MSc in Computer Science 2018-19

Project Dissertation

Graphical Grammar
+
Graphical Completion of Monoidal Categories

Trinity 2019

Vincent Wang

Abstract

This is a thesis in two complementary parts.

The first part informally suggests the graphical calculus for monoidal categories as a unifying framework for compositional semantics in generative and typelogical grammars, one in which the two approaches are category-theoretic duals.

The second part formally establishes a construction that completes arbitrary monoidal categories with respect to near-arbitrary graphical equations encoding structural monoidal natural transformations, offering freedom for theorists to explore and calculate with structures of their own devising.

Acknowledgements

To PM, for everything.
To PK, for inspiring me to think big picture.
To BC, for the gift of diagrams. This result is my gift to you.
To friends and family. They know who they are.

Contents

Part I
Graphical Grammar

The purpose of this part is to provoke a connection between compositional semantics, and two kinds of approaches to formal grammar analyses of natural language: Generative Grammars and Typeological Grammars.

We assume that meaning is compositional. Given this assumption, these are the visions we wish to communicate: we believe that the role of grammar is to direct the composition of meaning, *and* we believe that in doing so, grammar and semantics do not impose constraints on one another. We believe that the mathematical structures that generate grammatically correct sentences are, suitably conceived, *categorially dual* to the mathematical structures that evaluate the meaning of sentences.

We are not in the business of providing a Theory of Language, rather we seek to invite applied category theorists and formal linguists towards consideration of a *lingua franca* for Theories of Language. As such, in this work we are concerned about empirical linguistic data only insofar as they recommend constraints on expressivity in the theories we are attempting to establish common ground for.

Also, we'll do it in pictures.

1 Compositional Semantics

1.1 Compositionality

The proposed ideological glue is Frege's (Principle/Conjecture) of Compositionality, which is the bare minimum of any *analysis* – from the greek *άνα-* (above/throughout) and *λὺσις* (decomposition/disintegration). We let the man speak for himself [Max66, p.31] (emphasis mine):

> . . . we split up the sentence
>
> "Caesar conquered Gaul"
>
> into "Caesar", and "Conquered Gaul." The second part is "unsaturated" – it contains an empty place; only when this place is filled up with a proper name, or with an expression that replaces a proper name, does a complete sense appear. Here too I give the name "function" to what this "unsaturated" part stands for. In this case the argument is Caesar.

Compositionality as *functional application* is a good first approximation, and one that has enjoyed staying power. The Set Theory that Frege and Cantor developed eventually found favour with Bourbaki[Paw], whose structuralist philosophy is evident in the presentation of nearly all undergraduate mathematics even today: if the mathematics was developed earlier than the 20^{th} century, there is a great chance that it is formally presented by sets and functions between them.

At first, 'function' was defined in terms of analyticity. In the precursor to the mathematical field of Analysis, (single variable) functions were conflated with analytic representations – algebraic composites of 'simple' monovariate functions, such as polynomials – which today we recognise as differentiable functions. Fourier made the distinction between functions and their analytic representations – fourier series, though analytic, could represent discontinuous modern functions. In

parallel, logicians[Boo10] began their investigations into symbolic logic with functions defined by monovariate algebraic expressions. With time, the notion of function converged upon its present day form, as a many-one binary relation.

The essence of the modern day function is arguably *determinism*. Functions relate 'inputs' to 'outputs', in such a way that the value of the inputs determine the values of the outputs. In practice, this determinism yields an implicit *temporality*[noab] via epistemic asymmetry: if we know the inputs of a known process, we can calculate the outputs, but if we only know the outputs, in general we are left uncertain about what the inputs were. This ontology of function as *process* that transforms inputs to outputs in a deterministic way is familiar to our everyday intuitions of physical processes, and this will be our preferred ontology.

The ideas above are old. The fresh approach is a new mathematical language for speaking and reasoning about old ideas: Applied Category Theory, a branch of mathematics that concerns itself with Compositionality. Category Theory is 21^{st}-century mathematics, borne of a desire to seek unification in the various structuralist presentations of mathematics. The 21^{st}-century philosophy that Category Theory engenders is that of *synthesis*: broadly characterised, a structuralist concerns themselves with the inner structure and mechanics of a mathematical object, whereas a category theory concerns themselves with how different mathematical structures interact.

In particular, we are interested in the theory of monoidal categories, which is all about the sequential and parallel composition of processes. Do not be fooled by the simplicity of the diagrams that we draw in this part: the second part of this work will place them on formal grounds as solid as any other.

1.2 Process Theories

The presentation in this section is a quick overview of concepts expressed in more detail in [BC17, Ch.3]. We just introduce circuit diagrams in the context of **Set**, and we detour just before reaching string diagrams for compact closed categories and **Rel**, instead introducing Delpeuch's free autonomous construction[Del14].

A Process Theory ontologises the world as stuff and processes. There are types of stuff, and that's all we know about stuff. Processes take stuff as input, and produce stuff as output, provided that the types of stuff going in and out match the type specification of the process. Processes can be composed with one another to form new processes. The 'Theory' part of 'Process Theory' refers to the notion of equality of processes that one wishes to consider.

Example 1 (An analogy)**.** Consider power socket converters between international standards. A converter accepts input of certain standards and provides outputs of certain standards; these are the processes. A US-to-UK converter can't be plugged into a Brazil-to-Tonga converter; this is what is meant by well-typedness. A US-to-UK converter can be plugged into a UK-to-Brazil converter, yielding a US-to-Brazil converter; when typing is satisfied, processes can be composed. Suppose your friend needs a US-to-Brazil converter, and you have both a single device that does the job, and the composite US-to-UK-to-Brazil converter from before. Though these are different devices, to your friend *they are just as good as one another*; this kind of equality is what a Process Theory is about.

1.2.1 The basic graphical language of processes

Let us consider a particularly simple and expressive family of process theories, where there are only two modes of composition: vertical, and horizontal. It would not be remiss for the time being to imagine that vertical composition is temporal or sequential in nature, and that horizontal composition is spatial or concurrent in nature. In Figure 1 we present the elements of a graphical syntax for expressing such processes.

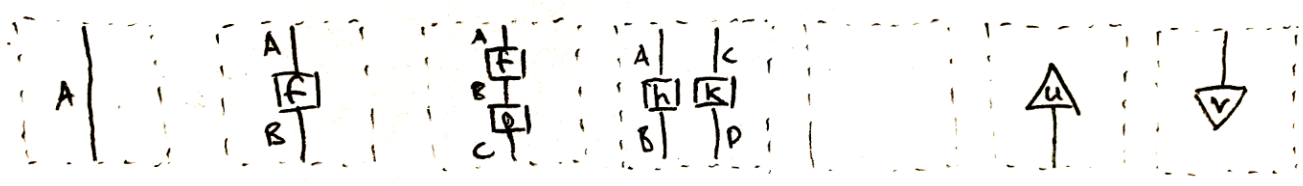


Figure 1: Each dotted box represents a bounding box, in which we form our graphical expressions. Each well-formed dotted box is itself a process, and can be used in other composite processes formed by the rules below. From left to right:

- A wire of type A, which ferries stuff of that type.
- A process named f, drawn as an axis-aligned box, which takes stuff of type A as input, and outputs stuff of type B.
- A vertical composite process of named processes f and g. First f 'transforms' A-stuff to B-stuff, and then g transforms that B-stuff into C-stuff.
- A horizontal composite process of named processes h and k. Since we can name h and k separately, why not invoke those processes concurrently? This composite takes A-stuff and C-stuff, and returns B-stuff and D-stuff.
- The special 'nothing' process.
- A process u that gives some output, but takes no input. We call these 'states'.
- A process v that accepts some input, but gives no output. We call these 'effects'.

Observe that in the graphical syntax, really all that is left is *processes*. Specific states might "create" *stuff*, but this stuff is hidden under and ferried by wires[1].

[1]Wires themselves are special processes known as *identities*.

Our graphical syntax has few constraints, but many affordances. Here are the rules:

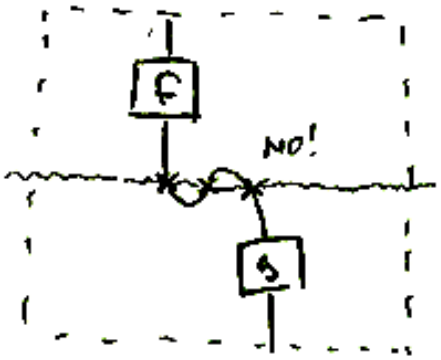


Figure 2: We assume that wires are *processive*, in that for any horizontal slice of a diagram, any single wire only intersects that slice once.

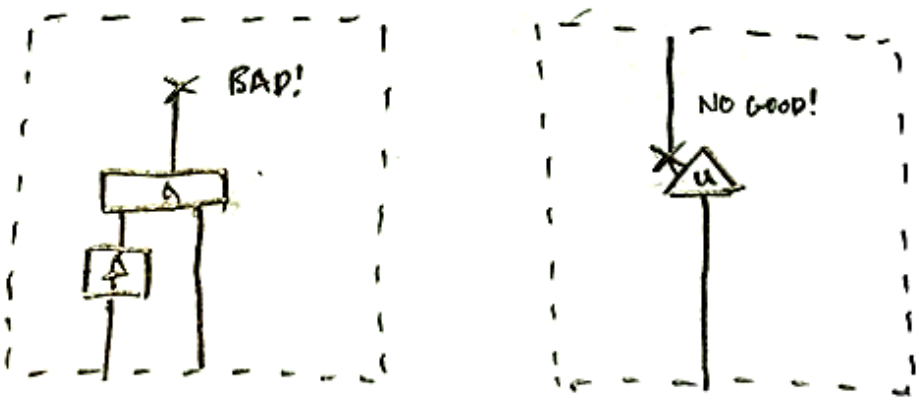


Figure 3: We assume that all wires terminate either at the top or bottom of the bounding box, or at the top or bottom of a process-box, and no wires should terminate at the 'nothing' end of a state or effect.

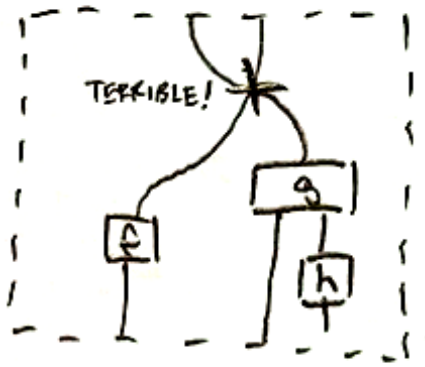


Figure 4: Wires should not cross.

Figure 5: If two diagrams obey the above rules, and they can be deformed into one another by *planar isotopy*, which is to say, continuous transformations of the plane, then they are equal.

Sets and functions between them form a process theory in this way. Wires are sets, and boxes are functions. If a box takes multiple wires $A, B, \ldots, Z$ as input, or outputs multiple wires, we consider the function to be taking in tuples $(a, b, \ldots, z) \in A \times B \times \ldots Z$ as input, and we treat outputs similarly. States of type A are elements of the set A. Effects in set are boring, because we take an arbitrary singleton set to stand in for 'nothing', and there is only one function from any set into a singleton. The reason we take singletons $\{\star\}$ to be 'nothing' is because we interpret parallel composition as the product of sets, and $A \times \{\star\} \simeq A$; so it really behaves like 'nothing', in that products with a singleton don't express any extra data. Formally, we say that the category **Set** with tensor product as the categorical product forms a monoidal category.

However, in for instance, **Set**, there may be graphical equations that hold beyond planar isotopies. For instance, we know that there is function $(+1) : \mathbb{N} \to \mathbb{N}$ that maps any natural number $n \mapsto n + 1$.

1.2.2 Equations and Substitution

If you know that certain equations hold in your monoidal category, you can use them to reason graphically. Suppose we know that that the process $(+1)$ applied to the state n is equal to the state $n+1$. We can express this fact in a system of graphical equations as follows:

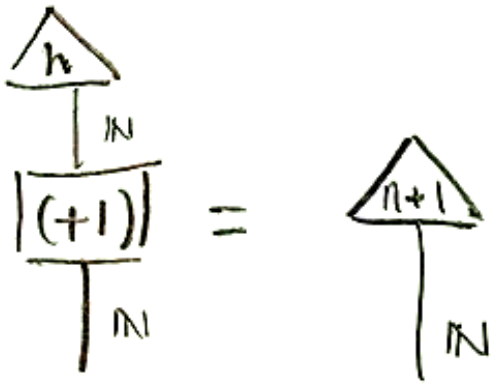


Figure 6: Simple stuff. We will stop drawing the bounding boxes now.

The general rule for well-formedness of such graphical equations is that the incoming and outgoing wires on the bounding boxes of the two sides of the equation must match. We can use these equations to calculate via diagrammatic substitutions.

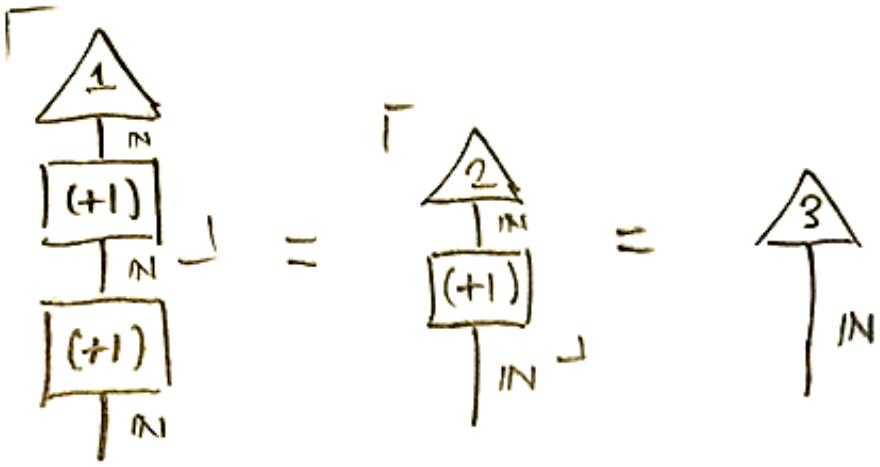


Figure 7: Reading left to right, we mark just the top-left and bottom-right corners of the subdiagrams we apply an equation to.

In general, we might write out a system of substitution rules for **Set** that look like this:

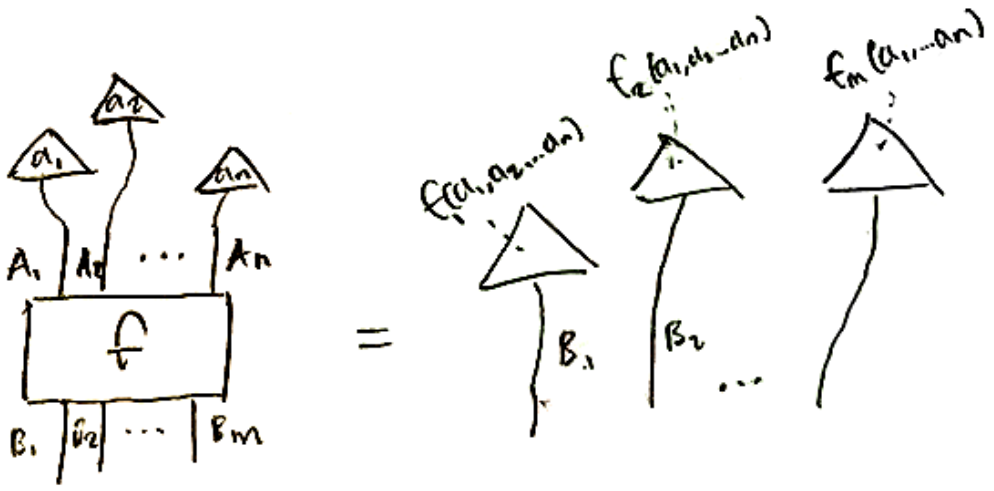


Figure 8: Still simple stuff.

This is all well and good, but sometimes we might wish to specify certain properties about our processes that are cumbersome to encode by a system of equations in this way. For instance, consider the binary addition operation $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. We know by arithmetic that for any natural numbers a, b, that $a + b = b + a$:

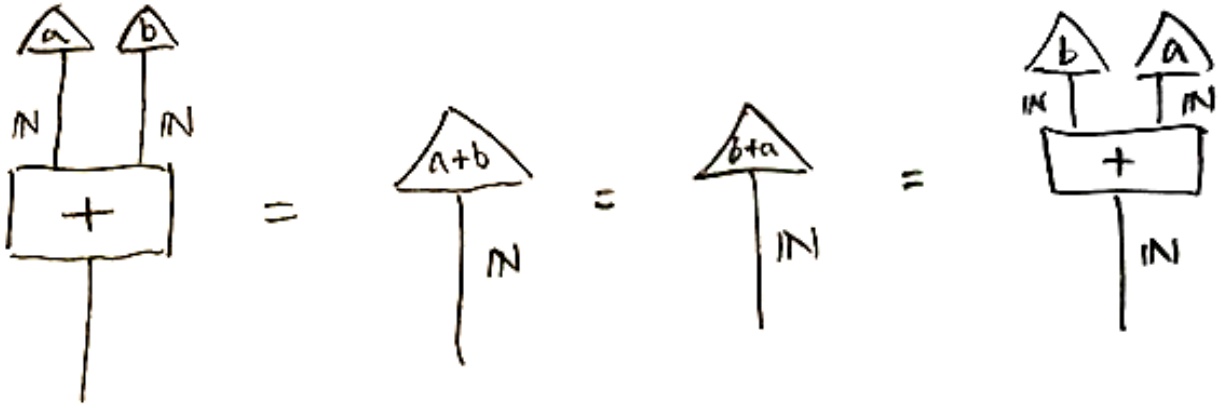


Figure 9: Here, it doesn't matter what a and b are: this equation always holds, because that's just how addition works.

What we really want to say is that addition is *symmetric*: it doesn't matter what order we feed in the arguments. We might want to write an equation like this:

Nobody needs to be shot just yet. We are proud to announce that centuries of mathematical development has culminated in a method to simulate crossing wires.

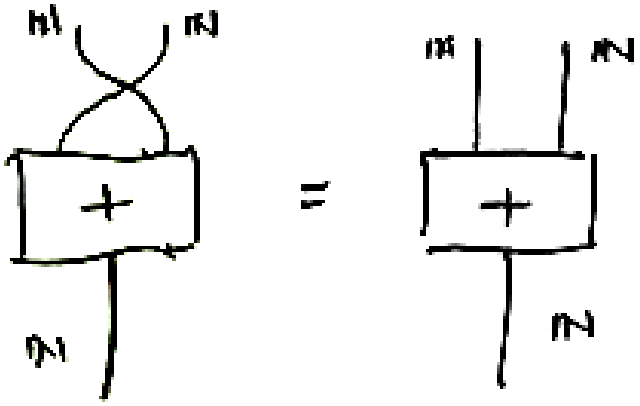


Figure 10: But the wires cross! Wasn't that forbidden?

1.2.3 As if the wires cross

Suppose, for all pairs of wires in the monoidal category – which is to say in **Set**, all sets A, B – we have a special process $\theta_{A,B}$ that satisfies the following equations.

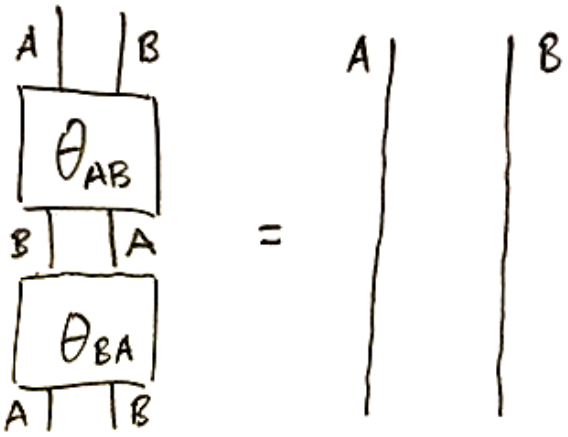


Figure 11: Two of these special processes in succession cancel each other out.

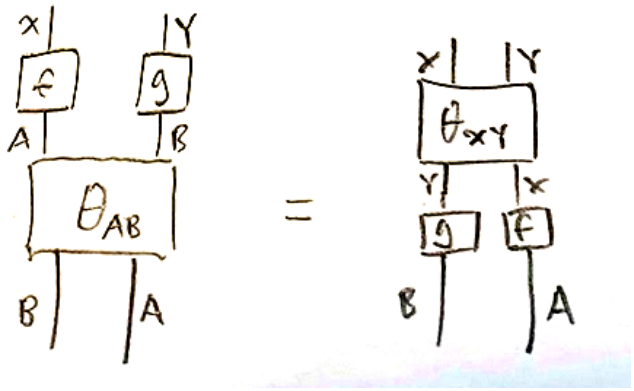


Figure 12: The special processes swap places with other processes like so.

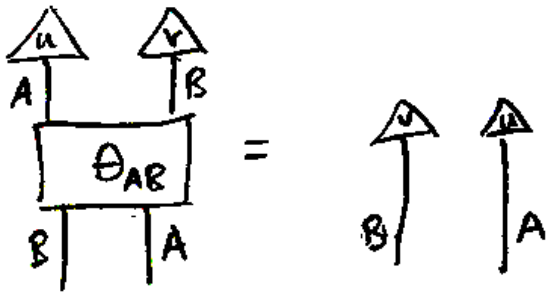


Figure 13: And a special case for states.

We might choose to label the boxes differently, and perhaps suggestively:

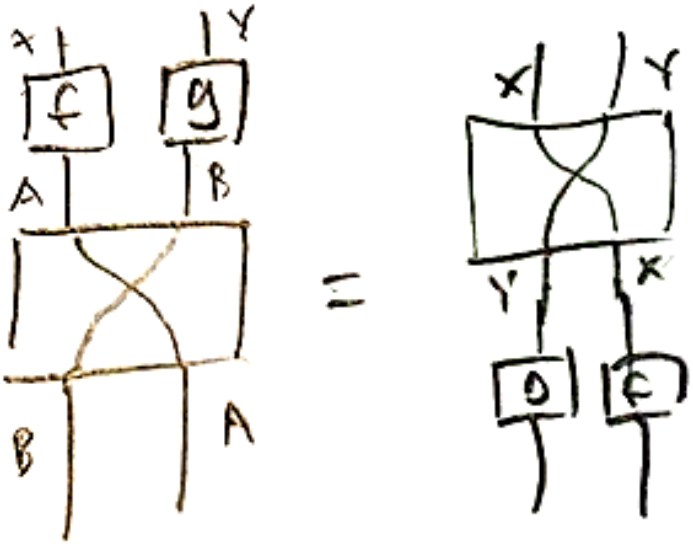


Figure 14: Now it looks as if the processes f and g slide down the wires.

So there's the trick: we use processes to simulate the mathematics of crossing wires. Thinking

of the special processes θ as 'hiding' crossed wires in this way, the figures above seem intuitively clear: crossing wires over twice is the same as doing nothing[2], and processes can slide along crossed wires. In **Set**, such a process exists for all sets A, B: $\theta_{A,B} := A \times B \to B \times A/(a, b) \mapsto (b, a)$. So we say that **Set** is *symmetric monoidal*. As graphical shorthand in symmetric monoidal categories, we draw the special processes $\theta_{A,B}$ as crossed wires, though secretly, they still have invisible boxes around them.

1.2.4 As if the wires bend backwards

So we have cheated the rule that wires cannot cross. What about the rule that wires cannot bend backwards? Well, suppose that we have special processes η_A and ϵ_A for all wire labels A, which look like this:

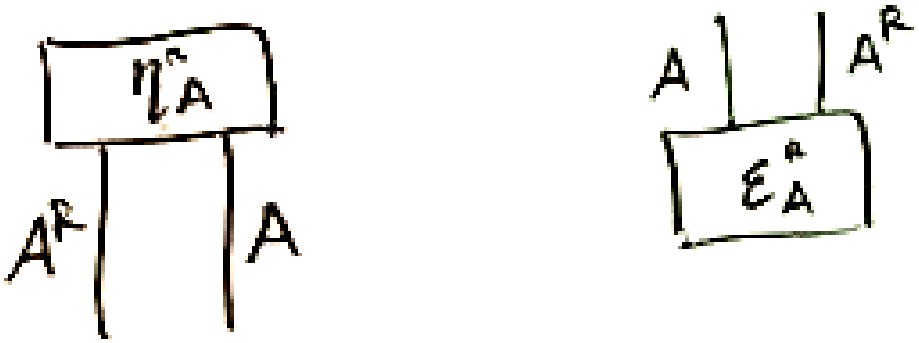


Figure 15: The unit and counit of a right-autonomous monoidal category. A fancy way to say that these processes will pretend to bend the wire to the right. Or left. Plus I keep forgetting which one is the unit and which is the counit. Ultimately not that important. There are also vertically reflected versions with A^L instead of A^R, and those give left-autonomous monoidal categories.

The A^R is a special wire label, functionally determined by A. We might consider the alternative presentation $R(A)$, where R is a function from the collection of wire labels (objects in our monoidal category, in this case sets) to itself, though we haven't evaluated the function yet. For the left and right units and counits, we want the following equations to hold:

[2]Not so in braided monoidal categories.

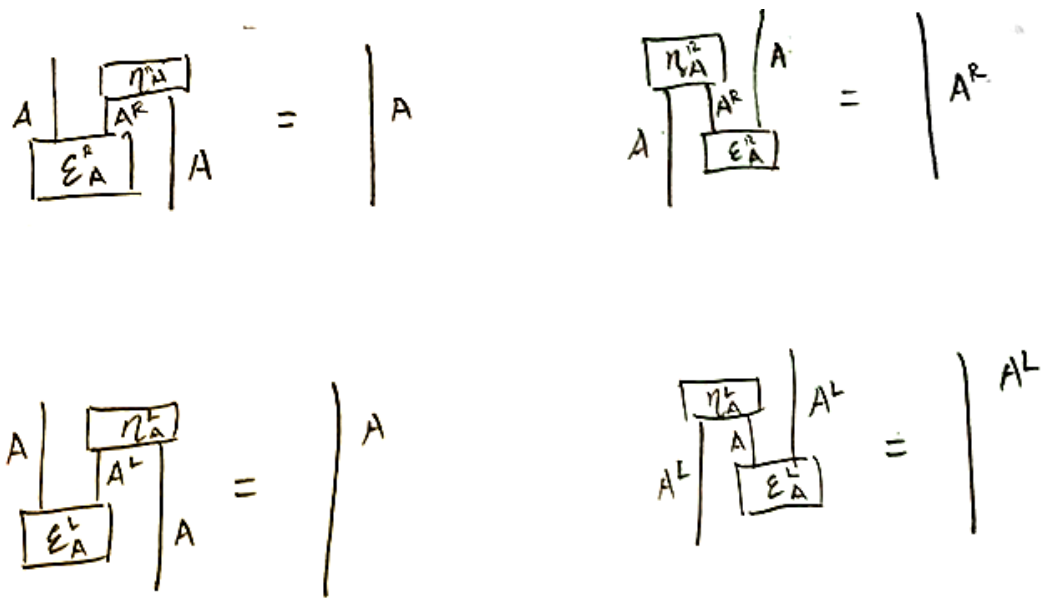


Figure 16: See where we're going? Our physical intuitions should kick in once we redecorate the boxes and wires more suggestively in the next figure.

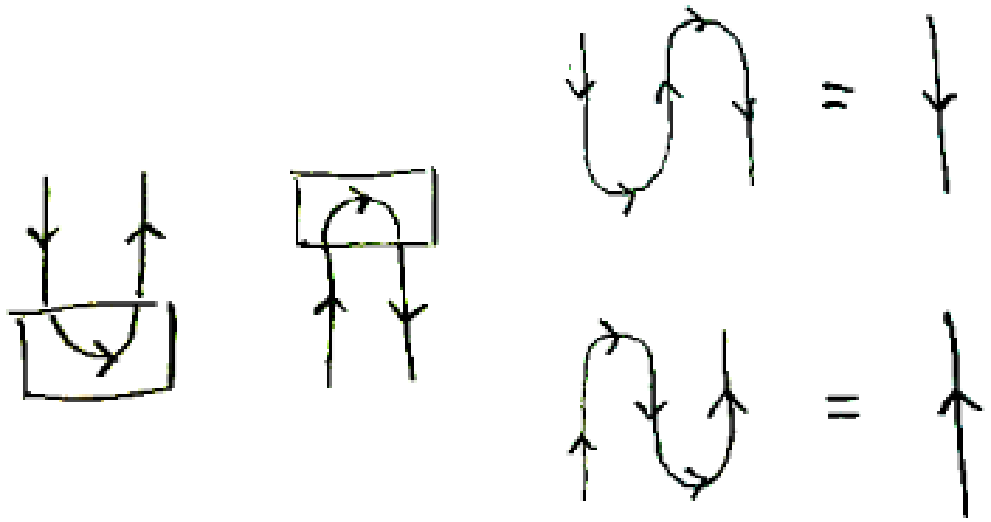


Figure 17: Cups, Caps, and the *Yanking Equations*. Technically speaking, this notation where wires are marked with direction is only correct for compact monoidal categories, which are planar autonomous (left + right autonomous) symmetric. Really formally we would have to keep track of the number of times the wire bends back left and right.

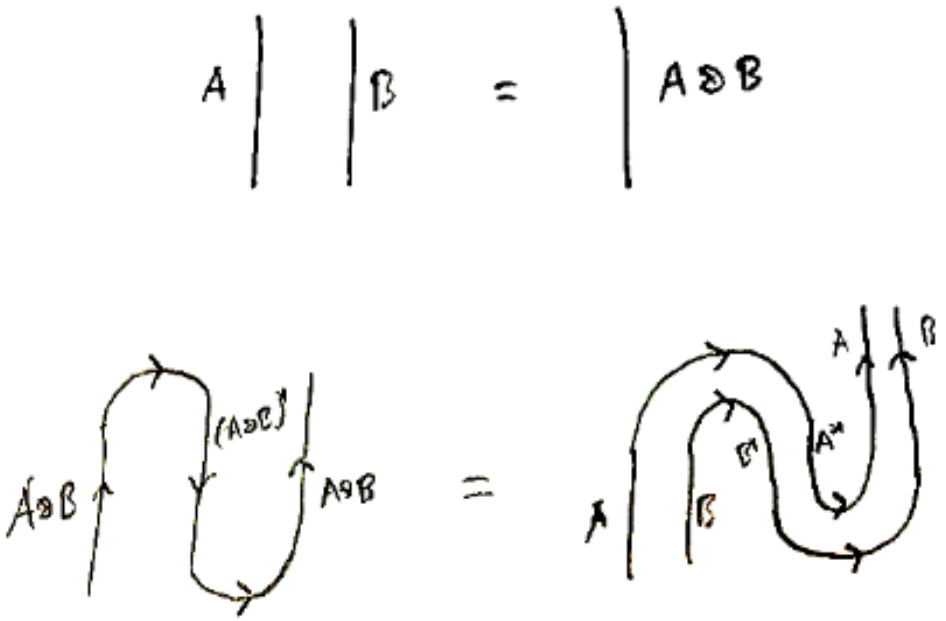


Figure 18: There is also, technically, a graphical rule that allows one to 'compress' and 'decompress' wires (above). A useful shorthand, but not really necessary in plain monoidal categories. When we are using processes to fake bending wires, we have to additionally specify how autonomous structure behaves with respect to it (below). If we didn't specify this rule, $(A^* \otimes B^*)^*$ (not depicted) would define the multiplicative connective of linear logic [KU17, p.9-10][Gir11, p.188]

Sadly, **Set** doesn't have processes that behave in this way. The reason is that there is only one effect in **Set**. Recalling that the 'nothing' set is an arbitrary singleton $\{\star\}$, there's a unique function from any set to the singleton, so we have the following graphical equations in **Set**:

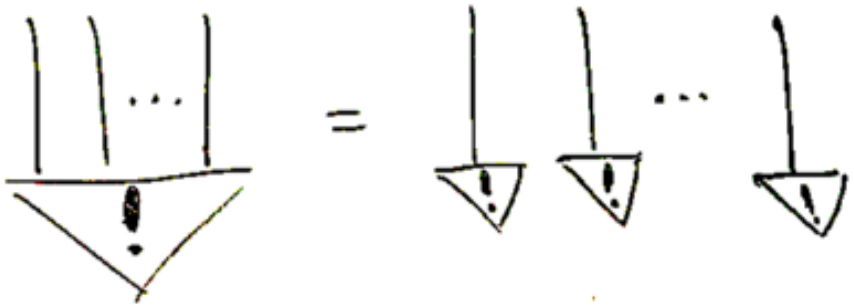


Figure 19: We notate the unique effect with **!**.

Thus in **Set**, starting with one side of a yanking equation results in a broken wire, as shown in figure 20.

So, we have the equation in figure 21 for all sets in **Set**:

Thus there are no interpretations of cups and caps in **Set**. There are, though, in **Set**'s nondeterministic cousin **Rel**, the category of sets and relations between them, and also in **FdHilb**, the category of finite dimensional Hilbert Spaces and bounded linear maps between them. The presence of cups and caps is fairly useful for modelling phenomena in quantum mechanics[BC17, §4.1.2].

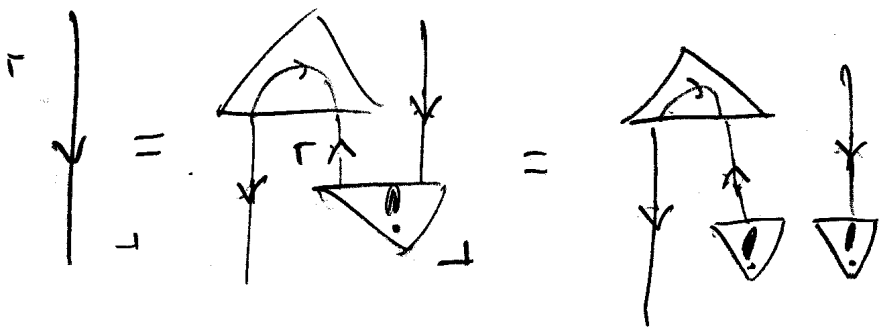


Figure 20: Quite dangerous. The passage from the first to second diagram is the yanking equation, and the passage from the second to the third is the unique-effect equation.

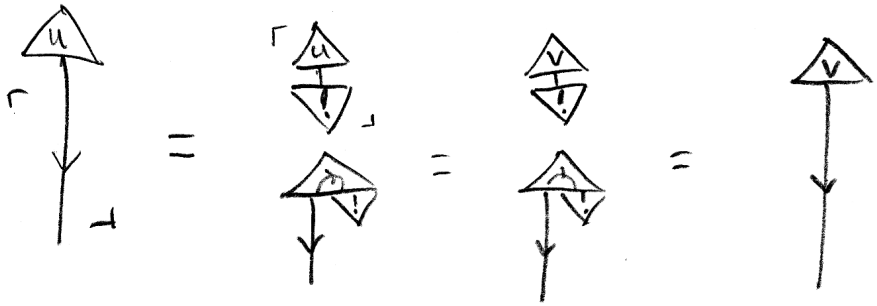


Figure 21: The first equation follows from above. The replacement in the second equation follows because there is only one function from the singleton to the singleton. The final equation simply reverses the first two, now with v in place of u. The consequence of equating the leftmost and rightmost diagrams is that all elements of any set are equal, as u and v were arbitrary: contradiction.

But we have a different trick to pull. Recall that the graphical calculus concerns the sequential and parallel composition of processes *modelled in* what we have been calling a monoidal category, where sequential composition in the monoidal category is depicted vertically and respects ingoing and outgoing types, and parallel composition is depicted horizontally. We can define a (monoidal) category of *diagrams* drawn in this way, by modelling vertical composition of diagrams as sequential composition in the monoidal category, and horizontal composition of diagrams as parallel composition in the monoidal category.

In this monoidal category of diagrams, we can ask that the cups and caps η and ϵ are included in the stock of diagrams, and we can ask that they obey the yanking equations. We call this category $\mathcal{L}(\mathbf{Set})$: the free autonomous completion of **Set**[Del14]. The η and ϵ diagrams aren't from the same universe as diagrams from the graphical calculus of **Set**, and we keep track of this fact by drawing veteran diagrams from **Set** in bold, and new, added diagrams thin in the diagrammatic calculus of $\mathcal{L}(\mathbf{Set})$.

There is an upside to this trick: note that in the figure 22, the diagram on the left of the equation has these uninterpretable elements, but the diagram on the right doesn't, so the diagram on the right is the same thing as a diagram from the diagrammatic calculus in **Set**, and refers to

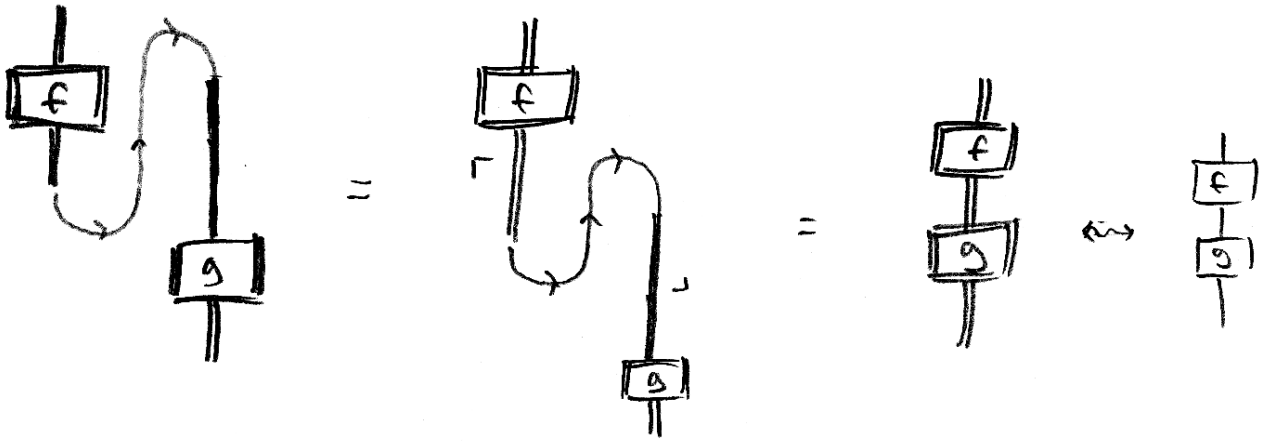


Figure 22: Now the yanking equations do hold: the formal cup and cap are no longer interpretable in **Set**, so we avoid the problem of disintegrating wires as before, as we cannot apply **Set**-equations to them. This is also the catch: the formal cup and cap have no interpretation in **Set**. They're just drawings.

something that is actually *in* **Set**.

In other words, sometimes we can add made-up processes that obey new made-up equations to a graphical calculus, and when the diagrams we draw are "well-formed" in some way, we can derive a diagram that doesn't contain any of those made-up processes. It's as if the made-up processes and equations stuck around just long enough to help guide calculation and composition, and then disappeared. This is worth keeping in mind.

2 What is Syntax?

The chief object of study of linguistics is Natural Language. Natural Language is, loosely conceived, any language that is, or was once the mother tongue of a group of human beings[Tra14, p.57]. What differentiates human language from the signalling systems, of say, plants[Kar08]? In 1960, Hockett[CC] undertook a cross-linguistic study, and identified a list of 13 design features, shared by all – and the stronger claim is, only – human languages. Among these, we draw attention to the third design feature: *transitoriness*. Spoken languages do not stick in the air as speech bubbles; they are transmitted in the medium of air, quickly fading. In conjunction with the design feature of *discreteness*, by which *phonemes* (perceptually distinct units of sound in a language) constitute words – which all languages also have – it seems that physics has forced the spoken word into linear streams of phonemic data[3]. By all accounts the spoken word predates writing systems[4], a technology that allows the persistence of linguistic data over time. Thus, though writing systems for natural languages may have nonlinear symbolic arrangements of morphemic – units of meaning – data, such as in ideographic (early cuneiform and egyptian) or logographic languages (later chinese), in written form, all natural written text is presented as linear strings of

[3]There is the muddy issue of sign languages, where a visual medium of communication allows simultaneous transmission of morphemes.

[4]Though there cannot exist evidence to the contrary.

words – which are the smallest elements of language that can be taken in isolation to carry meaning.

Grammar is the study of the formation of these linear strings of data. Grammar in the modern conception is also known as *morphosyntax*[Rad10, p.1], where *morphology* is the study of how morphemes form words, and *syntax* is the study of how words form sentences. Of course, grammarians have identified many other interstital layers of rules and structure in natural languages between, above, and below the distinction we have just outlined, such as the highly influential school of pragmatics[grice], but we cannot do them justice here.

There are, broadly speaking, two kinds of formal approaches to formal syntax in the modern day[PS01]. They are the *Generative-Enumerative*, and the *Model-Theoretic*, of which the former is currently mainstream, in no small part due to Chomsky's seminal contributions. The Generative-Enumerative approach treats grammar as a finite device that generates a set of strings or other structures, and the Model-Theoretic approach holds that a grammar is a set of axioms – known as *constraints* – in a formal logical system. The approaches are naturally complementary: the former determines 'what is allowed' while the latter determines 'what is not allowed' in grammar, and indeed modern syntactic theories make use of tools developed by both schools of thought[Dal].

Semantics is the study of the meanings of words and sentences, and tends to follow in the wake of formal theories of grammar, or really, formal logics of any kind. As a result of this geneaology, the dominant approach since Montague towards semantics is truth-theoretic. It has, however, been a recurring desire throughout the centuries to capture meaning within formal systems, and its pursuit has been fruitful: Wiener argues that Leibniz's dream of a *Calculus Ratiocinator* – a universal logical calculation framework – is the forerunner of the *Machina Ratiocinatrix*, the reasoning machine that modern computers have become[Wie61, p.12]. There are of course deeper complexities to the meaning and function of language than mere verity[Aus05], enough so that some would argue that the task is futile[WR07], but we remain broadly sympathetic to Leibniz's vision.

We will continue in a very small corner of a very small patch of the study of language. We are interested in the *Phrase Structure* and *Typeological* approaches to formal syntax – which are generative-enumerative and model-theoretic, respectively – and even then only of a small toy-model fragment of either, and we are interested in allowing the structures of both grammatical theories to direct semantic composition.

3 Phrase Structure Grammars

3.0.1 Historical Context

What follows is a condensed history that contextualises Chomsky's context-free grammars, which loosely follows a more detailed exposition given in [Tra14].

The conceptual origins of phrase structure grammars are ancient greek. Aristotle divided sentences into *subject* and *predicate* – a distinction that remains in our first-order logic today – and the greek school developed through the course of around 400 years and culiminated in the classification system devised by Dionysus Thrax. By studying the behaviour of Greek, he concluded that greek words fell into one of eight *syntactic categories*: Nouns, Verbs, Articles, Pronouns,

Prepositions, Conjunctions, Adverbs, and Participles. The Greek school of grammar was adopted by the Romans for Latin, and eventually the Greaco-Roman grammar came to dominate European schools of grammar into the 20^{th} century with little modification, and in some instances, remains taught today.

In late 19^{th} century Europe, Saussure[Sau95] inspired a major and lasting ideological change, positing that languages were not mere atomistic collections of phonemes and words, but rather structured systems of data. Saussure's structuralism remains *de rigeur*. Structuralist thought eventually inspired the branching off of *semiotics* from linguistics: the study of signs and meaning. The chief exponenent of semiotics was Roman Jakobson, who, during his exile from Russia to America in the midst of WW2, encountered the American school of semiotics founded by C.S. Peirce[Har78], which had by the time extended the theory of signs and meaning to biology in the work of Sebeok[Smi74]. Structuralist linguistics paired naturally with semiotics, in that language provided structures that could produce meaning.

In time, the research programmes of American and European structuralists diverged, along the lines of the Analytic and Continental traditions of philosophy. American structuralists lost interest in the meaning and function of words, and focused on placing the study of linguistic form on firm scientific footing. It was in this zeitgeist in mid-20^{th} century America that Zellig Harris grew interested in the analysis of linguistic form by formal, abstract, and algebraic means. Though Harris' approach was viewed by his colleagues as eccentric, he created a deep impression on one of his students, Noam Chomsky.

3.0.2 Context-Free Grammars as a Theory of Language

In Chomsky's seminal 1957 work, Syntactic Structures[CL02], the generative grammar proposed produces syntactically correct linguistic strings by means of production rules that expand upon strings, beginning from an initial symbol. The symbols themselves were aligned with syntactic categories, and the terminal symbols at which no further production rules could be applied were words, belonging to their parent syntactic category in a production tree.

Production rules work on the free monoid Σ^* over an alphabet Σ (that is, finite strings of symbols from Σ, including the empty string ϵ), where Σ contains a special start symbol S, and a subset of terminal symbols Υ, which are disallowed in the codomain of production rules. Production rules are most generally of the form $\Sigma* \to \Sigma^*$. The received presentation of such production rules is BNF: Backus-Naur Form[HU79, p.80], where each rule is presented without loss of generality as:

$$A \to b \mid C \cdot d$$

Where $A, B, C, D \in \Sigma^*$, and the above rule is to be read: "a string A can be replaced by either a symbol b, or the concatenation of the string C with the symbol d." Vertical bars indicate that a rule might license different replacements, and the we indicate concatenation of lowercase symbols and uppercase string variables in Σ^* with a dot. A collection of these rules is called a grammar, and a string is said to be in the language of a grammar if the string consists solely of terminals, and the string can be derived from the start symbol S by means of the production rules of the grammar.

Example 2. Suppose that we have two syntactic categories: `N`, for 'noun', and `V`, for 'verb', where we have two nouns: `chickens` and `roads`, and one verb `cross`[5]. Define the alphabet Σ : {`S`, `VP`, `NP`, `N`, `V`, `chickens`, `cross`, `roads`}[6], where `S` stands for 'sentence' (and 'start'), `VP` for 'verb phrase', `NP` for 'noun phrase', `N` for 'noun', and `V` for 'verb', and the three words are terminals. Let's take the following to be our production rules:

$$S \to NP \cdot VP$$
$$VP \to V \cdot NP$$
$$NP \to N$$
$$V \to \texttt{cross}$$
$$N \to \texttt{chickens}$$
$$N \to \texttt{roads}$$

The first rule says essentially that a sentence is composed of a subject and a predicate. The second tells us that predicates are formed of a (transitive) verb, and another noun phrase (recall Frege's 'unsaturated' component in "Caeasar conquered Gaul"!) Since our example is fairly simple, nouns are just as good as noun phrases (typically one allows noun phrases to be composed of a determiner along with another noun phrase.) Our final three rules capture the words of our language and the syntactic categories they belong in. Given these rules, we might have a production that looks like Figure 23.

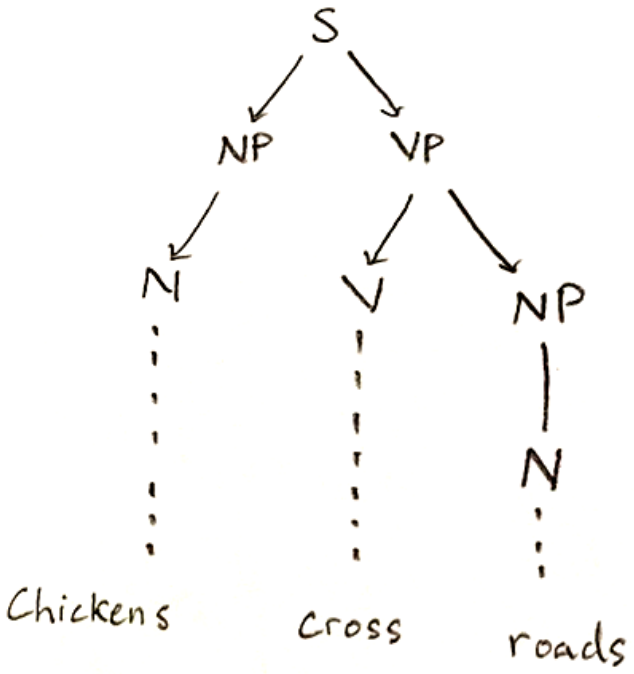


Figure 23: We mark terminal productions with dotted lines. Doesn't this look familiar?

[5]Ok, properly speaking our nouns would be NP, not N, but whatever.
[6]Yes, suppose chickens, cross, and roads are atomic symbols.

So `chickens cross roads` is grammatically correct in this toy model, as is `roads cross chickens`, `chickens cross chickens`, and `roads cross roads`. Note that only the first of these has a comfortable reading. Chomsky exemplifies the problem with the sentence `Colorless green ideas sleep furiously`: grammatically correct, but nonsensical. Thus he argues for a separation of concerns for syntax and semantics.

Figure 23 looks a lot like a diagram from a process theory, and that's because it is. Every rule in BNF with multiple productions on the right can be broken up into distinct rules with unique productions. For instance, the rule $A \to b \mid C \cdot d$ from before would break into:

$$A \to b$$
$$A \to C \cdot d$$

Each rule in this way can be interpreted as a process in a process theory, so the production tree that we have just drawn could just as well have been depicted so:

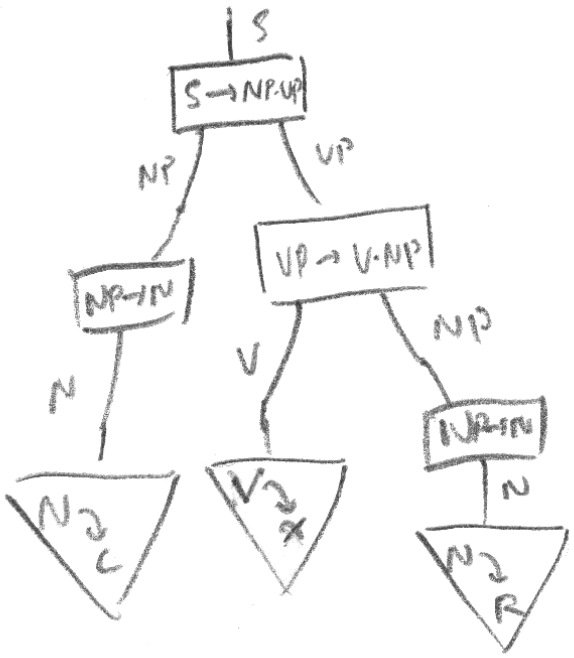


Figure 24: Nice.

Shortly after Syntactic Structures, Chomsky and Schützenberger [HM14] worked to place classes of production rules in relation to the models of computation that could recognise them.

Language Family	Production Rule Form	Model of Computation
$\mathcal{L}_0$: Recursively Enumerable	$A \to B : A \in (\Sigma - \Upsilon)^*, B \in \Sigma^*$	Turing Machines
$\mathcal{L}_1$: Context Sensitive	$C \cdot a \cdot B \to C \cdot D \cdot B : a \in (\Sigma - \Upsilon), D \in \Sigma^+, (B, C) \in \Sigma^*$	Linear Bounded Automata
$\mathcal{L}_2$: Context Free	$a \to B : a \in (\Sigma - \Upsilon), B \in \Sigma^*$	Pushdown Automata
$\mathcal{L}_3$: Regular	$a \to b \cdot C : a \in (\Sigma - \Upsilon), b \in \Upsilon, C \in (\Sigma \cup \{\epsilon\})$	Finite-state Automata

Figure 25: The Chomsky Hierarchy. Σ^+ denotes non-empty strings of Σ. Language families of a lower index strictly contain those of a higher index.

If for context sensitive languages, we rule out those languages that generate the empty string, we obtain the non-contracting grammars [RS97, §4][HU79, p.230, ex. 9.9], which are those whose production rules all (weakly) monotonically increase the length of the string operated upon. This is not a massive concession when we require context sensitive grammars to serve as models of language, because the empty string would correspond to the empty utterance, whose grammaticality is suspect. Making this concession gives us a nice graphical correspondence with the geometry of the process theories that implement production trees.

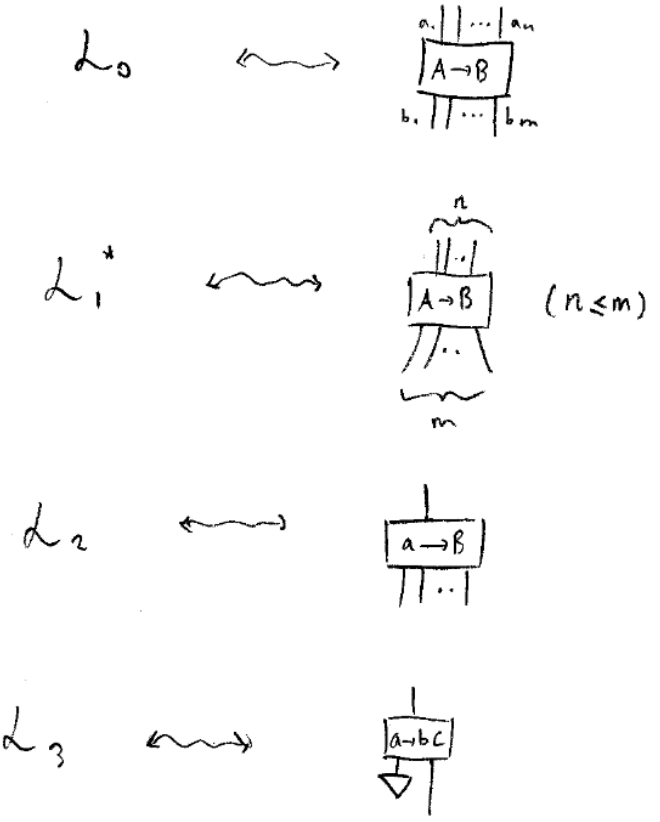


Figure 26: $\mathcal{L}_0$ rules have no restrictions on inputs and outputs. Non-contracting $\mathcal{L}_1$ rules have fewer inputs than outputs. $\mathcal{L}_2$ rules have exactly one input, and $\mathcal{L}_3$ rules have one input and one output, leaving behind terminal effects. This correspondence will be more suggestive after we consider semantics in generative grammar.

The toy example we gave was a context-free grammar. Interest in context-free grammars as theories of natural language waned after it was demonstrated that human languages exhibited mild context-sensitivity [Shi85], though formal language theory in models of computation remains a mainstay of any undergraduate computer science education. We're happy to continue with our toy context-free example.

3.0.3 Semantics in Generative Grammar by diagrams

In 1998, Heim and Kratzer applied truth conditional married truth-conditional semantics with generative grammars [HK98]. Their work is a landmark in the field. They adopt a rigid interpretation of Frege's compositionality as functional, set-theoretic composition, and their approach

can be summarised as follows: give the terminal nodes of production (which correspond to words) set-theoretic semantics, in particular taking transitive verbs to be 'schönfinkelized"[7] functions in two arguments, and inductively traverse the structure of the production tree to determine what composes with what.

We have very little to gain from a play-by-play exposition of their formalism. We exhibit instead an op-functor[8] from the monoidal category of production rules in context free grammars to $\mathcal{L}(\mathbf{Set})$, which faithfully captures the first three chapters of material.

Recall our previous example `chickens cross roads`. Suppose we do the naïve thing, and model `chickens` and `roads` as elements of some noun set, and the transitive verb `cross` as a process that takes two nouns, and returns a sentence type. For all we care, the noun type can be the set of JPEGs, and `cross` can be a neural net that takes images and does whatever with them. We're going to treat the terminals of production first:

[7]Essentially just currying.

[8]Structure preserving map that reverses directions of arrows

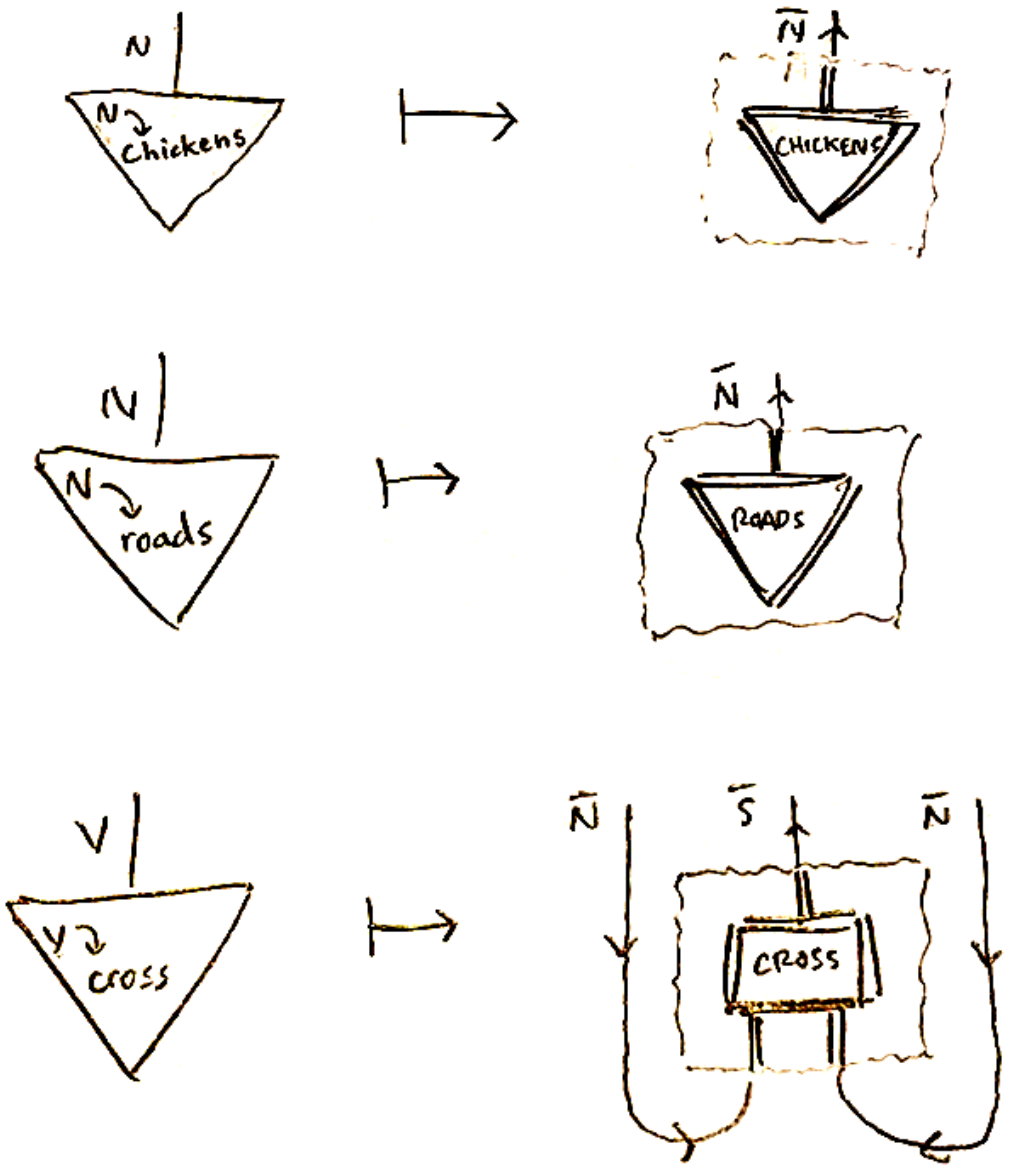


Figure 27: This is the action of the op-functor on the terminal production processes; we should be presenting the right-hand diagrams upside down, but we're keeping them this way to stress the structural nature of the map. Recall that what's drawn in bold is a 'native' of **Set**, whereas what's drawn thin is a formal element. To stress that they are different, we have boxed off the natives. The reasons for the particular choices of mapping will become evident shortly.

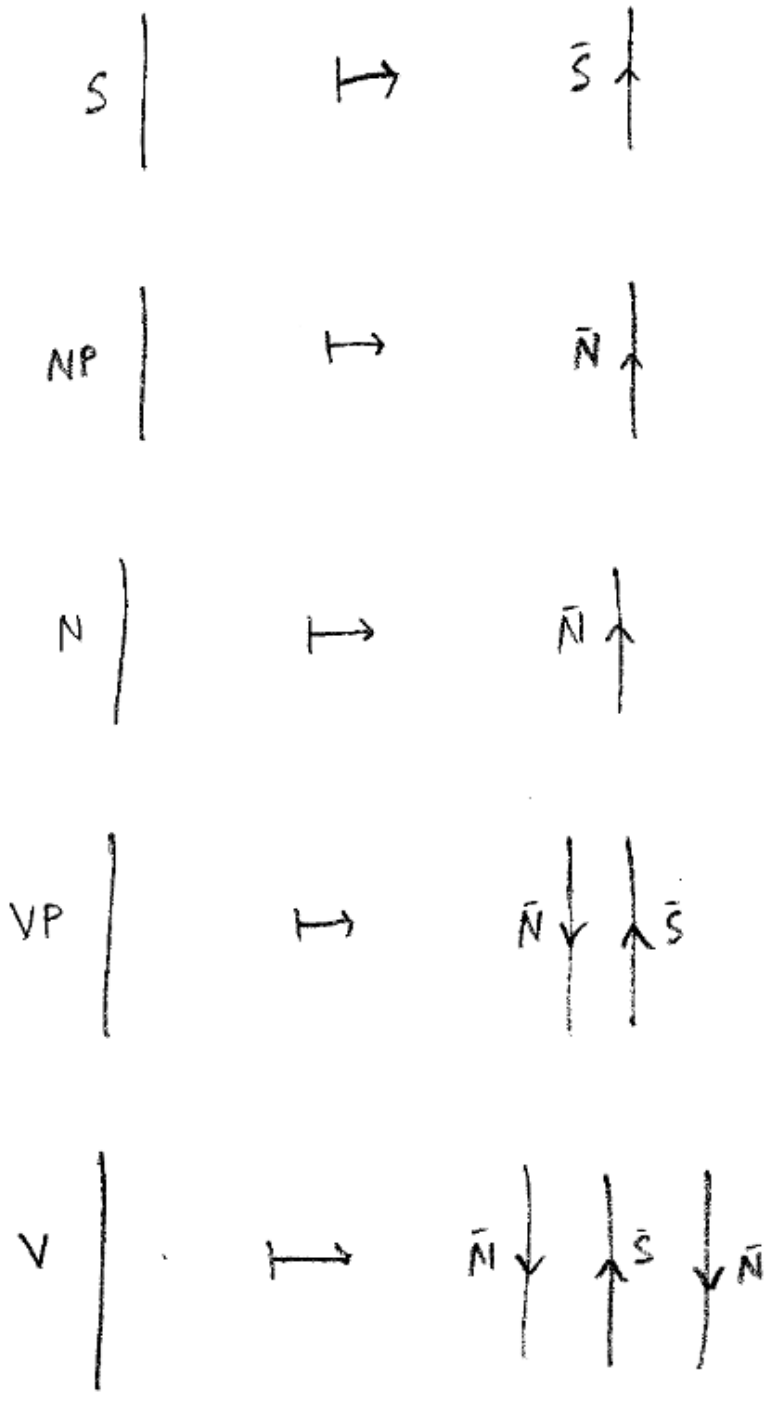


Figure 28: The objects of the source category are the labels from Σ. The objects of the target category are the two sets corresponding to the noun- and sentence- types, which we have notated with a bar and directional arrows to distinguish them from `N` and `S`.

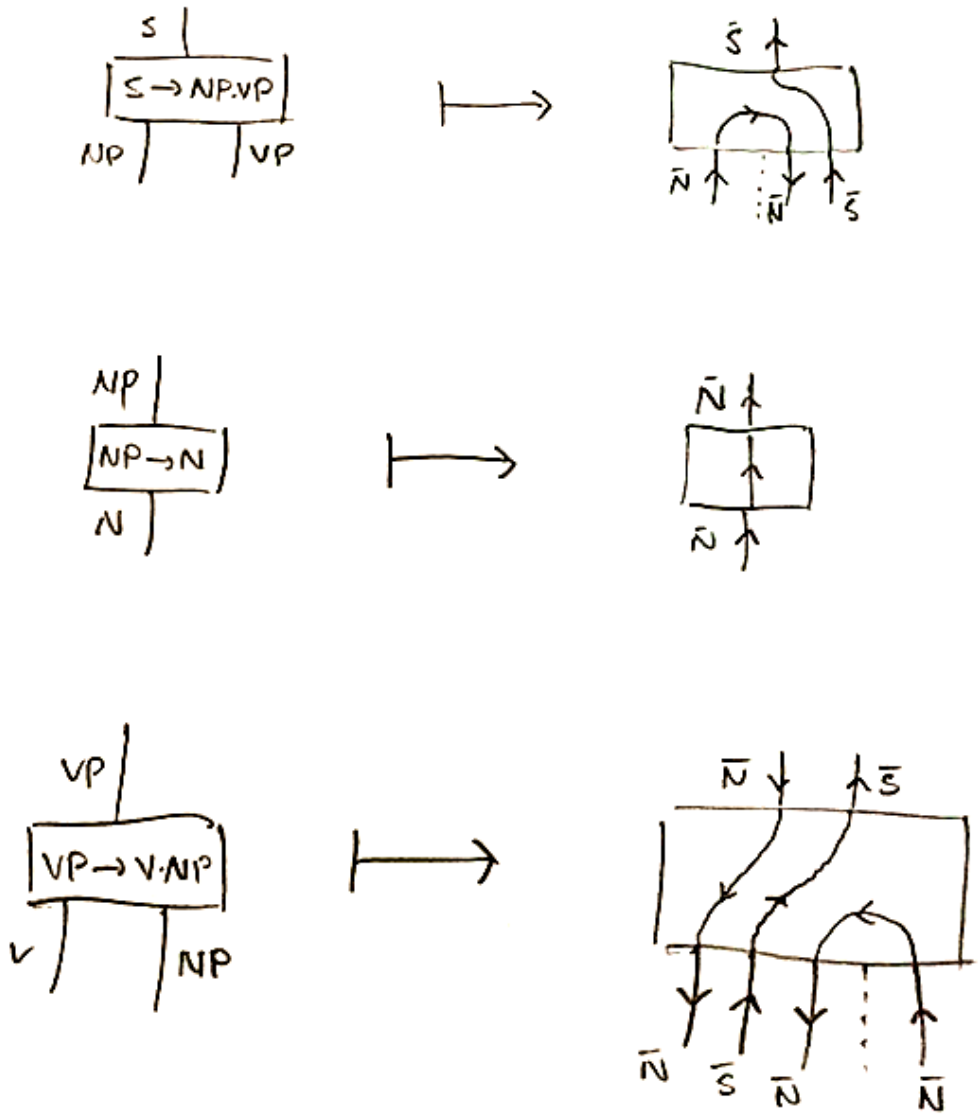


Figure 29: For the processes in the source category, which are production rules, we choose new processes in $\mathcal{L}(\mathbf{Set})$ like so. Note that in doing so, we have respected the mapping choices we have made for objects.

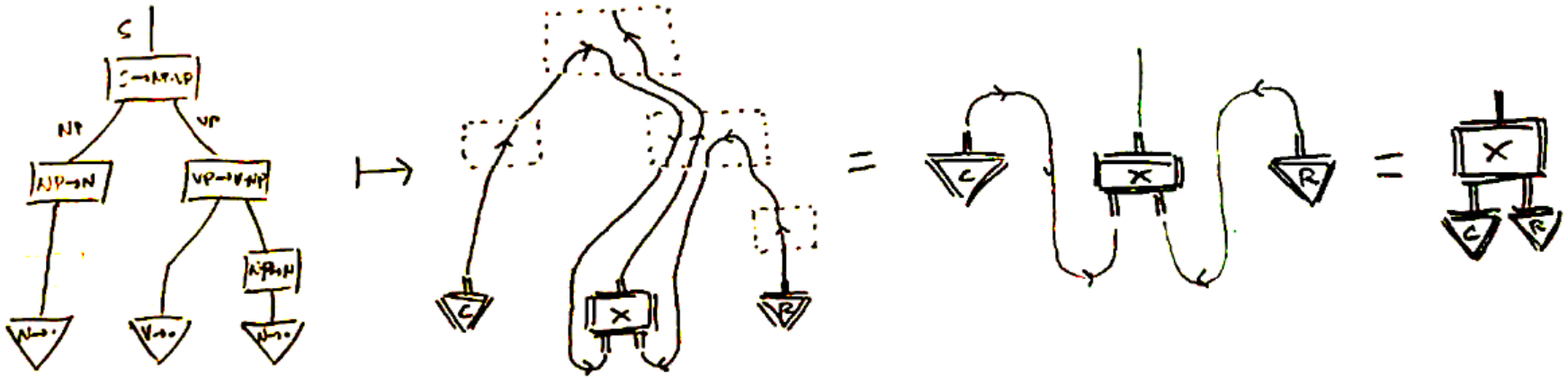


Figure 30: The op-functor gives us a diagram in $\mathcal{L}(\mathbf{Set})$, but with directions reversed, so starting from the second diagram, we start reading diagrams from bottom to top. The first equation is just applying planar isotopy. The second equation follows by the yanking equation applied to the left and right wings. The last diagram is plain functional composition in **Set**!

4 Typeological Grammars

The aim of this paper is to obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages such as English, or at least for fragments of such languages.

Joachim Lambek, The Mathematics of Sentence Structure (1958)

Knowing sufficiently powerful models of computation that will recognise languages is, to an extent, an academic concern: what matters from a computational perspective is knowing concrete deductive procedures that can determine the grammaticality of a sentence after it has been uttered, without access to the deep syntactic structure that may have generated it.

In his seminal work published shortly after Chomksy's, Lambek began the undertaking of a complementary approach to the generative-enumerative school. The residuated type system he introduced was among the first substructural logics, and enjoyed fruitful cross-pollination with linear logic three decades later. Over time, the algebraic structures Lambek considered evolved, the most modern incarnation being pregroups, which are much like groups, except with left and right inverses for all objects, which behave like group inverses but only under left or right multiplication respectively. We defer the reader to [CGS13] for a detailed treatment and historical perspective.

Typeological (sometimes Typological) grammars operate at the level of syntactic categories. The essence of typeological grammar is the assignment of an algebraic type to each syntactic category, from an algebra that at has at least the structure of a monoid: there is at least a binary tensor operation $\otimes$ on types. The $\otimes$ typically models the separation of distinct but consecutively uttered words. The assignment of types is paired with a system of gentzen-sequent style logical rules by which one can determine the well-formedness of a sentence by the outcome of a process of deduction[Moo14].

Example 3. We translate our `chickens cross roads` example into the typelogical system **NL**.

NL[9] has a residuated type algebra $(\otimes, /, \backslash)$, where all three operations are binary over a carrier alphabet encoding types. We will take our alphabet to just be $\{\mathtt{N}, \mathtt{S}\}$, where `N` informally corresponds to 'noun', and `S` 'sentence'. The types available to us will contain entries such as $\mathtt{N}, (\mathtt{N} \otimes \mathtt{S}), (\mathtt{N} \otimes (\mathtt{N}/\mathtt{S}))$, *etc.*.

In our example, we have two syntactic categories, that of nouns and verbs. We assign nouns the type `N`, and we assign verbs the type $(\mathtt{N}/(\mathtt{S}\backslash\mathtt{N}))$. We model gaps between words as the connective $\otimes$. So the sentence `chickens cross roads` is assigned the type

$$((\mathtt{N} \otimes (\mathtt{N}/(\mathtt{S}\backslash\mathtt{N}))) \otimes \mathtt{N})$$

Now we need to begin a derivation with this as the premise in a suitable logic, and we treat a sentence as grammatical if we can derive `S`, the sentence type, from the type of the sentence. The only inference rules of **NL** we need for this example are the application rules: for all types A, B,

[9] **NL** stands for the 'Mon-associative Lambek Calculus', referring to the non-associativity (and non-commutativity) of the binary $\otimes$ connective. The original lambek calculus **L** considered an associative tensor product, and was later 'gradated' into non-associative and non-commutative versions.

$(A \otimes (A/B)) \to B$, and $((A\backslash B) \otimes B) \to A$ (linear logicians will be well aware that the (A/B) notation is a precursor to Girard's linear maps $A \multimap B$.) The derivation follows in Figure 31.

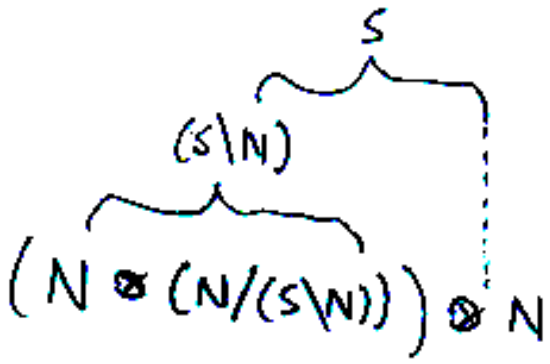


Figure 31

The algebraic models for the NL logic are known as residuated pregroups. In later iterations, Lambek simplified the type system to $(\otimes, 1, (-)^L, (-)^R)$, where $\otimes$ is considered associative (but not commutative), 1 is a nullary operation (constant) unital for $\otimes$, and $(-)^L$ and $(-)^R$ unary type forming operations known as the left and right adjoints respectively. The inference rules for this type system are, for all types A:

$$A \otimes A^R \to 1 \to A^R \otimes A$$
$$A^L \otimes A \to 1 \to A \otimes A^L$$

The algebraic models for this system are known as pregroups. Rewriting our example in this system, we would keep the same base of atomic types $\{\mathtt{N}, \mathtt{S}\}$, and we would type nouns as $\mathtt{N}$, and verbs as $\mathtt{N}^R \otimes \mathtt{S} \otimes \mathtt{N}^L$. Note that now the role of $\otimes$ is no longer to model *solely* the gaps between words occurring in sentences: word-types themselves share the same tensor product. In this system, the sentence `chickens cross roads` is typed (dropping the brackets for $\otimes$ due to associativity):

$$\mathtt{N} \otimes \mathtt{N}^L \otimes \mathtt{S} \otimes \mathtt{N}^R \otimes \mathtt{N}$$

And recalling that 1 is unital with respect to $\otimes$, the derivation would be as in figure 32

4.1 Semantics in Typelogical Grammars by diagrams

A striking connection was noted by Coecke et. al[CSC10] between Lambek's pregroup grammars and the diagrammatic calculus for compact closed categories: the left and right adjoint types corresponded to bent wires, and the reduction rules corresponded to cups and caps.

The connection to application was made via Firth's distributional semantics. In 1960, the linguist Firth put forward the dictum "You shall know a word by the company it keeps"[Fir57]:

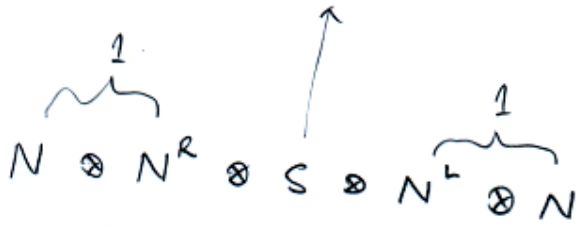


Figure 32: Only S remains uneliminated by the end of the derivation.

the proposal was to treat the kinds of contexts a word appeared in as the semantics of that word. It was only with the advent of Moore's law and the availability of large digital corpora of text that this proposal became practical in the field of computational linguistics. The *de rigeur* approach today is conceptually simple. First, concoct a bag of several hundred context words, and assign each of these context words a dimension in a vector space. Represent every other word in a corpus as a vector in this vector space, with entries corresponding to a histogram of how often the word appears within the same context (say, within the same sentence) as one of the context words.

Example 4. For example, if the context words are {`green`,`bark`,`furry`}, and we have a large enough corpus of text, we would expect the word `tree` to have high values in the `green` and `bark` dimensions, and the word `dog` to have high values in the `bark` and `furry` dimensions.

Distributional semantics came to favour when it was discovered that the linear algebra of these vector space semantics seemed to reflect deep semantic relations between words: if one takes the vector for `King`, subtracts the vector for `Man`, and adds the vector for `Woman`, the closest vector to the result is the vector for `Queen`. Exciting stuff[10].

After investigations in the programme of categorical quantum mechanics, an ingenious marriage was devised: since the category of finite dimensional vector spaces is monoidal and compact closed, and Lambek's pregroup grammars are also compact closed, why not take a distributional semantics approach to model the meanings of individual words, and then use linear cup and cup maps arranged in a manner dictated by the pregroup grammars to compose these meanings and obtain vector space representations of the meanings of sentences?

Example 5. Following from Figure 32, if we have vector representations of `chickens`, `cross`, and `roads`, we can draw the following diagram in **FinVect**, the category of finite dimensional vector spaces. The backwards bending wires become dual spaces, and finite dimensional vector spaces are isomorphic to their double-duals.

Fantastic. Why aren't we all doing this? In short, our computers aren't big enough. The category of finite dimensional vector spaces has two monoidal structures, the direct sum and the classic tensor product. Taking the direct sum is additive in dimension: the direct sum $v \oplus w$ of two vectors of dimensions n and m respectively yields a vector of dimension $n+m$. This is totally manageable, but the trouble is that with the direct sum as the tensor product, **FinVect** is not

[10]Everyone uses this example to talk about how great distributional semantics is, but analogical reasoning with word vectors is fraught with difficulty in practice[RDL17]. The closest word to $\overrightarrow{\texttt{King}} - \overrightarrow{\texttt{Man}} + \overrightarrow{\texttt{Woman}}$ is not $\overrightarrow{\texttt{Queen}}$, but rather $\overrightarrow{\texttt{King}}$. This isn't to mention other problems with distributional semantics, such as the fact that it can't tell apart antonyms.

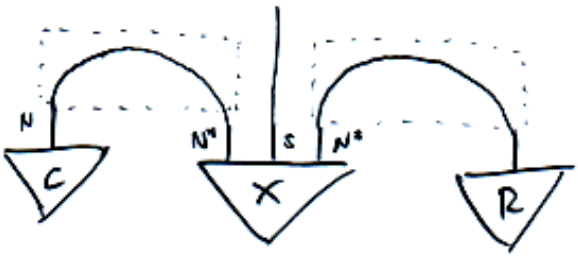


Figure 33: Read from bottom to top. Looks very similar to the derivation in Figure 32. The dotted boxes are enclosing the linear maps that act as caps in **FinVect**, also known as *metrics*[BC17, p.146].

compact closed, and we can't get at those juicy caps. **FinVect** with the classic tensor product *is* compact closed, but the dimension of $v \otimes w$ is $n \times m$. This multiplicative blowup in dimension is not great: a distributional categorical semantics approach to our `chickens cross roads` example – assuming the dimensions of `N` and `S` are about 1000, and we are being charitable, as deep neural models can take data of dimensions orders of magnitudes larger – would require about a petabyte worth of storage to encode the initial input $\mathtt{N} \otimes \mathtt{N}^L \otimes \mathtt{S} \otimes \mathtt{N}^R \otimes \mathtt{N}$. For a sense of scale, this is enough data to take a picture documenting every minute of your expected natural lifespan (around 80 years) with a 12-megapixel camera on a decent smartphone, with enough to show each of the about 10,000 people you will meet in that lifetime a Netflix-4k quality (3Mbps) rendition of Stanley Kubrick's '2001: A Space Odyssey' (runtime 2h44m), and to take a copy of the library of congress (200TB) to your grave when you're done. All to evaluate `chickens cross roads`.

What are the workarounds? The first solution is to wait for, and then use, a quantum computer. We have an alternative suggestion.

4.2 An alternative suggestion

Just use $\mathcal{L}(\mathbf{Set})$ instead of **FinVect**.

And as a bonus, the diagrams we get are categorially dual to the generative-enumerative approach, up to planar isotopy.

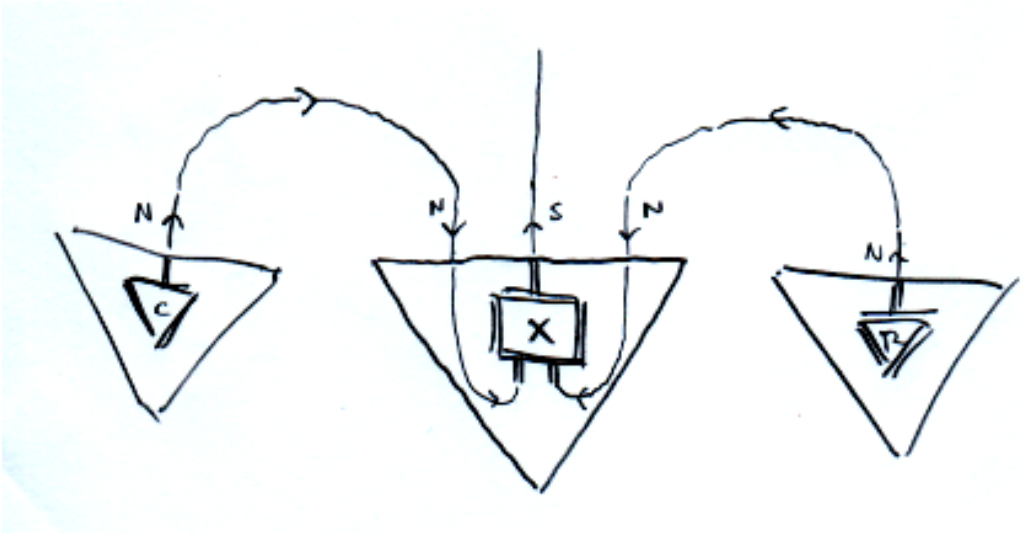


Figure 34: Read from bottom to top.

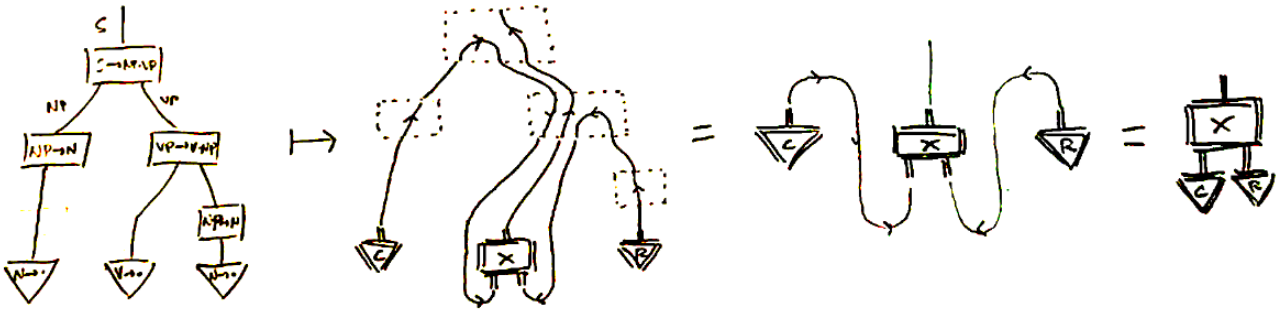


Figure 35: Here's Figure 30 again. The diagram in Figure is the second diagram from the right.

5 Apologia

You've only shown one example.

True, but the connection we have drawn is surely compelling. If we take generative and typelogical grammars as seriously as cognitive models, it is natural to conclude that generative grammars run typelogical proofs in reverse, such that the recipient of the sentence can have an easy time putting together the intended meaning. We are aware of the technical result that context-free grammars are weakly equivalent to pregroup grammars[Bus01], but that is a result that holds no force here: the interest should lie in finding *the* complementary pair of generative-enumerative and typelogical grammars that serve as a good empirical model.

What should we use, if not vectors?

It's **Set**! Just about anything you can structurally specify lives there. Since the autonomisation trick works for any monoidal category, you can work in **Cat** and have your states be databases[SSVW16]. Or, if you don't feel like anything fancy, in $\mathcal{L}(\mathbf{FinVect}^{\oplus})$ you can keep your vectors and get your compact closure, too.

CFGs and Typelogical grammars are old-school, what about X?

Some unsubstantiated speculation: I suspect that transformational grammars can be expressed as homotopies of diagrams. I regret not having the time to bring in dependency grammars and lexical functional grammar in this work. The work of Tésniere[Tes] and Mel'cuk[MMMM88] I believe will be particularly appealing to the category-theoretic eye for unifying structures, and I believe that lexical functional grammar can be upgraded to lexical *functorial* grammar. I don't know enough about other systems to comment, but I am fairly sure they will succumb to diagrams.

Gotcha! cups and caps aren't enough to do X, we need Y!

This objection is most likely to come from the Compositional Distributional Semantics community (DisCoCat, for short.) It must be admitted that they have worked miracles in the in the arid desert of vector space semantics, and great progress has been made in semantic and syntactic modelling. To name a few contributions, work in the DisCoCat community has covered the phenomena of negation [Vin19], montague semantics [dFMT], ambiguity [Rob14], entailment [KS], cognitive conceptual modelling [Yaa15] , logical connectives [Mar10], inquisitive semantics [dFMT19], pronouns [CCS13], updating information across sentences [BCG$^+$17], and the list goes on. All this variety has been squeezed from essentially two features of **FinVect**: Density Matrices and Frobenius Algebras. The former is typically used in semantic modelling, and the latter in syntactic modelling, and neither is fungible in **Set**.

Arguably, the chief purpose of Density Matrices in semantic modelling is to capture ambiguities or superpositions of information, which can be done in more naïve manners in more suitably structured settings, such as over probability distributions and multisets of alternatives. Frobenius algebras are harder to reconcile, and we will revisit them in the conclusion, to see what can be done about them.

Yet, we have still not answered this charge in full generality. So coming up next, we'll show how to add just about any diagrammatic gadgets you can dream of into your favourite monoidal categories. Skip ahead to the conclusion to see some more pictures.

Part II
Graphical Completions of Monoidal Categories

6 Introduction

Given a graphical calculus for a monoidal category, and a collection of extra graphical components and equations, we aim to demonstrate a monoidal category with a graphical calculus that supports all the old and new graphical data and equations. So we will have a graphical completion of arbitrary monoidal categories. If you are reading on from the first part, process theories are functors from the free monoidal category over a monoidal signature, and the free monoidal category over a monoidal signature is the category of diagrams in the diagrammatic calculus over that signature.

Our approach is a generalisation of Delpeuch's approach to the free autonomization of monoidal categories. In a sentence, Delpeuch adds cups and caps to the diagrams of a monoidal category by *just drawing them*, and then specifying that the diagrams must obey the yanking equations.

More at length, idea underlying Delpeuch's construction was to first embed the original monoidal category $\mathcal{K}$ into a syntactic category of graphical diagrams representing morphisms in $\mathcal{K}$ (taking equivalence classes under planar isotopies, hence making good use of the coherence theorem.) Then, he introduces *purely formal* cups and caps in the syntactic category, and he *decrees* that these formal cups and caps obey the yanking equations; which is to say, he imposes additional equalities. Then this construction turns out to be the free autonomous completion.

There are points we seek to improve upon in Delpeuch's construction. There is a degree of bureaucracy that must be dealt with when reasoning about upward planar graphs composed of linear segments, which Delpeuch deals with ultimately by taking graphically induced equivalence classes to obtain what is fundamentally a category composed of algebraic – rather than graphical – syntactic representations of diagrams. In contrast, we will start with the algebraic representations, introducing a sequent calculus to derive morphisms and equations between them, and the eventual term category we construct we will prove equivalent to a category of diagrams. The term calculus we introduce is independently conceived, but we note that there are similarities between our system and the approaches of [Mau], and of Hoare Logics.

Approaching in this algebraic manner will also give us the freedom and rigour to push to the logical conclusion: given any structure representable by system of diagrammatic equations in the plain monoidal graphical calculus, we will be able construct the completion of any monoidal category with respect to that structure.

Simply conceived, what we propose in this section is painfully trivial. If you have a plain graphical calculus with graphical equations, nothing is stopping you from *just drawing in* new morphisms and labels for wires and calculating with respect to additional graphical equations that govern the novel graphical elements. The contribution is that if someone put a gun to your head, it is now possible to justify your activity formally.

The institutional and cultural challenge that applied category theory presents is more substantial. While forms of pictorial reasoning are workhorses of the mathematical arsenal, serving working formalists as private calculation tools and pedagogical aids, there remains prejudice against pictorial reasoning in formal presentations. The charges are manifold, and among them: pictures lack rigour; we may be tricked by graphical intuitions; they do not look as SERIOUS as pristinely typeset reams of greek.

These charges we answer in turn. Arbitrary pictures may lack rigour, but we will rigourously ground the doodling systems of the previous section in this work, and of many other doodles and ways to redoodle.

While indeed our spatial intuitions may lead us astray, the innate human faculties of spatial comphrension and manipulation have also enjoyed the *benefits* of millions of years of evolution. Progress that is labouriously wrested from understanding and manipulating reams of greek – a skill that takes years to learn – may be expedited and made widely accessible, as our comprehension of graphical systems is erotetically charged in light of those innate faculties.

The final charge, that reams of greek look more SERIOUS, is the least voiced aloud, but it the likely progenitor of all prejudice against pictorial reasoning, deeply rooted in the modern psychology and culture of mathematicians and mathematics. Understanding and manipulating reams of greek is *hard*, and to do so with sufficient art and passion to make a living of it is *harder*: hence the mere suggestion that *it needn't be so* is an egregious affront on personal identity and job security. Further, we have all noticed the tendency of formal presentation to pretend divine inspiration: propositions and theorems arranged to follow one another drily, without traces of the path of discovery, failures, struggles, and humanity. In this regard, seriousness and sacredness with sanctified reams of greek is a shibboleth, and a point of pride in mastery of the arcane. In constrast, pictures are profane and proletariat, presenting challenge to the cultural order. We will skirt psychology for philosophy in answering this objection. To platonists and formalists alike, the activity of mathematics is pushing names around 'correctly'; why not do it in two dimensions?

6.1 The plan

7. We review the basic notions of category theory leading to monoidal categories. We will link monoidal categories with the graphical calculus, and show that certain structural properties expressed as natural transformations on monoidal categories can be presented as systems of diagrammatic equations. In this section we place a particular emphasis on monoidal signatures, and on presentations of categories as free objects under equational constraints. We leave this section armed with a generalised notion of monoidal signature $\mathfrak{S}$ which carries 'all the data we care about' in a given monoidal category $\mathcal{K}$ with structure.

8. We introduce the sequent calculus $\mathtt{Mon}(\mathfrak{S})$, built from the data of $\mathfrak{S}$, which constructs diagrams and equations between them using the vocabulary of $\mathfrak{S}$.

9. We construct a category $\mathcal{M}(\mathfrak{S})$ using the derivable terms, formulas, and equations from $\mathtt{Mon}(\mathfrak{S})$. We show that $\mathcal{M}(\mathfrak{S})$ is strict monoidal and monoidally equivalent to $\mathcal{K}$.

10. We show that when $\mathfrak{S}$ only contains the data provided by a monoidal signature, $\mathcal{M}(\mathfrak{S})$ is equivalent to the category of diagrams in the graphical calculus over a monoidal signature. Thus the objects of $\mathcal{M}(\mathfrak{S})$ are precisely equivalence classes of diagrams in graphical calculus associated to $\mathcal{K}$.

11. We elaborate upon structure and graphical equations, and we show how we may use the technology we have developed to combine any monoidal category $\mathcal{K}$ with any structure expressible by a system of diagrammatic equations $\mathbf{S}$ to obtain a new monoidal category with a graphical calculus that includes elements and equations of $\mathcal{K}$ and $\mathbf{S}$, and nothing more.

12. Conclusion. Discussion. Examples of what we can do with this trick.

7 Some Category Theory

This presentation of basic category theory is directed towards reaching monoidal categories, so we assume familiarity with other basic concepts, such as products, limits, and duality. Our presentation of basic concepts up to functors follows the progression and conventions of [FS19, §3], as the presentation of categories as free objects that satisfy equivalence relations is closely related to the initiality of the free monoidal category over a monoidal signature in the category **Mon**.

7.1 Categories, Free and Finitely Presented

Definition 7.1 (Categories)**.** A category $\mathcal{C}$ consists of the following data:

- A collection[11] $\mathrm{Ob}(\mathcal{C})$, the elements of which we call **objects** of $\mathcal{C}$.
- For every pair of objects c, d, a set $\mathcal{C}[c, d]$, elements of which are called **morphisms**, or **arrows**, from c to d. We might denote such a morphism $f : c \to d$, or $c \xrightarrow{f} d$.
- For every object c, a specification of a morphism $\mathrm{id}_c \in \mathcal{C}[c, c]$, called the **identity morphism** on c.
- For every triple of objects c, d, e, and every pair of morphisms $f \in \mathcal{C}[c, d]$ and $g \in \mathcal{C}[d, e]$, a specification of a morphism in $\mathcal{C}[c, e]$ called the **composite** of f and g, denoted either $g \circ f$, or $f; g$. The former is read "g after f", and the latter is read "f, then g"[12].

Subject to the following conditions:

- *(Unitality)*: For any $c \xrightarrow{f} d$, we have $\mathrm{id}_c; f = f = f; \mathrm{id}_d$; that is, composing with identities does nothing.
- *(Associativity)*: For any three morphisms $a \xrightarrow{f} b$, $b \xrightarrow{g} c$, $c \xrightarrow{h} d$, we have $(f; g); h = f; (g; h)$.

Categories are fairly versatile. One can express complicated constructs quite cheaply, because of the abstraction afforded.

Example 6 (**Set** is a Category)**. Set** is the category whose objects are sets, and whose morphisms are functions between sets.

Example 7 (Preorders are Categories)**.** A preorder $(P, \leq)$ consists of a set P, and a binary relation $\leq$ on P that is reflexive, antisymmetric, and transitive (without necessarily being total, which is what promotes the preorder to a linear order.) Any preorder is a category in the following way: the objects $\mathrm{Ob}(\mathcal{C})$ are the elements of P, and for any $x, y \in P$, $\mathcal{C}[x, y]$ contains a single element if $x \leq y$, and is empty otherwise.

[11] We won't fuss about size issues and the foundations of mathematics here.

[12] The former notation is often mentally taxing for the uninitiated, so we will skew towards the latter.

Example 8 (Path Categories)**.** For any directed multigraph $G = (V, E, s, t)$ – where V is a set of vertices, E is a set of edges, and s, t are functions with domain E and codomain V that specify the source and target of each edge – we can obtain the **Path Category**, which is also denoted $\textsc{Free}(G)$, by taking the objects to be those of V, and the arrows to be all paths between any two objects (including the trivial path, which serves as the identity on objects.)

Preorders and Path Categories fall on two ends of a spectrum: whereas Preorders do not distinguish between parallel morphisms with the same source and target, Path Categories are "freely generated" given a digraph. We may access the intermediate points of the spectrum by means of an interpreted equality predicate, starting from free categories. This is a familiar idea in abstract algebra, where a particular mathematical structure such as a group, is presented as a "free object" that is then subjected to an interpreted equality relation.

Example 9 (The Klein 4-group, finitely presented)**.**

$$V = \langle a, b \mid a^2 = b^2 = (ab)^2 = e \rangle$$

To be read: the elements of the Klein group are the equivalence classes of the set of all strings freely generated by $\{a, b\}$ under the equality relation specified in the right hand side (where e is the identity element,)

We want to establish a similar formalism of presentation for categories, where free objects are interpreted under an equality relation.

7.1.1 Finite Graphical Presentations of Categories

In this section, we wish to show that finite digraphs with path equations always yield a category, by freely taking paths on the digraph, and then grouping paths by the equality relation specified by the path equations. While the notion of path on a digraph is fairly intuitive, it will not hurt to define.

Definition 7.2 (Digraphs)**.** $G = (V, E, s, t)$ where V is a set of vertices, E is a set of edges, and s, t are functions with domain E and codomain V that specify the source and target of each edge

Definition 7.3 (Path Equations)**.** Given a Digraph $G = (V, E, s, t)$, a **Path** in G is a (finite) sequence of edges $P = e_1 \dots e_n$ such that for all $1 \leq i < n$, $t(e_i) = s(e_{i+1})$; each morphism is a step, and paths are composed sensibly of steps. Two paths $P = e_1 \dots e_n$ and $Q = f_1 \dots f_m$ are **parallel** when $s(e_1) = s(f_1)$ and $t(e_n) = t(f_m)$; paths are parallel if they start in the same place, and end in the same (possibly different) place. A **Path Equation** in G is an equation of the form $P = Q$ for parallel paths P, Q in G.

Definition 7.4 (Finitely Presented Categories)**.** A finitely presented category $\langle G \mid \mathbf{E} \rangle$ consists of a (finite) digraph G, and a set (not necessarily finite) of path equations in G denoted $\mathbf{E}$. Let $[P]_{\mathbf{E}}$ denote the equivalence class under $\mathbf{E}$ of a given path P in G.

The objects of $\langle G \mid \mathbf{E} \rangle$ are the vertices V of G. The morphisms of $\langle G \mid \mathbf{E} \rangle$ are $\{[P]_{\mathbf{E}} \mid P \text{ is a path in } G\}$ the equivalence classes of freely generated paths; for any pair of objects c, d, $\langle G \mid \mathbf{E} \rangle[c, d] = [\textsc{Free}(G)[c, d]]_{\mathbf{E}}$ (by abuse of notation, we take this to be the set of equivalence classes represented in the set $\textsc{Free}(G)[c, d]$ under $\mathbf{E}$.) For each object c, the identity morphism is $[\mathrm{id}_c]_{\mathbb{E}}$, where $\mathrm{id}_c \in \textsc{Free}[c, c]$ is the trivial path beginning and ending at c. Composition is defined by declaring $[f]_{\mathbf{E}}; [g]_{\mathbf{E}} := [f; g]_{\mathbf{E}}$; note that the left composition lives in $\langle G \mid \mathbf{E} \rangle$, and that the right nested composition lives in $\textsc{Free}(G)$.

The associativity and identity composition conditions are inherited from $\textsc{Free}(G)$, and preserved by equality relations, so we indeed have a category.

Takeaway 1: Free and thin categories are two ends of a spectrum, and we can reach parts in between using equations between arrows.

7.1.2 Type-Theoretic Presentations of Categories

Type Theory gives a sequent-style proof language to construct morphisms in a category. We wish to establish a correspondence early on between the categorical and type-theoretic presentations, as we will be making use of *term models* [AT10, p.70] later on, also known as *syntactic categories*. The inter-translation scheme between type theories and categories is fairly straightforward:

- The types of the type theory are the objects of the category[13]
- The terms of the type theory are the morphisms of the category
- The axioms of the type theory correspond to 'generators' of named morphisms in the category: for instance, the initial axiom $\dfrac{}{c \vdash c}$ corresponds to the invocation of the identity morphism on the object c
- The cut rule in type theory performs variable substitution in the contexts, which corresponds to plain composition of arrows in the category

Remark 1. It is difficult to find definitive presentations of Type Theory, but good sources for the interested reader are [BD08], [Cro94], and [Pau][14]. The fundamental idea is that type theories are the proof systems for the internal logics of categories. We will skip the usual presentation of type theories in terms of variables, judgements, terms, and contexts, because we nothing to gain from such a presentation, as we will not be developing towards dependent type theories. So, while we will keep the proof trees, we will use the syntax of arrows between objects we have developed so far, rather than that of typed variables in contexts and sequents. After all, $x : A \vdash f(x) : B$ and $A \xrightarrow{f} B$ differ only notationally.

7.2 Isomorphisms and Universals

In the familiar world of sets and functions, we have notions of 'injectivity', 'surjectivity', and 'bijectivity'. In the language of categories, we have the following analogues:

Definition 7.5 (Monics, Epics, Isomorphisms)**.** In a category $\mathcal{C}$, we say that a morphism f is:

- **monic** if for all other (well-typed) morphisms g, h, $g; f = h; f \implies g = h$
- **epic** if for all other (well-typed) morphisms g, h, $f; g = f; h \implies g = h$
- an **isomorphism** if there exists a morphism f^{-1} such that $f; f^{-1} = \mathrm{id}_{(\mathrm{cod}\ f)}$

[13] Later, when we introduce monoidal categories, we will see that the objects of the category correspond to contexts, rather than types.

[14] and whereever nLab leads.

Monic and epic morphisms are 'cancellable' from the left or right of equations between composite morphisms, and it is not difficult to see that in the category **Set**, the monics are precisely the injections, and the epics are precisely the surjections. It should come as no surprise that isomorphisms are both monic and epic, and moreover cancel to the identity morphism in either order of composition.

When there exists an isomorphism $f : c \to d$ between objects c and d in a category, we say that c and d are isomorphic, and it is easy to see that this isomorphism relation is an equivalence relation on objects in a category. Isomorphism between objects is the arrow-theoretic way of saying that two objects in a category are 'just as good as one another', which is a weaker notion than equality, but a more versatile one: when category theorists say that an object is 'unique' in a category, they very often say that as shorthand for 'unique up to unique isomorphism'.

When would we want to specify that certain objects or arrows are unique? All the time. *Universality* is a fundamental categorical notion, which arises in the search for 'canonical constructions'; the mere existence of a solution is not satisfactory, we want unique existence (up to unique isomorphism), *i.e.* canonicity.

Definition 7.6 (Initial and Terminal Objects)**.** An object $\bot$ is **initial** in a category $\mathcal{C}$ just when for any other object c, there exists a unique morphism $\iota_c : \bot \to c$. Dually, a **terminal** object $\top$ is such that for any other object c, there is a unique arrow $\tau_c : c \to \top$.

Initial and terminal objects are unique up to unique isomorphism, and they are fairly useful things. When we encounter categories of interpretations, we will see that the initial objects are the free syntactic objects. For deeper discussion of universality and other basic constructs, we refer the reader to [AT10, §1.5].

7.3 Functors and Natural Transformations

As morphisms are to objects within a category, functors are to categories: functors are arrows that go between categories, which 'preserve structure'.

7.3.1 Functors

Definition 7.7 (Functor)**.** A **functor** F between categories $\mathcal{C} \to \mathcal{D}$ comprises two maps:

- An object map, sending each $c \in \mathrm{Ob}(\mathcal{C})$ to $F(c) \in \mathrm{Ob}(\mathcal{D})$
- A morphism map, sending each $f : x \to y$ in $\mathrm{Mor}(\mathcal{C})$ to a morphism $F(f) : F(x) \to F(y)$ in $\mathrm{Mor}(\mathcal{D})$

Such that:

- F preserves identities: for all $c \in \mathrm{Ob}(\mathcal{C})$, $F(\mathrm{id}_c) = \mathrm{id}_{F(c)}$
- F preserves composition: for all (well-typed) morphisms f, g in $\mathcal{C}$, $F(f; g) = F(f); F(g)$

Example 10. The identity functor $\mathrm{Id}_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$, which maps every object and morphism to itself is a functor. Furthermore, since the composition of functors $F : \mathcal{C} \to D$, $G : \mathcal{D} \to \mathcal{E}$ is again a functor $F; G : \mathcal{C} \to \mathcal{E}$, we can define a category from a collection of categories and functors between them!

Example 11 (Copresheaves as semantic frames)**.** Spivak *et al.*[SSVW16] have described how categories can be viewed as database ontologies, and covariant functors into **Set** as database instances. We can naturally apply this observation to the modelling of Semantic Frames[Bar92].

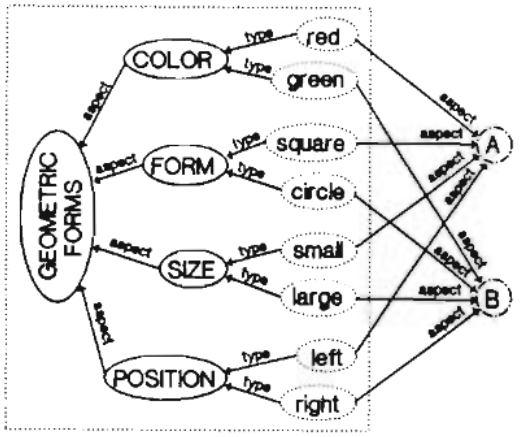


Figure 36: A semantic frame, from [Bar92, p.24]. On the right, two shapes A and B with attribute lists captured by an ontology on the right: 'Geometric Forms'.

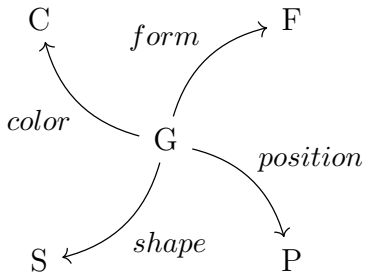


Figure 37: A category $\mathcal{G}$ that captures the ontology of 'Geometric Forms'. The full data of the figure above is captured by a covariant functor $F : \mathcal{G} \to$ **Set** which maps:

$$G \mapsto \{\text{A,B}\}$$

$$C \mapsto \{\text{red,green}\} \qquad \mathit{color} \mapsto \begin{cases} A \mapsto \text{red} \\ B \mapsto \text{green} \end{cases} \qquad F \mapsto \{\text{square,circle}\} \qquad \mathit{form} \mapsto \begin{cases} A \mapsto \text{square} \\ B \mapsto \text{circle} \end{cases}$$

$$S \mapsto \{\text{small,large}\} \qquad \mathit{size} \mapsto \begin{cases} A \mapsto \text{small} \\ B \mapsto \text{large} \end{cases} \qquad P \mapsto \{\text{left,right}\} \qquad \mathit{position} \mapsto \begin{cases} A \mapsto \text{left} \\ B \mapsto \text{right} \end{cases}$$

There is one more observation we wish to draw attention to about functors. It begins with the fact that a function between sets $f : X \to Y$ induces an equivalence relation on its codomain by grouping together objects with common image in Y: explicitly, the equivalence classes are $\{\{x' \mid f(x') = f(x)\} \mid x \in X\}$. Functors between categories consist of a pair of functions, one for objects, and one for morphisms, hence inducing equivalence classes of objects and morphisms in the codomain category. However, because functors must preserve identities and composition,

and due to the property in categories that every object has a unique identity morphism, we can faithfully capture the functorially induced equivalence classes on both objects and morphisms by just considering the equivalence classes induced on morphisms.

Lemma 1. *For any functor $F : \mathcal{C} \to \mathcal{D}$, and objects $a, b \in C$,*

$$F(a) = F(b) \iff F(id_a) = F(id_b)$$

Proof. For the forward direction, if $F(a) = F(b)$, by the uniqueness of identities on objects in categories, $\mathrm{id}_{F(a)} = \mathrm{id}_{F(b)}$ in $\mathcal{D}$. By functors preserving identities, we have $F(\mathrm{id}_a) = F(\mathrm{id}_b)$ as required.

For the converse, if $F(\mathrm{id}_a) = F(\mathrm{id}_b)$, by functors preserving identities, we have $\mathrm{id}_{F(a)} = \mathrm{id}_{F(b)}$, and by the uniqueness of identities, we have $F(a) = F(b)$. □

Generally, $a \xrightarrow{f} b$ and $c \xrightarrow{f} d$ with distinct objects as codomains and domains in $\mathcal{C}$ may be mapped by a functor $F : \mathcal{C} \to \mathcal{D}$ to the same image morphism in $\mathcal{D}$, *i.e.* $F(f) = F(g)$; in which case, we must have that $F(a) = F(c)$ and $F(b) = F(d)$, and by the above lemma, the following well-behavedness property.

Proposition 2. *Given $a \xrightarrow{f} b$, $c \xrightarrow{f} d$ in $\mathcal{C}$, and a functor $F : \mathcal{C} \to \mathcal{D}$,*

$$F(f) = F(g) \implies F(id_a) = F(id_c) \text{ and } F(id_b) = F(id_d)$$

Proof. By unitality of identity in $\mathcal{C}$, $\mathrm{id}_a; f = f = f; \mathrm{id}_b$, and since functors preserve composition, we must have $F(a) \xrightarrow{F(f)} F(b)$ in $\mathcal{D}$. Symmetrically, $F(c) \xrightarrow{F(g)} F(d)$. Since $F(f) = F(g)$, we have matching codomain and domain, so $F(a) = F(c)$ and $F(b) = F(d)$, and by the previous lemma, this holds iff $F(\mathrm{id}_a) = F(\mathrm{id}_c)$ and $F(\mathrm{id}_b) = F(\mathrm{id}_d)$. □

Takeaway 2: The data of functors can be encoded as equivalence classes of morphisms in the the codomain category.

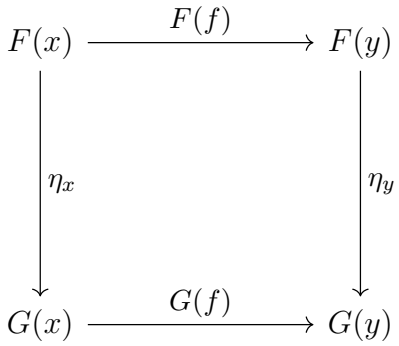
Later, we will show that free monoidal categories are initial objects in a category of monoidal categories and functors between them, so any particular monoidal category may be expressed as the data of a functor from the free monoidal category. With the above observation, *we can capture the essential data of arbitrary monoidal categories in terms of equivalence classes of morphisms in the free monoidal category.*

7.3.2 Natural Transformations

We have seen how functors are structure-preserving mappings between categories; a functor 'models' the source category within the structure of the target category. In this reading, the source category is a kind of 'blueprint', and the target category is a concrete realisation of the blueprint. The role of natural transformations is to allow comparison of distinct realisations/models.

Definition 7.8 (Natural Transformation)**.** Given categories $\mathcal{C}$ and $\mathcal{D}$, and functors F, G both from $\mathcal{C}$ to $\mathcal{D}$, a **natural transformation** $\eta : F \Rightarrow G$ assigns to every object $c \in \mathcal{C}$ a morphism $\eta_c : F(c) \to G(c)$ in $\mathcal{D}$ – called *the component of eta at c*, such that, for any morphism $f : x \to y$ in $\mathcal{C}$, the following diagram in $\mathcal{D}$ commutes:

Moreover, when every component of η is an isomorphism in $\mathcal{D}$, we call η a **natural isomorphism**, in which case write $F \simeq G$.



Natural morphisms map objects to morphisms, and morphisms to commuting squares. It is not easy to grasp an intuition for natural transformations without a broad inventory of concrete examples, which we won't provide here for space. Instead we direct the reader to [Milewski], for concrete applications of these categorical concepts to programming. The important takeaway here is that a natural isomorphism $\eta : F \Rightarrow G$ is a witness to the face that F and G are 'essentially the same'. Now we are equipped to tackle equivalence of categories succinctly.

Definition 7.9 (Equivalence)**.** Categories $\mathcal{C}$ and $\mathcal{D}$ are **equivalent**, written $\mathcal{C} \simeq \mathcal{D}$, if there are functors $F : \mathcal{C} \to \mathcal{D}$, $G : \mathcal{D} \to \mathcal{C}$, and natural isomorphisms such that $F; G \simeq \mathrm{Id}_{\mathcal{C}}$ and $G; F \simeq \mathrm{Id}_{C}$.

Note the familial resemblance to isomorphic arrows within a category! Asking for equality $F; G = \mathrm{Id}_{\mathcal{C}}$ is in a sense 'too strict', so this notion of equivalence of categories is a mild concession that nevertheless respects structures within the participating categories.

There is one more 'weakening' of the notion of equality that we will consider, where instead of asking for natural isomorphisms in the definition of equivalence, we ask for natural transformations to and from the identity functors.

Definition 7.10 (Adjunctions)**.** $L : \mathcal{C} \to \mathcal{D}$, $R : \mathcal{D} \to \mathcal{C}$ are **adjoint** when [15] for all objects $X \in \mathrm{Ob}(\mathcal{C})$ and $Y \in \mathrm{Ob}(\mathcal{D})$, we have

$$\mathrm{Mor}_{\mathcal{C}}[F(Y), X] \simeq \mathrm{Mor}_{\mathcal{D}}[Y, G]$$

In which case we say that F is **left adjoint** to G and G is **right adjoint** to F, sometimes written $F \dashv G$.

7.4 Monoidal Categories

7.4.1 Monoidal Signatures

To motivate the purpose of a monoidal signature before defining it requires a brief philosophical digression. When we speak of or reason about monoidal categories, we do so with a basic vocabulary provided to us by a monoidal signature, from which we construct a whole language using the composition of categories and the tensor product. The signature *qua* vocabulary plays the intermediary between us and whatever a monoidal category actually *is*. The notion of monoidal signature originally appeared under the name 'tensor scheme' [JS91, Dfn. 1.4], reproduced below.

[15] an equivalent definition is that there if exist natural transformations $\eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow L; R$ and $\epsilon : R; L \Rightarrow \mathrm{Id}_{\mathcal{D}}\eta$ and ϵ satisfy the yanking equations in the 2-category Cat.

Definition 7.11 (Tensor Scheme)**.** A **tensor scheme** $\mathcal{T}$ consists of two sets obj $\mathcal{D}$ and mor $\mathcal{D}$, together with a function which assigns to each element $d \in \text{mor } \mathcal{D}$ a pair $(d(0), d(1))$ of words in the elements of obj $\mathcal{D}$. Write

$$d : X_1 \cdots X_m \to Y_1 \cdots Y_n$$

for $d \in \text{mor } \mathcal{D}$ with $d(0) = X_1 \cdots X_m$, $d(1) = Y_1 \cdots Y_n$

The above definition implicitly assumes strictness, which Selinger [Sel10] dispenses with in his non-strict definition, which he calls 'Monoidal Signatures'.

Definition 7.12 (Monoidal Signatures)**.** Given a set of **object variables** Σ_0, let $\text{Mon}(\Sigma_0)$, the set of **object terms**, be the free $(I, \otimes)$-algebra – where I is nullary and $\otimes$ is binary – generated by Σ_0. A **Monoidal signature** consists of a set of object variables Σ_0, and a set of **morphism variables** Σ_1, along with interpretation functions dom() and cod(): $\Sigma_1 \to \text{Mon}(\Sigma_0)$.

Example 12. Writing $\otimes$ infix, the free $(I, \otimes)$-algebra over the set A, B would contain objects such as $((A \otimes I) \otimes B)$, A, $(I \otimes (A \otimes B))$, *etc.*

Example 13 (The monoidal signature of **Set**)**.** When we speak of **Set** as a strict monoidal category (with the categorical product as tensor product), the monoidal signature we use has the names of (all the nameable) sets as object variables, and the names of (all the nameable) functions between sets as morphism variables. Note that there is nothing yet to interpret the equations that hold in **Set**, such that the fact that the endomorphisms on $\mathbb{N}$ $(+1); (+1)$ and $(+2)$ are equal. But then, we have not yet defined what ; means. We only have some names. The role of equations between morphisms and objects is that of identifying denotation: just as 'the Morning Star' and 'the Evening Star' are different names with the same referent (though it took time to demonstrate!), an equation $f = g$ is the assertion that the two names f and g, built from the monoidal signature, in the end refer to the same thing, *whatever that thing is*.

7.4.2 Free algebras and adding type constructors to a signature

To formalise the notion of 'free algebra', we recourse to the notion of signature from Universal Algebra; there is an unfortunate clash of terminology, so we will call these Logical Signatures.

Definition 7.13 (Logical Signatures)**.** A (single-sorted) **logical signature** σ is a triple $(\texttt{Fn}, \texttt{Rl}, \texttt{Ar})$, consisting of:

1. `Fn`: a set of **function symbols**
2. `Rl`: a set of **relation symbols** (disjoint from `Fn`)
3. `Ar`: a function $\texttt{Fn} \cup \texttt{Rl} \to \mathbb{N}$ that assigns all function and relation symbols a natural number **arity**

When `Fn` and `Rl` are finite, σ is **finite**. When `Fn` is empty, σ is **relational**. When `Rl` is empty, σ is **algebraic**.

Adding in the object variables, we have:

Definition 7.14 (Structures)**.** A structure Σ is a triple $(D, \sigma, (-)^\Sigma)$, consisting of:

1. D: an arbitrary set known as the **domain**, or **carrier**

2. σ: a logical signature

3. $(-)^\Sigma$: an assignment (**interpretation**) of:

   - a function $f^\Sigma : D^{\mathtt{Ar}(f)} \to D$ for each function symbol $f \in \mathtt{Fn}$
   - an n-ary relation $R^\Sigma \subseteq D^{\mathtt{Ar}(R)}$ for each relation symbol $R \in \mathtt{Rl}$

When the logical signature is algebraic, the free algebra contains 'all possible terms'.

Definition 7.15 (Free algebra on an algebraic signature)**.** Given an algebraic signature σ, the **free σ-algebra** over a set of object variables D is the structure with:

- σ: an algebraic signature
- Domain D^σ, which is defined to be closed under the following grammar:

$$\tau := d \in D \mid f(\tau_1, \ldots, \tau_n)$$

  For all f in $\mathtt{Fn}$ of σ with arity n.

- Each object $\tau \in D^\sigma$ and function f interpreted *syntactically*; *i.e.*, $(\tau)^\Sigma = \tau$, and f maps n-ary tuples $(\tau_1, \ldots, \tau_n)$ of objects in $\mathbf{D}$ to $f(\tau_1, \ldots, \tau_n)$.

Evidently, if σ is an algebraic signature, D^σ has the same objects as $(D^\sigma)^\sigma$; the free generation is a fixed point.

Remark 2. We choose this presentation of free algebra, as we will eventually want to add more function symbols in addition to $(I, \otimes)$ in the algebraic signature, and these new function symbols will serve as constructors for new objects.

Example 14 (Elements of a free residuated pregroup on D)**.** Given a set $D = \{A, B\}$, and an algebraic logical signature σ with two binary functions $(\_/\_)$ and $(\_\backslash\_)$, the elements of D^σ are, in no particular order:

$$\{A, B, (A/B), (B/A), ((A/A)\backslash B), \ldots\}$$

The interpretation of the product operation is the job of the tensor product to be mixed in later; all we are doing now is constructing the basic types.

Example 15 (A free pre-autonomous structure on D)**.** We'll begin a running example which will eventually become a free autonomous construction. Take Aut to be the signature consisting of two unary function symbols $(-)^*$ and ${}^*(-)$, which will play the role of left and right adjoint-type constructors for objects. If $D = \{A, B\}$, then D^{Aut} consists of:

$$\{A, B, (A)^*, {}^*(A), ((A)^*)^*, {}^*({}^*(B)), \ldots\}$$

What's missing here, as opposed to the residuated pregroup case, is an interpretation rule saying that left and right adjoints cancel: ${}^*((x)^*) = x = ({}^*(x))^*$.

Takeaway 3: Monoidal Signatures provide a basic stock of named objects and morphisms, and we can expand the stock of named objects by expanding the algebraic signature.

7.4.3 Monoidal Categories

Definition 7.16 (Monoidal Category)**.** A **monoidal category** is a tuple $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$ where

- $\mathcal{C}$ is a category
- $\otimes$, the **tensor**, is a functor $C \times C \to C$, and we use the infix notation $A \otimes B$ for $\otimes(A, B)$
- I, the **unit**, is an object of C
- α, λ, ρ are natural isomorphisms, with types (for all X, Y, Z in $\mathcal{C}$):
  - $\alpha_{X,Y,Z} : ((X \otimes Y) \otimes Z) \to (X \otimes (Y \otimes Z))$ – the **associator**
  - $\rho_X : X \otimes I \to X$ – the **right unitor**
  - $\lambda_X : I \otimes X \to X$ – the **left unitor**

  The natural isomorphisms above must satisfy the **triangle** and **pentagon** equations, which are expressed succinctly as the condition that the following diagrams commute for all objects in the category.

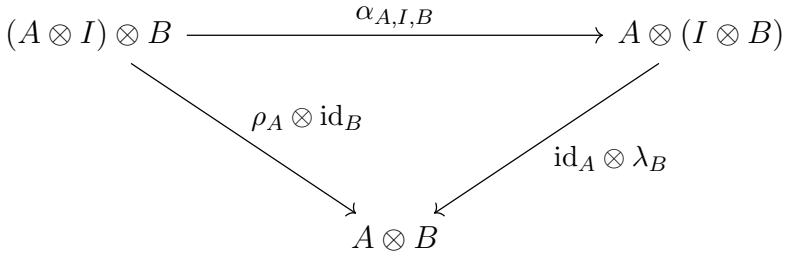


Figure 38: The triangle equation

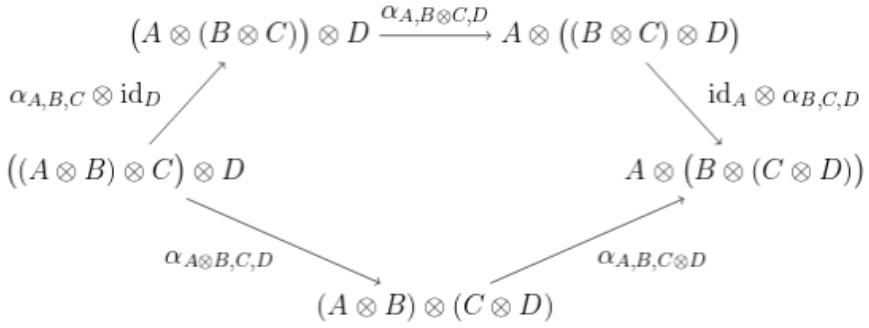


Figure 39: The pentagon equation

Definition 7.17 (Structures on Monoidal Categories)**.** A monoidal category is further:

- **strict** if the natural transformations above are identities
- **symmetric** if there is a **twist** natural transformation $\theta_{X\otimes Y} : X \otimes Y \to Y \otimes X$ such that

$$\theta_{Y\otimes X} \circ \theta_{X\otimes Y} = \mathrm{id}_{X\otimes Y}$$

- **compact closed** if for each object X there are objects X^L (the **left dual**), X^R (the **right dual**), and natural transformations:

$$\eta^L_X : I \to X \otimes X^L \qquad \eta^R_X : I \to X^R \otimes X$$
$$\epsilon^L_X : X \otimes X^L \to I \qquad \epsilon^R_X : X^R \otimes X \to I$$

  Which satisfy the **yanking equations**:

$$(\mathrm{id}_X \otimes \epsilon^L_X) \circ (\eta^L_X \otimes \mathrm{id}_X) = \mathrm{id}_X \qquad (\epsilon^R_X \otimes \mathrm{id}_X) \circ (\mathrm{id}_X \otimes \eta^R_X) = \mathrm{id}_X$$
$$(\epsilon^L_X \otimes \mathrm{id}_{X^L}) \circ (\mathrm{id}_{X^L} \otimes \eta^L_X) = \mathrm{id}_{X^L} \qquad (\mathrm{id}_{X^R} \otimes \epsilon^R_X) \circ (\eta^R_{X^R} \otimes \mathrm{id}_{X^R}) = \mathrm{id}_{X^R}$$

Notice that the objects of a monoidal category are generated by the free algebra from a monoidal signature. Further, observe that in the case of compact closed structure, the η and ϵ natural transformations require that the algebra of the monoidal signature is expanded to include unary functions $(-)^L, (-)^R$, such that the free algebra supports objects such as $(X^L \otimes I)^R$.

Monoidal categories admit an elegant graphical calculus. Penrose first used the graphical calculus in 1971 as a private tool for tensor contraction calculations [Pen71]. With the advent of category theory, two major theorems established sound foundations for the calculus. The first was MacLane's Coherence theorem, which showed that the triangle and pentagon equations were necesary and sufficient for 'all well-typed expressions built from the associator, unitors, and their inverses commute': essentially removing the bureaucracy of bracketing induced by $\otimes$. Joyal and Street then formalised the graphical calculus in 1991 [JS91], proving that well-typed diagrams are sound and complete for monoidal categories up to planar isotopies: the important aspect of the diagrams is their connectivity, not the specific geometry of any individual wire. The standard reference that surveys such graphical calculi is [Sel10].

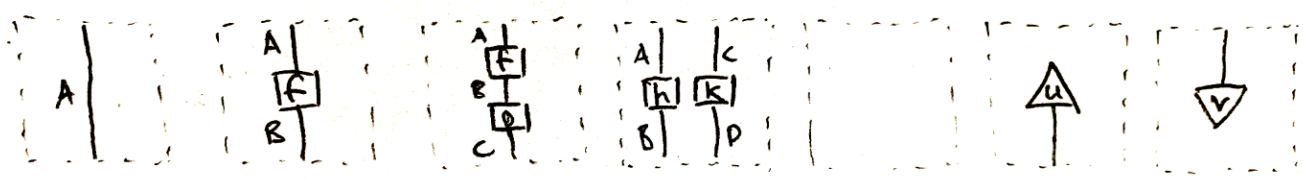


Figure 40: We adopt the convention that composition reads from top to bottom. From left to right: an object A; a morphism $A \xrightarrow{f} B$; the composite $A \xrightarrow{f} B \xrightarrow{g} C$; the tensor of $A \xrightarrow{h}$ and $C \xrightarrow{k} D$; a **state** morphism $u : I \to A$; an **effect** morphism $v : A \to I$. I is the empty diagram.

Notice that we can also express the equations governing compact closed structure diagrammatically:

The drawing of cups and caps as bent wires is really a graphical liberty. Formally, η and ϵ live inside boxes like any other morphism; we simply omit the bounding box around them. Similarly, we can also express symmetric structure graphically:

Takeaway 4: Structure consists of natural transformations, which may introduce type constructors, and equations that hold between natural transformations. The type constructors are passed to the algebra in the monoidal signature, and the natural transformations and equations between them can be expressed *graphically*. Conversely, graphical equations between natural transformations induce structural constrants on categories.

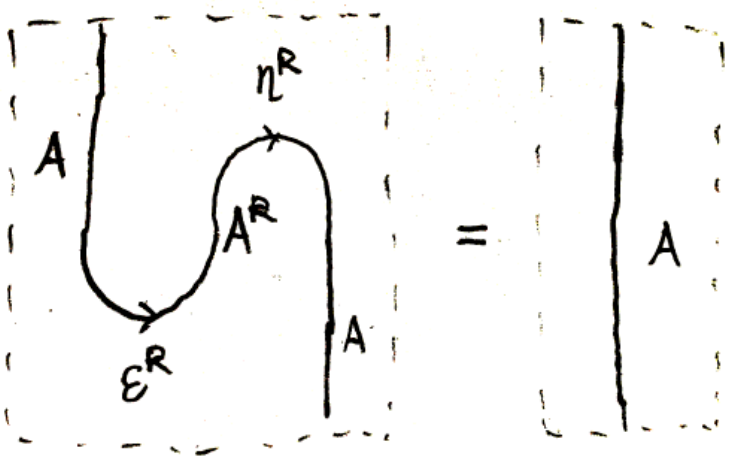


Figure 41: The ϵ, η morphisms are depicted as **cups** and **caps** respectively (take care, as we read diagrams from top to bottom), and are depicted along with the yanking equations above. The single equation above is actually a system of equations, natural in A: recall that natural transformations are parameterised over the objects in a category.

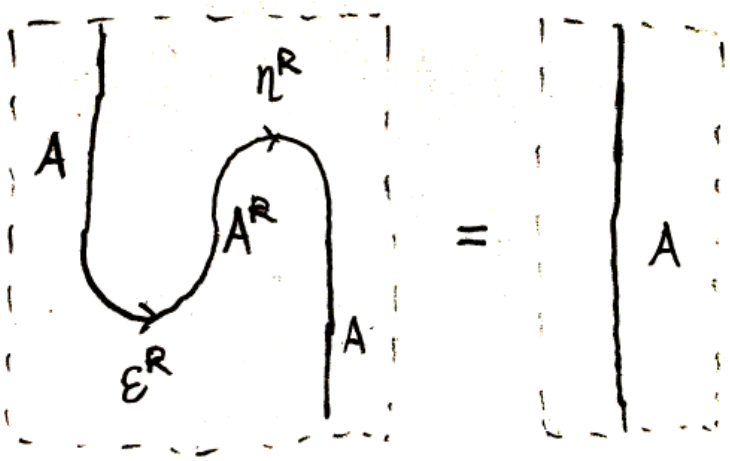


Figure 42: Symmetry

7.4.4 Monoidal Functors and Natural Transformations

Monoidal functors and natural transformations between monoidal categories are just like their regular counterparts, except that they carry additional coherence maps that preserve the monoidal structure in their sources and targets.

Definition 7.18 (Monoidal Functor)**.** A (**lax**) **monoidal functor** between monoidal categories $(\mathcal{C}, \otimes_\mathcal{C}, I_\mathcal{C})$ and $(\mathcal{D}, \otimes_\mathcal{D}, I_\mathcal{D})$ (with associator and unitor natural transformations implicit) consists of:

- A functor $F : \mathcal{C} \to \mathcal{D}$
- A morphism $\epsilon : I_\mathcal{D} \to F(I_\mathcal{C})$ in $\mathcal{D}$
- a natural transformation $\mu_{a,b} : F(a) \otimes_\mathcal{D} F(b) \to F(a \otimes_\mathcal{C} b)$

The latter two data are called **coherence maps**, and they must satisfy the *associativity* and *unitality* equations, presented as commuting diagrams below:

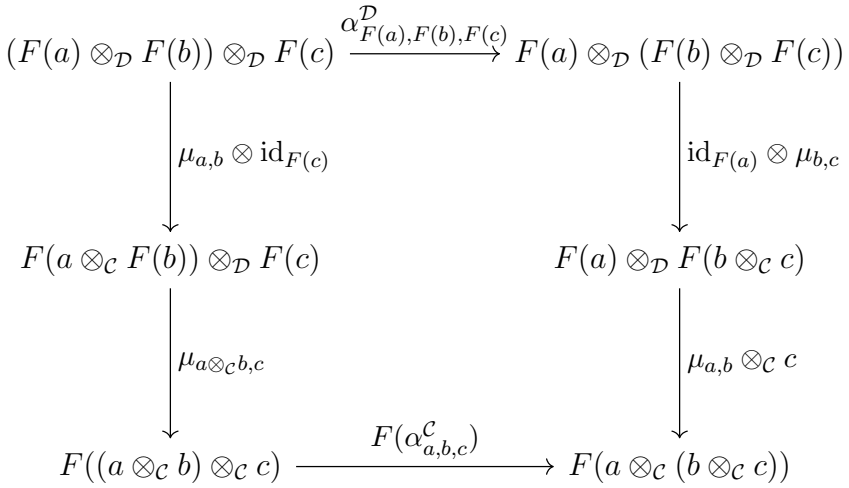


Figure 43: Associativity: the diagram above commutes for all $a, b, c \in \mathcal{C}$.

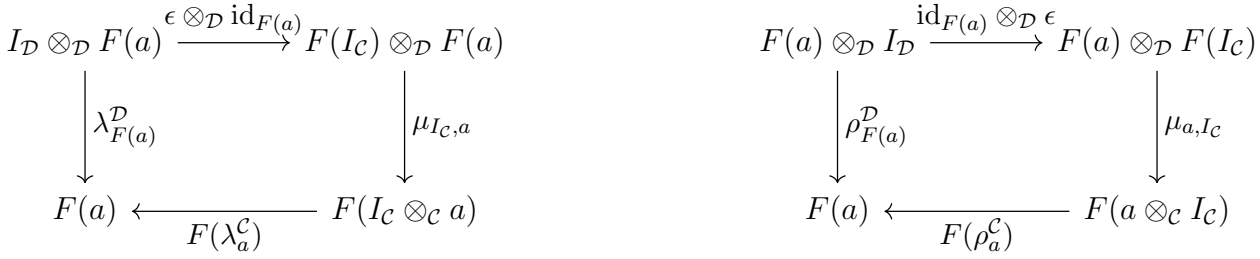


Figure 44: Unitality: the diagrams above commutes for all $a \in \mathcal{C}$.

If the coherence maps are isomorphisms, we have a **strong monoidal functor**. If the coherence maps are identities, we have a **strict monoidal functor**.

Definition 7.19 (Monoidal Natural Transformation)**.** If (F, μ, ϵ) and (G, μ', ϵ') are monoidal functors from the monoidal categories $\mathcal{C}$ to $\mathcal{D}$, a **monoidal natural transformation** $\phi : F \to G$ is a natural transformation between the functors F and G such that the following diagram commutes.

If the components of the natural transformation are isomorphisms, then we have a **monoidal natural isomorphism**.

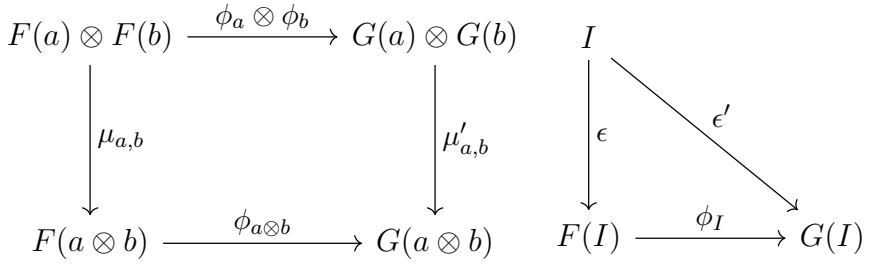


Figure 45: Monoidal Naturality Conditions (subscripts for domain and codomain categories of F, G omitted)

Definition 7.20 (Equivalence of monoidal categories)**.** A monoidal functor is an equivalence of monoidal categories if it is an equivalence of ordinary categories.

Now, we can state a theorem that makes life a lot simpler[16]; we reproduce below its statement from [Johb]:

Theorem 3 (MacLane's Coherence Theorem)**.** *Given a monoidal category $\mathcal{C}$, there exists a strict monoidal category $\mathcal{C}'$ for which there is a monoidal equivalence $F : \mathcal{C} \to \mathcal{C}'$.*

Recall that in strict monoidal categories, the associator and unitor natural isomorphisms are all identities, so all objects are equivalent up to different bracketings and absorbing monoidal units: we have shed a lot of bureaucracy! So, without loss of generality, we will work with just strict monoidal categories, and strong monoidal functors between them.

Definition 7.21 (**Mon**)**. Mon** is the category with strict monoidal categories as objects, and strong monoidal functors as morphisms.

7.5 Interpretations of, and Free Monoidal Categories over Monoidal Signatures

For this section, it is useful to keep in mind Takeaway 1:

Takeaway 1: Free and thin categories are two ends of a spectrum, and we can reach parts in between using equations between arrows.

Definition 7.22 (Interpretations of Monoidal Signatures)**.** Given a monoidal signature Σ (recall definition 7.12) and a monoidal category $\mathcal{K}$, an **interpretation** $[\![\_]\!] : \Sigma \mapsto \mathcal{K}$ consists of:

- An object function $[\![\_]\!]_0 : D^\Sigma (= D) \to \mathrm{Ob}(\mathcal{K})$, which extends in a unique way to an object function $[\![\_]\!]_0^* : \mathbf{T} \to \mathrm{Ob}(\mathcal{K})$, such that:
  - $[\![aB]\!]_0^* = [\![a]\!]_0 \otimes_\mathcal{K} [\![B]\!]_0^*$; where $a \in D$ (see remark)
  - $[\![\epsilon]\!]_0^* = I_\mathcal{K}$
- A morphism function $[\![\_]\!]_1$, that maps $f \in \mathbf{Name}$ to $[\![f]\!]_1 : [\![\mathrm{dom}\ f]\!]_0 \to [\![\mathrm{cod}\ f]\!]_0$

Remark 3. Regarding uniqueness, Selinger's original definition is directly transliterated as:

$$[\![A \otimes B]\!]_0^* = [\![A]\!]_0^* \otimes_\mathcal{K} [\![B]\!]_0^*$$

Which is due to the fact that he uses a non-strict definition, whereas we are working in anticipation of a generalisation in strict monoidal categories. The codomain of his interpretation function is free algebra $(D, \otimes, I)$, rather than the free monoid, so the splitting point between A and B is always uniquely defined as the topmost $\otimes$ connective. In the strict case, the uniqueness condition is implicitly forbidding situations like $[\![a]\!] = A$, $[\![b]\!] = B$, $[\![ab]\!] = C$, whereupon splitting ab as (a, b) and (ab, ϵ) yield distinct values. Since what we really want is a coalgebra on lists [Jac16, §2.4], we have adapted this rule to reflect the structure of the initial coalgebra on lists, with no loss of generality, since we will be working exclusively with strict monoidal categories.

Now that we have defined interpretations, we can define 'freeness'. Selinger, again paraphrasing Joyal and Street, defines free monoidal categories over a monoidal signature as follows:

[16] oddly difficult to track down the original source for MacLane's Coherence Theorem.

Definition 7.23 (Free monoidal categories)**.** A monoidal category $\mathcal{K}$ is a **free monoidal category** over a monoidal signature Σ if it is equipped with an interpretation $[\![\_]\!]_0 : \Sigma \mapsto \mathcal{K}$ such that for any monoidal category $\mathcal{J}$ and interpretation $[\![\_]\!] : \Sigma \mapsto \mathcal{J}$, there exists a strong monoidal functor $F : \mathcal{K} \to \mathcal{J}$, such that $[\![\_]\!] = [\![\_]\!]_0; F$, and F is unique up to a unique monoidal natural isomorphism.

Freeness so conceived is a universal property (specifically, initiality) in a suitable category of interpretations.

Definition 7.24 (Category of strict interpretations of a monoidal signature Σ)**.** Given a monoidal signature Σ, the category $\mathbf{Mon}(\Sigma)$ is defined to have:

- Objects pairs $(\iota, \mathcal{D})$ of interpretations $\iota : \Sigma \to \mathcal{D}$, and strict monoidal categories $\mathcal{D}$, (which, recalling remark **??**, determine strict monoidal subcategories of $\mathcal{D}$)
- Morphisms strong monoidal functors between the objects

Observe that $\mathbf{Mon}(\Sigma)$ is evidently a subcategory of $\mathbf{Mon}$.

The following theorem relates the free monoidal category over a monoidal category to the graphical calculus:

Theorem 4. *[Sel10, Thm. 3.3] The graphical language of monoidal categories over a monoidal signature Σ, up to planar isotopy of diagrams, forms a free monoidal category over Σ.*

Takeaway 5: The consequence of definition 10 and theorem 4 is that any category that we can prove is initial in $\mathbf{Mon}(\Sigma)$ is monoidally equivalent to the category formed by the graphical language over Σ.

7.6 General Monoidal Signatures

Now we recount the last four takeaways, to motivate our gameplan moving forward.

Takeaway 2: The data of functors can be encoded as equivalence classes of morphisms in the the codomain category.

Takeaway 3: Monoidal Signatures provide a basic stock of named objects and morphisms, and we can expand the stock of named objects by expanding the algebraic signature.

Takeaway 4: Structure consists of natural transformations, which may introduce type constructors, and equations that hold between natural transformations. The type constructors are passed to the algebra in the monoidal signature, and the natural transformations and equations between them can be expressed *graphically*. Conversely, graphical equations between natural transformations induce structural constrants on categories.

Takeaway 5: The consequence of definition 10 and theorem 4 is that any category that we can prove is initial in $\mathbf{Mon}(\Sigma)$ is monoidally equivalent to the category formed by the graphical language over Σ.

Suppose we have a (without loss of generality) strict monoidal category $\mathcal{K}$, which possesses some specified structural properties $\mathbf{S}$. We know that $\mathcal{K}$ is equivalently specified up to monoidal natural isomorphism by some strong monoidal functor K from the free monoidal category over Σ, which is the category of diagrams. By takeaway 4, the structural properties $\mathbf{S}$ may be expressed graphically. By takeaway 3, the type constructors of $\mathbf{S}$ may be included by expanding the underlying algebraic signature appropriately. By takeaway 2, the strong monoidal functor K must induce

equivalence classes of diagrams such that the structural equations of **S** are respected. Moreover, knowing these equivalence classes means knowing 'everything' about K, by the initiality of the free category over Σ.

Now suppose we have a system to construct objects and morphisms between them, and we feed this system the data from Σ, such that the category of derivable terms is the free category over Σ. By takeaway 5, any such system is equivalent to the category formed by the graphical language over Σ, so the system would be a *source code* for building diagrams. Suppose further that we may also derive equations between terms within the system, and we feed the system data from K and **S** (expanding the algebraic signature of Σ as necessary) such that the derivable equivalence classes of objects and morphisms are precisely those induced by K and **S**: we would have a system equivalent to the graphical calculus with equations that capture the specific properties of the monoidal category $\mathcal{K}$. Finally, suppose that we have a method to encode extra structural properties **T** in the source code, such that the diagrams generated may contain novel objects and morphisms not originally present in the signature of $\mathcal{K}$, but such that one can still reason graphically.

Definition 7.25 (General Monoidal Signatures)**.** Given a monoidal signature Σ, which consists of:

- an algebraic signature σ with at least a nullary I and binary $\otimes$
- object variables Σ_0
- morphism variables Σ_1 with domain and codomains in the free algebra $(\Sigma_0)^\sigma$

and structural properties **S**, which consists of:

- A set of type constructors for objects, each of which is interpreted as a function symbol with associated arity in σ
- A set of natural transformations $\star$, such that each type constructor is used in at least one. These include the associators and unitors.
- A system of graphical equations **Str** that involve the natural transformations

and a strong monoidal functor K from the free monoidal category on Σ to a target category $\mathcal{M}$, which induces an equivalence relation **Den** on morphisms in the free monoidal category on Σ that, by Proposition 2, also captures the action of K on objects of the free category over Σ,

define the associated **general monoidal signature** $\mathfrak{S}$ to have data:

- the algebraic signature σ
- object variables Σ_0
- morphism variables Σ_1, with accompanying domain and codomain data
- structural morphisms $\star$, freely instantiated over $(\Sigma_0)^\sigma$
- an equivalence relation $=_{\mathbf{Den+Str}}$ defined on morphisms in the free monoidal category over Σ plus all components of the natural transformations $\star$ instantiated freely over $(\Sigma_0)^\sigma$, which is the reflexive, symmetric, and transitive closure of the two equivalence relations **Den** and **Str**

Let's hack diagrams.

8 The calculus $\mathtt{Mon}(\mathfrak{S})$

We adopt the convention that lowercase romans a, b, c, d are used whenever the side condition states "For all $a, b, c, d \in (\Sigma_0)^\sigma$". Uppercase romans occurring in the premise and conclusion of rules are contexts requiring syntactic matches. Lowercase romans f, g are used for general constructed morphisms. We include a copy of the rules in the appendix for easy reference.

Object and Morphism constructors:

Initial sequents:

$$\text{(Id)}\ \frac{}{a \overset{\mathrm{id}_a}{\to} a} \qquad \text{(Unit)}\ \frac{}{I \overset{\mathbf{1}}{\to} I} \qquad \text{(Den-Id)}\ \frac{}{a \overset{\mathrm{id}_X}{\to} b}\ [\text{If } id_a =_{\mathbf{Den+Str}} \mathrm{id}_b;\ X \text{ can be either } a \text{ or } b]$$

$$\text{(Name)}\ \frac{}{\mathrm{cod}(f) \overset{f}{\to} \mathrm{dom}(f)}\ [\text{For all } f \in \Sigma_1] \qquad (\star)\ \frac{}{\mathrm{cod}(\omega(\overrightarrow{a})) \overset{\omega_a}{\to} \mathrm{dom}(\omega(\overrightarrow{a}))}\ [\text{For all } \omega \text{ in } \star]$$

$$(\lambda)\ \frac{}{(I \otimes a) \overset{\lambda_a}{\to} a} \qquad (\rho)\ \frac{}{(a \otimes I) \overset{\rho_a}{\to} a} \qquad (\alpha)\ \frac{}{((a \otimes b) \otimes c) \overset{\alpha_{a,b,c}}{\to} (a \otimes (b \otimes c))}$$

$$(\lambda^{-1})\ \frac{}{a \overset{\lambda_a^{-1}}{\to} (I \otimes a)} \qquad (\rho^{-1})\ \frac{}{a \overset{\rho_a^{-1}}{\to} (a \otimes I)} \qquad (\alpha^{-1})\ \frac{}{(a \otimes (b \otimes c)) \overset{\alpha_{a,b,c}^{-1}}{\to} ((a \otimes b) \otimes c)}$$

Non-initial sequents:

$$(;)\ \frac{A \overset{f}{\to} B \qquad B \overset{g}{\to} C}{A \overset{(f;g)}{\to} C} \qquad\qquad (\otimes)\ \frac{A \overset{f}{\to} B \qquad C \overset{g}{\to} D}{(A \otimes C) \overset{(f \otimes g)}{\to} (B \otimes C)}$$

Equation constructors:

Equality Axioms:

$$\text{(R)}\ \frac{A \overset{f}{\to} B}{A \overset{f=f}{\to} B} \qquad \text{(S)}\ \frac{A \overset{f=g}{\to} B}{A \overset{g=f}{\to} B} \qquad \text{(T)}\ \frac{A \overset{f=g}{\to} B \qquad A \overset{g=h}{\to} B}{A \overset{f=h}{\to} B}$$

Categorical Structure: (double line indicates reversible rule)

$$\text{(Left Unitality)}\ \frac{A \overset{(\mathrm{id}_A;f)=g}{\to} B}{A \overset{f=g}{\to} B} \qquad \text{(Right Unitality)}\ \frac{A \overset{(f;\mathrm{id}_B)=g}{\to} B}{A \overset{f=g}{\to} B} \qquad \text{(Assoc)}\ \frac{A \overset{((f;g);h)=j}{\to} B}{A \overset{(f;(g;h))=j}{\to} B}$$

(Strict) Monoidal Structure:

$$\text{(Interchange)}\ \frac{A \overset{f}{\to} X \qquad X \overset{g}{\to} C \qquad (A \otimes B) \overset{((f;g)\otimes(h;j))=k}{\to} C \otimes D \qquad B \overset{h}{\to} Y \qquad Y \overset{j}{\to} D}{(A \otimes B) \overset{((f\otimes h);(g\otimes j))=k}{\to} (C \otimes D)}$$

$$\text{(Id-Tensor)}\ \frac{}{(a \otimes b) \overset{\mathrm{id}_{(a\otimes b)}=(\mathrm{id}_a\otimes\mathrm{id}_b)}{\to} (a \otimes b)}$$

$$(\lambda_=)\ \frac{}{(I \otimes a) \overset{\lambda_a=\mathrm{id}_{(I\otimes a)}}{\to} a} \qquad (\rho_=)\ \frac{}{(a \otimes I) \overset{\rho_a=\mathrm{id}_{(a\otimes I)}}{\to} a} \qquad (\alpha_=)\ \frac{}{((a \otimes b) \otimes c) \overset{\alpha_{a,b,c}=\mathrm{id}_{((a\otimes b)\otimes c)}}{\to} (a \otimes (b \otimes c))}$$

$$(\lambda^{-1}_=)\ \frac{}{a \overset{\lambda^{-1}_a=\mathrm{id}_a}{\to} (I \otimes a)} \qquad (\rho^{-1}_=)\ \frac{}{a \overset{\rho^{-1}_a=\mathrm{id}_a}{\to} (a \otimes I)} \qquad (\alpha^{-1}_=)\ \frac{}{(a \otimes (b \otimes c)) \overset{\alpha^{-1}_{a,b,c}=\mathrm{id}_{(a\otimes(b\otimes c))}}{\to} ((a \otimes b) \otimes c)}$$

Diagram Manipulation:

$$(=^{\leftrightarrow})\ \frac{A \overset{f=f'}{\to} B \qquad (A \otimes C) \overset{(f\otimes g)=h}{\to} (B \otimes D) \qquad C \overset{g=g'}{\to} D}{(A \otimes C) \overset{(f'\otimes g')=h}{\to} (B \otimes D)}$$

$$(=^{\updownarrow})\ \frac{A \overset{f=f'}{\to} B \qquad A \overset{(f;g)=h}{\to} C \qquad B \overset{g=g'}{\to} C}{A \overset{(f';g')=h}{\to} C}$$

Denotations and Structure:

$$\text{(Den)}\ \frac{A \overset{f}{\to} B \qquad C \overset{g}{\to} D}{A \overset{f=g}{\to} D}\ [\text{For all } f =_{\mathbf{Den}} g]$$

$$\text{(Str)}\ \frac{A \overset{f}{\to} B \qquad A \overset{g}{\to} B}{A \overset{f=g}{\to} B}\ [\text{For all } f =_{\mathbf{Str}} g]$$

8.1 A guided tour of $\mathtt{Mon}(\mathfrak{S})$

This sequent calculus is a simple type theory, which takes contexts from the free algebra $(\Sigma_0)^\sigma$: the arrows can also be read as turnstiles. The calculus only keeps one context on each side of the arrow, which reflects the property that every derivation is meant to be of a particular diagram.

First we consider the morphism constructors. The initial sequent (Id) is moonlighting as a type declaration $a : \mathrm{Type} \vdash a : \mathrm{Type}$; we choose to omit explicit type declarations, as we will not consider dependent type theories. (Unit) is a special case of (Id) for the monoidal unit.

(Den-Id) is worth elaborating: we will later take equivalence classes of objects in $(\Sigma_0)^\sigma$ as objects in a strict monoidal category on the basis of whether $\mathrm{id}_a = \mathrm{id}_b$ is derivable. The side

condition splits this rule into two, allowing either the domain or codomain to claim the identity. The function of (Den-Id) in the calculus is to license type-changes in the domain or codomain, in conjunction with (Left Unitality) and (Right Unitality). The broader purpose of (Den-Id) is to reflect the objects identified by the interpreting strong monoidal functor K over the free category: recalling example 15, we may wish to have that left and right adjoints of objects cancel via equations $a = (a^L)^R$, which (Den-Id) implements.

(Name) allows us to introduce named morphisms from Σ_1 as basic vocabulary elements for building morphisms, and $(\star)$ allows the introduction of instantiated components of the natural transformations carried in the structure $\mathbf{S}$: $\vec{a}$ is shorthand for a nonempty list of objects $a_i \in (\Sigma_0)^\sigma$. For a concrete example, supposing that $\mathbf{S}$ is symmetric structure in the form of a twist natural transformation $\theta : X \otimes Y \to Y \otimes X$, the $(\star)$-rule would allow initial sequents of the form $\theta_{a,b} : (a \otimes b) \to (b \otimes a)$ for every pair of objects a, b from the free algebra $(\Sigma_0)^\sigma$. Worth remarking is that the structure $\mathbf{S}$ may contain nullary function symbols, which introduce named objects in whatever category possess that structure (such as the monoidal unit!) While we will generally not consider such cases, such cases are easy to accommodate by subsuming them under the (Id)-rule much like (Unit).

The remaining initial sequents are the associator and unitor natural isomorphisms, presented with their inverses. Using these rules invokes a particular component of that natural transformation, much like $(\star)$.

The only two rules for building new morphisms out of old are to vertically (;) and horizontally ($\otimes$) compose them, respecting domains and codomains in the case of vertical composition.

It should be evident by inspection that the derivable morphisms are precisely those of the free monoidal category over Σ (if we do not include any extra data from K and $\mathbf{S}$.) Even if that isn't convincing, we will prove freeness by other means later. Before moving onto the equation constructors, we remark on a very important property of the morphism building rules: analyticity. The name of every constructed morphism in a premiss appears in the constructed morphism in the conclusion. Further, since the ; and $\otimes$ connectives are introduced by unique rules and properly bracketed, just by looking at the syntax of any derived morphism, one can reconstruct the entire unique proof tree that built it.

The equality axioms are in the form required to make $=$ an equivalence relation on morphisms, and are straightforward.

The categorical structure rules enforce the equalities that we want present in any category, and will suffice to prove that the term category we eventually construct really is a category.

The rules for strict monoidal structure require some elaboration. The first two rules govern the interaction of the ; and $\otimes$ connectives, as the interaction of the two forms of composition must respect the bifunctoriality of the tensor product functor $\otimes$. The presentation of the two rules is a translation of the conditions from [Sel10, §3.1]. (Id-Tensor) enforces coherence of identities with the $\otimes$ functor. (Interchange) enforces the the Interchange Law, which only holds when the participating morphisms form two $\otimes$-separable ;-chains. We could have equivalently placed the typing requirements in a side condition, but we have avoided that here to preserve a form of analyticity

we will remark upon later. The placement of the participating morphisms in the wings is purely aesthetic; the arrangement doesn't matter.

The remaining strict monoidal structure equality rules simply force the associators, unitors, and their inverses to be identities, which we have presented explicitly here. Alternatively, we could have explicitly encoded the property of isomorphism (*e.g.* $\alpha; \alpha^{-1} = \mathrm{id}$) and the triangle and pentagon equations as such rules, which would have worked just as well. It is worth remarking that in each case, we have given possession of the identity to the codomain, but this is not important, as we can always derive an equality between the identity on the domain and the identity on the codomain. We give an example below:

Example 16 (Equalities between identities)**.**

$$\dfrac{(\lambda_=)\dfrac{}{(I\otimes a)\overset{\lambda_a=\mathrm{id}_{(I\otimes a)}}{\to} a} \quad (\mathrm{S})\dfrac{(\lambda)\dfrac{}{(I\otimes a)\overset{\lambda_a}{\to} a} \quad (\mathrm{Id})\dfrac{}{a\overset{\mathrm{id}_a}{\to} a}}{(;)\dfrac{(I\otimes a)\overset{(\lambda_a;\mathrm{id}_a)}{\to} a}{(I\otimes a)\overset{(\lambda_a;\mathrm{id}_a)=(\lambda_a;\mathrm{id}_a)}{\to} a}} \quad (\mathrm{R})\dfrac{(\mathrm{Id})\dfrac{}{a\overset{\mathrm{id}_a}{\to} a}}{a\overset{\mathrm{id}_a=\mathrm{id}_a}{\to} a}}{(\mathrm{S})\dfrac{(I\otimes a)\overset{(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)=(\lambda_a;\mathrm{id}_a)}{\to} a}{(I\otimes a)\overset{(\lambda_a;\mathrm{id}_a)=(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)}{\to} a}}(=^{\updownarrow})$$

Continuing and compressing for space:

$$\dfrac{(\lambda_=)\dfrac{}{(I\otimes a)\overset{\lambda_a=\mathrm{id}_{(I\otimes a)}}{\to} a} \quad \dfrac{\vdots}{(I\otimes a)\overset{(\lambda_a;\mathrm{id}_a)=(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)}{\to} a} \quad (\mathrm{R})\dfrac{(\mathrm{Id})\dfrac{}{a\overset{\mathrm{id}_a}{\to} a}}{a\overset{\mathrm{id}_a=\mathrm{id}_a}{\to} a}}{(\text{Left Unitality})\dfrac{(I\otimes a)\overset{(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)=(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)}{\to} a}{(\mathrm{S})\dfrac{(I\otimes a)\overset{\mathrm{id}_a=(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)}{\to} a}{(\text{Right Unitality})\dfrac{(I\otimes a)\overset{(\mathrm{id}_{(I\otimes a)};\mathrm{id}_a)=\mathrm{id}_a}{\to} a}{(I\otimes a)\overset{\mathrm{id}_{(I\otimes a)}=\mathrm{id}_a}{\to} a}}}(=^{\updownarrow})$$

One can see by the length of the proofs the cost of parsimonious rules! Note that in the above proof, we make use of the $(=^{\updownarrow})$ rule, which allows replacement of vertical slices in a diagram on the basis of derivable equalities. Its sister rule $(=^{\leftrightarrow})$ licenses replacements of horizontal slices. We'll just remark that by using the rules judiciously, one can perform targeted replacement of subdiagrams, chiefly by the (Assoc) and (α) rules. We won't pursue such an avenue, because we will show that the eventual category we build out of terms is equivalent to the category formed by the graphical calculus by other means, but the proof sketch of replacement of 'subdiagrams' purely within the logic, along with other lemmas, are in the appendix material.

The final rule (Den/Str) introduces equalities between derivable terms, reflecting the equations in **Den** and **Str**. There are some features to note. Firstly, the domains and codomains do not necessarily match, reflecting the fact that in general, functors K may map type-mismatched morphisms in a source category to the same image morphism in the target. However, by proposition

2, whenever $A \neq C$ syntactically, we will also have a (Den-Id)-rule $A \overset{\mathrm{id}}{\to} C$, which will resolve morphism typing when we take equivalence classes. While in the conclusion of the rule we have set the equality between the codomain of the 'first' premiss and the domain of the 'second' premise, we allow the premisses to occur in either order, and using the present (Den-Id)-rules, we can always derive any combination of domain and codomain we wish, for example:

Example 17 (Choosing domains and codomains for (Den))**.**

$$\dfrac{\dfrac{\dfrac{}{A \overset{\mathrm{id}_A}{\to} C}(\text{Den-Id})}{A \overset{\mathrm{id}_A = \mathrm{id}_A}{\to} C}(\text{R}) \qquad \dfrac{\dfrac{\dfrac{}{A \overset{\mathrm{id}_A}{\to} A}(\text{Id}) \quad \dfrac{\vdots(f)}{A \overset{f}{\to} B}}{A \overset{\mathrm{id}_A ; f}{\to} B}(;)}{A \overset{\mathrm{id}_A ; f = \mathrm{id}_A ; f}{\to} B}(\text{R}) \qquad \dfrac{\dfrac{\vdots(g)}{C \overset{g}{\to} D} \quad \dfrac{\vdots(f)}{A \overset{f}{\to} B}}{C \overset{g=f}{\to} B}(\text{Den})}{A \overset{\mathrm{id}_A ; f = f}{\to} B}(\text{S})}{A \overset{\mathrm{id}_A ; g = f}{\to} B}(=^{\updownarrow}) \qquad \dfrac{}{A \overset{g=f}{\to} B}(\text{L.U.})$$

8.2 Useful notions and Lemmas for later

Definition 8.1 (Derivability)**.** Write $\mathtt{Mon}(\mathfrak{S}) \vdash A \overset{f}{\to} B$ if there is a proof of $A \overset{f}{\to} B$ in $\mathtt{Mon}(\mathfrak{S})$. We say then that the **name** f **is derivable** in $\mathtt{Mon}(\mathfrak{S})$. Write $\mathtt{Mon}(\mathfrak{S}) \vdash A \overset{f=g}{\to} B$ if there is a proof of $A \overset{f=g}{\to} B$ in $\mathtt{Mon}(\mathfrak{S})$ for some A and B. We say then that the **equation** $f = g$ **is derivable** in $\mathtt{Mon}(\mathfrak{S})$.

The derivable names f are terms composed of bracketed binary ; and $\otimes$ operations, and atomic names provided by $\mathfrak{S}$. For instance, we might have $((f;g);h)$ and $(f;(g;h))$ as (so far) distinct derivable names. Next we interpret the equality relation to get rid of the bureaucracy of brackets.

Definition 8.2 ($\mathfrak{S}$-Equivalence)**.** Given $A \in (\Sigma_0)^\sigma$, we write $[A]_{\mathfrak{S}}$ for the set of all objects $B \in (\Sigma_0)^\sigma$ for which $\mathrm{id}_A = \mathrm{id}_B$ is derivable in $\mathtt{Mon}(\mathfrak{S})$. Explicitly:

$$[A]_{\mathfrak{S}} := \{B \in (\Sigma_0)^\sigma \mid \mathtt{Mon}(\mathfrak{S}) \vdash \mathrm{id}_A = \mathrm{id}_B\}$$

Similarly, given a constructed morphism f, we write $[f]_{\mathfrak{S}}$ for the set of all morphisms g for which $f = g$ is derivable in $\mathtt{Mon}(\mathfrak{S})$. Explicitly,

$$[f]_{\mathfrak{S}} := \{g \mid \mathtt{Mon}(\mathfrak{S}) \vdash f = g\}$$

Further, we write $A =_{\mathfrak{S}} B$ if $[A]_{\mathfrak{S}} = [B]_{\mathfrak{S}}$, and $f =_{\mathfrak{S}} g$ if $[f]_{\mathfrak{S}} = [g]_{\mathfrak{S}}$.

Now, we want to show that what we have defined is indeed an equivalence relation, for which we establish the following lemma.

Definition 8.3 (Names)**.** f is a **name** if it is a well-bracketed substring without the $=$ symbol appearing over an arrow in a proof.

Lemma 5. *Let f be the name on one side $=$ in the conclusion of a proof tree of $A \overset{f=g}{\to} B$. Then there exists a derivable (hence analytic) morphism $A \overset{f'}{\to} B$ such that $A \overset{f'=f}{\to} B$ is derivable. Further, f' differs from f at most up to pre- and post-composed identities in subnames.*

Proof. We can construct f' from the proof tree of the equality $f = g$ by an inductive argument on proofs. Throughout, we denote syntactic equality with $\equiv$. The equality rules induce the following cases for the final step in proving the equality $f = g$. The inductive hypothesis is the claim of the lemma.

- Base cases:
  - If the final proof step is (R), f required to be derivable, and hence $f = f$ is derivable by the (R) rule.
  - If the final proof step is a $(\omega_=)$-rule (where ω is an associator, unitor, or an inverse of either), or (Id-Tensor), f is either an associator, unitor, or identity, all of which are derivable by initial sequents for morphisms. Applying (R) satisfies the inductive hypothesis.
  - If the final proof step is (Den), f appears unchanged as a name on one side of a premiss equality, but possibly with differing domain or codomain. Without loss of generality by symmetry, we obtain f' by the transforming the (Den) rule instance as follows:

$$\text{(Den)}\ \frac{\dfrac{\vdots}{A \overset{f}{\to} B} \qquad \dfrac{\vdots}{C \overset{g}{\to} D}}{A \overset{f=g}{\to} D}$$

$$\Downarrow$$

$$\text{(;)}\ \frac{\dfrac{\vdots}{A \overset{f}{\to} B} \qquad \text{(Den-Id)}\ \dfrac{}{B \overset{\mathrm{id}_D}{\to} D}\ [\dagger]}{\text{(R)}\ \dfrac{A \overset{f;\mathrm{id}_D}{\to} D}{\text{(R.U.)}\ \dfrac{A \overset{f;\mathrm{id}_D = f;\mathrm{id}_D}{\to} D}{A \overset{f = f;\mathrm{id}_D}{\to} D}}}\ [\ddagger]$$

  The presence of the † (Den-Id) rule is guaranteed by Proposition 2 and the definition of $\mathfrak{S}$. The derivation at ‡ gives us $f' \equiv f; \mathrm{id}_D$, and the conclusion of the proof gives us a derivation of $f' = f$. A symmetric argument follows for g. The cases above are the only rules that introduce $=$, and so cover the base cases for the induction.

- Inductive cases:
  - If the final proof step is (S), (T), or (Str), f appears unchanged as a name on one side of a premiss equality, so the inductive hypothesis carries.
  - If the final proof step is (Assoc), without loss of generality, write $f \equiv (g; (h; j))$. By the inductive hypothesis, the other bracketing $((g; h); j)$ is derivable. By analyticity of morphism building, without loss of generality in typing, we know that the construction of $((g; h); j)$ terminates as below:

$$\text{(;)}\ \frac{\dfrac{\vdots}{A \overset{g}{\to} B} \qquad \dfrac{\vdots}{B \overset{h}{\to} C} \qquad \dfrac{\vdots}{C \overset{j}{\to} D}}{\text{(;)}\ \dfrac{A \overset{(g;h)}{\to} C \qquad C \overset{j}{\to} D}{A \overset{((g;h);j)}{\to} D}}$$

  Where g, h, j are derivable, so we can construct a proof of $f \equiv (g; (h; j))$ as follows:

$$(;)\ \dfrac{\dfrac{\vdots}{A \overset{g}{\to} B} \qquad (;)\ \dfrac{\dfrac{\vdots}{B \overset{h}{\to} C} \qquad \dfrac{\vdots}{C \overset{j}{\to} D}}{B \overset{(h;j)}{\to} D}}{A \overset{(g;(h;j))}{\to} D}$$

And we may derive the equality $A \overset{((g;h);j)=(g;(h;j))}{\to} D$ by (R), (Assoc), and (T). Recall that the (Assoc) rule is reversible, so the alternative bracketing case is settled symmetrically.

- If the final proof step is (Interchange), without loss of generality, write $f \equiv (g \otimes j);(h \otimes k)$. The inductive hypothesis grants that $(g;h) \otimes (j;k)$ is derivable, and the form of the (Interchange)-rule grants that g, h, j, k are each themselves derivable. $(g \otimes j);(h \otimes k)$ is evidently derivable by the latter fact, and the conjunction of the two facts gives us that $(g \otimes j);(h \otimes k) = (g;h) \otimes (j;k)$ is derivable, by joining the derivations with the (R), (Assoc), (T) strategy outlined in the previous case.
- If the final proof step is $(=^{\leftrightarrow})$, then either:
  - f appears unchanged in some premiss, in which case the inductive hypothesis carries.
  - $f \equiv (f_1 \otimes f_2)$. By the inductive hypothesis, there exists a derivable f_1' such that $f_1' = f_1$ is derivable, and symmetrically an f_2' for f_2. So, we have the following proof tree:

$$(=^{\leftrightarrow})\ \dfrac{\dfrac{\vdots(Ind.Hyp.)}{A \overset{f_1'=f}{\to} B} \qquad (\mathrm{R})\ \dfrac{(\otimes)\ \dfrac{\dfrac{\vdots(Ind.Hyp.)}{A \overset{f_1'}{\to} B} \qquad \dfrac{\vdots(Ind.Hyp.)}{C \overset{f_2'}{\to} D}}{(A \otimes B) \overset{(f_1' \otimes f_2')}{\to} (C \otimes D)}\,[\dagger]}{(A \otimes B) \overset{(f_1' \otimes f_2')=(f_1' \otimes f_2')}{\to} (C \otimes D)} \qquad \dfrac{\vdots(Ind.Hyp.)}{C \overset{f_2'=f_2}{\to} D}}{(A \otimes B) \overset{(f_1 \otimes f_2)=(f_1' \otimes f_2')}{\to} (C \otimes D)}$$

  The † step and the conclusion of the proof satisfy the inductive hypothesis.
- If the final proof step is $(=^{\updownarrow})$, the proof is very similar to the $(=^{\leftrightarrow})$ case, so we omit it.
- If the final proof step is (Left Unitality) or (Right Unitality), either f remains unchanged in the premiss, or, without loss of generality by symmetry, the inductive hypothesis grants that $\mathrm{id}; f$ is derivable. By analyticity, we know that the proof of $\mathrm{id}; f$ terminates with an (Id) and (;) rule, and that f by itself is derivable. Thus $\mathrm{id}; f = f$ is derivable maintaining codomain and domain types by appending (R), (S), and (Left Unitality) rules to a derivation of f.

We can observe by the cases constructing f' that it differs from f at most up to pre- and post-composed identities in subnames, which only occurs when a (Den) rule base case is reached. □

Proposition 6. $=_{\mathfrak{S}}$ *is an equivalence relation on derivable objects and morphisms.*

Proof. Ensured by the reflexivity, symmetry, and transitivity rules, and the definition of derivable equations.

For reflexivity of objects, since the (Id) rule draws objects from the free algebra, the following proof demonstrates that $\mathrm{id}_A = \mathrm{id}_A$ is always derivable. Reflexivity for morphisms follows in the

same way, replacing the (Id)-rule with an arbitrary derived morphism.

$$\text{(R)}\ \frac{\text{(Id)}\ \dfrac{}{A \overset{\mathrm{id}_A}{\to} A}}{A \overset{\mathrm{id}_A = \mathrm{id}_A}{\to} A}$$

For symmetry, it suffices to show that if $\mathrm{id}_A = \mathrm{id}_B$ is derivable, so is the reverse. The case for morphisms is again similar.

$$\text{(S)}\ \frac{\dfrac{\vdots}{X \overset{\mathrm{id}_A = \mathrm{id}_B}{\to} Y}}{X \overset{\mathrm{id}_B = \mathrm{id}_A}{\to} Y}$$

Transitivity requires some subtlety. The case for objects is settled by the proof strategy used in Example 16, by replacing the λs with (Den-Id)-rules. One can repeat that proof strategy to demonstrate the equality of identities for all objects in a denotationally induced equivalence class. For morphisms, in general we will have $A \overset{f=g}{\to} B$ and $C \overset{g=h}{\to} D$, and writing $\equiv$ for syntactic equality, we are not guaranteed that $A \equiv C$ and $B \equiv D$ due to the presence of (Den-Id) and (Den) rules, so we cannot apply (T) directly. Since we have settled the case for objects, replace every object name A with a symbolic representative for $[A]_{\mathfrak{S}}$ denoted $A^{@}$ across the proofs of $A \overset{f=g}{\to} B$ and $C \overset{g=h}{\to} D$. Observe that by choosing representatives for equivalence classes, (Den-Id) is indistinguishable from (Id), and (Den) is indistinguishable from (T). In this setting, Lemma 5 applies, so we granted some derivable $X^{@} \overset{g'}{\to} Y^{@}$ such that $X^{@} \overset{g'=g}{\to} Y^{@}$, and by applying the lemma again for the f and h, we must have $A^{@} = X^{@} = C^{@}$ and $B^{@} = Y^{@} = D^{@}$ (*i.e.*, that $A =_{\mathfrak{S}} C$ and $B =_{\mathfrak{S}} D$), and hence that the (T) rule can be used to prove transitive closure for morphisms. Note that this also proves that taking equivalence classes under objects and morphisms at once preserves well-typedness of domains and codomains of equivalence classes of morphisms.

□

9 The Monoidal Category $\mathcal{M}(\mathfrak{S})$

With these notions in hand, we can define a term model. As remarked at the close of the proposition above, we already have that the codomains and domains of equivalence classes of morphisms are well-defined.

Definition 9.1 (The strict monoidal category $\mathcal{M}(\mathfrak{S})$)**.** Given $\mathfrak{S}$, with prestructure Σ, objects D, types $\mathbf{T}$, named morphisms **Name**, and equations **Den** and **Str**, we define the category $\mathcal{M}(\mathfrak{S})$ as follows:

- The objects of $\mathcal{M}(\mathfrak{S})$ are the equivalence classes $(\Sigma_0)^{\sigma}$ under $=_{\mathfrak{S}}$
- Given objects $[A]_{\mathfrak{S}}, [B]_{\mathfrak{S}}$, the morphisms $\mathcal{M}(\mathfrak{S})\big[[A]_{\mathfrak{S}}, [B]_{\mathfrak{S}}\big]$ are equivalence classes of derivable morphisms with matched domain and codomain. Explicitly:

$$\mathcal{M}(\mathfrak{S})\big[[A]_{\mathfrak{S}}, [B]_{\mathfrak{S}}\big] := \{[f]_{\mathfrak{S}} \mid \mathtt{Mon}(\mathfrak{S}) \vdash C \overset{f}{\to} D \text{ and } A =_{\mathfrak{S}} C, B =_{\mathfrak{S}} D\}$$

- The identity morphism on each object $[A]_{\mathfrak{S}}$ is $[\mathrm{id}_A]_{\mathfrak{S}}$

- For morphisms $[f]_{\mathfrak{S}} : [A]_{\mathfrak{S}} \to [B]_{\mathfrak{S}}$ and $[g]_{\mathfrak{S}} : [B]_{\mathfrak{S}} \to [C]_{\mathfrak{S}}$, composition is defined as:
$$[f]_{\mathfrak{S}}; [g]_{\mathfrak{S}} := [(f; g)]_{\mathfrak{S}}$$
- The Monoidal Unit is $[I]_{\mathfrak{S}}$
- The Tensor product functor $\otimes_{\mathcal{M}(\mathfrak{S})} : \mathcal{M}(\mathfrak{S}) \times \mathcal{M}(\mathfrak{S}) \to \mathcal{M}(\mathfrak{S})$ is defined...
  - ...on objects as: $[A]_{\mathfrak{S}} \otimes_{\mathcal{M}(\mathfrak{S})} [B]_{\mathfrak{S}} := [(A \otimes B)]_{\mathfrak{S}}$
  - ...on morphisms as: $[f]_{\mathfrak{S}} \otimes_{\mathcal{M}(\mathfrak{S})} [g]_{\mathfrak{S}} := [(f \otimes g)]_{\mathfrak{S}}$
- The associator and unitor natural isomorphisms are all identities.

Proposition 7. $\mathcal{M}(\mathfrak{S})$ *is a category*

Proof. We need to check unitality and associativity. Conveniently, we have rules in place for that.

Unitality: For any derivable morphism f, we can construct the following proof for left unitality.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \overset{\mathrm{id}_A}{\to} A} (\mathrm{Id}) \quad \dfrac{\vdots}{A \overset{f}{\to} B}}{A \overset{\mathrm{id}_A; f}{\to} B} (;)}{A \overset{\mathrm{id}_A; f = \mathrm{id}_A; f}{\to} B} (\mathrm{R})}{A \overset{f = \mathrm{id}_A; f}{\to} B} (\text{Left Unitality})$$

So so we have, by definition of composition, $[\mathrm{id}_A]_{\mathfrak{S}}; [f]_{\mathfrak{S}} = [\mathrm{id}_A; f]_{\mathfrak{S}}$, and we have just shown $[\mathrm{id}_A; f]_{\mathfrak{S}} = [f]_{\mathfrak{S}}$, so left unitality holds. A symmetric argument applies for right unitality.

Associativity: While we were a little lax with bracketing for the previous exercise, this one is nothing but syntax, so we'll be a little more careful. For well-typed derivable morphisms f, g, h, we have the following proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{A \overset{f}{\to} B} \quad \dfrac{\vdots}{B \overset{g}{\to} C}}{A \overset{(f;g)}{\to} C} (;) \quad \dfrac{\vdots}{C \overset{h}{\to} D}}{A \overset{((f;g);h)}{\to} D} (;)}{A \overset{((f;g);h) = ((f;g);h)}{\to} D} (\mathrm{R})}{A \overset{(f;(g;h)) = ((f;g);h)}{\to} D} (\text{Assoc})$$

So, we have:

$$\begin{aligned}
&([f]_{\mathfrak{S}}; [g]_{\mathfrak{S}}); [h]_{\mathfrak{S}} \\
&= [(f; g)]_{\mathfrak{S}}; [h]_{\mathfrak{S}} && \text{Composition} \\
&= [((f; g); h)]_{\mathfrak{S}} && \text{Composition} \\
&= [(f; (g; h))]_{\mathfrak{S}} && \text{Proof Tree} \\
&= [f]_{\mathfrak{S}}; [(g; h)]_{\mathfrak{S}} && \text{Composition} \\
&= [f]_{\mathfrak{S}}; ([g]_{\mathfrak{S}}; [h]_{\mathfrak{S}}) && \text{Composition}
\end{aligned}$$

As required.

□

Let's be sure that the tensor product we have defined really is a bifunctor, and not a pun. Though, since we are building a category out of syntax, we are literally playing with words.

Proposition 8. $\otimes : \mathcal{M}(\mathfrak{S}) \times \mathcal{M}(\mathfrak{S}) \to \mathcal{M}(\mathfrak{S})$ *is a bifunctor.*

Proof. Recall that the product category $\mathcal{M}(\mathfrak{S}) \times \mathcal{M}(\mathfrak{S})$ consists of:

- Objects pairs $([A]_{\mathfrak{S}}, [B]_{\mathfrak{S}})$ of objects $[A]_{\mathfrak{S}}, [B]_{\mathfrak{S}} \in \mathcal{M}(\mathfrak{S})$
- Morphisms between $([A]_{\mathfrak{S}}, [B]_{\mathfrak{S}}), ([C]_{\mathfrak{S}}, [D]_{\mathfrak{S}})$ pairs $([f]_{\mathfrak{S}}, [g]_{\mathfrak{S}})$ of morphisms $[A]_{\mathfrak{S}} \overset{[f]_{\mathfrak{S}}}{\to} [C]_{\mathfrak{S}}, [B]_{\mathfrak{S}} \overset{[g]_{\mathfrak{S}}}{\to} [D]_{\mathfrak{S}}$ of $\mathcal{M}(\mathfrak{S})$. For the following, we'll drop the nesting $[_]_{\mathfrak{S}}$s.
- Identities as pairs of identities: $\mathrm{id}_{(A,B)} = (\mathrm{id}_A, \mathrm{id}_B)$
- Composition defined componentwise: $(f, g); (f', g') := (f; f', g; g')$

Now, we need to check that our definition preserves composition and identities. We will write the functor $\otimes$ outfix, to distinguish it from the syntactic $\otimes$

Identities: For a pair of objects (A, B) of $\mathcal{M}(\mathfrak{S})$, we aim to show that $\otimes(\mathrm{id}_{(A,B)}) = [\mathrm{id}_{\otimes((A,B))}]_{\mathfrak{S}} = [\mathrm{id}_{AB}]_{\mathfrak{S}}$. We have:

$$
\begin{aligned}
&\otimes\,(\mathrm{id}_{(A,B)}) \\
&= \otimes([\mathrm{id}_A]_{\mathfrak{S}}, [\mathrm{id}_B]_{\mathfrak{S}}) && \text{Product Category Identity} \\
&= [\mathrm{id}_A \otimes \mathrm{id}_B]_{\mathfrak{S}} && \text{Functor} \\
&= [\mathrm{id}_{(A \otimes B)}]_{\mathfrak{S}}
\end{aligned}
$$

Where the final equality follows from the proof tree below:

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{A \overset{\mathrm{id}_A}{\to} A}(\mathrm{Id}) \quad \dfrac{}{B \overset{\mathrm{id}_B}{\to} B}(\mathrm{Id})}{(A \otimes B) \overset{\mathrm{id}_A \otimes \mathrm{id}_B}{\to} (A \otimes B)}(\otimes)}{(A \otimes B) \overset{\mathrm{id}_A \otimes \mathrm{id}_B = \mathrm{id}_A \otimes \mathrm{id}_B}{\to} (A \otimes B)}(\mathrm{R})}{(A \otimes B) \overset{\mathrm{id}_{(A \otimes B)} = \mathrm{id}_A \otimes \mathrm{id}_B}{\to} (A \otimes B)}(\text{Id-Unit})
$$

Composition: We need to show that $\otimes(([f]_{\mathfrak{S}}, [h]_{\mathfrak{S}}); ([g]_{\mathfrak{S}}, [j]_{\mathfrak{S}})) = \otimes(([f]_{\mathfrak{S}}, [h]_{\mathfrak{S}})); \otimes(([g]_{\mathfrak{S}}, [j]_{\mathfrak{S}}))$. Note that the composition in the left is that within the product category, and that composition in the right is within the plain $\mathcal{M}(\mathfrak{S})$. We have, as desired:

$$
\begin{aligned}
&\otimes\,(([f]_{\mathfrak{S}}, [h]_{\mathfrak{S}}); ([g]_{\mathfrak{S}}, [j]_{\mathfrak{S}})) \\
&= \otimes(([f]_{\mathfrak{S}}; [g]_{\mathfrak{S}}, [h]_{\mathfrak{S}}; [j]_{\mathfrak{S}})) && \text{Product Composition} \\
&= \otimes(([(f; g)]_{\mathfrak{S}}, [(h; j)]_{\mathfrak{S}})) && \text{Composition in } \mathcal{M}(\mathfrak{S}) \\
&= [(f; g) \otimes (h; j)]_{\mathfrak{S}} && \text{Functor} \\
&= [(f \otimes h); (g \otimes j)]_{\mathfrak{S}} && \text{Proof Tree} \\
&= [(f \otimes h)]_{\mathfrak{S}}; [(g \otimes j)]_{\mathfrak{S}} && \text{Composition in } \mathcal{M}(\mathfrak{S}) \\
&= \otimes(([f]_{\mathfrak{S}}, [h]_{\mathfrak{S}})); \otimes(([g]_{\mathfrak{S}}, [j]_{\mathfrak{S}})) && \text{Functor definition}
\end{aligned}
$$

Where the proof tree in question is as follows. We omit the typing of the morphisms in the wings for display, and we rely on the fact that if a morphism is derivable, it is multiply so.

$$\text{(Interchange)}\ \dfrac{\dfrac{\vdots}{f}\quad\dfrac{\vdots}{g}\qquad \text{(R)}\ \dfrac{(\text{;})\ \dfrac{\dfrac{\vdots}{A \overset{f}{\to} X}\quad\dfrac{\vdots}{X \overset{g}{\to} C}\qquad (\text{;})\ \dfrac{\dfrac{\vdots}{B \overset{h}{\to} Y}\quad\dfrac{\vdots}{Y \overset{j}{\to} D}}{(\otimes)\ \dfrac{A \overset{f;g}{\to} C \qquad\qquad Y \overset{h;j}{\to} D}{(A \otimes B) \overset{(f;g)\otimes(h;j)}{\to} (C \otimes D)}}}{(A \otimes B) \overset{(f;g)\otimes(h;j)=(f;g)\otimes(h;j)}{\to} (C \otimes D)}\qquad\dfrac{\vdots}{h}\quad\dfrac{\vdots}{j}}{(A \otimes B) \overset{(f\otimes h);(g\otimes j)=(f;g)\otimes(h;j)}{\to} (C \otimes D)}$$

□

Though it looks obvious, we may as well prove that this category is monoidal, and in fact, strict. Our workload is halved, as we have made all the associators and unitors identities, so we already have coherence for objects, as each object is an equivalence class that contains all possible $\otimes$ bracketings. However, the morphisms we are constructing have not escaped the bureaucracy of brackets, so we still have to demonstrate that different bracketings in constructing morphisms with $\otimes$ are equal.

Proposition 9. *$\mathcal{M}(\mathfrak{S})$ is a strict monoidal category.*

Proof. As mentioned, we don't need to show coherence for objects, so as shorthand, we will write $A \otimes B$ as the concatenation AB, and we will treat the monoidal unit as the empty string ϵ. By MacLane's Theorem, for strictness and monoidal structure in one go, we need only demonstrate that $(f \otimes g) \otimes h =_{\mathfrak{S}} f \otimes (g \otimes h)$ and $\mathbf{1} \otimes f =_{\mathfrak{S}} f =_{\mathfrak{S}} f \otimes \mathbf{1}$, where f, g, h are arbitrary well-typed derivable terms.

For the associator equation, we use the following proof tree, omitting precomposition with the associator α, as it is an identity.

$$\dfrac{(\alpha)\ \dfrac{(\text{R})\ \dfrac{(\otimes)\ \dfrac{(\otimes)\ \dfrac{\dfrac{\vdots}{A \overset{f}{\to} B}\quad\dfrac{\vdots}{C \overset{g}{\to} D}}{AC \overset{(f\otimes g)}{\to} BD}\qquad\dfrac{\vdots}{E \overset{h}{\to} F}}{ACE \overset{(f\otimes g)\otimes h}{\to} BDF}}{ACE \overset{(f\otimes g)\otimes h=(f\otimes g)\otimes h}{\to} BDF}}{ACE \overset{f\otimes(g\otimes h)=(f\otimes g)\otimes h}{\to} BDF}$$

Since we have established the bifunctoriality of $\otimes$, we will treat ourselves to the infix notation for the following. Using the proof tree above, we have:

$$\begin{aligned}
&[f]_{\mathfrak{S}} \otimes [g]_{\mathfrak{S}}) \otimes [h]_{\mathfrak{S}} \\
&= [(f \otimes g) \otimes h]_{\mathfrak{S}} && \text{Functor} \\
&= [f \otimes (g \otimes h)]_{\mathfrak{S}} && \text{Proof Tree} \\
&= [f]_{\mathfrak{S}} \otimes ([g]_{\mathfrak{S}}) \otimes [h]_{\mathfrak{S}}) && \text{Functor}
\end{aligned}$$

For the unitor equations, we just show one case, as the other follows symmetrically, and we will reason informally. By (Id-Tensor), $[\mathbf{1} \otimes \mathrm{id}_A]_{\mathfrak{S}} = [\mathrm{id}_{I \otimes A}]$, and we have that $A =_{\mathfrak{S}} I \otimes A$, so:

$$
\begin{array}{ll}
\mathbf{1} \otimes [f]_{\mathfrak{S}} & \\
= [\mathbf{1} \otimes f]_{\mathfrak{S}} & \text{Functor} \\
= [f]_{\mathfrak{S}} & \text{By precomposition with identity and unitality}
\end{array}
$$

□

Now we have a nice theorem relating $\texttt{Mon}(\mathfrak{S})$ and $\mathcal{M}(\mathfrak{S})$.

Theorem 10 ($\texttt{Mon}(\mathfrak{S})$ is sound and complete for $\mathcal{M}(\mathfrak{S})$)**.** *For objects or morphisms* Φ, Ψ

$$[\Phi]_{\mathfrak{S}} = [\Psi]_{\mathfrak{S}} \textit{ in } \mathcal{M}(\mathfrak{S}) \iff \Phi =_{\mathfrak{S}} \Psi$$

Proof. By definitions of $\mathcal{M}(\mathfrak{S})$ and $=_{\mathfrak{S}}$. □

10 Interpretations of $\mathfrak{S}$ in Monoidal Categories

We perform a comparative reading of the definition of general monoidal signatures with that of monoidal interpretations (Definition 7.22), and that of free monoidal categories over a signature (Definition 10) given earlier from [Sel10, p. 12], which is itself a non-strict paraphrase of the original given by Joyal and Street. We recall the definitions for convenience below.

Definition (General Monoidal Signatures)**.** Define the associated **general monoidal signature** $\mathfrak{S}$ to have data:

- the algebraic signature σ, containing at least nullary I and binary $\otimes$
- object variables Σ_0
- morphism variables Σ_1, with accompanying domain and codomain data
- structural morphisms $\star$, freely instantiated over $(\Sigma_0)^\sigma$
- an equivalence relation $=_{\mathbf{Den+Str}}$ defined on morphisms in the free monoidal category over Σ plus all components of the natural transformations $\star$ instantiated freely over $(\Sigma_0)^\sigma$, which is the reflexive, symmetric, and transitive closure of the two equivalence relations **Den** (obtained from an interpretation functor) and **Str** (obtained by graphical equations)

Definition (Interpretations of Monoidal Signatures)**.** Given a monoidal signature Σ (recall definition 7.12) and a monoidal category $\mathcal{K}$, an **interpretation** $[\![\_]\!] : \Sigma \mapsto \mathcal{K}$ consists of:

- An object function $[\![\_]\!]_0 : D^\Sigma (= D) \to \mathrm{Ob}(\mathcal{K})$, which extends in a unique way to an object function $[\![\_]\!]_0^* : \mathbf{T} \to \mathrm{Ob}(\mathcal{K})$, such that:
  - $[\![aB]\!]_0^* = [\![a]\!]_0 \otimes_{\mathcal{K}} [\![B]\!]_0^*$; where $a \in D$
  - $[\![\epsilon]\!]_0^* = I_{\mathcal{K}}$
- A morphism function $[\![\_]\!]_1$, that maps $f \in \mathbf{Name}$ to $[\![f]\!]_1 : [\![\mathrm{dom}\ f]\!]_0 \to [\![\mathrm{cod}\ f]\!]_0$

Definition (Free monoidal categories)**.** A monoidal category $\mathcal{K}$ is a **free monoidal category** over a monoidal signature Σ if it is equipped with an interpretation $[\![\_]\!]_0 : \Sigma \mapsto \mathcal{K}$ such that for any monoidal category $\mathcal{J}$ and interpretation $[\![\_]\!] : \Sigma \mapsto \mathcal{J}$, there exists a strong monoidal functor $F : \mathcal{K} \to \mathcal{J}$, such that $[\![\_]\!] = [\![\_]\!]_0; F$, and F is unique up to a unique monoidal natural isomorphism.

The first observation is that if a general monoidal signature $\mathfrak{S}$ is only provided a monoidal signature Σ, $\mathcal{M}(\mathfrak{S})$ is the free monoidal category over Σ.

Proposition 11. *If $\mathfrak{S}$ and Σ express the same data, $\mathcal{M}(\mathfrak{S})$ is the free monoidal category over Σ.*

Proof. Equip $\mathcal{M}(\mathfrak{S})$ with the interpretation $[\![\_]\!]_0 : \Sigma \mapsto \mathcal{M}(\mathfrak{S})$, which we specify as follows:

- $[\![\_]\!]_0$ embeds object variables $a \in \Sigma_0$ as $a \in (\Sigma_0)^\sigma = \mathrm{Ob}(\mathcal{M}(\mathfrak{S}))$. This embedding extends uniquely to an embedding (the identity) from $(\Sigma_0)^\sigma$ in $\mathrm{Ob}(\mathcal{M}(\mathfrak{S}))$
- $[\![\_]\!]_0$ maps $f \in \Sigma_1$ to $[f]_{\mathfrak{S}} \in \mathrm{Mor}(\mathcal{M}(\mathfrak{S}))[[\![\mathrm{cod}/f]\!]_0, [\![\mathrm{dom}\ f]\!]_0]$

Observe that if $\mathfrak{S}$ is only provided Σ, in the form of object variables Σ_0 and typed morphism variables Σ_1, the derivable equations in $\mathtt{Mon}(\mathfrak{S})$ are restricted to those of categorical and strict monoidal structure, as there is no data for (Den-Id), ($\star$), (Den), and (Str) rules. Thus $[\![\_]\!]_0$ is the identity map on $(\Sigma_0)^\sigma$, so for any other object map $[\![\_]\!] : (\Sigma_0)^\sigma \to \mathrm{Ob}(\mathcal{J})$ obtained by an interpretation $[\![\_]\!] : \Sigma \mapsto \mathcal{J}$ for some strict monoidal category $\mathcal{J}$, we may write the factorisation $[\![\_]\!] = [\![\_]\!]_0; [\![\_]\!]$. The same holds for morphism variable maps.

What remains to be shown is that the data in an arbitrary interpretation $[\![\_]\!] : \Sigma \mapsto \mathcal{J}$ corresponds to a strict monoidal functor $F : \mathcal{M}(\mathfrak{S}) \to \mathcal{J}$, and that this functor is unique up to unique monoidal isomorphism.

For the first task, recall that a functor is defined as a pair of functions, from the objects of the source category to the objects of the target category, and ditto for the morphisms. For objects $X \in \mathtt{Mon}(\mathfrak{S})$, which we write as strings in Σ_0, define a function F_{ob} from $(\Sigma_0)^\sigma$ into objects of the target category $\mathcal{J}$ recursively as follows:

$$F_{\mathrm{ob}}(X) := \begin{cases} I_{\mathcal{J}} & \text{if } X =_{\mathfrak{S}} I \in (\Sigma_0)^\sigma \quad \langle\dagger\rangle \\ [\![x]\!] & \text{if } X = a \in (\Sigma_0)^\sigma \\ F(A) \otimes_{\mathcal{J}} F(B) & \text{if } X = (A \otimes B) \in (\Sigma_0)^\sigma \end{cases}$$

Since $\mathcal{M}(\mathfrak{S})$ and $\mathcal{J}$ are both strict, and the objects of $\mathcal{M}(\mathfrak{S})$ are equivalence classes from $(\Sigma_0)^\sigma$ up to bracketing by design, by coherence we have a function from $\mathrm{Ob}(\mathcal{M}(\mathfrak{S}))$ to $\mathrm{Ob}(\mathcal{J})$.

For morphisms, define the following function from $\mathrm{Mor}(\mathcal{M}(\mathfrak{S}))$ into $\mathrm{Mor}(\mathcal{J})$.

$$F_{\mathrm{mor}}([f]_{\mathfrak{S}}) := \begin{cases} \mathrm{id}_{F_{\mathrm{ob}}(X)} & \text{if } f = [\mathrm{id}_X]_{\mathfrak{S}} \\ [\![f]\!] & \text{if } f \in \Sigma_1 \quad \langle\ddagger\rangle \\ F(g);_{\mathcal{J}} F(h) & \text{if } f = [g; h]_{\mathfrak{S}} \quad \langle\ddagger\ddagger\rangle \\ F(g) \otimes_{\mathcal{J}} F(h) & \text{if } f = [g \otimes h]_{\mathfrak{S}} \quad \langle\dagger\dagger\rangle \end{cases}$$

Note that the $[\mathrm{id}_X]_{\mathfrak{S}}$ identify all well-formed bracketings, and by strictness of the two categories, it is a well-defined map. By design of $\mathcal{M}(\mathfrak{S})$, one can pick any derivable morphisms f

and g to begin the recursive translation above, and by soundness and completeness of $\mathtt{Mon}(\mathfrak{S})$, $f =_{\mathfrak{S}} g \implies [f]_{\mathfrak{S}} = [g]_{\mathfrak{S}} \implies F([f]_{\mathfrak{S}}) = F([g]_{\mathfrak{S}})$, so we have a well defined function for morphisms.

The marked ‡ and ‡‡ clauses together guarantee that the pair of functions we have defined are a functor. The † and †† clauses implement the coherence maps that make this a monoidal functor. The † coherence is one-to-one, and since $[g]_{\mathfrak{S}} \otimes_{\mathcal{M}(\mathfrak{S})} [h]_{\mathfrak{S}} := [g \otimes h]_{\mathfrak{S}}$, the †† coherence is also invertible, hence we have a strong monoidal functor. Also since $[g]_{\mathfrak{S}} \otimes_{\mathcal{M}(\mathfrak{S})} [h]_{\mathfrak{S}} := [g \otimes h]_{\mathfrak{S}}$, we have actually defined F as the componentwise action of a monoidal natural transformation from the identity functor on $\mathcal{M}(\mathfrak{S})$, so we have that F is unique up to unique natural isomorphism. □

Now, recall Theorem 4.

Theorem. *The graphical language of monoidal categories over a monoidal signature Σ, up to planar isotopy of diagrams, forms a free monoidal category over Σ.*

By the definition of free monoidal categories over Σ as initial objects, we are immediately granted the following.

Proposition 12 ($\mathcal{M}(\Sigma)$ is the graphical language over Σ)**.** *If $\mathfrak{S}$ and Σ express the same data, $\mathcal{M}(\mathfrak{S})$ is monoidally equivalent to the graphical language of monoidal categories over Σ*

Proof. By proposition 11, Theorem 4, and definition of initiality. □

So, by freeness *qua* initiality, when we have a monoidal signature to provide a basic stock of named objects and morphisms, an arbitrary monoidal category over this signature, such as nameable portion of **Set**, is obtained by a functor from the category of diagrams in the graphical calculus. As we have elucidated, the role of the functor is to enforce denotational equations: so functors provide equations between diagrams in the graphical calculus to reflect calculation within particular monoidal categories.

Proposition 13. *Any monoidal category $\mathcal{K}$ over a monoidal signature Σ is monoidally equivalent to $\mathcal{M}(\mathfrak{S})$ for $\mathfrak{S}$ constructed from the data Σ and the unique (up to unique natural isomorphism) strong monoidal functor K from the free monoidal category over Σ to $\mathcal{K}$.*

Proof. Let $\mathcal{M}(\Sigma)$ denote the free monoidal category over Σ, obtained by giving the general monoidal signature only the data from Σ. By initiality, there exists a strong monoidal functor from $\mathcal{M}(\Sigma)$ to $\mathcal{K}$, unique up to unique natural isomorphism. Call this functor K. Construct $\mathfrak{S}$ out of the data of Σ and K, recalling the definition of general monoidal signatures. The objects and morphisms of $\mathcal{M}(\mathfrak{S})$ are precisely the equivalence classes on objects and morphisms in $\mathcal{M}(\Sigma)$ induced by K[17] so we have a monoidal equivalence in the obvious way. Further observe that applying natural isomorphisms to K does not change the construction of $\mathfrak{S}$ in this way. □

The above proposition extends to any reasonable conception of monoidal categories that satisfy graphical equations, as we can always take the forgetful inclusion functor from any reasonably conceived 'category of strict monoidal categories that satisfy additional graphical equations' into **Mon**. In the next sections, we expand upon structure **S** induced by graphical equations, and construct a reasonable category **SMon**.

[17] Another way to think about it is that we have an epi-mono factorisation of the object and morphism maps of K through $\mathcal{M}(\mathfrak{S})$

11 Denotational Capture, Structural Augmentation, Free graphical completion

11.0.1 Examples of Structure

Recall that a general monoidal structure $\mathfrak{S}$ carries a stock of what we have called 'structural data' in the data **Str** and $\star$. In the logic $\texttt{Mon}(\mathfrak{S})$, this is reflected in eponymous rules:

$$(\star)\ \frac{}{\mathrm{cod}(\omega(\vec{a})) \overset{\omega_a}{\to} \mathrm{dom}(\omega(\vec{a}))}\ [\text{For all } \omega \text{ in } \star] \qquad (\mathrm{Str})\ \frac{A \overset{f}{\to} B \qquad A \overset{g}{\to} B}{A \overset{f=g}{\to} B}\ [\text{For all } f =_{\mathbf{Str}} g]$$

The form of the rules above is perhaps too abstract to glean intuition, so we provide a stock of examples. The underlying idea is very simple: $(\star)$ implements natural transformations, and (Str) enforces equalities that hold between them.

Example 18 (Twists)**.** Recall that a braided monoidal category is braided in virtue of a *twist* natural isomorphism $\theta_{AB} : A \otimes B \to B \otimes A$, which must satisfy two 'hexagon axioms', shown in Figure 46. The equations are captured diagrammatically in Figure 49[Sel10, p. 14-15].

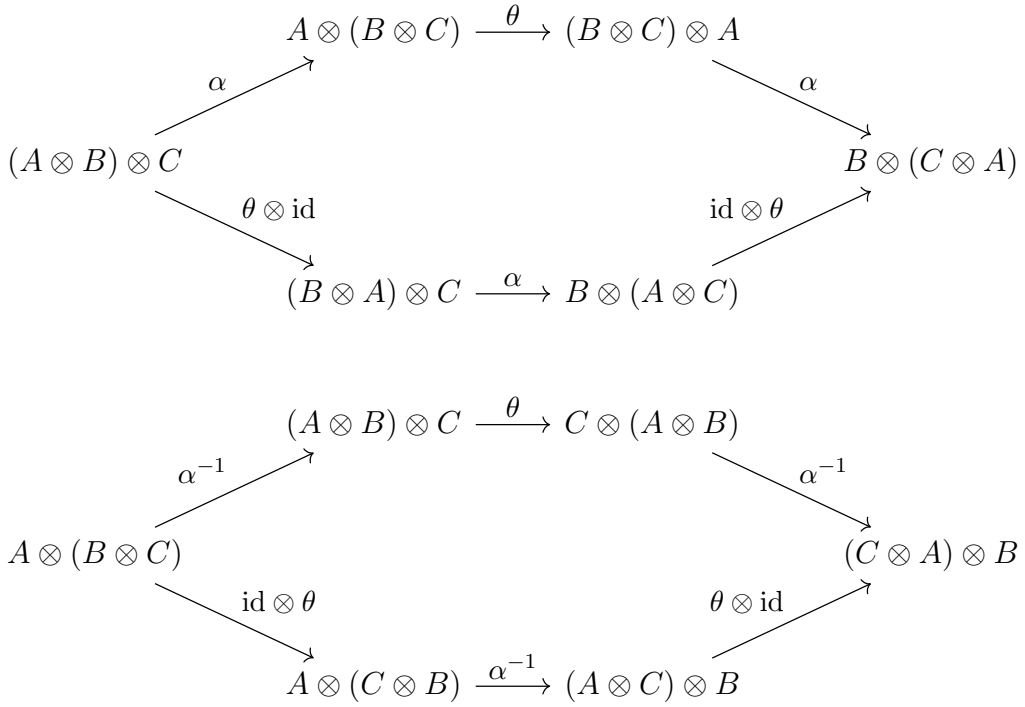


Figure 46: The hexagon axioms

Figure 47: The braiding morphism is drawn as a wire crossing under another.

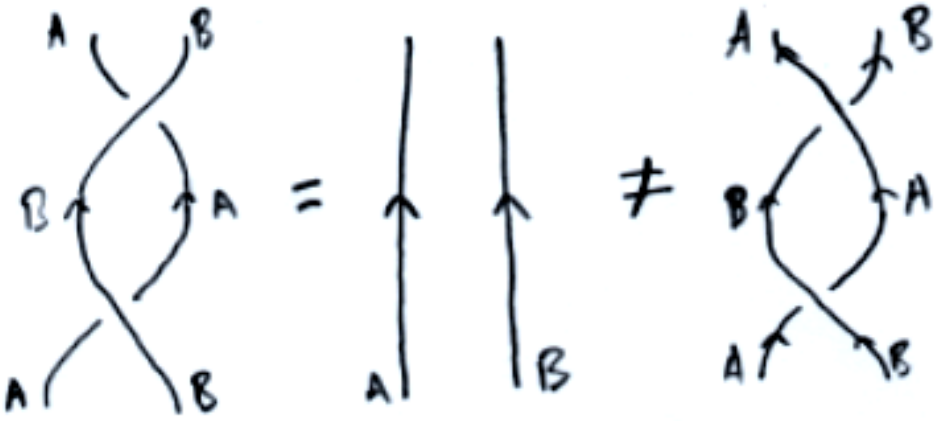


Figure 48: The fact that it's a natural isomorphism can be expressed by the equation between the first two diagrams, where we draw the inverse of braiding as the right wire going under the left. Note that we can't derive the equation between the second and third.

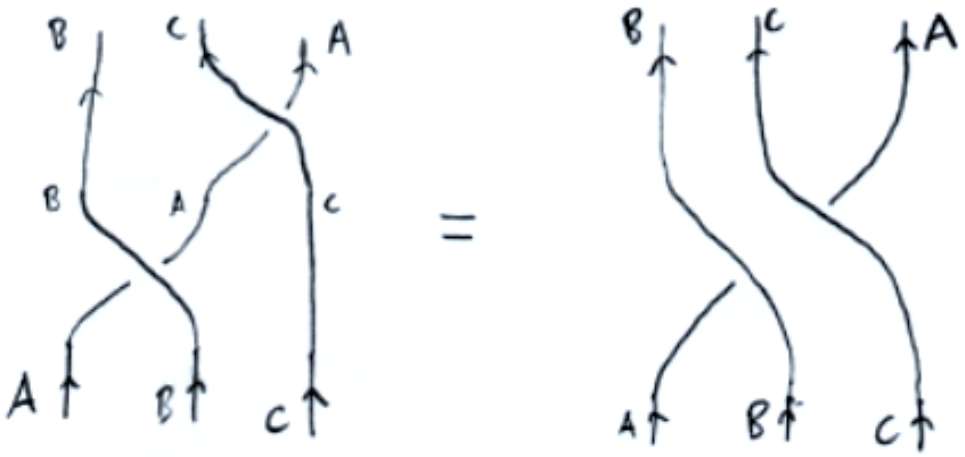


Figure 49: The hexagon axioms, diagrammatically. The big gap in the second is due to the braiding occuring between A and $(B \otimes C)$.

We can equivalently capture these conditions in a suitably augmented $\mathfrak{S}$, which induces the following rules in $\mathtt{Mon}(\mathfrak{S})$. For the final rule, all morphisms are typed $(A \otimes B) \otimes C \to B \otimes (C \otimes A)$.

$$(\theta)\ \frac{}{(a \otimes b) \overset{\theta_{ab}}{\to} (b \otimes a)}$$

$$(\text{Braid-Iso})\ \frac{\theta_{ab}; \theta_{ba} \qquad \mathrm{id}_{(a \otimes b)}}{\theta_{ab}; \theta_{ba} = \mathrm{id}_{(a \otimes b)}}$$

$$(\text{Hex.})\ \frac{(\theta_{A,B} \otimes \mathrm{id}_C); \alpha_{A,B,C}; (\mathrm{id}_B \otimes \theta_{A,C}) \qquad \alpha_{A,B,C}; \theta_{B,(C \otimes A)}; \alpha_{B,C,A}}{(\theta_{A,B} \otimes \mathrm{id}_C); \alpha_{A,B,C}; (\mathrm{id}_B \otimes \theta_{A,C}) = \alpha_{A,B,C}; \theta_{B,(C \otimes A)}; \alpha_{B,C,A}}$$

Both rules draw a, b from the free algebra $(\Sigma_0)^\sigma$, and in this way simulate naturality *qua* family of morphisms parameterised by all objects present in a category. We have lapsed on the arrow-overset notation for the latter for brevity. The (Non-Braid) rule 'eliminates' occurrences of $\theta_{AB}; \theta_{BA}$ by establishing their equality with identities. Not that (Str) *must* factorise identities, but it may. Note that there is no rule that 'eliminates' $\theta_{AB}; \theta_{AB}$, which actually braids wires, so properly braided morphisms cannot be equated with identity wires. Note that since we are dealing with strict categories, we have actually implemented the hexagon axioms via this equality, by totally capturing the graphical equation associated with braiding,

Example 19 ("Weakening")**.** (To see that this actually provides the structure of weakening in a logic, see the appendix that proves the admissibility of PRO-composition in $\mathtt{Mon}(\mathfrak{S})$)

$$(\Delta)\ \frac{}{a \overset{\Delta_a}{\to} (a \otimes a)}$$

$$(\text{Copy})\ \frac{f; \Delta_{\mathrm{dom}(f)} \qquad \Delta_{\mathrm{cod}(f)}; (f \otimes f)}{f; \Delta_{\mathrm{dom}(f)} = \Delta_{\mathrm{cod}(f)}; (f \otimes f)}$$

$$(\text{CoComm})\ \frac{(\Delta_a; (\mathrm{id}_a \otimes \Delta_a)) \qquad (\Delta_a; (\Delta_a \otimes \mathrm{id}_a))}{(\Delta_a; (\mathrm{id}_a \otimes \Delta_a)) = (\Delta_a; (\Delta_a \otimes \mathrm{id}_a))}$$

Example 20 ("Contraction")**.**

$$(\text{Discard})\ \frac{}{a \overset{\mathbf{d}_a}{\to} I} \qquad\qquad (\text{Causality})\ \frac{f; \mathbf{d}_{\mathrm{dom}(f)} \qquad \mathbf{d}_{\mathrm{cod}(f)}}{f; \mathbf{d}_{\mathrm{dom}(f)} = \mathbf{d}_{\mathrm{cod}(f)}}$$

$$(\text{Left Counitality})\ \frac{A \overset{(\Delta_A; (\mathbf{d}_A \otimes \mathrm{id}_A))}{\to} A \qquad A \overset{\mathrm{id}_A}{\to} A}{A \overset{(\Delta_A; (\mathbf{d}_A \otimes \mathrm{id}_A)) = \mathrm{id}_A}{\to} A}$$

Right counitality is symmetric.

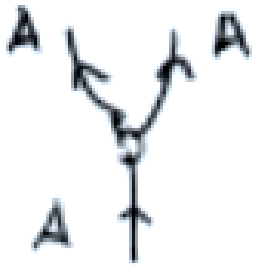


Figure 50: Graphically, we'll represent the copy operation like so.

Figure 51: The (Copy) rule stipulates that all derivable morphisms are homomorphisms with respect to the Δ structure.

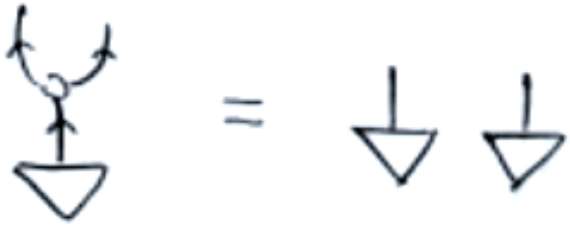


Figure 52: Observe the nice fact that $\Delta_I = \mathbf{1}$ is derivable by use of the equational rules enforcing strictness, so we can copy states leaving no trace of the structural morphism Δ.

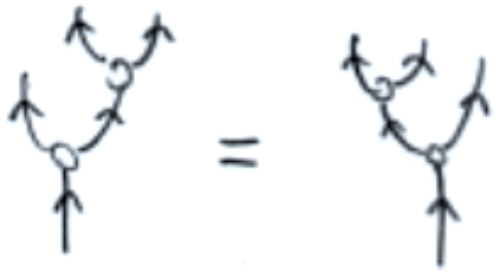


Figure 53: However, the (Copy) rule by itself leaves $\Delta_A; (\mathrm{id}_A \otimes \Delta_A)$ and $\Delta_A; (\Delta_A \otimes \mathrm{id}_A)$ in distinct equivalence classes of morphisms: while it is extensionally the same thing to make three copies by first making one copy, and then either copying on the left or right, the two are procedurally different. We can explicitly encode cocommutativity as follows above, but it's not strictly necessary.

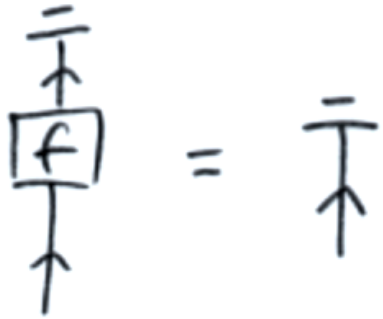


Figure 54: (**d**) refers to 'discard', and (Causality) to a similar global-homomorphism condition as in the case of (Copy) – *cf.* causal categories and processes [KU17] [BC17, §6]. We depict the discard morphism as on the right diagram.

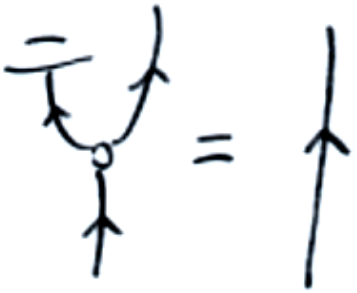


Figure 55: We can also specify counitality with respect to copy, if we'd like.

Remark 4. We put Weakening and Contraction in quotation marks, because we are working in the opposite direction to the regular approach to categorial logic [AT10, p. 74], and this makes sense: instead of using proof rules *backwards* to reduce the task of proving a sequent to various subtasks, we are using the proof rules *forwards* to construct morphisms.

Example 21 (Autonomous structure)**.** We have already introduced the graphical calculus of autonomous monoidal categories in the first part. The cup and cap we can capture with the following rules.

$$(\eta^L)\ \frac{}{I \overset{\eta_a^L}{\to} (a^L \otimes a)} \qquad\qquad (\epsilon^L)\ \frac{}{(a \otimes a^L) \overset{\epsilon_a^L}{\to} I}$$

The yanking equations we can capture as follows:

$$(\text{Yank})\ \frac{(\eta_a^L \otimes \mathrm{id}_{a^L}); (\mathrm{id}_{a^L} \otimes \epsilon^L) \qquad \mathrm{id}_{a^L}}{(\eta_a^L \otimes \mathrm{id}_{a^L}); (\mathrm{id}_{a^L} \otimes \epsilon^L) = \mathrm{id}_{a^L}} \qquad\qquad \frac{(\mathrm{id}_a \otimes \eta_a^L); (\epsilon^L \otimes \mathrm{id}_a) \qquad \mathrm{id}_a}{(\mathrm{id}_a \otimes \eta_a^L); (\epsilon^L \otimes \mathrm{id}_a) = \mathrm{id}_a}$$

The case for right adjoints is symmetric, but we are not technically done yet: we still have to dictate how the type constructors $(-)^L$ and $(-)^R$ interact with the tensor structure: if we do not 'break up' tensors nested in adjoints, we admit multiple tensor structures, such as the multiplicative connective of linear logic. Here we actually have two choices. Orthodox autonomy is captured by the following rule: we force $(A \otimes B)^L =_{\mathfrak{S}} B^L \otimes A^L$ in $\mathcal{M}(\mathfrak{S})$.

$$(L/R : \otimes)\ \frac{\mathrm{id}_{(A \otimes B)^{(L/R)}} \qquad \mathrm{id}_{B^{(L/R)}} \otimes \mathrm{id}_{A^{(L/R)}}}{\mathrm{id}_{(A \otimes B)^{(L/R)}} = \mathrm{id}_{B^{(L/R)}} \otimes \mathrm{id}_{A^{(L/R)}}}$$

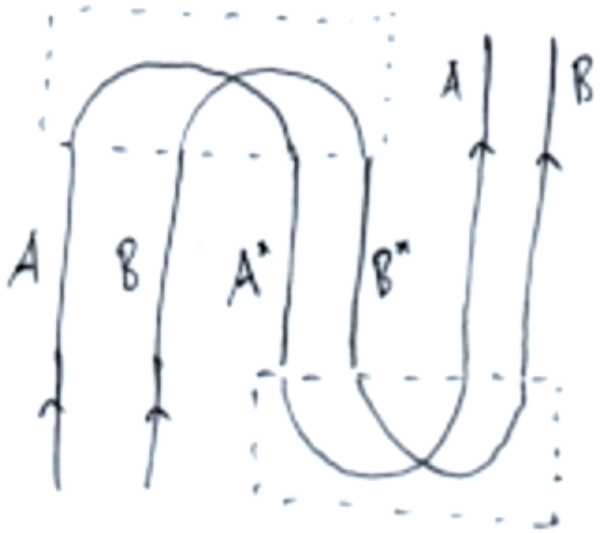


Figure 56: If we take $((A \otimes B))^L = (A)^L \otimes (B)^L$, we have a perfectly serviceable 'covariant' cup and cap, which preserves the order of types. We can draw these as twisted cups and caps.

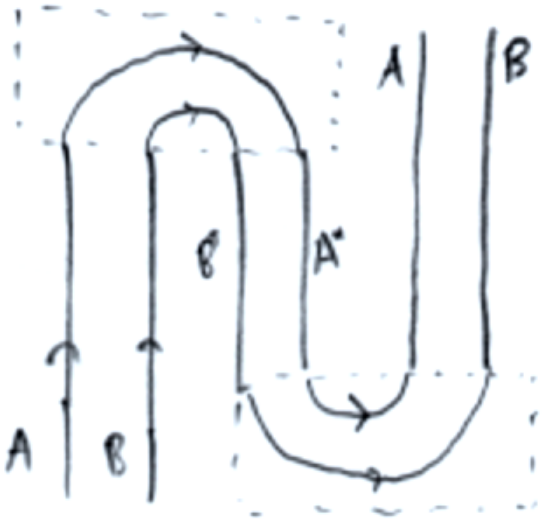


Figure 57: Otherwise, if we take the orthodox route, $((A \otimes B))^L = (B)^L \otimes (A)^L$, it's easy to see by a strong induction argument that this reverses the order of types. We can draw these as nested cups and caps.

The first observation is that graphical equations that govern natural transformations have their wires labelled with placeholder object variables, reflecting the naturality of the morphisms involved.

The second observation is that all graphical equations we consider are well-typed, in that the placeholder variables in the source and target of the two sides of the equations match.

The third observation is that we also allow placeholder morphism variables in the statement of graphical equations, in the case of equations governing homomorphism properties. However, thus far we have considered examples that obey an implicit 'semantic monotonicity' condition, which is an intuitive property of structural morphisms. At length, if we have derived a diagram that contains some specific named morphisms – which we may think of as carrying some form of semantic information – alongside structural morphisms, we may spatially rearrange, duplicate and discard the semantic information we have (*cf.* the *structural* rules of classical logic), but it *shouldn't* be the role of structure to introduce *novel* semantic elements. This separation of concerns between semantics and structure/syntax is reflected in naturality *qua* morphisms parameterised over all objects in a category: for a graphical equation involving natural transformations to introduce a *specific* – rather than arbitrary – named morphism on one side presupposes that a monoidal signature has been provided such that the named morphism can even be named. So we ask that graphical equations are 'semantically monotonous' with respect to morphism variables, more succinctly expressed as the constraint that the set of morphism variables appearing on one side of a graphical equation is either a superset or subset of the set of morphism variables appearing on the other side.

Definition 11.1 (Structural Graphical Equations)**.** A **structural graphical equation** governing monoidal structural natural isomorphisms $\omega \in \star$ equates two diagrams from the strict monoidal category obtained from the general monoidal signature which takes:

- object variables Σ_0 to be a fresh set of **placeholder object variables**
- algebraic signature σ containing I, $\otimes$, and whatever type constructors are warranted by the natural transformations ω – such as unary left and right adjoint constructors in the case of autonomous structure
- morphism variables Σ_1 to be the union of:
  - a fresh set of **placeholder morphism variables**, arbitrarily many for each choice of codomain and domain from the free algebra $(\Sigma_0)^\sigma$
  - the morphisms $\star$, each of which has a particular codomain and domain in $(\Sigma_0)^\sigma$

We consider graphical equations up to α-equivalence. We require at least one side of the equation must contain an occurrence of some morphism from $\star$. If both sides do, we call the equation **computational**. If only one side does, we call the equation **evaluative**. Further, we require that the domain and codomain of the two diagrams of any equation must be equal up to strictness of monoidal structure.

If a collection of graphical equations specified in this way are such that it is possible to derive an equation between homotopically distinct diagrams only composed of placeholder morphism variables, we consider the structure **nondeterministic**. Otherwise, we say that the structure is **deterministic**

Remark 5. Determinism is a nice condition for graphical equations to obey, as it guarantees that if one is able to eliminate all occurrences of $\star$-morphisms by applying particular series of graphical equations, the resulting diagram is unique up to planar isotopy for any other sequence of rewrites. Demonstrating necessary and sufficient conditions on graphical equations such that this graphical Church-Rosser condition holds is beyond the scope of this work. However, there are two particular necessary condition that we would draw attention to.

If the equation is evaluative, we require that the set of morphism variables appearing on the side with no $\star$-morphisms is a subset of the set of morphism variables appearing on the other side.

This condition is fairly straightforward: if it were possible to introduce a novel placeholder morphism with such an equation, that placeholder morphism can eventually be instantiated with every morphism that satisfies codomain and domain typing, which merges equivalence classes of distinct diagrams.

If the equation is computational, the set of placeholder morphism variables appearing on one side of a structural graphical equation must be either a superset or subset of the set of placeholder morphism variables occurring in the other.

Again, the same motivation applies. We observe that just about every graphical calculus with structural equations obey these rules, but one doesn't need to study a collection of rules to arrive at these conditions: if one accepts semantic monotonicity as an *a priori* requirement of 'structure', one can easily arrive at the same conditions.

Thus, by the soundness and completeness of the monoidal graphical calculus, any system of structural natural isomorphisms whose properties can be specified by graphical equations can be faithfully captured with ($\star$) and (Str) rules from a suitable $\mathfrak{S}$ up to planar isotopy of diagrams, and conversely.

The reason we have made the computational/evaluative distinction in structural equations reflects the purposes in mind we have for graphical augmentation of a monoidal category $\mathcal{K}$: we would like for 'well-formed' diagrams in the graphical language of $\mathcal{K}$ augmented with additional structural morphisms to *evaluate* via graphical equations to a diagram that contains only graphical elements from the calculus of $\mathcal{K}$. With this teleology in mind, graphical equations that do not eliminate the foreign graphical elements are intermediate steps in a *computation* towards evaluation.

11.1 Monoidal Categories with Structure S

While we were able to define arbitrary monoidal categories as arising from strong monoidal functors from the free monoidal category over a monoidal signature, we cannot directly lift the definition to define arbitrary monoidal categories with some structure **S** as defined by strong monoidal functors from a general monoidal signature with data **S**. For a counterexample, suppose that there is a monoidal category $\mathcal{K}$ with only two objects that enjoy a symmetry morphism in the presence of other objects that don't, and suppose that this monoidal category is defined over a monoidal signature Σ. We can define a strong monoidal functor from the general monoidal signature $\mathfrak{S}$ made up of Σ and symmetry structure **Sym** whose image is just the two objects that enjoy symmetry in $\mathcal{K}$. On some reading, this is satisfactory, and perhaps even desirable, but we have lost the naturality

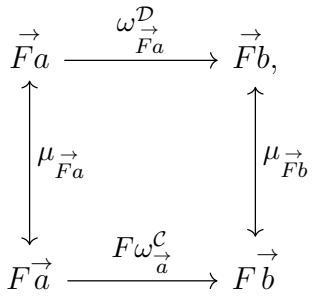


Figure 58: We have superscripted the structural natural isomorphism placeholder ω assuming $F : \mathcal{C} \to \mathcal{D}$, according to where ω lives. $\overrightarrow{a}$ is shorthand for $\mathsf{cod}(\omega)$, which is generally of the form $\bigotimes a_i$. Similarly, $\overrightarrow{b}$ is shorthand for $\mathsf{dom}(\omega)$. $\overrightarrow{Fa}$ denotes $\bigotimes F(a_i)$, and $F\overrightarrow{a}$ denotes $F(\bigotimes a_i)$. The μ are composites of coherence maps $F(a) \otimes F(b) \to F(a \otimes b)$, which are required to be invertible, since F is a strong monoidal functor.

of structural data.

What we desire is an interpretation of every structural natural transformation component as a native morphism of the category: *i.e.* one that is derivable from just a general monoidal signature with denotations and no structural data. For example, when we write the symmetry-twist $\theta_{A,B}$ in **Set**, we know from experience that there is a native morphism in **Set** that interprets it, namely the function $\theta_{A,B} := (a, b) \mapsto (b, a) : a \in A, b \in B$.

Suppose we have some structural data **S**, arbitrary but fixed. We will approach by defining an augmentation functor $\mathbb{A} : \mathcal{M}(\mathfrak{S}_\varnothing) \mapsto \mathcal{M}(\mathfrak{S}_{\mathbf{S}})$, which maps free monoidal categories on general monoidal signatures with no structural data to free monoidal categories on the same signatures with structural data **S** appended. Since every strict monoidal category is monoidally equivalent to some $\mathcal{M}(\mathfrak{S}_\varnothing)$, we will have a functor from **Mon** to 'something'. Since all $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$ have structure **S** *by fiat*, this 'something' is halfway to becoming a category of monoidal categories with structure **S**. Recalling that we want interpretations of natural transformation components as native morphisms, we have the following provisional definition.

Definition 11.2 (**SMon**)**.** Given structural natural isomorphisms and equations expressed in data **S**, let the category **SMon** consist of:

- Objects all monoidal categories equivalent to some $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$
- Morphisms strong monoidal functors, such that the diagram in Figure 58 commutes for all structural natural transformations ω in **S**

An immediate consequence of this definition is that the action of functors on objects in the target category determines the action of functors on natural transformations in the source category:

$$F(\omega^{\mathcal{C}}_{\overrightarrow{a}}) = \mu^{-1}_{\overrightarrow{F(a)}} ; \omega^{\mathcal{D}}_{\overrightarrow{F(a)}} ; \mu_{\overrightarrow{F(b)}}$$

Remark 6. The constraint on functors in the definition of **SMon** 'the obvious thing to do'; we are just asking that the functor and natural transformation data cohere. One can see that this definition subsumes that of symmetric monoidal functors [Johb, p. 4-5]: we have just powered up the definition to apply to all natural transformations present. We further discuss this definition in the discussion section.

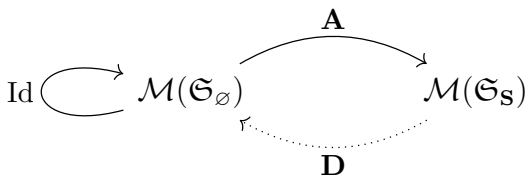


Figure 59: The motivation for the definition of **SMon** is the diagram above. If we have a functor $\mathbf{A}$ that grants $\mathcal{M}(\mathfrak{S}_\varnothing)$ extra morphisms that implement structure, and there exists a 'denotational' strong monoidal functor $\mathbf{D}$ that interprets/equates all named natural transformation components as native to $\mathcal{M}(\mathfrak{S}_\varnothing)$ such that $\mathbb{A};\mathbb{D} = \mathsf{Id}$, then we say $\mathcal{M}(\mathfrak{S}_\varnothing)$ has structure **S** to begin with.

11.1.1 Denotational Capture

When we perform graphical augmentation, we do so with respect to a general monoidal signature with no structural data. The claim in this section is that given a general augmented monoidal structure $\mathfrak{S}_\mathbf{S}$ with structural morphisms and equations, captured by data $\mathbf{S}$, we can always construct a *monoidal signature* $\mathfrak{S}^\mathbf{S}_\varnothing$, with no structural morphisms and equations, such that $\mathcal{M}(\mathfrak{S}^\mathbf{S}_\varnothing)$ is monoidally equivalent to $\mathcal{M}(\mathfrak{S}^\mathbf{S})$. The title of this section ought to be suggestive of the cheap trick we are about to pull: we simply replace all $(\star)$-rule instances with new (Name)-rule instances, and all (Str)-rule instances with new (Den)-rule instances, expanding the monoidal signature as necessary.

Definition 11.3 ($\mathfrak{S}^\mathbf{S}_\varnothing$)**.** Given $\mathfrak{S}_\mathbf{S}$ with structural data $\mathbf{S}$ in the form of morphisms $\star$ and equations $\mathbf{Str}$, obtain $\mathfrak{S}^\mathbf{S}_\varnothing$ by taking the same object variables Σ_0 and algebraic signature σ from $\mathfrak{S}$, but replacing the type constructors of $\mathbf{S}$ with fresh function symbols of the same arity. We assume that these name changes propagate forwards for the remainder of the definition, with respect to typing the domains and codomains of morphisms.

Define the morphism variables of $\mathfrak{S}^\mathbf{S}_\varnothing$ as follows:

$$\mathbf{Name}_{\mathfrak{S}^\mathbf{S}_\varnothing} := \mathbf{Name}_\mathfrak{S} \cup \star((\Sigma_0)^\sigma)$$

Where $\star((\Sigma_0)^\sigma)$ is the collection of morphisms obtained by freely instantiating the contextual slots in the domain and codomain of each $\omega \in \star$ with objects from $(\Sigma_0)^\sigma$, and assigning a fresh morphism variable for each such instantiation. Similarly define $\mathbf{Den}_{\mathfrak{S}^\mathbf{S}_\varnothing}$ by freely instantiating equations in $\mathbf{Str}_\mathbf{S}$ with morphisms derivable in $\mathtt{Mon}(\mathfrak{S})$ and replacing instances of natural transformations ω with appropriately instantiated morphism variables from $\star((\Sigma_0)^\sigma)$. Leave $\star_{\mathfrak{S}^\mathbf{S}_\varnothing}$ and $\mathbf{Str}_{\mathfrak{S}^\mathbf{S}_\varnothing}$ empty, such that $\mathfrak{S}^\mathbf{S}_\varnothing$ is a monoidal signature with denotations.

Proposition 14. *Where Ξ is a morphism or equation,*

$$\mathit{Mon}(\mathfrak{S}_\mathbf{S}) \vdash \Xi \iff \mathit{Mon}(\mathfrak{S}^\mathbf{S}_\varnothing) \vdash \Xi$$

Up to syntactic representation of instanted ω natural transformations from $\mathfrak{S}_\mathbf{S}$ in the morphism variables of $\mathfrak{S}^\mathbf{S}_\varnothing$.

Proof. Proof by proof-tree horticulture. Let $\omega(\vec{a})^\circ$ denote the unique novel morphism variable in $\mathfrak{S}^\mathbf{S}_\varnothing$ obtained by instantiating ω in $\mathfrak{S}_\mathbf{S}$ with the nonempty list $\vec{a}$ of objects from $(\Sigma_0)^\sigma$. Note further that by construction, $\circ$ is one-to-one, and the domains and codomains of $\omega(\vec{a})^\circ$ and $\omega(\vec{a})$

match. First let us consider the case for morphisms. In the proof tree for any morphism f in $\mathtt{Mon}(\mathfrak{S})$, we have the following one-to-one correspondence for every instance of $(\star)$-rule in proofs.

$$(\star)\ \frac{}{\mathrm{cod}(\omega(\overrightarrow{a})) \overset{\omega_q}{\to} \mathrm{dom}(\omega(\overrightarrow{a}))}$$

$$\Updownarrow$$

$$(\mathbf{Name}^\star)\ \frac{}{\mathrm{cod}(\omega(\overrightarrow{a})^\circ) \overset{\omega_q^\circ}{\to} \mathrm{dom}(\omega(\overrightarrow{a})^\circ)}$$

Where the latter proof lives in $\mathtt{Mon}(\mathfrak{S}^{\mathbf{S}}_{\varnothing})$, and (Name*) is just a notational indication that this is a captured $(\star)$-rule. Each such transformation evidently preserves well-formedness of proof trees, since we have by construction that $\mathrm{cod}(\omega(\overrightarrow{a})^\circ) = \mathrm{cod}(\omega(\overrightarrow{a}))$ and $\mathrm{dom}(\omega(\overrightarrow{a})^\circ) = \mathrm{dom}(\omega(\overrightarrow{a}))$ by construction. The case for the (Str) rule follows similarly.

□

11.2 Graphical Completions

Definition 11.4 (Augmenting General Monoidal Signatures with $\mathbf{S}$)**.** When a general monoidal signature $\mathfrak{S}$ has no structural data, we write it $\mathfrak{S}_{\varnothing}$. $\mathfrak{S}_{\mathbf{S}}$ is the general monoidal signature obtained from the data of $\mathfrak{S}_{\varnothing}$ and $\mathbf{S}$, expanding the algebraic signature of $\mathfrak{S}_{\varnothing}$ with type constructors as necessary to support $\mathbf{S}$.

Proposition 15. *For objects or names Φ and Ψ only recruiting syntactic elements contained in $\mathfrak{S}_{\varnothing}$, for any* deterministic *structure $\boldsymbol{S}$,*

$$\Phi =_{\mathfrak{S}_{\varnothing}} \Psi \iff \Phi =_{\mathfrak{S}_{\boldsymbol{S}}} \Psi$$

Proof. The forward direction is easy, as any proof in $\mathtt{Mon}(\mathfrak{S}_{\varnothing})$ is also a proof in $\mathtt{Mon}(\mathfrak{S}_{\mathbf{S}})$. For the reverse implication, we prove the contrapositive. It will suffice to prove the statement for morphisms, as the definition of $=_{\mathfrak{S}}$ for objects follows by equivalence between identity morphisms. Suppose that $\Phi = \Psi$ is derivable in $\mathtt{Mon}(\mathfrak{S}_{\mathbf{S}})$. We aim to show that such a derivation can be performed without invoking $(\star)$ and (Str). In general, suppose that the equality is achieved by the existence of a Γ containing novel syntactic elements introduced by $\mathbf{S}$, such that $\Phi = \Gamma$ and $\Gamma = \Psi$ are derivable in $\mathtt{Mon}(\mathfrak{S}_{\mathbf{S}})$. Moreover, we know there must exist some name Δ that does not contain novel syntactic elements such that $\Gamma = \Delta$ is derivable, since we must at least have that $\Phi = \Gamma$ and $\Gamma = \Psi$. By the determinism of $\mathbf{S}$, if $\Gamma = \Delta$ and $\Gamma = \Delta'$ are derivable in $\mathtt{Mon}(\mathfrak{S}_{\mathbf{S}})$, and both Δ and Δ' only contain syntactic elements from $\mathfrak{S}_{\varnothing}$, then $\Delta =_{\mathfrak{S}_{\varnothing}} \Delta'$: as all equations that aren't planar isotopies are achieved by (Den)-rules, which are shared by $\mathtt{Mon}(\mathfrak{S}_{\varnothing})$ and $\mathtt{Mon}(\mathfrak{S}_{\mathbf{S}})$. Thus $\Phi =_{\mathfrak{S}_{\varnothing}} \Psi$ as required. □

So, given any monoidal category $\mathcal{J}$ over a monoidal signature Σ, and structural data $\mathbf{S}$, we may define $\mathfrak{S}_{\mathbf{S}}$ as the general monoidal structure obtained from Σ, the denotational equations obtained from the unique functor from $\mathcal{M}(\Sigma)$ to $\mathcal{J}$, and the structural data $\mathbf{S}$. The above proposition grants that when $\mathbf{S}$ is deterministic, there is a faithful inclusion functor from diagrams of $\mathcal{J}$ to diagrams of $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$, and so that any derivation in the graphical calculus of $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$ that ends with a diagram

with elements only from $\mathcal{J}$ is *really referring to* that diagram in $\mathcal{J}$.

Put another way, if $\mathcal{J}$ is taken to be an arbitrary monoidal category that represents *semantics*– we might interpret the meanings of words as morphisms in $\mathcal{J}$ – we may take any deterministic structure $\mathbf{S}$ and treat morphisms in $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$ as *syntax*. The syntactically well-formed morphisms in $\mathcal{M}(\mathfrak{S}_{\mathbf{S}})$ are those that we can eventually reduce – by means of graphical equations granted by $\mathcal{J}$ and $\mathbf{S}$ – to diagrams that live in $\mathcal{J}$.

This generality is significant, because it removes the requirements upon the semantic category to carry interpretations of all structural natural transformations that may be required by syntax.

12 Discussion

12.1 What did we do?

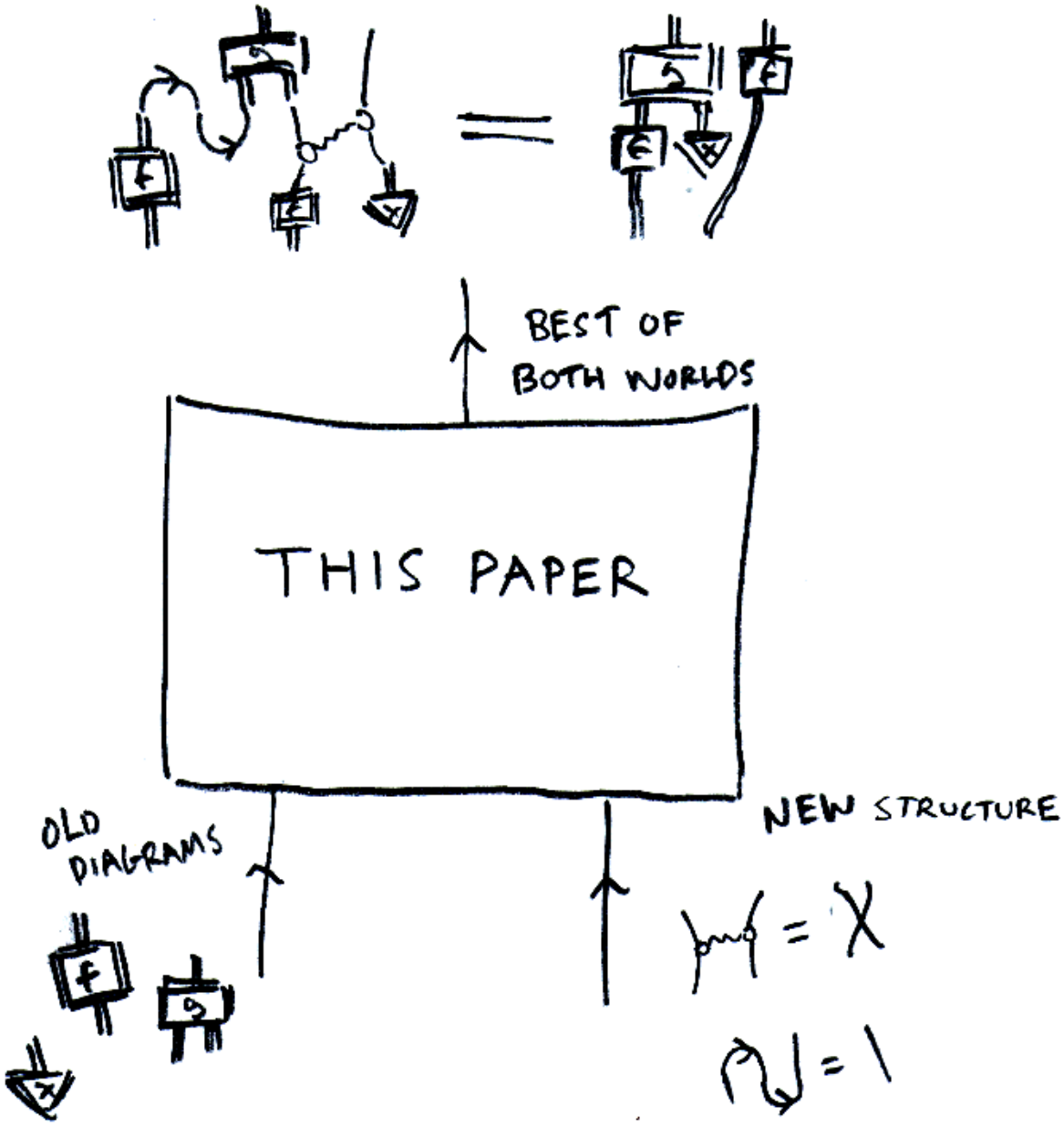


Figure 60: What we did.

12.2 Is it a free construction?

Generally, no. At least not in the sense of forming an obvious free-forgetful adjunction.

While we can describe the augmentation process as a functor from **Mon** to **SMon** (appendix material), there is no obvious candidate partner functor **SMon** to **Mon** that would make this a free-forgetful adjunction in the general case. I have considered 'inclusion' and 'structural-deletion' functors as candidates, and have considered weakening adjunction to 'adjunction up to adjunction'[18], but I have not pursued these paths to fruition.

The primary reason for this failure is that most structures aren't uniquely determined in a category, so augmentation with a structure **S** with respect to a monoidal category that already carries that structure is generally not the trivial operation. The functor **D** in Figure 59, which interprets the components of natural transformations, was originally conceived as the (pointwise) left Kan extension of the identity functor over **A**. **D** is necessarily a little opaque, because it might have to make a lot of choices under the hood. Consider a monoidal category with two syntactically distinct 'copy' natural transformations. Augmentation with another 'copy' structure will yield a new monoidal category with three syntactically distinct but extensionally equivalent 'copy' natural transformations, and the work of the functor **D** is to choose interpretations from the first two for the newly introduced third, for each object.

It is only in the event that some structure is uniquely defined in a monoidal category – such as dual structure – that augmentation with the same structure is the trivial operation. So, although our approach follows Delpeuch's, we cannot claim 'free graphical completion' as he did 'free autonomous completion': if a monoidal category already possesses autonomous structure by virtue of exact pairings for all objects [AR93], adding formal duals that satisfy the same graphical equations does nothing, because duals are unique. The same does not hold for 'accidental' structures of the general kind.

12.3 On definitional choices

It is unclear what the correct definition of **SMon** is, which frustrated attempts to prove freeness. As far as we are aware, there aren't notions in the literature similar to general monoidal signatures supporting arbitrary graphical equations. We are wary that our given definition is heavy handed and crude: having a 'one-sided' equivalence in the manner sketched in Figure 59 is problematic, so we have powered up the definition to equivalences.

Defining **S**-monoidal categories in this way however respects the spirit, if not the letter, of Joyal and Street's autonomous signatures: as we have defined general monoidal signatures to encode the data of the free **S**-structured monoidal category in its algebraic signature and structural equations, and we have encoded the action of a functor from the free category in the denotational equations.

Though general monoidal signatures as we have defined them allowed us the freedom to specify additional graphical equations in a monoidal category, we would have liked to carry out this work with more high-powered categorical tools, such as colored PROs[Yau08, §2.2], or in terms of enriched category theory. Regarding the latter, we considered viewing this construction as 'enrichment in proofs' of a thin category of algebraic terms as objects and derivability as morphisms (as in pregroups). Explicitly interpreting inequalities as monoidal natural transformations would then allow us to recast 'enrichment in proofs' as 'enrichment in diagrams'.

[18] Lax 2-Adjunction on nLab.

We have dealt mostly with equalities and strict categories for ease of reasoning. The type-theoretic approach we have taken can be extended to interpret an inequality relation, which would have (crudely) captured the notion of diagrammatic homotopies. Extending the logic in this way would permit formalisation of process theories with irreversible transformations.

The definitions of nondeterministic and deterministic collections of graphical equations hide a painful amount of complexity, and one may rightly argue that without a sound procedure to determine whether a given collection of graphical equations are deterministic, their utilisation in directing semantic composition is suspect. Determining Church-Rosser for one-dimensional rewrite systems is hard enough, but two?

12.4 What can we do with it?

Rather than start with a particular monoidal category and develop a graphical calculus supported by the category, we can now work the other way around: draw pictures first, and ask questions later.

12.4.1 Semantics in residuated pregroup grammars

With strategically introduced 'control modalities', we can make arbitrary monoidal semantics compatible with a residuated monoidal type system. The trick is, we explicitly interpret the application interpretation rules as (natural) morphisms, except we give them extra duties with respect to control modalities that 'sandwich' processes we wish to treat as semantic elements.

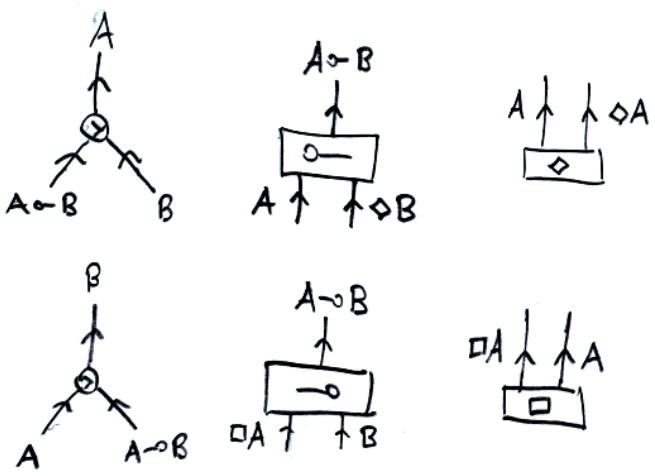


Figure 61: The two structures on the left column implement the application inference rule, where we use the lollipop rather than slashes out of personal preference. The associated control modalities for each application rule lie in the same row.

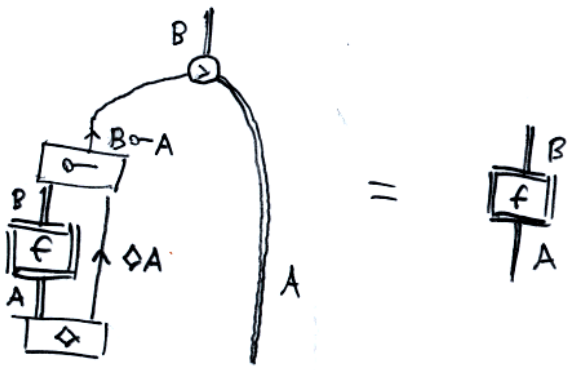


Figure 62: This is the structural equation that allows us to smuggle in arbitrary semantics. The other equation follows symmetrically with the other control modality, though strictly speaking one control modality is enough for both left and right application. Note how appending a state to the available wire on the bottom right of the left diagram resolves to composition on the right diagram.

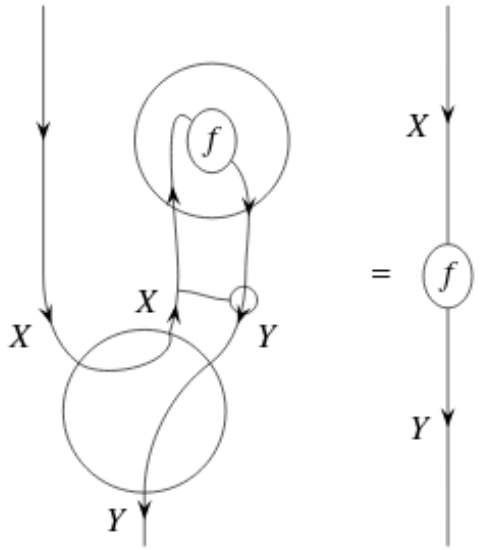


Figure 63: Observe the similarity of our trick to this diagram from [baezrosetta p33] for taking names and evaluating in closed monoidal categories. A cup is more than strictly sufficient to 'bend back' the input wire of the morphism f: we use a control modality. Our lollipop notation matches with Baez's 'clasp' notation.

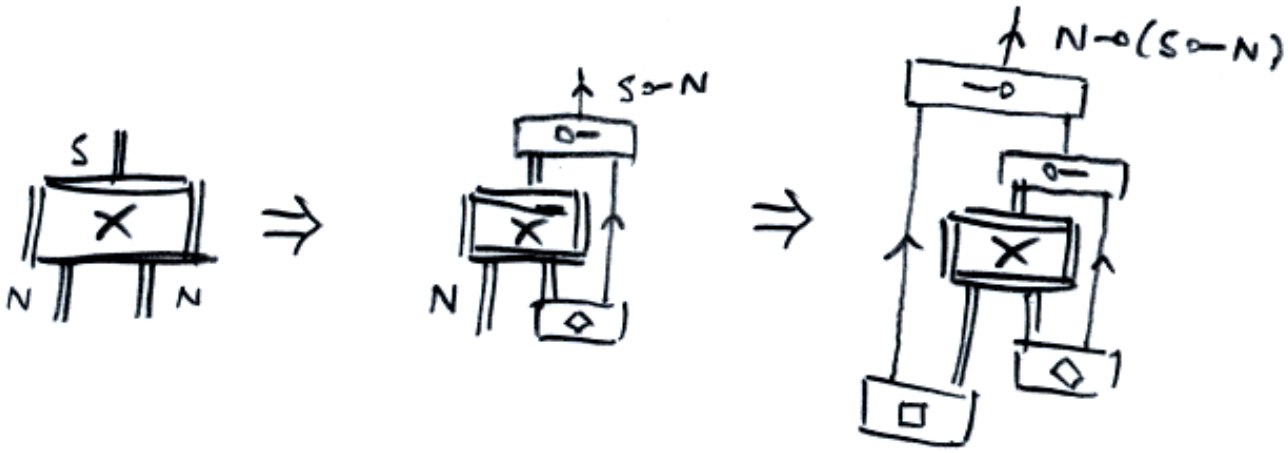


Figure 64: We may apply above visual recipe to turn semantic morphisms into typed states, such that there is a natural correspondence between the typing of words and their process-theoretic semantic interpretations.

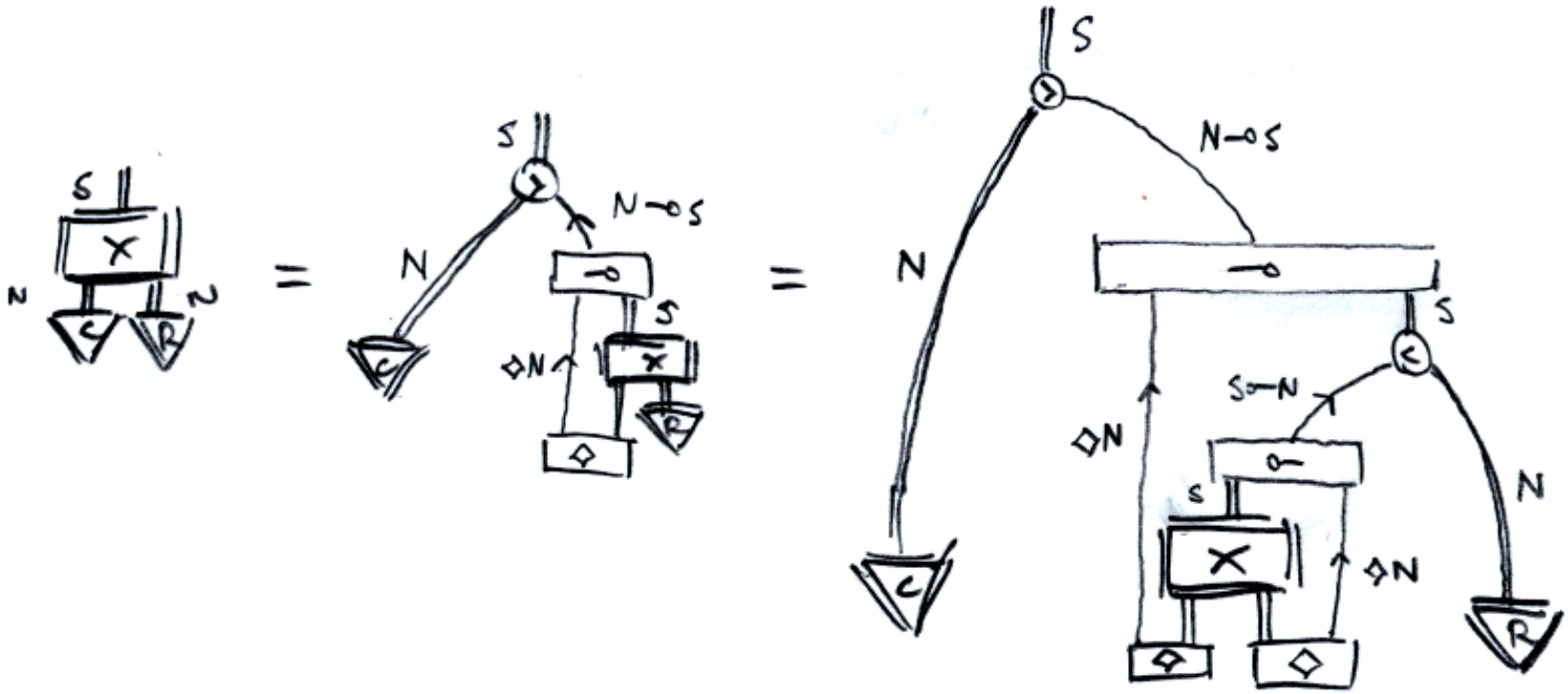


Figure 65: We can actually take things a step further into the cognitive. Starting with the completed composition corresponding to `chickens cross roads`, we can run the equations we have specified *backwards* to obtain a generative-grammar style placement of each word in a linear order as a state.

12.4.2 Delayed Symmetries

Figure 66: We take the above four morphisms in the context of a symmetric monoidal category, or one that is also suitably augmented with symmetric structure. We draw the 'control modalities' in this case as wiggly wires.

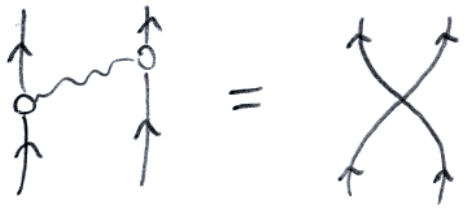


Figure 67: We factorise the symmetry twist like so.

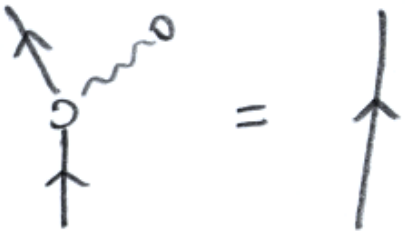


Figure 68: But we also grant the wiggly emissions a unitality property. In this way, in the course of a generative grammar that models both active and passive verb forms, it can be decided during generation whether a twist is warranted, and the twist itself is delayed and 'transmittable' long distance across the production tree.

12.4.3 Portals and the weirdness of nondeterministic graphical equations

As discussed, when a system of graphical equations is nondeterministic, even after we eliminate all novel graphical elements from a diagram in the augmented calculus, we may end up with equations between morphisms in our original category that did not hold before. This is not always an undesirable phenomenon. Consider the following nondeterministic graphical equation:

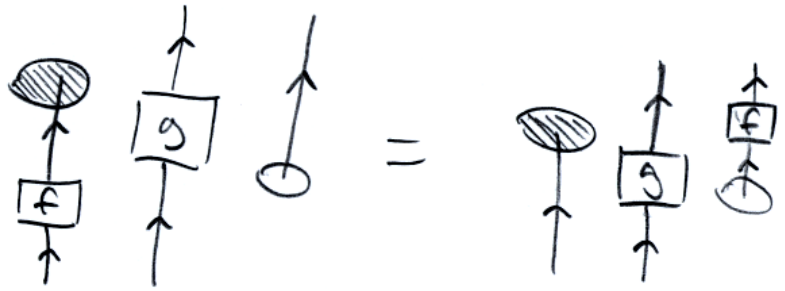


Figure 69: We have a 'black hole' and 'white hole' structure, which behave like portals.

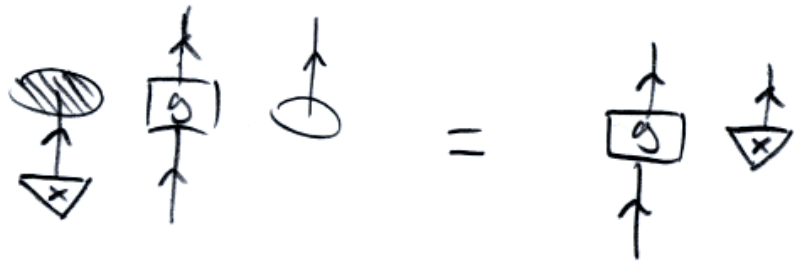


Figure 70: On states, the effect is that of teleportation, but only if there is exactly one entrance and exit.

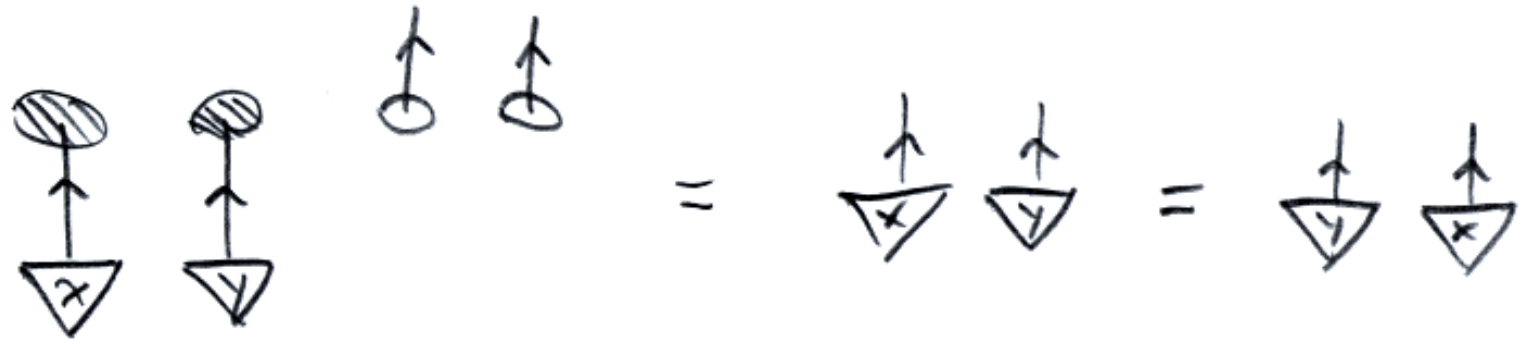


Figure 71: When there are multiple entrances and exits (and the same number of each), we have the above situation, where depending on the order of applying the portal-equation, we end up with distinct morphisms. Whereas $x \otimes y$ and $y \otimes x$ as morphisms in our original category may have been distinct, in this graphical calculus, they are made equal. Apparently, by equations of this sort, we have an implied functor from our original category to a closely related one, where all tensor-separable states and processes on objects have become *permutation invariant*. It would be interesting to know what structures induce what kinds of functors in this way.

12.4.4 Frobenius?

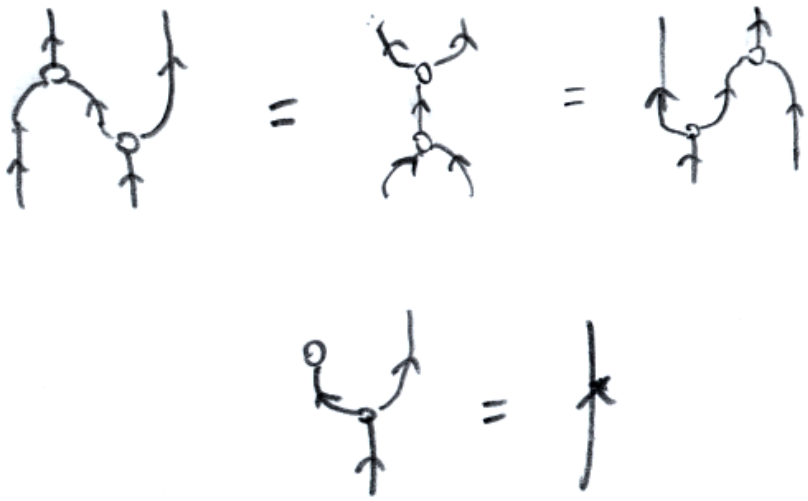


Figure 72: The frobenius equations. There is a monoid and comonoid that interact in the manner specified above, and both are unital as in the bottom equation. Frobenius algebras admit a graphical normal form as spiders [pqp spiders].

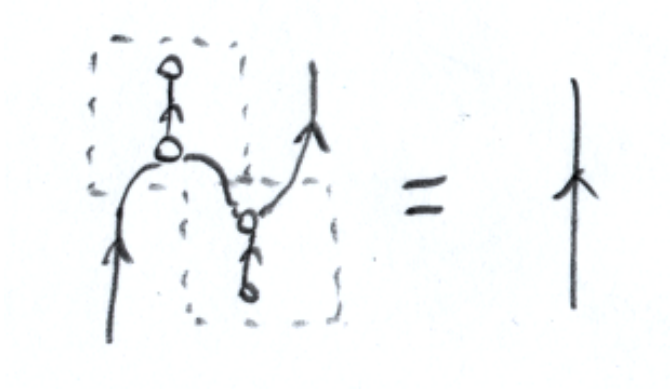


Figure 73: Frobenius structure implies a very strong form of compact closure: we can specify cups and caps (the equation above follows from the frobenius axioms) such that all objects are their own duals.

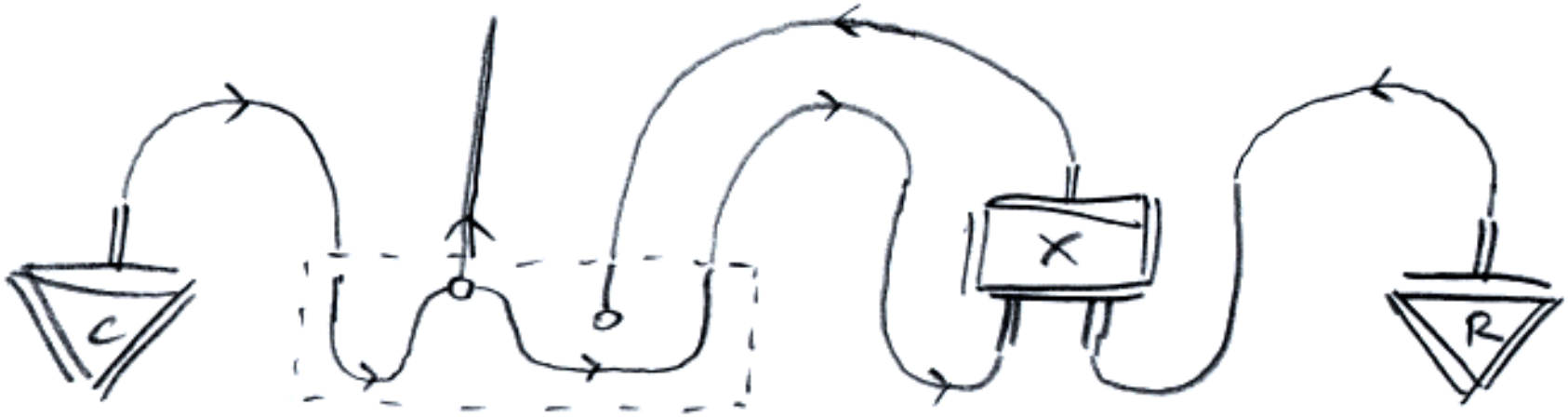


Figure 74: From [frobpronoun], the modelling of a relative pronoun using frobenius technology. The diagram above corresponds to the sentence `chickens that cross roads`. Note the nice fact that grammatical words such as `that` in this programme are 'semantically bleached': they do not contain any semantic elements, which is in accord with the received view of grammar v. semantics.

Figure 75: Resolving the previous diagram as far as we can, we reach a dead end here. Note that there remains one frobenius comonoid and one frobenius counit. Frobenius algebras are intimately related to orthonormal bases in vector spaces [coeckedusko], which `Set` simply doesn't have. We propose two possible directions for exploration.

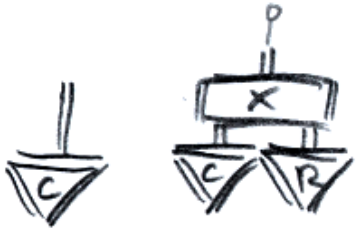


Figure 76: If we grant extra equations allowing all structures to be homomorphisms with respect to the frobenius structure, we may take one step further to arrive at the above diagram. This is a problem on several fronts. Firstly, we have an uninterpretable scalar on the right (as it still carries a frobenius counit, not to mention that **Set** has only one scalar.) Secondly, the sentence is either meant to be a proposition to be evaluated truth theoretically, or a statement of fact meant as an update to some epistemic state. In the former case, if it is not true that chickens cross roads, it is inappropriate to allow the `chickens` state to pass through on the left, and it is unclear how the scalar could stop this from occurring. In the latter case, we commit ourselves to a view where the `chickens` 'state' stands for an arbitrary chicken, and it is unclear how the scalar is to interpret a property of the arbitrary chicken.

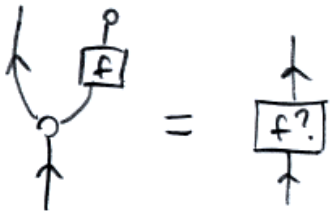


Figure 77: If we take a truth-theoretic view of the function providing the semantics for `cross`, *a priori*, we know roughly what we want: $\{x \in \mathtt{N} \mid \mathtt{cross}(x, \mathtt{roads}) = 1\}$. We're asking for a pullback of sorts, so a possible hack is to work in the category of pointed sets (sets with a special 'basepoint' element, which we will take as a standin for a $\perp$ element), and declare the above equation, where $f?$ is the characteristic function on those chickens that do cross roads, sending all other chickens to the basepoint. This solution is not nice, and further raises the spectre of dependent types in the general case of pullbacks, as we are essentially asking for the typing of the outgoing wire on the left to be dependent on the particular morphism f. Another option is to work instead in the richer monoidal category **Cat** of categories and functors between them, where there is the possibility of treating the diagram on the left as a database query, but the details must be left for future work. For now, we must concede that frobenius algebras are indispensable until proven otherwise!

A $\mathtt{Mon}(\mathfrak{S})$

Side conditions, coherence maps and equations omitted, tensor writen as concatenation.

$$\text{(Id)}\ \frac{}{a \overset{\mathrm{id}_a}{\to} a} \qquad \text{(Unit)}\ \frac{}{I \overset{\mathbf{1}}{\to} I} \qquad \text{(Den-Id)}\ \frac{}{a \overset{\mathrm{id}_X}{\to} b}$$

$$\text{(Name)}\ \frac{}{\mathrm{cod}(f) \overset{f}{\to} \mathrm{dom}(f)} \qquad (\star)\ \frac{}{\mathrm{cod}(\omega(\vec{a})) \overset{\omega_a}{\to} \mathrm{dom}(\omega(\vec{a}))}$$

$$(;)\ \frac{A \overset{f}{\to} B \qquad B \overset{g}{\to} C}{A \overset{(f;g)}{\to} C} \qquad (\otimes)\ \frac{A \overset{f}{\to} B \qquad C \overset{g}{\to} D}{AC \overset{(f\otimes g)}{\to} BD}$$

$$\text{(R)}\ \frac{A \overset{f}{\to} B}{A \overset{f=f}{\to} B} \qquad \text{(S)}\ \frac{A \overset{f=g}{\to} B}{A \overset{g=f}{\to} B} \qquad \text{(T)}\ \frac{A \overset{f=g}{\to} B \qquad A \overset{g=h}{\to} B}{A \overset{f=h}{\to} B}$$

$$\text{(L.U.)}\ \frac{A \overset{(\mathrm{id}_A;f)=g}{\to} B}{A \overset{f=g}{\to} B} \qquad \text{(R.U.)}\ \frac{A \overset{(f;\mathrm{id}_B)=g}{\to} B}{A \overset{f=g}{\to} B} \qquad \text{(Assoc)}\ \frac{A \overset{((f;g);h)=j}{\to} B}{A \overset{(f;(g;h))=j}{\to} B}$$

$$\text{(Interchange)}\ \frac{A \overset{f}{\to} X \qquad X \overset{g}{\to} C \qquad AB \overset{((f;g)\otimes(h;j))=k}{\to} CD \qquad B \overset{h}{\to} Y \qquad Y \overset{j}{\to} D}{AB \overset{((f\otimes h);(g\otimes j))=k}{\to} CD}$$

$$\text{(Id-Tensor)}\ \frac{}{ab \overset{\mathrm{id}_{(a\otimes b)}=(\mathrm{id}_a\otimes\mathrm{id}_b)}{\to} ab}$$

$$(=^{\leftrightarrow})\ \frac{A \overset{f=f'}{\to} B \qquad AC \overset{(f\otimes g)=h}{\to} BD \qquad C \overset{g=g'}{\to} D}{AC \overset{(f'\otimes g')=h}{\to} BD} \qquad (=^{\updownarrow})\ \frac{A \overset{f=f'}{\to} B \qquad A \overset{(f;g)=h}{\to} C \qquad B \overset{g=g'}{\to} C}{A \overset{(f';g')=h}{\to} C}$$

$$\text{(Den)}\ \frac{A \overset{f}{\to} B \qquad C \overset{g}{\to} D}{A \overset{f=g}{\to} D} \qquad \text{(Str)}\ \frac{A \overset{f}{\to} B \qquad A \overset{g}{\to} B}{A \overset{f=g}{\to} B}$$

B The Functor $\mathbb{A} : \mathbf{Mon} \to \mathbf{SMon}$

Before we define the augmentation functor, we'll show how our definition **SMon** allows us to lift strong monoidal functors **Mon** to functors in **SMon**. Throughout, we suppose that some arbitrary but fixed $\mathbf{S}$ is provided.

Definition B.1 (Lifting Functors)**.** For any functor $F : \mathcal{J} \to \mathcal{K}$ between $\mathcal{J}$, $\mathcal{K}$ in **Mon**, apply unique monoidal isomorphisms by Proposition 13 to obtain $G : \mathcal{M}(\mathfrak{S}_\varnothing) \to \mathcal{M}(\mathfrak{T}_\varnothing)$. Define $\mathbb{A}G : \mathcal{M}(\mathfrak{S}_\mathbf{S}) \to \mathcal{M}(\mathfrak{T}_\mathbf{S})$ on objects as:

$$\mathbb{A}G[X]_{\mathfrak{S}_\mathbf{S}} = \begin{cases} [I]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = I \\ \left[\mathbb{A}G[A]_{\mathfrak{S}_\mathbf{S}} \otimes \mathbb{A}G[A]_{\mathfrak{S}_\mathbf{S}}\right]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = (A \otimes B) \\ \left[\rho(\mathbb{A}G[A_1]_{\mathfrak{S}_\mathbf{S}}, \ldots, \mathbb{A}G[A_n]_{\mathfrak{S}_\mathbf{S}})\right]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = \rho(A_1, \ldots, A_n),\ \rho \in \sigma \\ \left[G[a]_{\mathfrak{S}_\mathbf{S}}\right]_{\mathfrak{T}_\mathbf{S}} (= \left[G[a]_{\mathfrak{S}_\varnothing}\right]_{\mathfrak{T}_\varnothing}) & \text{if } X = a \in \Sigma_0 \end{cases}$$

And on morphisms:

$$\mathbb{A}G[X]_{\mathfrak{S}_\mathbf{S}} = \begin{cases} [\mathrm{id}_{\mathbb{A}GA}]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = \mathrm{id}_A \\ \left[\mathbb{A}G[f]_{\mathfrak{S}_\mathbf{S}}; \mathbb{A}G[g]_{\mathfrak{S}_\mathbf{S}}\right]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = (f; g) \\ \left[\mathbb{A}G[f]_{\mathfrak{S}_\mathbf{S}} \otimes \mathbb{A}G[g]_{\mathfrak{S}_\mathbf{S}}\right]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = (f \otimes g) \\ \left[G[f]_{\mathfrak{S}_\mathbf{S}}\right]_{\mathfrak{T}_\mathbf{S}} (= \left[G[f]_{\mathfrak{S}_\varnothing}\right]_{\mathfrak{T}_\varnothing}) & \text{if } X = f \in \Sigma_1 \\ \left[\omega_{\overrightarrow{Ga}}\right]_{\mathfrak{T}_\mathbf{S}} & \text{if } X = \omega_{\overrightarrow{a}},\ \omega \in \star \end{cases}$$

Remark 7. Lifting functors in this way is basically an algebra homomorphism on the objects and morphisms, with respect to the algebraic signature and $(;, \otimes)$ structure.

Definition B.2 ($\mathbb{A}$: the augmentation functor)**.** As shorthand, we write $\mathcal{J}_\mathbf{S}$ for the $\mathbf{S}$-augmentation of the monoidal category $\mathcal{J}$. Given a structure $\mathbf{S}$, we define $\mathbb{A} : \mathbf{Mon} \to \mathbf{Smon}$ to take monoidal categories $\mathcal{J}$ to $\mathcal{J}_\mathbf{S}$, and strong monoidal functors $F : \mathcal{K} \to \mathcal{L}$ to their lift $\mathbb{A}F : \mathcal{K}_\mathbf{S} \to \mathcal{L}_\mathbf{S}$, as described above.

C More Lemmas for $\mathtt{Mon}(\mathfrak{S})$

We can actually leverage Lemma 5 into more powerful forms. At the moment, f' and f may still differ up to associativity, and unitality. We can sharpen the statement of the Lemma to make f' agree with f up to just unitality.

Lemma 16 (Each side of derivable equation is derivable up to type-matching identities)**.** *If $f = g$ is derivable, then f'' and $f'' = f$ are derivable, where f'' differs from f at most up to pre- and post-composition of type-matching identities from (Den-Id) in subnames of f''.*

Proof. We will just focus on f, as the argument for g is symmetric. In the proof strategy of Lemma 5, we have observed that in the construction of f' in the cases associativity and interchange, both syntactic variants are derivable: *i.e.* in the derivation of f' corresponding to an (Assoc) equality

in the induction, if $f' \equiv ((g;h);j)$, we may also derive $(g;(h;j))$ by transforming the proof tree, and similarly for interchange. In this way we may derive an f'' by transforming the proof tree of f', such that f'' agrees with f up to pre- and post-composition of type-matching identities. A similar proof strategy as Lemma 5 grants that $f'' = f$. □

Recall that in a sequent calculus, a new rule is *admissible* if any derivation in a system with the addition of that rule is also derivable without it. We consider a particularly admissible rule: where $\Gamma\langle f\rangle$ denotes that f is a subname appearing as a (located) substring of a name Γ, we consider the following (Cut&Paste) rule.

Lemma 17 (Replacement is Admissible)**.** *The following rule is admissible*

$$(Cut\&Paste)\ \frac{A \overset{\Gamma\langle f\rangle}{\to} B \qquad C \overset{f=g}{\to} D}{A \overset{\Gamma\langle g'\rangle}{\to} B}$$

Where g' and $g' = g$ is derivable, and g' differs from g at most up to pre- and post-composition of type-matching (Den-Id) identities in subnames occuring in g.

Proof. Firstly, since f occurs as a well-bracketed subname, by analyticity, a derivation of $A' \overset{f}{\to} B'$ occurs as a unique subproof in the derivation of $A \overset{\Gamma\langle f\rangle}{\to} B$. By Lemma 5, a derivation of $C \overset{f=g}{\to} D$ yields a derivation of $C \overset{g'}{\to} D$ such that $g' = g$ is derivable. By the well-definedness of $=_{\mathfrak{S}}$ on objects, we can pre- and postcompose g' with type-matching identities from (Den-Id) to obtain a derivation of $A' \overset{g''}{\to} B'$, by matching C with A' and D with B'. Now we can replace the subproof of $A' \overset{f}{\to} B'$ in that of $A \overset{\Gamma\langle f\rangle}{\to} B$ by the derivation of $A' \overset{g''}{\to} B'$, and the remainder of the derivation will yield $A \overset{\Gamma\langle g''\rangle}{\to} B$ as desired. □

Of note is that the new proof constructed in the lemma above no longer contains a derivation of the replaced f.

Proposition 18 (Cut and Paste for Equations)**.** *If $\Gamma\langle f\rangle$, $\Gamma\langle f\rangle = h$ and $f = g$ are derivable, then $\Gamma\langle g'\rangle = h$ is derivable, where g' and $g' = g$ is derivable, and g' differs from g at most up to pre- and post-composition of type-matching (Den-Id) identities in subnames occuring in g.*

Proof. We can consider the proof of $\Gamma\langle f\rangle = h$ to 'leave $\Gamma\langle f\rangle$ unmolested' in every equality step: it is derived once, introduced into an equality, and then travels syntactically unchanged towards the conclusion $\Gamma\langle f\rangle = h$. In light detail, we can achieve this by eliminating occurrences of (Assoc), (Interchange), and diagram manipulation equality rules as in the proof strategies outlined in Lemmas 5 and 16. We can also consider the proof of $\Gamma\langle f\rangle = h$ to contain no instances of Left and Right Unitality rules that alter $\Gamma\langle f\rangle$. An informal argument for this second point is to consider the length of name to be the number of connectives and atomic name symbols obtained from initial morphism sequents in that name, and to note that after eliminating instances of $(=^{\leftrightarrow})$, $(=^{\updownarrow})$ as above, all equality rules either preserve or monotonically decrease (in the case of left and right unitality) the length of names they operate upon. Since $\Gamma\langle f\rangle$ is derivable, it is analytic, and hence contains all its type-matching identities explicitly, so we know that no left and right unitality rules operate upon $\Gamma\langle f\rangle$ on its way to the conclusion.

In this proof form of $\Gamma\langle f\rangle = h$, we have a unique derivation of $\Gamma\langle f\rangle$ that travels unmolested to the conclusion. We can then use the admissible (Replacement) rule to convert derivations of $\Gamma\langle f\rangle = h$ and $f = g$ into a derivation of $\Gamma\langle g'\rangle = h$, which must also be a valid proof, because by specification, $\Gamma\langle f\rangle$ is unmolested, so replacing it with $\Gamma\langle g'\rangle$ still grants safe passage to the conclusion.

Now consider the base cases. The first is where $\Gamma\langle f\rangle$ is introduced as one side of an equality by a (Den) rule, but this is easily dealt with: We can prove by (R) that $\Gamma\langle f\rangle = \Gamma\langle f\rangle$, and by a strong inductive argument on proof tree length, assuming the claim as the inductive hypothesis, we may derive $\Gamma\langle f\rangle = \Gamma\langle g'\rangle$, with which we can apply the (T) rule to the (Den) case. The case where $\Gamma\langle f\rangle$ is introduced into an equality by the (R) rule follows similarly. □

The above lemma allows graphical reasoning by replacement of equal subdiagrams (so long as one trusts that judicious use of interchange and associativity can isolate any well-typed subdiagram)

C.1 PRO-composition

PROs are PROPs (product and permutation categories) without symmetry. We have essentially been working with coloured-PROs throughout. There does not appear to be a definition of PRO-composition in the literature, so we are only taking this name for its suggestivity. What we wish to specify is a liberal form of composition for lists of objects from a set of colours, such that a list of domain variables X may compose with a list of codomain variables Y if either: X contains Y as a sublist, Y contains X as a sublist, a prefix of X is equal to a suffix of Y, or a suffix of X is equal to a prefix of Y. Graphically, the options correspond to the essentially different ways two planar diagrams can be pasted together sequentially without crossing wires.

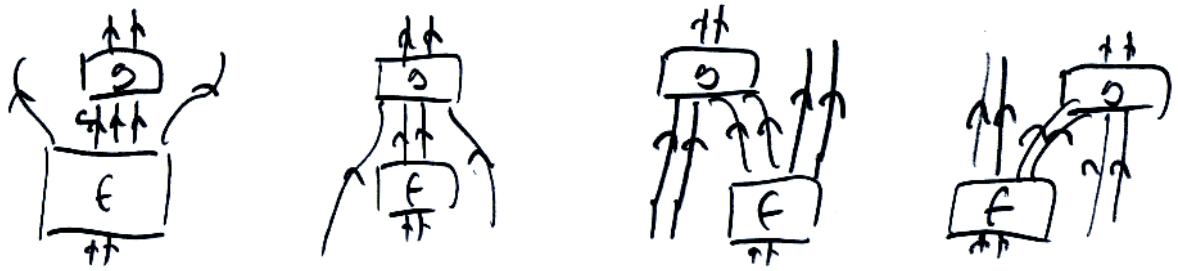


Figure 78: The four options for vertical composition of processes f and g, corresponding to the four possibilities expressed in terms of sublists and prefixes

It is in general insufficient to state just the matched sublists, as we may have a situation where a domain $BCCD$ must be matched with a codomain C, in which case there is ambiguity. The solution is to specify just the prefix and postfix of unmatched outputs of the domain, which does uniquely specify the sublist where composition is performed. So we write $f^B +^D g$ to indicate the PRO-composition of f with g, where B and D are a prefix and postfix of the domain type of f, replacing either B or D with $-$ if the sublist is empty. For the following, we will omit tensors, and just write concatenations.

Proposition 19. *PRO-composition is admissible.*

Proof. The idea is to tensor in sufficiently many identity wires in the plain calculus such that PRO-composition becomes plain composition. There are essentially three distinct cases of the PRO-composition rule:

- The first case is where the codomain of s strictly contains the domain of t, which, without loss of generality, appears as such:

$$\dfrac{\dfrac{\vdots}{A \xrightarrow{s} BC} \quad \dfrac{\vdots}{CD \xrightarrow{t} E}}{s^B +^D t : A \to BCD}$$

  Instances of this form can be replaced by trees of the following form:

$$\dfrac{\dfrac{\vdots}{A \xrightarrow{s} BCD} \quad \dfrac{(\mathrm{Id})\dfrac{}{B \xrightarrow{\mathrm{id}_B} B} \quad \dfrac{\vdots}{C \xrightarrow{t} E} \quad (\mathrm{Id})\dfrac{}{D \xrightarrow{\mathrm{id}_D} D}}{BCD \xrightarrow{\mathrm{id}_B \otimes t \otimes \mathrm{id}_D} BED}\,(\otimes \times 2)}{s^B +^D t : A \to BCD}\,(+)$$

- The second case is where the codomain of s strictly partially overlaps the domain of t, which can either be from the left or the right. Without loss of generality, this appears as such:

$$\dfrac{\dfrac{\vdots}{A \xrightarrow{s} BC} \quad \dfrac{\vdots}{CD \xrightarrow{t} E}}{s^B +^- t : AD \to BE}$$

  Instances of this form can be replaced by trees of the following form:

$$\dfrac{(\otimes)\dfrac{\dfrac{\vdots}{A \xrightarrow{s} BC} \quad (\mathrm{Id})\dfrac{}{D \xrightarrow{\mathrm{id}_D} D}}{AD \xrightarrow{s \otimes \mathrm{id}_D} BCD} \quad (\mathrm{Id})\dfrac{\dfrac{}{B \xrightarrow{\mathrm{id}_B} B} \quad \dfrac{\vdots}{CD \xrightarrow{t} E}\,(\otimes)}{BCD \xrightarrow{\mathrm{id}_B \otimes t} BE}}{s^B +^- t : AD \to BE}\,(+)$$

- The third case is where the codomain of s is strictly contained within the domain of t, which, without loss of generality, appears as such.

$$\dfrac{\dfrac{\vdots}{A \xrightarrow{s} C} \quad \dfrac{\vdots}{BCD \xrightarrow{t} E}}{s^- +^- t : BAD \to E}$$

  Instances of this form can be replaced by trees of the following form:

$$\dfrac{(\otimes \times 2)\dfrac{(\mathrm{Id})\dfrac{}{B \xrightarrow{\mathrm{id}_B} B} \quad \dfrac{\vdots}{A \xrightarrow{s} C} \quad (\mathrm{Id})\dfrac{}{D \xrightarrow{\mathrm{id}_D} D}}{BAD \xrightarrow{\mathrm{id}_B \otimes s \otimes \mathrm{id}_D} BCD} \quad \dfrac{\vdots}{BCD \xrightarrow{t} E}}{s^- +^- t : BAD \to E}\,(+)$$

□

References

[Abr05] Samson Abramsky. Abstract Scalars, Loops, and Free Traced and Strongly Compact Closed Categories. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Jos Luiz Fiadeiro, Neil Harman, Markus Roggenbach, and Jan Rutten, editors, *Algebra and Coalgebra in Computer Science*, volume 3629, pages 1–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[AHS02] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, October 2002.

[AL19] Andrea Aler Tubella and Lutz Straburger. Introduction to Deep Inference, 2019.

[AR] Andr Joyal and Ross Street. Planar Diagrams and Tensor Algebra.

[AR93] Andr Joyal and Ross Street. Braided Tensor Categories. *Advances in Mathematics*, 102:20–78, 1993.

[AT10] Samson Abramsky and Nikos Tzevelekos. Introduction to Categories and Categorical Logic. *arXiv:1102.1313 [cs, math]*, 813:3–94, 2010. arXiv: 1102.1313.

[Aus05] J. L. Austin. *How to Do Things with Words*. Harvard University Press, Cambridge, Mass, 2005.

[Bar] Bartosz Milewski. Category Theory for Programmers.

[Bar92] Lawrence Barsalou. Frames, Concepts, and Conceptual Fields. In E. Kittay and A. Lehrer, editors, *Frames, Fields, and Contrasts: New Essays in Semantic and Lexical Organization*, pages 21–74. Erlbaum, 1992.

[Bar99] Michael Barr. *-Autonomous Categories: Once More Around The Track. In *And Chu Constructions: Cousins? 149*, pages 5–24, 1999.

[BB] Marco Baroni and Raffaella Bernardi. Frege in Space: A Program for Compositional Distributional Semantics. page 107.

[BBE11] David Barkerplummer, Jon Barwise, and John Etchemendy. *Language, Proof and Logic*. Center for the Study of Language and Information, Stanford, Calif, 2nd revised edition edition edition, October 2011.

[BBMM97] J. van Benthem, Johan van Benthem, Alice G. B. ter Meulen, and A. ter Meulen. *Handbook of Logic and Language*. Elsevier, 1997. Google-Books-ID: BFTF_6uD14EC.

[BC17] Aleks Kissinger Bob Coecke. *Picturing Quantum Processes*. Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA, March 2017.

[BCG+17] Joe Bolt, Bob Coecke, Fabrizio Genovese, Martha Lewis, Dan Marsden, and Robin Piedeleu. Interacting Conceptual Spaces I : Grammatical Composition of Concepts. *arXiv:1703.08314 [cs]*, March 2017. arXiv: 1703.08314.

[BD08] Alexandre Buisse and Peter Dybjer. Towards Formalizing Categorical Models of Type Theory in Type Theory. *Electronic Notes in Theoretical Computer Science*, 196:137–151, January 2008.

[Bla88] Barry B. Blake. Russell S. Tomlin, Basic word order. Functional principles. London: Croom Helm, 1986. Pp. 308. *Journal of Linguistics*, 24(1):213–217, 1988.

[Boo10] George Boole. *An Investigation of the Laws of Thought*. Watchmaker Publishing, Erscheinungsort nicht ermittelbar, April 2010.

[BS00] Francis Borceux and Isar Stubbe. Short Introduction to Enriched Categories. In Bob Coecke, David Moore, and Alexander Wilce, editors, *Current Research in Operational Quantum Logic*, pages 167–194. Springer Netherlands, Dordrecht, 2000.

[BS09] John C. Baez and Mike Stay. Physics, Topology, Logic and Computation: A Rosetta Stone. *arXiv:0903.0340 [quant-ph]*, March 2009. arXiv: 0903.0340.

[Bus01] Wojciech Buszkowski. Lambek Grammars Based on Pregroups. In *Proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics*, LACL '01, pages 95–109, Berlin, Heidelberg, 2001. Springer-Verlag.

[CC] Charles F. Hockett and Charles H. Hockett. Vol. 203, No. 3, September 1960 of Scientific American on JSTOR: Origins of Speech.

[CCS13] Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. The Frobenius Anatomy of Relative Pronouns. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 41–51, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[CFS16] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Information and Computation*, 250:59–86, October 2016. arXiv: 1409.5531.

[CGS13] Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. Lambek vs. Lambek: Functorial vector space semantics and string diagrams for Lambek calculus. *Annals of Pure and Applied Logic*, 164(11):1079–1100, November 2013.

[CL] Bob Coecke and Ray Lal. Causal categories: a backbone for a quantum- relativistic universe of interacting processes. page 10.

[CL02] Noam Chomsky and David W. Lightfoot. *Syntactic Structures*. Walter de Gruyter, 2002. Google-Books-ID: SNeHkMXHcd8C.

[Coe19] Bob Coecke. The Mathematics of Text Structure. *arXiv:1904.03478 [quant-ph]*, April 2019. arXiv: 1904.03478.

[CPV13] Bob Coecke, Dusko Pavlovic, and Jamie Vicary. A new description of orthogonal bases. *Mathematical Structures in Computer Science*, 23(3):555–567, June 2013. arXiv: 0810.0812.

[Cro94] Crole. *Categories for Types*. Cambridge University Press, Cambridge ; New York, 1 edition edition, March 1994.

[CS] Edited J G Carbonell and J Siekmann. Lecture Notes in Artificial Intelligence. page 372.

[CS99] J.R.B. Cockett and R.A.G. Seely. Linearly distributive functors. *Journal of Pure and Applied Algebra*, 143(1-3):155–203, November 1999.

[CSC10] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning. *arXiv:1003.4394 [cs, math]*, March 2010. arXiv: 1003.4394.

[Dal] Mary Dalrymple. Lexical Functional Grammar. page 27.

[Deb00] Ralph Debusmann. An introduction to dependency grammar. February 2000.

[Del14] Antonin Delpeuch. Autonomization of Monoidal Categories. *arXiv:1411.3827 [cs, math]*, November 2014. arXiv: 1411.3827.

[dFMT] Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Montague Semantics for Lambek Pregroups. page 12.

[dFMT19] Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Functorial Question Answering. *arXiv:1905.07408 [cs, math]*, May 2019. arXiv: 1905.07408.

[DP05] K. Dosen and Z. Petric. Coherence for Star-Autonomous Categories. *arXiv:math/0503306*, March 2005. arXiv: math/0503306.

[Fir57] J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

[FL05] Marcelo Fiore and Tom Leinster. An abstract characterization of Thompson's group F. *arXiv:math/0508617*, August 2005. arXiv: math/0508617.

[Fon19] Brendan Fong. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, July 2019.

[FS02] Fernanda Ferreira and Benjamin Swets. How Incremental Is Language Production? Evidence from the Production of Utterances Requiring the Computation of Arithmetic Sums. *Journal of Memory and Language*, 46(1):57–84, January 2002.

[FS18] Brendan Fong and Maru Sarazola. A recipe for black box functors. *arXiv:1812.03601 [math]*, December 2018. arXiv: 1812.03601.

[FS19] Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 1 edition, July 2019.

[FST17] Brendan Fong, David I. Spivak, and Rmy Tuyras. Backprop as Functor: A compositional perspective on supervised learning. *arXiv:1711.10455 [cs, math]*, November 2017. arXiv: 1711.10455.

[Gir11] Jean-Yves Girar. *The Blind Spot: Lectures on Logic*. European Mathematical Society, Zrich, September 2011.

[Gol14] R. Goldblatt. *Topoi: The Categorial Analysis of Logic*. North Holland, 2 edition edition, June 2014.

[GP18] Richard Garner and John Power. An enriched view on the extended finitary monad–Lawvere theory correspondence. *Logical Methods in Computer Science ; Volume 14*, page Issue 1 ; 18605974, 2018.

[Gri91] P. Grice. *Studies in the Way of Words*. Harvard University Press, Cambridge, Mass., new ed edition edition, May 1991.

[Har78] Charles S. Hardwick. *Semiotics and Significs: Correspondence Between Charles S.Peirce and Lady Victoria Welby*. Indiana University Press, Bloomington, April 1978.

[Heu] Chris Heunen. Categories and Quantum Informatics: Dual objects. page 14.

[HK98] Irene Heim and Angelika Kratzer. *Semantics in generative grammar*. Number 13 in Blackwell textbooks in linguistics. Blackwell, Malden, MA, 1998.

[HM14] G H. Matthews. Chomsky N. and Schtzenberger M. P.. The algebraic theory of context-free languages. Computer programming and formal systems, edited by Braffort P. and Hirschberg D., Studies in logic and the foundations of mathematics, North-Holland Publishing Company, Amsterdam 1963, pp. 118161. *The Journal of Symbolic Logic*, 32:388–389, October 2014.

[HMP98] C. Hermida, M. Makkai, and J. Power. Higher dimensional multigraphs. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.98CB36226)*, pages 199–206, June 1998.

[Hof95] Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Leszek Pacholski, and Jerzy Tiuryn, editors, *Computer Science Logic*, volume 933, pages 427–441. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[HR15] Philip Hackney and Marcy Robertson. On the category of props. *Applied Categorical Structures*, 23(4):543–573, August 2015. arXiv: 1207.2773.

[HS05] Esfandiar Haghverdi and Philip Scott. From Geometry of Interaction to Denotational Semantics. *Electronic Notes in Theoretical Computer Science*, 122:67–87, March 2005.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *An Introduction to Automata Theory, Languages, and Computation*. Pearson, Reading, Mass, 1 edition edition, January 1979.

[Jac16] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge University Press, Cambridge, 1 edition edition, December 2016.

[Jak61] Roman Jakobson. *Structure of Language and Its Mathematical Aspects*. American Mathematical Soc., 1961. Google-Books-ID: w0vHCQAAQBAJ.

[Jam19] James. A julia package for representing and manipulating model semantics: jpfairbanks/SemanticModels.jl, July 2019. original-date: 2018-11-01T22:18:11Z.

[Joha] John Baez. The n-Category Caf.

[Johb] John Baez. Some Definitions Everyone Should Know. Technical report.

[Joh02] Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: 2 Volume Set*. Oxford University Press UK, 2002.

[JS91] Andr Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991.

[Kar08] Richard Karban. Plant behaviour and communication. *Ecology Letters*, 11(7):727–739, 2008.

[Kis12] Aleks Kissinger. Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing. *arXiv:1203.0202 [quant-ph]*, March 2012. arXiv: 1203.0202.

[KM12] Karl H. Hofmann and Michael Mislove. Compact affine monoids, harmonic analysis and information theory. In Samson Abramsky and Michael Mislove, editors, *Proceedings of Symposia in Applied Mathematics*, volume 71, pages 125–182. American Mathematical Society, Providence, Rhode Island, 2012.

[KPBB16] Germn Kruszewski, Denis Paperno, Raffaella Bernardi, and Marco Baroni. There Is No Logical Negation Here, But There Are Alternatives: Modeling Conversational

Negation with Distributional Semantics. *Computational Linguistics*, 42(4):637–660, December 2016.

[KS] Dimitri Kartsaklis and Mehrnoosh Sadrzadeh. Distributional Inclusion Hypothesis for Tensor-based Composition. page 12.

[KU17] Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. *arXiv:1701.04732 [math-ph, physics:quant-ph]*, January 2017. arXiv: 1701.04732.

[LaF15] Adrienne LaFrance. An Unusual Way of Speaking, Yoda Has, December 2015.

[Lal] Raymond Lal. Causal Structure in Categorical Quantum Mechanics. page 173.

[Lam] Joachim Lambek. THE MATHEMATICS OF SENTENCE STRUCTURE. page 18.

[Lam68] Joachim Lambek. The Mathematics of Sentence Structure. *Journal of Symbolic Logic*, 33(4):627–628, 1968.

[Lam08] J. Lambek. Pregroup Grammars and Chomskys Earliest Examples. *Journal of Logic, Language and Information*, 17(2):141–160, April 2008.

[Law69] F. William Lawvere. Adjointness in Foundations. *dialectica*, 23(3-4):281–296, December 1969.

[Lei04] Tom Leinster. *Higher Operads, Higher Categories*. Cambridge University Press, July 2004. Google-Books-ID: K8nPLSAzhAcC.

[LLP99] Alain Lecomte, Franois Lamarche, and Guy Perrier, editors. *Logical aspects of computational linguistics: second international conference, LACL '97, Nancy, France, September 22-24, 1997: selected papers*. Number 1582 in Lecture notes in computer science ; Lecture notes in artificial intelligence. Springer, Berlin ; New York, 1999.

[Lon] John Longley. Complexity and Character of Human Languages - Informatics 2a: Lecture 25. page 26.

[Mar10] Marco Fernandez. *Encoding Logical Words as Quantum Gates: The Higher Dimensional Case*. PhD thesis, University of Oxford, 2010.

[Mat07] Matteo Capelletti. *The Non-Associative Lambek Calculus*. PhD thesis, University of Bologna, 2007.

[Mau] Luca Mauri. ALGEBRAIC THEORIES IN MONOIDAL CATEGORIES. page 26.

[Max66] Geach Peter & Black Max. *Translations from the philosophical writings of Gottlob Frege*. Basil Blackwell, 1966.

[Mel] Paul-Andr Mellis. Categorical Semantics of Linear Logic. page 213.

[Mel12] Paul-Andre Mellies. Game Semantics in String Diagrams. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 481–490, Dubrovnik, Croatia, June 2012. IEEE.

[MMMM88] Igor Aleksandrovic Mel'cuk, Igor? A. Mel??uk, Igor A. Mel?uk, and Igor? Aleksandrovi? Mel??uk. *Dependency Syntax: Theory and Practice*. SUNY Press, January 1988. Google-Books-ID: diq29vrjAa4C.

[MNP] Luke Maurits, Dan Navarro, and Amy Perfors. Why are some word orders more common than others? A uniform information density account. page 9.

[Moo14] Michael Moortgat. Typelogical Grammar. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2014 edition, 2014.

[MR12] Richard Moot and Christian Retor. Lambek Calculus and Montague Grammar. In Richard Moot and Christian Retor, editors, *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*, Lecture Notes in Computer Science, pages 65–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[MT15] Robert McColl Millar and Larry Trask. *Trask's Historical Linguistics*. Routledge, February 2015. Google-Books-ID: gGzABgAAQBAJ.

[noaa] Martin-Lf dependent type theory in nLab.

[noab] The n-Category Caf.

[noac] On certain formal properties of grammars - ScienceDirect.

[noad] Practical Foundations of Mathematics.

[noae] PROP in nLab.

[noaf] r/compsci - I've heard of context-free grammars (CFGs), but are context-sensitive grammars used in any applications?

[noag] r/compsci - Why is there relatively little work done on linear-bounded automata compared to other types?

[noah] ResearchGate.

[noai] r/math - Interesting but short examples of context-sensitive grammars.

[noaj] star-autonomous category in nLab.

[noak] syntactic category in nLab.

[noal] term model in nLab.

[Pau] Paul Taylor. Category theory and type theory.

[Paw] Pawel Sobocinski. Nicolas Bourbaki.

[Pel94] Francis Jeffry Pelletier. The Principle of Semantic Compositionality. *Topoi*, 13(1):11–24, March 1994.

[Pen71] Roger Penrose. Applications of negative-dimensional tensors. 1971.

[Per17] Paolo Perinotti. Causal structures and the classification of higher order quantum computations. *arXiv:1612.05099 [quant-ph]*, pages 103–127, 2017. arXiv: 1612.05099.

[Pet] Wiebke Petersen. Representation of Concepts as Frames. page 25.

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor Optics: Modular Data Accessors. *The Art, Science, and Engineering of Programming*, 1(2):7, April 2017.

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, Cambridge, Mass, September 1991.

[Pin15] Steven Pinker. *Language, Cognition, and Human Nature*. OUP USA, reprint edition edition, October 2015.

[Pir01] Teimuraz Pirashvili. On the PROP corresponding to bialgebras. *arXiv:math/0110014*, October 2001. arXiv: math/0110014.

[PL07] Anne Preller and Joachim Lambek. Free compact 2-categories. *Mathematical Structures in Computer Science*, 17(2):309–340, April 2007.

[Pot] Chris Potts. Distributional approaches to word meanings. page 20.

[PS01] Geoffrey K. Pullum and Barbara C. Scholz. On the Distinction between Model-Theoretic and Generative-Enumerative Syntactic Frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retor, editors, *Logical Aspects of Computational Linguistics*, Lecture Notes in Computer Science, pages 17–43. Springer Berlin Heidelberg, 2001.

[Pul] Georey K Pullum. Contrasting Applications of Logic in Natural Language Syntactic Description. page 23.

[Rad10] Andrew Radford. *Syntax: A Minimalist Introduction*. Cambridge University Press, Cambridge England ; New York, NY, USA, January 2010.

[Ram] Revantha Ramanayake. An introduction to the display calculus. page 67.

[RDL17] Anna Rogers, Aleksandr Drozd, and Bofang Li. The (too Many) Problems of Analogical Reasoning with Word Vectors. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 135–148, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[Ril18] Mitchell Riley. Categories of Optics. *arXiv:1809.00738 [math]*, September 2018. arXiv: 1809.00738.

[Rob14] Robin Piedeleu. *Ambiguity in Categorical Models of Meaning*. PhD thesis, University of Oxford, 2014.

[RS97] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*. January 1997.

[RVHV] Lectured David Reutter, Jamie Vicary, Notes Chris Heunen, and Jamie Vicary. Categorical Quantum Mechanics. page 299.

[Sau95] Ferdinand de Saussure. *Cours de linguistique gnrale*. Payot, Paris, September 1995.

[SCC13] Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. The Frobenius anatomy of word meanings I: subject and object relative pronouns. *Journal of Logic and Computation*, 23(6):1293–1317, December 2013. arXiv: 1404.5278.

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. *arXiv:0908.3347 [math]*, 813:289–355, 2010. arXiv: 0908.3347.

[Shi85] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343, August 1985.

[Sim] Alex Simpson. Context-sensitive languages - Informatics 2a: Lecture 28. page 25.

[Smi74] W. John Smith. Perspectives in Zoosemiotics by Thomas A. Sebeok. *Bird-Banding*, 45:82–83, January 1974.

[SSVW16] Patrick Schultz, David I. Spivak, Christina Vasilakopoulou, and Ryan Wisnesky. Algebraic Databases. *arXiv:1602.03501 [cs, math]*, February 2016. arXiv: 1602.03501.

[Str04] Ross Street. Frobenius monads and pseudomonoids. *Journal of Mathematical Physics*, 45(10):3930–3948, October 2004.

[Str12] Ross Street. Monoidal categories in, and linking, geometry and algebra. *arXiv:1201.2991 [math]*, January 2012. arXiv: 1201.2991.

[SVA96] Ross Street, Dominic Verity, and Andr Joyal. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, April 1996.

[Tes] Lucien|Osborne Tesniere. *Elements of Structural Syntax*. John Benjamins Publishing Company.

[Tra14] R. L. Trask. *Introducing Linguistics: A Graphic Guide*. Icon Books Ltd, 2 edition edition, June 2014.

[Tro] S N Tronin. Abstract Clones and Operads. page 10.

[Tro08] S. N. Tronin. Multicategories and varieties of many-sorted algebras. *Siberian Mathematical Journal*, 49(5):944–958, September 2008.

[Tro16] S. N. Tronin. On algebras over multicategories. *Russian Mathematics*, 60(2):52–61, February 2016.

[Vin19] Vincent Wang. Concept Functionals. In *SEMSPACE*, Riga, 2019.

[Vor01] Alexander A. Voronov. Notes on universal algebra. *arXiv:math/0111009*, November 2001. arXiv: math/0111009.

[VSW94] K. Vijay-Shanker and D. J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546, November 1994.

[Wey16] Hermann Weyl. *Symmetry*. Princeton University Press, Princeton, New Jersey, reprint edition edition, October 2016.

[Wie61] Wiener. *Cybernetics: Or Control and Communication in the Animal and the Machine*. MIT Press, Cambridge, Mass, second edition edition edition, January 1961.

[WP03] Dominic Widdows and Stanley Peters. Word Vectors and Quantum Logic Experiments with negation and disjunction. *Stanford University*, page 14, 2003.

[WR07] Ludwig Wittgenstein and Bertrand Russell. *Tractatus Logico-Philosophicus*. Cosimo Classics, New York, NY, May 2007.

[Yaa15] Yaared Al-Mehairi. *Compositional Distributional Cognition*. PhD thesis, University of Oxford, 2015.

[Yau08] Donald Yau. Higher dimensional algebras via colored PROPs. *arXiv:0809.2161 [math-ph]*, September 2008. arXiv: 0809.2161.