative-pronoun-spiders to pick out the participating nouns to compose relationships onto.

Figure 20: A coherent conservative generalisation of DisCoCat with less baggage had emerged, or rather, DisCoCirc was placed to formally subsume DisCoCat. It was now understood that the sentence type was a formal syntactic ansatz for the sake of grammar, which was to be interpreted in the semantic domain not as a single wire, but as a sentence-dependent collection of wires. It was further realised that the complexity of pregroup diagrams was due to grammar – the topological deformation of semantic connections to fit the one-dimensional line of language – whereas the essential connective content of language could be expressed in a simple form that distilled away the bureaucracy of syntax.

Figure 21: We wrote up the story about spaces in CITE , the spiritual successor to *interacting conceptual spaces I*. We could formally calculate the meanings of sentences that used linguistic spatial relations, all using a simple and tactile diagrammatic calculus.



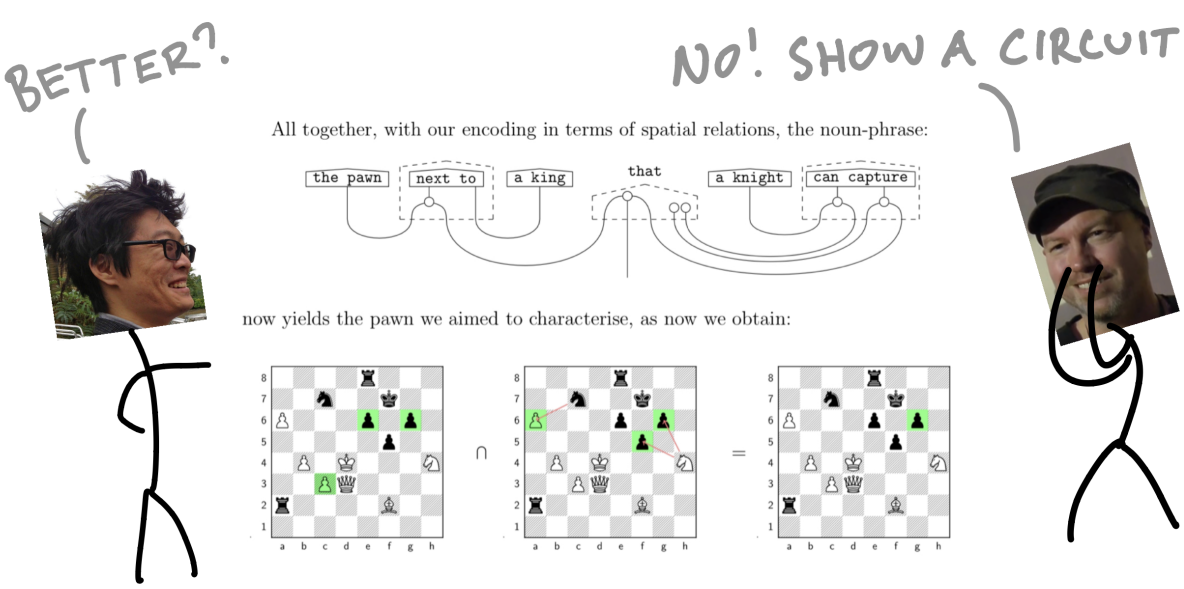
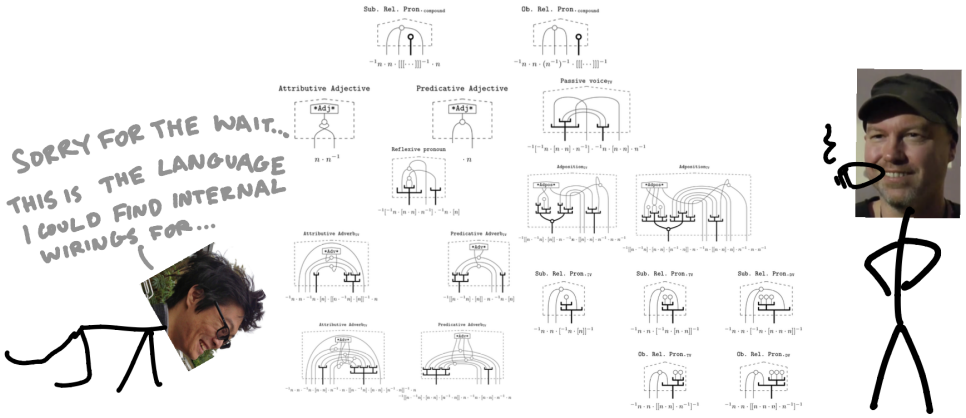
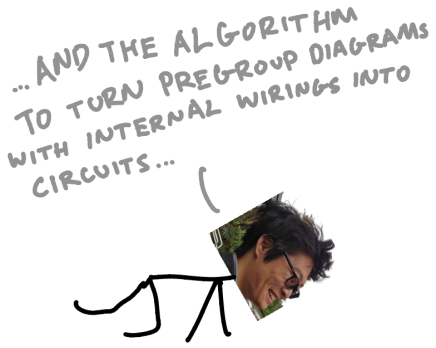


Figure 22: The paper on spatial relations actually came very late, because I was busy with Bob's ludicrous request to go turn "all of language" into circuits. I bitched and moaned about how I wasn't a linguist and how it was an impossible task, but I was in too deep to back out.

Figure 23: I suppose the nice thing about aiming for the moon is that even failure might mean you leave orbit. So I settled for what I thought was a sensible fragment of English, for which I devised internal wirings and an algorithm that transformed pregroup diagrams with the internal wirings into circuit form. Many tiring diagrams later, I presented my results in the first draft of "distilling text into circuits".





Algorithm 1: Grammar to Circuit Algorithm

Input: A sentence S
Result: A text circuit
Data: ***Parse***, ***PrnRes***, NOUNS, PRONOUNS
BASICWIRES $\leftarrow \{w \in S \cap \text{NOUNS}\}$; // 'BASICWIRES' is nouns of S
DIAGRAM $\leftarrow$ ***Parse***(S) ; // 'DIAGRAM' is pregroup diagram with wrapping
if *'DIAGRAM' has single output wire* o **then**
  Append appropriate deletion to o in DIAGRAM ; // delete output wire
  for $w \in$ *BASICWIRES* **do**
    Erase w's box from DIAGRAM, creating open wire ; // copy nouns
    Append rightmost output of a copy-spider to the open wire;
    Pull free input of copy-spider to the top of DIAGRAM;
    Pull free output of copy-spider to the bottom of DIAGRAM;
    Label this new wire with w;
  end
  while *There remain wrapped wires in DIAGRAM* **do**
    Apply unwraping from Definition 3 ; // unwrap
  end
  for $w_1, w_2 \in$ *BASICWIRES* **do**
    if $w_1 \in$ *PRONOUNS and* $w_2 \notin$ *PRONOUNS and* ***PrnRes***(w_1, w_2) **then**
      Attach outputs of a copy-spider to w_1, w_2 inputs;
      Attach inputs of upside-down copy-spider to w_1, w_2 outputs;
      relabel merged wire as w_2 ; // resolve pronouns
    end
  end
  while *There remain deletions, cups, or caps of non-wrapped wires* **do**
    Use spider-fusion to remove deletions, cups, and caps;
  end
  Simplify to obtain text circuit ; // clean up
  return DIAGRAM ; // return text circuit
else
  return False ; // (Fail if input ungrammatical)
end

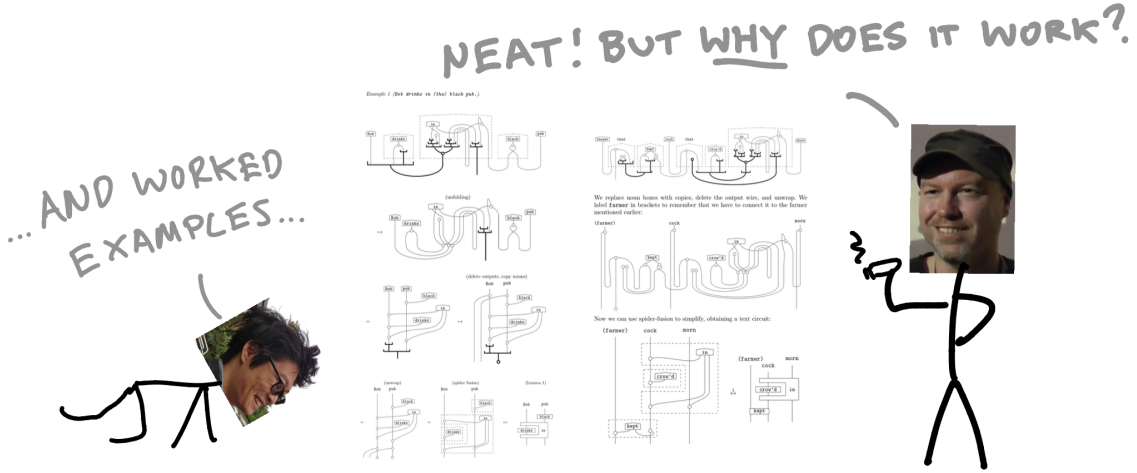


Figure 24: Bob had a good point. Everything worked, but we had no understanding as to why, and accordingly, whether or not it would all break. At this point in time, Jonathon Liu, a masters' student I tutored during COVID, had committed the grave error of thinking diagrams were cool, and was now hanging out with me and Bob. After understanding the procedure, Jono independently devised the same arcane internal wirings as I had, but neither of us could explain how we did it. So we had evidence of an underlying governing structure that was coherent but inarticulable.

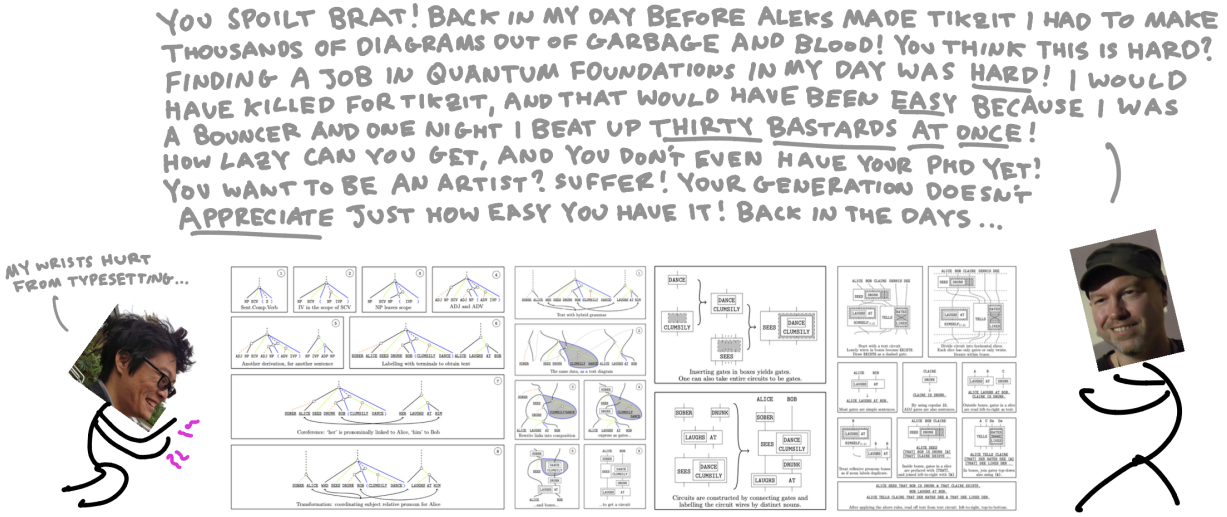


Figure 25: I realised that our intuitions were coming from an implicit productive grammar, rather than a parsing one, and that the path of least resistance for obtaining formal guarantees for the language-to-circuit procedure was to just handcraft a generative grammar for the fragment of language we were interested in. This meant scrapping everything in the first draft and starting again from scratch. Bob always had a word of gentle encouragement, giving me the motivation to persevere.

So now we had two ways to obtain text circuits. One from pregroups (which Jono had extended the technique for to CCGs in his master's thesis CITE), and one from handcrafted productive grammars. Then came time for me to write my thesis, and there were three salient questions I wanted to address.

Firstly, what are internal wirings?

Secondly, how do text circuits relate to other generative grammars?

Thirdly, what are text circuits good for?

These questions are now what the rest of the thesis seeks to answer.

1
Bibliography

[Fri05] Harvey Friedman. [FOM] Characterization of R/Simple proof, February 2005.