WHAT IS APPLIED CATEGORY THEORY?
What is applied category theory?

Upon hearing the phrase “applied category theory,” you might be thinking either one of two thoughts:

#1  Applied category theory? Isn’t that an oxymoron?

#2  Applied category theory? What’s the hoopla? Hasn’t category theory always been applied?

For those thinking thought #1, I hope to convince you that the answer is No way! It’s true that category theory sometimes goes by the name of general abstract nonsense, which might incline you to think that category theory is too pie-in-the-sky to have any impact on the “real world.” My hope is that these notes will convince you that that’s far from the truth!

For those thinking thought #2, yes it’s true that ideas and results from category theory have found applications in computer science and quantum physics (not to mention pure mathematics itself), but these are not the only applications to which the word applied in applied category theory is being applied. So what is applied category theory?

Read on.

A quick note to the reader

Before we get started, I’ll mention that this document is a collection of notes I amassed while participating in the 2018 Applied Category Theory Adjoint School—a wonderful online seminar that ran from January - April 2018 and culminated in a two-week workshop at the Lorentz Center in May 2018. I had a blast learning from the folks there, and I want to share some of the things I learned with anyone who’s interested. (So thanks for being interested!) Later, I’ll describe a couple of the research projects discussed during the workshop. Much of the information in this PDF can be found in various journal articles, blog posts, and videos of conference talks, most of which are freely available online. I’ve provided citations to these throughout. Here are a few other things to know:

• I’ll assume the reader is comfortable with the basics of category theory: categories, functors and natural transformations. For a friendly introduction to these topics, feel free to browse through the articles from my blog Math3ma listed in the margin.

• I’ll make heavy use of hyperlinks, as I have already, and I’ll also incorporate the occasional use of color throughout the text. For these reasons, it’s probably best to read this PDF on a computer rather than in print form.

• Finally, a fair warning: I use italics a lot (along with frequent parenthetical remarks). I also like exclamation points! And many of my sentences begin with a conjunction.

For a gentle introduction to (pure) category theory, here are a few places to start:

– What is Category Theory, Anyway?
– What is a Category?
– What is a Functor?
– What is a Natural Transformation?

At the first link, you’ll find a list of other recommended resources for learning about category theory.

My gratitude goes to the participants and mentors of the 2018 ACT Workshop from whom I learned a great deal. I also thank John Baez, Joseph Hirsh, Maximilien Péroux, and Todd Trimble for providing valuable feedback on a first draft of these notes.
Introduction

One of the great features of category theory, birthed in the 1940s, is that its organizing principles have been used to reshape and reformulate problems within pure mathematics, including topology, homotopy theory and algebraic geometry. Category theory has light on those problems, making them easier to solve and opening doors for new avenues of research. Historically, then, category theory has found immense application within mathematics. As John Baez recently noted, “[category theory] was meant to be applied.”

More recently, however, category theory has found applications in a wide range of disciplines outside of pure mathematics—even beyond the closely related fields of computer science and quantum physics. These disciplines include chemistry, neuroscience, systems biology, natural language processing, causality, network theory, dynamical systems, and database theory to name a few. And what do they all have in common? That’s much of what current-day applied category theory is seeking to discover. In other words, the techniques, tools, and ideas of category theory are being used to identify recurring themes across these various disciplines with the purpose of making them a little more formal. And that’s what the phrase applied category theory (ACT) refers to in these notes. As explained on the ACT 2018 workshop webpage,...we should treat the use of categorical concepts as a natural part of transferring and integrating knowledge across disciplines. The restructuring employed in applied category theory cuts through jargon, helping to elucidate common themes across disciplines. Indeed, the drive for a common language and comparison of similar structures in algebra and topology is what led to the development category theory in the first place, and recent hints show that this approach is not only useful between mathematical disciplines, but between scientific ones as well.

Of course, one of the challenges of using category theory to transfer and integrate knowledge across disciplines is making category theory itself accessible to the broader scientific audience. John Baez and Brendan Fong address this very point in their 2016 paper on electrical circuit diagrams¹:

While diagrams of networks have been independently introduced in many disciplines, we do not expect formalizing these diagrams to immediately help the practitioners of these disciplines. At first the flow of information will mainly go in the other direction: by translating ideas from these disciplines into the language of modern mathematics, we can provide mathematicians with food for thought and interesting new problems to solve. We hope that in the long run mathematicians can return the favor by bringing new insights to the table.

Although their comments refer to a particular project, they can apply to the field at large, too.

The goal of this document is to give a taste of applied category from a graduate student’s perspective. In doing so, I’ll share two themes and two constructions that appeared frequently during the ACT 2018 workshop. The math underlying these themes and constructions is not new. The newness, rather, is in how they are being applied. To illustrate the themes and constructions, I’ll also share two examples—two research projects in the field of ACT. The first project relates to chemistry and the second to natural language processing, though the expositions are weighted unevenly. I’ll devote considerably more time on the second example since that’s where my own research interests lie. And that’s what’s on the carte du jour! Two themes and two constructions and two examples, along with a few crumbs (i.e. digressions) in between. Here’s the menu in more detail:

¹ A Compositional Framework for Passive Linear Networks, https://arxiv.org/pdf/1504.05625.pdf
Contents

1  Two Themes  5
  1.1  Functorial Semantics  5
  1.2  Compositionality  11
  1.3  Further Reading  12
2  Two Constructions  13
  2.1  Monoidal Categories  13
  2.2  Decorated Cospans  28
  2.3  Further Reading  29
3  Two Examples  30
  3.1  Chemical Reaction Networks  30
  3.2  Natural Language Processing  35
  3.3  Further Reading  48
4  But Wait! There’s More...  50

Although the items are listed linearly, they are very much intertwined. The themes motivate the constructions; the constructions embody the themes, and both the themes and the constructions come to life in the examples.

I like to imagine that category theory is like a cup of black coffee, while fields outside of pure mathematics are like fresh cream. Both are lovely on their own, but blending them makes for a beverage par excellence.

I hope you’ll enjoy it as much as I do!
1 Two Themes

Two themes that appear over and over (and over and over and over) in applied category theory are functorial semantics and compositionality. Let’s talk about the first one first.

1.1 Functorial Semantics

Functorial semantics relates to the idea that a structure-preserving functor between categories

\[ C \to D \]

can be viewed as an interpretation of \( C \) within \( D \). It’s often helpful to think of \( C \) as somehow encoding for syntax while \( D \) provides semantics. Syntax refers to rules for putting things together and semantics refers to the meaning of those things. A functor

\[ \text{syntax} \to \text{semantics} \]

provides a way to bring the syntax to life.

To get a better idea of syntax vs. semantics, think of the English language where two important features of communication are 1) grammar, which provides rules for combining words to form sentences, and 2) the actual meaning conveyed by those words and sentences. Grammar is the syntax, and the meaning is the semantics.

\[ \text{grammar} \leftrightarrow \text{syntax} \quad \text{meaning} \leftrightarrow \text{semantics}. \]

Of course, neither is useful on their own. For instance, it’s easy to come up with a sentence that is grammatically correct and yet has no meaning. That’s the whole point behind MadLibs! As another example, here’s a sentence attributed to linguist Noam Chomsky:

Colorless green ideas sleep furiously.

It is grammatically correct, yet it has no meaning. The point here is that syntax vs. semantics is nothing new. So when an applied category theorist wants to model some phenomena in the “real world,” don’t be surprised if their model is ultimately a functor from a syntax category to a semantics category!
A small-ish digression...

Even though the idea goes by the fancy name of functorial semantics, it is not just a “category theory thing.” Mind if I digress for a while to elaborate on this?

If you know a little bit about groups, then you’ve seen functorial semantics in action before! How so? A group is a set endowed with some extra structure, though that tells us nothing about why groups are useful. It’s better to think of a group as encoding for some kind of action or transformation. And this is why group representations are so great! A group representation provides a way to view your abstract group elements as concrete linear transformations of some vector space. Explicitly, given a vector space \( V \), a group representation is a group homomorphism from \( G \) to \( \text{Aut}(V) \), the group of all automorphisms of \( V \)

\[
G \rightarrow \text{Aut}(V)
\]

It assigns to each group element a linear isomorphism \( V \rightarrow V \).

As a quick example, suppose our group is \( D_3 \), the dihedral group of order 6, which is the group of symmetries of an equilateral triangle. If we were to look at a presentation of the group,

\[
D_3 = \langle r, s \mid r^3 = s^2 = rsrs = 1 \rangle
\]

it might not seem to have anything to do with triangles. Fortunately, a representation of \( D_3 \) makes the connection clearer by assigning to each group element \( r \) and \( s \) a linear transformation of the real plane. Specifically, the standard representation of \( D_3 \rightarrow \text{Aut}(\mathbb{R}^2) \) assigns to each of \( r \) and \( s \) an invertible \( 2 \times 2 \) matrix with real entries:

\[
r \mapsto R = \begin{bmatrix}
\cos(2\pi/3) & -\sin(2\pi/3) \\
\sin(2\pi/3) & \cos(2\pi/3)
\end{bmatrix} \quad s \mapsto S = \begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\]

Here \( R \) is a rotation by 60° while \( S \) is reflection across the x-axis. Moreover \( R^3 \) and \( S^2 \) and \( RSRS \) each are equal to the \( 2 \times 2 \) identity matrix, which is exactly what we would expect: rotating an equilateral triangle by one full revolution leaves it unchanged, as does reflecting it twice in a row, and so on.

More generally then, we can think of a group \( G \) as providing the syntax while automorphisms \( \text{Aut}(V) \) provide the semantics

\[
G \leftrightarrow \text{syntax} \quad \text{Aut}(V) \leftrightarrow \text{semantics}
\]

So a group representation is like a (structure-preserving) morphism

\[
\text{syntax} \rightarrow \text{semantics}
\]
In fact... it’s not *like* that. It **IS** that. If we view both the groups $G$ and $\text{Aut}(V)$ as one-object\(^2\) categories, then a group representation

$$G \to \text{Aut}(V)$$

**IS** a functor from syntax to semantics. That’s because every group homomorphism *is* a functor when the groups are viewed as one-object categories! So although *functorial semantics* has the word “functor” in it, don’t think that the idea behind it is unique to category theory. Indeed, representation theory capitalizes on the relationship between syntax and semantics: a *representation* assigns to an abstract algebraic gadget (the syntax) some concrete meaning (the semantics).

I **COULD END OUR DIGRESSION** here, but I’d like to share one more instance of functorial semantics at work in pure mathematics. The next few examples involve monoids and monoidal categories, so I’ll assume you are familiar with those words. If you are not familiar with those words, don’t fret—you’re in luck! Section 2.1 is all about monoids and monoidal categories, so feel free to read that section first then come back here. In either case, let’s proceed with another neat example of functorial semantics in action:

**Example:** a *monoid* is the image of a functor from a certain syntax category to a certain semantics category.

More specifically,\(^3\)

\[
\begin{array}{ccc}
\text{a monoid} & \quad & \text{a lax monoidal functor} \\
\mathbb{1} & \xrightarrow{\cong} & \text{Set}
\end{array}
\]

Here I’m viewing both $\mathbb{1}$ and $\text{Set}$ as monoidal categories. The symbol $\mathbb{1}$ is meant to represent the category with one object and only one morphism (the identity), which we can view as a monoidal category $(\mathbb{1}, \otimes, \mathbb{1})$ in exactly one way. The category of sets has a monoidal structure given by the Cartesian product with the set containing one element, denoted {*} , as the monoidal unit. *Technically* then,

\(^3\) Every group $G$ gives rise to a category having a single object $\bullet$ (the group itself) and a morphism $\bullet \mapsto \bullet$ for each group element $g \in G$. Composition is given by the group operation.

Here’s another example I can’t resist sharing: *operads!* If you’re not familiar with operads, just know that this is a souped-up version of the group theory example. If you are familiar with operads, then you know this is the souped-up version of the group theory example.

An operad is an example of syntax, while an algebra over that operad provides the semantics. For example, given a vector space $V$, an operad homomorphism from the [commutative, associative, Lie, Poisson,...] operad to the endomorphism operad on $V$ **IS** a [commutative, associative, Lie, Poisson,...]-algebra! That is, the structure-preserving homomorphism provides an interpretation of each abstract $n$-ary operation as a actual, concrete operation $V^\otimes_n \to V$.

This next comment is really digressing from the digression, but: I also like to think of simplicial sets as an instance of functorial semantics. A simplicial set $X$ is a bit like syntax, while a topological space is like semantics. **Geometric realization** $X \mapsto |X|$ provides a map from one to the other.

\(^3\) A functor $F : \mathcal{C} \to \mathcal{D}$ between monoidal categories is called **lax monoidal** if for every pair of objects $c, c'$ in $\mathcal{C}$ there is a morphism $Fc \otimes Fc' \to F(c \otimes c')$ (which assembles into a natural transformation.) It’s called **strong monoidal** if $Fc \otimes Fc' \cong F(c \otimes c')$, and it’s called **strict monoidal** if $Fc \otimes Fc' = F(c \otimes c')$. 
In footnote 3 you’ll notice I dropped parentheses and wrote \( Fc \) rather than \( F(x) \). The reason for my preference is categorical! Let me explain by saying a few words about sets: Did you know that an element \( x \) in a set \( X \) is the same thing as a function \( \{\ast\} \to X \), where \( \{\ast\} \) denotes the one-element set? It’s true. The function \( \{\ast\} \to X \) is uniquely determined by where it sends that one point \( \ast \). So since an element \( x \) is the same thing as an arrow \( \{\ast\} \to X \), we might as well label that arrow by \( x \), \( x: \{\ast\} \to X \). Now if we have another function \( f: X \to Y \), then an element \( f(x) \in Y \) is precisely the composition

\[
\ast \xrightarrow{f} X \xrightarrow{x} Y
\]

That is,

\[
f(x) = f \circ x = fx
\]

where on the right hand side, I’ve omitted the composition symbol \( \circ \) because it’s cleaner. So there you have it! An element \( f(x) \in Y \) is the same as a function \( fx \). And since categorically-minded folks (such as you and I) prefer arrows over elements (Because of Yoneda. Also, we might be rethinking set theory.), the notation \( fx \)—and more generally, \( Fc \) as above—is preferred. By the way, this is all related to Lawvere’s philosophy of generalized elements, which is the idea that a morphism \( A \to B \) is really an “\( A \)-shaped element in \( B \).” For some examples, check out the articles “A Diagram is a Functor” as well as “The Yoneda Embedding” on Math3ma. Generalized elements are closely related to functorial semantics, so both links are worth a read!

Why is this true? First observe that a functor \( F: \mathbf{1} \to \text{Set} \) picks out a set, \( F\mathbf{1} := M \). And the data of a lax monoidal functor \( F: (\mathbf{1}, \otimes, \mathbf{1}) \to (\text{Set}, \times, \{\ast\}) \) consists of a morphism

\[
\bullet: F\mathbf{1} \times F\mathbf{1} \to F(\mathbf{1} \otimes \mathbf{1}) \quad \text{i.e.} \quad \bullet: M \times M \to M
\]

along with a morphism

\[
1: \{\ast\} \to F\mathbf{1} \quad \text{i.e.} \quad 1: \{\ast\} \to M
\]

both of which are required to fit into some commuting diagrams. I won’t write them here, but one diagram says “\( \bullet \) is associative” and the other diagram says, “\( 1 \) serves as an identity for \( \bullet \).” In summary, the data of a lax monoidal functor \( F: (\mathbf{1}, \otimes, \mathbf{1}) \to (\text{Set}, \times, \{\ast\}) \) are

i) a set \( M \)

ii) an associative binary operation \( \bullet: M \times M \to M \)

iii) a special element \( 1 := 1(\{\ast\}) \in M \) that serves as a “multiplicative identity” for \( \bullet \).

This triple \( (M, \bullet, 1) \) is precisely a monoid! Or to borrow from Lawvere’s terminology, the functor (equivalently, the monoid) is one interpretation of the category \( \mathbf{1} \) in the \( \text{Set} \). Interestingly, \( \mathbf{1} \) may be interpreted in other categories as well. This leads to other familiar monoidal structures. Indeed, if we replace \( (\text{Set}, \times, \{\ast\}) \) by any monoidal category \( (\mathbf{C}, \otimes, 1) \), then a lax monoidal functor

\[
(\mathbf{1}, \otimes, \mathbf{1}) \to (\mathbf{C}, \otimes, 1)
\]

is a monoid in the category \( \mathbf{C} \). Sometimes this monoid goes by a familiar name. Here are some examples.
1. **Topological Monoid.** Let \((\text{Top}, \times, *)\) denote the category of topological spaces and continuous functions, viewed as a monoidal category with the Cartesian product \(\times\), with the one-point space \(*\) as monoidal unit. A lax monoidal functor
\[
(1, \otimes, 1) \to (\text{Top}, \times, *)
\]
is a topological monoid. That is, **a topological monoid is a monoid in the category of topological spaces.**

2. **Ring.** Let \((\text{AbGroup}, \otimes, \mathbb{Z})\) denote the category of abelian groups (viewed as \(\mathbb{Z}\)-modules) and abelian group homomorphisms, viewed as a monoidal category with the tensor product \(\otimes\), with the integers \(\mathbb{Z}\) as monoidal unit. A lax monoidal functor
\[
(1, \otimes, 1) \to (\text{AbGroup}, \otimes, \mathbb{Z})
\]
is a ring (with unit). That is, **a ring is a monoid in the category of abelian groups.**

3. **Algebra.** Let \((\text{FVect}, \otimes, k)\) denote the category of finite-dimensional vector spaces over a field \(k\) and linear maps, viewed as a monoidal category with the tensor product \(\otimes\), with \(k\) as monoidal unit. A lax monoidal functor
\[
(1, \otimes, 1) \to (\text{FVect}, \otimes, k)
\]
is an algebra (with unit). That is, **an algebra is a monoid in the category of vector spaces.**

4. **Monad.** Let \(\mathcal{C}\) be a category and let \(\text{End}_{\mathcal{C}}\) denote the category whose objects are functors \(\mathcal{C} \to \mathcal{C}\) and whose morphisms are natural transformations. (So \(\text{End}_{\mathcal{C}}\) is the category of *endofunctors* on \(\mathcal{C}\).) Note that \(\text{End}_{\mathcal{C}}\) can be given the structure of a monoidal category: the monoidal product is composition of functors (i.e. if \(F, G\) are objects in \(\text{End}_{\mathcal{C}}\), then the monoidal product of \(F\) and \(G\) is \(F \circ G\)), and the monoidal unit is the identity functor \(1_{\mathcal{C}}\) on \(\mathcal{C}\) (i.e. \(1_{\mathcal{C}}\) assigns each object and morphism in \(\mathcal{C}\) to itself). Then a lax monoidal functor
\[
(1, \otimes, 1) \to (\text{End}_{\mathcal{C}}, \circ, 1_{\mathcal{C}})
\]
is a monad. That is, **a monad is a monoid in the category of endofunctors on \(\mathcal{C}\).**

You’ll notice that in each of these examples, a change in the semantics category \(\mathcal{C}\) gives rise to a different interpretation of \(1\), which served as our syntax category. Pretty neat, right? For more details on the examples, see Emily Riehl’s *Category Theory in Context* Definitions 1.6.3 and 5.1.1.
That’s the idea behind functorial semantics. Now, how is it used in applied category theory? We’ll see the answer when we look at the two examples—two research projects from the field—one from chemistry and one from natural language processing. In both examples, the key is the existence of a (structure-preserving) functor from a syntax category to a semantics category. Here’s a sneak preview:

In Section 3.1 we’ll see how the behavior of a chemical reaction network is modeled by a functor

\[ \text{Petri nets} \rightarrow \text{dynamical systems} \]

as shown in “A Compositional Framework for Reaction Networks” by John Baez and Blake Pollard. In Section 3.2, we’ll see how a model for natural language can be described by a functor

\[ \text{grammar} \rightarrow \text{meanings of words} \]

via the work of Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark in “Mathematical Foundations for a Compositional Distributional Model of Meaning”. And perhaps you’re wondering, “How do Petri nets, dynamical systems, grammar, and meanings of words form categories? And what’s a Petri net, anyway?” We’ll answer these questions in the pages to come, but first I’d like to introduce another important theme in applied category theory: compositionality.
1.2 Compositionality

Compositionality, also known as the principal of compositionality, also known as Frege’s principle, is the idea that the meaning of a complex expression is determined by

1. the meanings of its constituent parts, and
2. the rules for how those parts are combined.

Or, as succinctly stated on the homepage of the brand new journal of applied-category-theory-and-related-fields,

compositionality describes and quantifies how complex things can be assembled out of simpler parts.

As it turns out, the name of that journal is itself Compositionality, which hints at the importance of this concept within the field.

In Section 3.1, which contains our example from chemistry, the complex expression will be a network—a big complicated directed multigraph, if you like. Its constituent parts are simply smaller chunks of the network.

In Section 3.2, our example from natural language, the complex expression will be a sentence; its constituent parts are the words that comprise the sentence.

In both examples, functorial semantics and the principle of compositionality will go hand-in-hand. The former prompts us to model behavior using a functor between syntax and semantics categories. The latter encourages us to take things one at a time: To model a huge system, compositionality tells us, it’s enough to model smaller pieces of it and then stick those pieces together. Simple enough. But what does it mean to “stick pieces together” mathematically? The answer is provided by the structure of a monoidal category. And that is the first of our two main constructions in ACT.

\[ M = UDV^* \]

where \( U \) and \( V \) are unitary square matrices (here \( V^* \) denotes the conjugate transpose of \( V \)) and \( D \) is a rectangular diagonal matrix. Intuitively then, the linear transformation \( M \) can be broken down into a rotation followed by a shear followed by another rotation. So you can analyze your transformation (or your data set, if that’s what \( M \) is encoding) by understanding its constituent pieces—the factors—and how they compose together. More generally, I like to think that tensor networks are a good example of compositionality, but such a discussion might take us too far off course. Perhaps another day!
1.3 Further Reading

For more on functorial semantics and compositionality:

• Take a look at (this small notice on) William Lawvere’s 1963 PhD thesis “Functorial Semantics of Algebraic Theories” for the formal foundations for functorial semantics.

• You might also enjoy this discussion on doctrines over at the n-Category Café: https://golem.ph.utexas.edu/category/2006/09/doctrines.html

• The preface to Brendan Fong’s PhD thesis, “The Algebra of Open and Interconnected Systems”, has a nice discussion on the principal of compositionality and includes various references. And while you’re at it, take a look at the entire thesis, which is wonderfully written and provides the backbone of much of John Baez’s current research in network theory, which we’ll talk a little bit about in Section 3.1.

• Applied category theorist Jules Hedges has also written a nice exposition on compositionality, appropriately entitled “On Compositionality.” In the article, you’ll find a link to the Stanford Encyclopedia on Philosophy’s entry on compositionality, which gives a thorough overview of the topic.
2 Two Constructions

Two constructions that appear over and over (and over and over and over) in (some projects in) applied category theory are monoidal categories and decorated cospans. Let’s talk about the first one first.

2.1 Monoidal Categories

Actually, before we talk about monoidal categories, let’s talk about monoids. Here are three examples of monoids: the integers \( \mathbb{Z} \), the rational numbers \( \mathbb{Q} \), and the set of all \( n \times n \) matrices with real-number entries \( M_n(\mathbb{R}) \). Well, technically these are monoids:

\[
\begin{align*}
(Z, +, 0) & \\
(Q, \cdot, 1) & \\
(M_n(\mathbb{R}), \cdot, I) &
\end{align*}
\]

Each example consists of a set \( X \) equipped with an associative binary operation, which I’ll denote by \( \bullet \). Moreover, there is a special element in the set, let’s call it \( 1 \), that serves as an identity for the operation. Those three things—a set, an associative binary operation, an identity—comprise a monoid. Usually, we write this triple as

\[
(X, \bullet, 1)
\]

Not too bad, right?

Great. Now imagine replacing the set \( X \) by a category \( \mathcal{C} \), and replacing the binary operation \( \bullet : X \times X \to X \) by a functor \( \bullet : \mathcal{C} \times \mathcal{C} \to \mathcal{C} \), and replacing the identity element \( 1 \in X \) by an object \( 1 \) in \( \mathcal{C} \). The resulting triad

\[
(\mathcal{C}, \bullet, 1)
\]

is called a monoidal category. The object \( 1 \) is often called the monoidal unit, and people usually prefer to write \( \otimes \) (and call it the monoidal product) instead of \( \bullet \) so let’s do that too:
What is applied category theory? 14

There's a little more to the story here since we want \( \otimes \) to be associative: for any objects \( A, B, C \) in the category \( C \), we want \( A \otimes (B \otimes C) = (A \otimes B) \otimes C \). Alas, things are rarely equal on the nose. To compensate for this, we ask instead that there exist an isomorphism 
\[
A \otimes (B \otimes C) \cong (A \otimes B) \otimes C,
\]
which should behave nicely. I won't go into the details here, but of course you can find more on the Wikipedia page on monoidal categories. For a delightful exposition on the richness of monoidal categories, I strongly recommend "A Rosetta Stone" by John Baez and Mike Stay. It is a gem.

In short, a monoidal category is a category in which it makes sense to "combine" objects and morphisms. As we'll see in Sections 3.1 and 3.2, each of the four categories mentioned on page 10—Petri nets, dynamical systems, grammar, and meanings of words—are monoidal categories! Here are some more examples.

\[
\begin{align*}
\text{(Set, } \times, \{\ast\}\text{)} & \quad \text{the category of sets, with the Cartesian product. The monoidal unit is the set with one element, } \{\ast\}. \\
\text{(Top, } \sqcup, \emptyset\text{)} & \quad \text{the category of topological spaces with disjoint union. The monoidal unit is the empty set.} \\
\text{(FVect, } \otimes, \mathbb{R}\text{)} & \quad \text{the category of finite-dimensional vector spaces over a field } \mathbb{R} \text{ with tensor product. The monoidal unit is } \mathbb{R}.
\end{align*}
\]

By the way, if there is an isomorphism \( A \otimes B \cong B \otimes A \) for all objects \( A \) and \( B \) that behaves nicely in a sense that can be made precise, then we say that \( (C, \otimes, 1) \) is a symmetric monoidal category. Each of the three examples above are symmetric monoidal. Monoidal categories come in other flavors too (braided, Cartesian, closed, Cartesian closed, closed braided,...), depending on which properties are satisfied.

The main takeaway here is that monoidal categories are the bread

\( 5 \) Allow me to explain the "and morphisms" part. First remember, \( \otimes \) is a functor! That means it's an assignment on objects \textit{and} on morphisms. Consider \( (\text{Set}, \times, \{\ast\}) \), for example, where \( \times \) assigns to a pair of sets \( (A, B) \) their Cartesian product \( A \times B \). And given two functions \( f : A \to B \) and \( g : A' \to B' \), it assigns to the pair \((f, g)\) the function \( f \times g : A \times A' \to B \times B' \), which is defined by: \( (f \times g)(a, b) := (fa, gb) \). This is one example of the action of the monoidal product on morphisms. More generally, what \( f \otimes g \) is depends on the explicit definition of \( \otimes \).
and butter of many applied category theorists. One reason for this is that monoidal categories provide a good setting in which to view morphisms as *physical processes* and objects $A, B, \ldots$ as *states*. As a non-technical example, let’s suppose $A$ is a bunch of lemon meringue pie ingredients while $B$ is a fully-assembled-yet-unbaked lemon meringue pie. We might view a morphism $A \to B$ as the process of mixing the raw ingredients together and then pouring the resulting concoction into a pre-baked crust.

As it turns out, this pie example isn’t so silly after all. It’s one of the motivating examples that Brendan Fong and David Spivak use in their excellent book *Seven Sketches in Compositionality: An Invitation to Applied Category Theory* to illustrate both the ubiquity and the simplicity of monoidal categories. (If you haven’t read *Seven Sketches* yet, you really must.) Below is a copy of their lemon meringue pie diagram, where I’ve drawn our $A$ and $B$ on the left as input and right as output.

Now that we’ve zoomed in, we can see that our process $A \xrightarrow{\text{prepare lemon meringue pie}} B$ is actually made up of a bunch of other processes! This isn’t too surprising as there are several steps that go into preparing a lemon pie: *separating the eggs, making the lemon filling, filling the crust*, and so on. Fong and Spivak’s diagram illustrates just how those individual steps combine to form the single process *prepare lemon meringue pie*. What’s neat is that we can describe these steps using the language of monoidal categories! We’ll go into more detail later in this section, but here’s a quick preview:
• The category’s composition \( \circ \) corresponds to using one box’s output wire as another box’s input wire. For example,

![Diagram of composition](image)

is really \( g \circ f \)

• The monoidal product \( \otimes \) corresponds to stacking boxes on top of each other. For example,

![Diagram of monoidal product](image)

is really \( f \otimes g \)

In other words, \( \circ \) means “do the processes in series” while the monoidal product \( \otimes \) means “do the processes in parallel.” The resulting picture is called a **string diagram**—a graphical representation of a process (or equation of processes) in a monoidal category. I’ll give more detail on how string diagrams work in a second. But first, I’m reminded of something else about monoidal categories that I want to tell you! So let me tell you this new bit of information, then we’ll come back to string diagrams. This small digression will, in fact, tie things together quite nicely. Bear with me.

**Earlier, I mentioned** that the word “symmetric” can be used as an adjective for “monoidal categories”:

![Symmetric monoidal categories](image)

Similarly, there is another flavor of monoidal categories that we should know about. This one will provide the main setting for our example in Section 3.2:

![Compact closed categories](image)

I’ll explain.
Interlude: Compact Closed Categories and String Diagrams

Do you know what a finite dimensional vector space over $\mathbb{R}$ is? Then you know what a compact closed category is! Or rather, you know an example of one:

$\text{FVect}$ is a compact closed category.

What makes that sentence true? Answer: every finite dimensional vector space $V$ has a dual space $V^* = \text{hom}(V, \mathbb{R})$.

That’s it.

A compact closed category just the name for a monoidal category in which every object has a dual. But what does “has a dual” mean? In other words, what makes a dual dual? Before I tell you the answer, let’s think back to the category $\text{FVect}$, by way of motivation, and let’s assume each vector space comes with an inner product. In this case, there are two very important linear maps between the ground field—let’s say it’s $\mathbb{R}$ for now—and a vector space $V$ tensored with its dual:

$$\eta_V : \mathbb{R} \to V \otimes V^* \quad \epsilon_V : V^* \otimes V \to \mathbb{R}$$

In fact, there’s a nice fact from linear algebra, namely that once we fix a basis $\{e_1, \ldots, e_n\}$ for $V$ then there is an isomorphism $V \cong V^*$. So let’s fix that basis (the standard one) and write $V$ instead of $V^*$. Also, the subscript is a little cumbersome, so let’s drop it for now. So we have two maps

$$\eta : \mathbb{R} \to V \otimes V \quad \epsilon : V \otimes V \to \mathbb{R} \quad (1)$$

The map $\eta$ is called the unit, and it assigns to every real number a vector in $V \otimes V$, namely:

$$\eta(1) = \sum_{i=1}^n e_i \otimes e_i \quad \text{(and extend linearly)}$$

The map $\epsilon$ is called the counit, and it assigns to every vector in $V \otimes V$ a real number, namely:

$$\epsilon \left( \sum_{i,j} c_{ij} v_i \otimes w_j \right) = \sum_{i,j} c_{ij} (v_i \cdot w_j) \quad \text{where} \cdot \text{is the inner product}$$

Intuitively, we can think of $\epsilon$ as an evaluation map. That’s because there is always a map $V^* \otimes V \to \mathbb{R}$ given by evaluation. Indeed, if $v \in V$ and $f \in V^*$, then we can pair the two together to obtain $fv \in \mathbb{R}$.
And if we view $f$ as a $1 \times n$ matrix and $v$ as an $n \times 1$ matrix, then $fv$ is their inner product:

$$\begin{bmatrix} \ldots f \ldots \\ \vdots \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ v \end{bmatrix} = \text{a number}$$

The $\epsilon$ map just extends this linearly. That is, if we now have any vector $\sum_{i,j} c_{ij} v_i \otimes w_j$ in $V \otimes V$, then $\epsilon : V \otimes V \to \mathbb{R}$ is given by

$$\epsilon \left( \sum_{i,j} c_{ij} v_i \otimes w_j \right) = \sum_{i,j} c_{ij} (v_i \cdot w_j)$$

as above.

Finally, the unit $\eta$ and counit $\epsilon$ interact nicely with each other because they satisfy some equations called the yanking equations, which I’ll explain shortly. The bottom line is that all the above—the maps $\eta$ and $\epsilon$ and the equations they satisfy—makes $V^*$ into a bona fide dual for $V$. The upshot is that compact closed categories generalize these notions.

**Definition 2.1.** A **compact closed category** is a monoidal category $(C, \otimes, 1)$ where for every object $c$ in $C$ there exists objects $c^l$ and $c^r$ and morphisms

$$\eta^l_c : 1 \to c \otimes c^l \quad \quad \epsilon^l_c : c^l \otimes c \to 1$$

$$\eta^r_c : 1 \to c^r \otimes c \quad \quad \epsilon^r_c : c \otimes c^r \to 1$$

that satisfy the “yanking (or snake) equations”

$$\left( \text{id}_c \otimes \epsilon^l \right) \circ (\eta^l \otimes \text{id}_c) = \text{id}_c \quad \quad \left( \epsilon^r \otimes \text{id}_c \right) \circ (\text{id}_c \otimes \eta^r) = \text{id}_c$$

$$\left( \epsilon^r \otimes \text{id}_c \right) \circ (\text{id}_c \otimes \eta^l) = \text{id}_{c^r} \quad \quad \left( \text{id}_{c^l} \otimes \epsilon^r \right) \circ (\eta^r \otimes \text{id}_{c^r}) = \text{id}_{c^l} \quad \quad (2)$$

Yikes. What do these equations MEAN?

I’m glad you asked.

To answer, it’s time to revisit our previous discussion on **string diagrams**! String diagrams are loved by applied category theorists far and wide because they make life SO much easier. As we saw earlier, a string diagram is a picture that represents morphisms in a monoidal category $C$. For now let’s take that category to be $\text{FVect}$ so that our objects are vector spaces $V, W, \ldots$. In this case, the left and right dual of space $V$ is its vector space dual

$$V^* = V^r = V^l$$
We’ll get to the yanking equations shortly, but first: If this document has been your first introduction into string diagrams, then here is THE KEY thing to know:

In category theory, we often draw an object as a dot $\bullet$ and a morphism as an arrow $\bullet \rightarrow \circ$. To draw a string diagrams, just do the opposite! (This goes back to Poincaré duality in topology.) To draw a string diagram, draw an object as an arrow and a morphism as a dot or, even better, a box.

The lemon pie diagram that we saw on page 15 is an example of a string diagram!

With this small artistic adjustment, we can represent the monoidal product $\otimes$ pictorially as well. The product of two spaces $V \otimes W$ is drawn as two lines, side-by-side. A similar picture holds for the product of two morphisms. Composition $\circ$ is represented by gluing strings together.

And as we saw above, every object in a compact closed category such as $\bf{FVect}$ has a dual. Its picture is given by an arrow pointing in the opposite direction.

Another special object in a compact closed category is the monoidal unit, for instance $\mathbb{R}$ in $\bf{FVect}$. Because the unit is an object, it’s depicted as an arrow, too. People like to draw this arrow in a special way, namely as the “empty” arrow. In other words, people don’t like to draw an arrow. That’s because the monoidal unit $\mathbb{R}$ satisfies

$$V \otimes \mathbb{R} \cong V \cong \mathbb{R} \otimes V \quad \text{for all } V,$$

More generally, the monoidal unit $1$ in a monoidal category $(C, \otimes, 1)$ satisfies $1 \otimes c \cong c \cong c \otimes 1$ for all objects $c$ in $C$. 

which suggests that the unit is “invisible.” But I like to draw it anyway, shaded:

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\mathbf{V} \boxtimes \mathbf{R} \\
\longrightarrow \mathbf{V} \\
\end{array} \\
\mathbf{V} \end{array} \\
\end{array} \cong \begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\mathbf{R} \boxtimes \mathbf{V} \\
\longrightarrow \mathbf{V} \\
\end{array} \\
\mathbf{V} \end{array} \end{array}
\]

Now we are ready to get back to the yanking equations. Remember, part of the data of a compact closed category is that each object \( \mathbf{V} \) has left and right duals \( \mathbf{V}' \), \( \mathbf{V}^l \) together with morphisms

\[
\begin{align*}
\eta^l_V &: \mathbf{R} \to \mathbf{V} \otimes \mathbf{V}' \\
\eta^r_V &: \mathbf{R} \to \mathbf{V}' \otimes \mathbf{V} \\
\epsilon^l_V &: \mathbf{V}' \otimes \mathbf{V} \to \mathbf{R} \\
\epsilon^r_V &: \mathbf{V} \otimes \mathbf{V}' \to \mathbf{R}
\end{align*}
\]

Again, to simplify the notation we’ll use the fact that that for vector spaces, \( \mathbf{V}^\ast = \mathbf{V}' = \mathbf{V}^l \). I’ll also drop the subscripts to keep things clean.

Graphically, the \( \eta \)s and \( \epsilon \)s are drawn as below. The reason we have two versions of each map is because the “information flow” can either flow up or it can flow down.

Alternatively, some folks will rotate the \( \epsilon \) and \( \eta \) diagrams by 90° clockwise and counterclockwise, respectively, which is the reason for their common nickname of “cups and caps.”

Note: the direction of the (invisible) arrow for the unit \( \mathbf{R} \) can go either way. The monoidal unit is always self dual!

And since \( \text{FVect} \) is a symmetric monoidal category, and since the left and right duals are both \( \mathbf{V}^\ast \), there is really only one unit and one counit for vector spaces.\(^\text{10}\)

\(^\text{10}\) Remember, the sentence “\( \text{FVect} \) is symmetric monoidal” means there is an isomorphism \( \mathbf{V} \otimes \mathbf{W} \cong \mathbf{W} \otimes \mathbf{V} \) for every pair of vector spaces \( \mathbf{V} \) and \( \mathbf{W} \). In string diagram calculus, this means that the order in which we draw our arrows doesn’t matter:
That is, in $\text{FVect}$

$$\eta = \eta' = \eta^! \quad \text{and} \quad \epsilon = \epsilon' = \epsilon^!,$$

and these are precisely the $\eta$ and $\epsilon$ defined on page 17! So in this example, the four yanking equations of (2) reduce down to just two:

$$\begin{align*}
(e \otimes \text{id}_V) \circ (\text{id}_V \otimes \eta) &= \text{id}_V \\
(id_{V^*} \otimes \epsilon) \circ (\eta \otimes \text{id}_{V^*}) &= \text{id}_{V^*}
\end{align*} \tag{3}$$

Graphically, these equations can be represented as follows:

For fun, verify that the unit and counit maps on page 17 do indeed satisfy these two equations.

After yanking the strings taut, you’ll notice that information flows rightwards in the first equation, while it flows leftwards in the second equation.
Since we’re in a symmetric monoidal category, nothing changes if we reverse the arrows in the pre-yanked strings. If, however, the monoidal product \( \otimes \) is not symmetric, then we obtain two more diagrams.

This gives a grand total of four equations—the four displayed in (2). And that’s what gives us a compact closed category. The category \( \text{FVect} \) together with the unit and counit defined in (1) will make another appearance in Section 3.2. In that same section, we’ll also seen an example of a compact closed category that is not symmetric.

By the way, a key feature of \( \text{FVect} \) (and more generally, all symmetric compact closed categories) is that processes, i.e. morphisms, \( V \to W \) are in bijection with states \( \mathbb{R} \to W \otimes V^* \cong V^* \otimes W \), which is the special name given to morphisms whose domain is the monoidal unit. This bijection is sometimes called process-state duality, and in the context of \( \text{FVect} \), it means we can view linear maps as vectors in a tensor product\(^{11}\) and vice versa!

\(^{11}\) While a linear map \( \mathbb{R} \to V^* \otimes W \) is not itself a vector in \( V^* \otimes W \), it can be identified with one, namely with the image of 1 in \( \mathbb{R} \). More generally, for any finite-dimensional vector space \( A \) over \( \mathbb{R} \), you can always think of \( \text{hom}(\mathbb{R}, A) \) as \( A \) itself, at least at the set level. That’s because the forgetful functor \( U: \text{FVect} \to \text{Set} \) is representable with representing object \( \mathbb{R} \). In other words, linear maps \( \mathbb{R} \to A \) are in one-to-one correspondence with the vectors in \( A \), viewed as elements of its underlying set, \( \text{hom}(\mathbb{R}, A) \cong UA \).

This is completely analogous to how functions \( \{*\} \to X \) from the one-point set to a set \( X \) are in one-to-one correspondence with the elements in \( X \), \( \text{hom}(\{*\}, X) \cong X \) and is another manifestation of the “probing” idea we saw in the margin on page 9.
I like to think of it this way: when $V$ and $W$ are $\mathbb{R}^n$ and $\mathbb{R}^m$ with the standard bases, process-state duality—taken together with the margin note on the previous page—is akin to the observation that matrices can be viewed as vectors and vice versa. That is, a linear map $\mathbb{R}^n \to \mathbb{R}^m$ has an $m \times n$ matrix representation which can be reshaped into an $nm \times 1$ column vector and then identified with a vector in $\mathbb{R}^n \otimes \mathbb{R}^m$. Conversely, there’s a way to identify a vector in $\mathbb{R}^n \otimes \mathbb{R}^m$ with an $nm \times 1$ column vector that can be reshaped into an $n \times m$ matrix, which gives rise to a linear map $\mathbb{R}^n \to \mathbb{R}^m$.

Aside: There is, I think, a nice categorical way to piece this together. First note that there is a category $\text{Mat}(\mathbb{R})$ whose objects are natural numbers $n, m, \ldots$ and whose morphisms $f : n \to m$ are $m \times n$ matrices with real entries. The identity $n \to n$ is the $n \times n$ identity matrix and composition is given by matrix multiplication. This category is actually a compact closed category! The monoidal product on objects is given by multiplication $n \otimes m := nm$ and on morphisms is given by the Kronecker product of matrices. The monoidal unit is $1 \in \mathbb{N}$. For the compact closed structure, each object is self-dual, $n^* := n$, and for each $n \in \mathbb{N}$ the unit map $\eta_n : 1 \to n^2$ is the $n^2 \times 1$ column vector obtained by stacking the standard bases vectors $e_1, \ldots, e_n$ on top of each other. In other words, $\eta$ is given by the Kronecker delta function $\eta_{ij} := \delta_{ij}$. The counit map $\epsilon_n : n^2 \to 1$ is the $1 \times n^2$ row vector obtained by taking the transpose of $\eta_n$. For example, if $n = 3$ then

$$
\eta_3 = \begin{bmatrix} 1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \end{bmatrix}^T \quad \text{and} \quad \epsilon_3 = \eta_3^\dagger.
$$

Since $\text{Mat}(\mathbb{R})$ is compact closed, it exhibits process-state duality, too:

$$
\text{hom}(n, m) \cong \text{hom}(1, nm)
$$

This correspondence is precisely the reshaping of $n \times m$ matrices into $nm \times 1$ column vectors and vice versa, which can be verified by using the unit and counit maps in a way analogous to the string diagrams shown at the bottom of the previous page. To tie this in to the remark about $\mathbb{R}^n$ and $\mathbb{R}^m$ in the previous paragraph, note that there is a functor $\text{FVect} \xrightarrow{\cong} \text{Mat}(\mathbb{R})$ sending a vector space $V$ to its dimension $\text{dim}(V)$ and a linear map $f : V \to W$ to its corresponding $\text{dim}(W) \times \text{dim}(V)$ matrix representation, and it defines an equivalence of categories! For details, see the discussion on page 30 as well as Corollary 1.5.11 of Emily Riehl’s *Category Theory in Context*.

As we’ll see in Section 3.2, process-state duality pairs very nicely with our intuition about language. There we’ll discover that a *verb* can either be represented as a vector in a tensor product of vector spaces or as a linear map, i.e. a process. In other words, a verb is an action in the eyes of both grammar and mathematics!
Another digression: A conjunction with adjunctions?

If you’re familiar with adjunctions in category theory, then you might wonder about this choice of naming and notation:

\[ \eta \rightarrow \text{“unit”} \quad \epsilon \rightarrow \text{“counit”} \]

Is it a coincidence that these two words are also used in the definition of an adjunction? NOPE. They are closely related. Specifically, the data \( V, V^*, \eta, \epsilon \) together with the yanking equations are an instance of a categorical adjunction! I think this is a neat fact, which appears on the first page of “Coherence for Compact Closed Categories” by Kelley and LaPlaza.

Let’s take yet another digression. Happily, it will tie in quite nicely with our discussion on string diagrams. We’ll begin by recalling the definition of an adjunction.

**Definition 2.2.** An adjunction between categories \( C \) and \( D \) is a pair of functors

\[
L : C \rightarrow D : R
\]

and a pair of natural transformations

\[
\eta : \text{id}_C \Rightarrow RL \\
\epsilon : LR \Rightarrow \text{id}_D
\]

called the unit and counit respectively, such that these two triangles commute:

\[
\begin{array}{ccc}
L & \cong & LRL \\
\downarrow \epsilon & & \downarrow \eta R \\
L & \cong & RLR \\
\end{array}
\]

The adjunction is denoted \( L \dashv R \), and \( L \) is said to be left adjoint to \( R \) while \( R \) is said to be right adjoint to \( L \).

Believe it or not, those commuting triangles—often called the triangle identities—are closely related to the yanking equations in (3)!

Indeed, “these triangles commute” means that these two equations hold:

\[
\begin{align*}
(e \circ L) \circ (L \circ \eta) & = \text{id}_L \\
(R \circ \epsilon) \circ (\eta \circ R) & = \text{id}_R
\end{align*}
\]

(4)

Now lets compare them to (3):

\[
\begin{align*}
(e \otimes \text{id}_V) \circ (\text{id}_V \otimes \eta) & = \text{id}_V \\
(\text{id}_{V^*} \otimes \epsilon) \circ (\eta \otimes \text{id}_{V^*}) & = \text{id}_{V^*}
\end{align*}
\]

Why are (3) and (4) so similar? What’s going on here? Is there a sense in which a vector space and its dual form an adjunction?

Is \( V \dashv V^* \) a thing?
Yes!

But to make sense of $V \to V^*$ we’ll need to venture into the world of 2-categories. A 2-category is an appropriate setting in which to talk about adjunctions, among other things. Here’s why. As we know from the definition above, an adjunction consists of

i. some **objects** (categories)

These form a category!

ii. some **arrows** (functors)

iii. some **arrows between the arrows** (natural transformations)

You’ll notice that the **objects** and the **arrows themselves** form a category, namely Cat, the category of all categories. The objects of Cat are **categories** and the morphisms are **functors**.

*Nice.*

It’d be even nicer, though, if the **natural transformations** were also part of the data. That is, it’d be *super nice* if the threesome itself constituted a known categorical construction. But as it stands, it doesn’t. There is no room for a notion of “arrows between arrows” in the definition of a category.

So what do we do?

We expand the definition. *Literally.* We add an extra dimension, which results in a 2-category. That is, a 2-category consists of

i. objects, now called **0-cells** $\bullet, \circ, \ldots$

ii. morphisms that go between objects, which are the usual arrows, but now we’ll call them **1-cells** $\bullet \to \circ$

iii. morphisms that go between 1-cells, which are not surprisingly called **2-cells** $\bullet \Rightarrow \circ$

As you might guess, the quintessential example of a 2-category is Cat, where the

i. **0-cells** are **categories** $C, D, \ldots$

ii. **1-cells** are **functors** $C \xrightarrow{F} D$

iii. **2-cells** are **natural transformations** $C \xRightarrow{\eta} D$

Recall: The data of an adjunction are functors between categories

$$L : C \longrightarrow D : R$$

and natural transformations

$$\eta : \text{id}_C \Rightarrow RL \quad \epsilon : LR \Rightarrow \text{id}_D$$

What’s more, in any 2-category there is a composition rule for 2-morphisms just like there is for 1-morphisms in an ordinary category! In fact, in a 2-category we require that the set $\text{hom}(\bullet, \circ)$ be *more* than a set. We ask that it be a category itself! Its objects are 1-cells $\bullet \to \circ$, and its morphisms are 2-cells $\bullet \Rightarrow \circ$. There is also an identity 1-cell $\text{id}_\bullet$ for each 0-cell $\bullet$ and there is an identity 2-cell $\text{id}_{\bullet}$ for each $\text{id}_\bullet$. Confusingly, both of these identity morphisms are sometimes denoted as $\bullet$. And of course, there are the usual identity and associativity axioms, though I won’t write them here.
So a 2-category is a good generalization of the relationship we see exhibited among categories, functors, and natural transformations. Having generalized this trio, it becomes very easy to talk about “adjunctions” in any 2-category. Parallel to Definition 2.2, we might lay down the following proposed definition:

**Definition (proposed).** An adjunction between objects and is a pair of 1-cells and ,

\[ l: \bullet \rightarrow \circ \rightarrow r \]

and a pair of 2-cells and , called the unit and counit respectively,

\[ \eta \quad \text{and} \quad \epsilon \]

such that these two triangles commute

\[ l \overset{\eta}{\Rightarrow} l \circ r \overset{\epsilon}{\Rightarrow} r \]

\[ r \overset{\epsilon}{\Rightarrow} r \circ l \overset{\eta}{\Rightarrow} l \]

i.e. such that the following equations hold\(^\text{13}\)

\[ (\epsilon \circ l) \circ (l \circ \eta) = \text{id}_l \quad \text{and} \quad (r \circ \epsilon) \circ (\eta \circ r) = \text{id}_r \]

where \( l \circ \eta := \text{id}_l \circ \eta \), and similarly for \( \epsilon \circ l \) and so on. We’ll say \( l \) is a left adjoint of \( r \), and \( r \) is a right adjoint of \( l \), and we’ll denote the adjunction by \( l \dashv r \).

Alright, fine. But what does this have to do with vector spaces?

The answer lies in the following neat fact.

**Neat Fact:** Every monoidal category \((C, \otimes, 1)\) can be viewed as a 2-category!

Er, actually, I shouldn’t spread rumors.

**Neat Fact:** Every monoidal category \((C, \otimes, 1)\) can be viewed as a 2-category!

Here’s the correct statement:

**Neat Fact:** Every monoidal category \((C, \otimes, 1)\) can be viewed as a bicategory!

A bicategory is basically a 2-category—the data is completely the same. There are 0-cells, 1-cells, and 2-cells. The only difference is what’s in the margin.\(^\text{14}\) So any monoidal category \((C, \otimes, 1)\) gives rise to a bicategory \(C\) where the

\[ V \otimes (W \otimes U) \stackrel{\cong}{\rightarrow} (V \otimes W) \otimes U! \]
i. only 0-cell is the category $C$

ii. 1-cells are the objects of $C$; composition is $\otimes$

iii. 2-cells are the morphisms of $C$; composition is composition $\circ$ in $C$

Therefore it makes sense to talk about 1-cells in $C$ (i.e. objects in $C$) having adjoints! And it makes sense to talk about 2-cells in $C$ (i.e. morphisms in $C$) being units and counits of the adjunction, vis-a-vis our Proposed Definition! In particular, this is true of the symmetric monoidal category $(\mathcal{FVect}, \otimes, R)$. It gives rise to a bicategory $\mathcal{F}\text{Vect}$ where the

i. only 0-cell is the category $\mathcal{FVect}$

ii. 1-cells are vector spaces; composition is the tensor product $\otimes$

iii. 2-cells are linear maps; composition is the usual composition $\circ$

So there is an adjunction of vector spaces $V \dashv W$ whenever the conditions of our Proposed Definition hold. Of course, those conditions hold precisely when $W = V^*$ and $\eta$ and $\epsilon$ are defined as in (1).

Explicitly:

There is an adjunction $V \dashv V^*$ in the bicategory $\mathcal{F}\text{Vect}$ since there are linear maps

$$R \xrightarrow{V} V^* \otimes V \quad \text{and} \quad V \otimes V^* \xrightarrow{\epsilon} R$$

so that the following triangles commute

$$
\begin{array}{ccc}
V \cong V \otimes R & \xrightarrow{V \otimes \eta} & V \otimes V^* \otimes V \\
\downarrow{\text{id}_V} & & \downarrow{\epsilon \otimes V} \\
R \otimes V & \cong V \\
\end{array}
\quad
\begin{array}{ccc}
V^* \cong R \otimes V^* & \xrightarrow{\eta \otimes V^*} & V^* \otimes V \otimes V^* \\
\downarrow{\text{id}_{V^*}} & & \downarrow{\epsilon} \\
V^* \otimes R \cong V^* \\
\end{array}
$$

i.e. so that the following equations hold

$$(\epsilon \otimes \text{id}_V) \circ (\text{id}_V \otimes \eta) = \text{id}_V \quad \text{and} \quad (\text{id}_{V^*} \otimes \epsilon) \circ (\eta \otimes \text{id}_{V^*}) = \text{id}_{V^*}$$

and these are precisely the string diagram equations shown in the chart on page 21.

Voila!

Finally, notice that the above holds for every vector space $V$ in $\mathcal{F}\text{Vect}$. On the other hand, there are certainly 2-categories in which not every 1-cell is dualizable, i.e. has an adjoint. Take $\text{Cat}$ for instance! Not every functor is part of an adjunction. There is, however, a special name given to those bicategories $\mathcal{C}$ that do arise from a monoidal category $C$ and in which every 1-cell has an adjoint.

That name is compact closed.
2.2 Decorated Cospans

A second construction that appears in some work within applied category is the **decorated cospan**. In any category, a diagram that looks like

\[ A \rightarrow C \leftarrow B. \]

is called a **cospan**. In the next section, we’ll only consider the case when \( A, B, \) and \( C \) are finite sets and the arrows are functions between them. A **decorated cospan** is a cospan where the middle set \( C \) has been endowed with some extra structure. That’s the intuitive definition, though I’d like to postpone a more precise definition until the next section.

Now you might think it strange to give a name to a simple diagram like \( A \rightarrow C \leftarrow B \), but cospans come in handy quite often! For instance, if for some reason you can’t possibly hope to find a morphism between objects \( A \) and \( B \), a common technique\(^\text{15} \) is to instead look for a “larger” object \( C \) that “contains” both \( A \) and \( B \). Then although you don’t have maps between \( A \) and \( B \), you do have maps \( A \rightarrow C \leftarrow B \). In that case, your cospan is the **next best thing**.

Admittedly, this section is *bite-sized* compared to the behemoth on monoidal categories that we just finished, but that’s not because decorated cospans are any less important! In fact, Brendan Fong developed the theory of decorated cospans as part of his PhD thesis “The Algebra of Open and Interconnected Systems”, which has served as the foundation for much progress in applied category theory, as I mentioned earlier. But in these notes, we’ll only use the cospan construction in our brief discussion on chemical reaction networks in Section 3.1. On the other hand, we will need the language of monoidal categories in both Sections 3.1 and 3.2. In fact, as we’ll soon see, decorated cospans themselves form a monoidal category!
2.3 Further Reading

For more on monoidal categories and string diagrams:

- Read Chapters 3 and 4 of *Picturing Quantum Processes* by Bob Coecke and Aleks Kissinger. There you’ll also find more information on the interpretation of morphisms in a monoidal category as processes and objects as systems.

- Take a look at TheCatsters videos on string diagrams, by Eugenia Cheng and Simon Willerton. On second thought, their entire collection of videos is great. Go watch them all!

- If you like $\infty$-categories, you’ll be delighted to know that a version of string diagrams (affectionately called “strictly undulating squiggles”) and the yanking equations (!) make an appearance in chapter 8 of *Elements of $\infty$-Category Theory*, a new book on model-independent $\infty$-category theory by Emily Riehl and Dominic Verity.

For more on decorated cospans:

- Read “Decorated Cospans” a blog post by John Baez on the n-Category Café.

- Read Chapter 6 of *Seven Sketches in Compositionality* by Brendan Fong and David Spivak. In Section 6.1, the authors give the following bit of motivation:

  ...we produce a certain monoidal category—namely that of cospans in a category $\mathcal{C}$, denoted $\text{Cospans}_\mathcal{C}$—that can conveniently package $\mathcal{C}$’s colimits in terms of its own basic operations: composition and monoidal structure. In summary, the first part of this chapter is devoted to the slogan ‘colimits model connection.’ (emphasis theirs)

As we’ll see in Section 3.1, objects in $\text{Cospans}_\mathcal{C}$ are cospans in $\mathcal{C}$ and a morphism between two cospans is given by a construction called a colimit. Like the composition $\circ$ and product $\otimes$ in a general monoidal category, a colimit is a categorical construction that allows you to connect things together. But for the sake of “time” (i.e. so that this document doesn’t accidentally turn into a book...), I’ll assume familiarity with colimits. But if you’d like to an intuitive introduction of colimits, as well as their dual construction, limits, I recommend that you

- Take a look at “Limits and Colimits (Part 1)” a blog post on Math3ma. Also see chapters 3 and 6 of *Seven Sketches* and chapter 3 of *Category Theory in Context* by Emily Riehl.
3 Two Examples

Having taken a leisurely stroll through two themes (functorial semantics and compositionality) and two constructions (monoidal categories and decorated cospans) within applied category theory, it’s time to see them come to life in two examples. As mentioned in the introduction, we’ll walk through the first example—chemical reaction networks—relatively quickly. There are several excellent resources available online, including John Baez’s expositions on the n-Category Café as well as on his personal webpage. (I’ve included a few links to these in Section 3.3.) Afterwards we’ll take a longer stroll through the second example—natural language processing—in Section 3.2.

3.1 Chemical Reaction Networks

The first example comes from a paper by John Baez and Blake Pollard called “A Compositional Framework for Reaction Networks.” Specifically, they provide a compositional framework for modeling chemical reaction networks. A chemical reaction network is, well, a network of chemical reactions. And a chemical reaction is exactly what you think it is. It’s what you learned back in high school: You start with some reactants and some products, and there’s a chemical process that takes one to the other.

\[
2\text{H}_2 + \text{O}_2 \rightarrow \text{H}_2\text{O} \\
\text{C} + \text{O}_2 \rightarrow \text{CO}_2
\]

What’s nice is that these reactions can be depicted graphically:

Of course, you can imagine that there might be lots of various reactants, products, and chemical processes. The corresponding network would then be a (possibly huge) collection of these graphs stacked
For instance, this chemical reaction network made a cameo appearance in Baez’s 2016 talk “The Mathematics of Networks”:

Graphs such as these are examples of Petri nets. A Petri net is essentially a bipartite directed (multi)graph that allows us to visually represent reactions, though they are used outside of chemistry as well.

But if we do wish to model chemical reactions, then an important thing we’d like to account for is the rate at which one or more chemicals change over to another. A Petri net with rates included is called, appropriately, a Petri net with rates. More specifically, it’s a bipartite directed graph whose two types of vertices are called places, which represent chemical species, and transitions, which represent chemical reactions. Moreover, each transition $\tau_i$ is assigned a rate $r_i$, a positive real number that describes how fast or how likely it is for $\tau_i$ to occur. These rates then allow us to write down differential equations that describe the system. A Petri net with rates is thus a pictorial representation of a set of differential equations that describe a system. So, for instance, if you did watch Baez’s “The Mathematics of Networks” talk then this example will look familiar:

It tells us that, for example, substances with concentrations $A$ and $B$ combine and produce a substance with concentration $C$ at a rate proportional to $r_1$. The differential equations you see are due to the
law of mass action, which says that the rate with which a chemical reaction will occur is equal to its rate constant $r$, multiplied by the product of the concentration of the reactants, i.e. the concentration of the “inputs” of the reaction.

By the way, the rates themselves could change with time, which might suggest the presence of a dynamical system. What’s more, a system such as the above could potentially interact with its environment, which is to say there might be some quantities that flow in and some quantities that flow out, resulting in an open Petri net with rates:

These quantities can be incorporated into the equations, too, resulting in an open dynamical system. As Baez and Pollard summarize, the goal is to use these observations “to build up a reaction network from smaller pieces, in such a way that its rate equation can be determined from those of the pieces.” This is what is meant by a compositional framework, and is a prime example of the principle of compositionality mentioned in Section 1.2. What’s more, a key step towards achieving this goal is given by functorial semantics! That is, we start by thinking of a Petri net as syntax and a set of differential equations as semantics. And if we have a collection of Petri nets that model a very large network, then—guided by the principal of compositionality—we would like to compose them by gluing graphs together, and we would like to aggregate them by stacking graphs on top of each other. In other words, we hope that Petri nets form a monoidal category! Similarly, one would hope that there is a sense in which dynamical systems form a monoidal category so that differential equations can be “composed” and “aggregated” as well. One would also wish for a monoidal functor Petri nets $\rightarrow$ dynamical systems.

That’s a lot of wishes, but amazingly they all come true, for this is precisely what Baez and Pollard proved in their paper! But how exactly? How was it all formalized? The key is the decorated cospan construction of Brendan Fong that we mentioned in Section 2.2.

(What’s amazing is that Fong’s construction is general enough to model other open reaction networks\textsuperscript{16} as well! But more on that later.)
A **cospan** in the category of **finite sets**, for example, is just a diagram of the form $X \to V \leftarrow Y$, where we’re meant to think of $V$ as the set of places (i.e. chemical species) in the Petri net. To account for the edges in the graph, we ask that $V$ is “decorated” with extra structure, namely source and target maps from the set of edges. This results in a **decorated cospan**, and Fong proved that these constructions form a category! That is, there is a category where objects are finite sets $X,Y,\ldots$, and a morphism $X \to Y$ is a decorated cospan whose feet are $X$ and $Y$. Composition is given by the pushout\(^{17}\), which amounts to gluing graphs together. (This composition is only associative up to isomorphism, so the morphisms are really **isomorphism classes** of cospans. Also, the identity $X \to X$ is the empty graph.)

What’s more, Fong showed that this category has a **symmetric monoidal** structure by stacking graphs on top of each other, i.e. by taking their disjoint union. In fact, it’s **compact closed** and also a **hypergraph category**!

In summary, Fong’s constructions quickly give rise to key results, which I’ll summarize here. The first is that the syntax category of Baez and Pollard is indeed a category:

**Theorem 3.1** (Baez, Pollard). There is a symmetric monoidal category $\text{Petri}$ where

- **objects** are finite sets $X,Y,\ldots$

- a **morphism** $X \to Y$ is a **open Petri net with rates**, i.e. a cospan

\[
\begin{array}{ccc}
X & \overset{i}{\rightarrow} & V \\
\downarrow & & \downarrow \circ \\
Y & \underset{o}{\leftarrow} & X
\end{array}
\]

**together with a Petri net with rates whose places are comprised of $V$.**

The reason the arrows from $X$ and $Y$ point in is that $X$ and $Y$ might be thought of as “leftputs and rightputs” rather than as inputs and outputs. In other words, you’d like the freedom to think of things as flowing either in or out of either end. For example, a physical pipe doesn’t know the different between left and right. Water can flow in or out at either direction. As Baez notes, “The main reason for these designations is to remember that when we screw together two pipes, we attach the output of the first to the input of the second.”

\(^{17}\) A pushout is a type of colimit, a major construction in category theory that’s a bit like like mathematical glue. Anytime you mush two mathematical objects together—like the graphs in the picture—you’ve probably got a colimit construction. More intuition behind colimits and their dual construction, limits, can be found on Math3ma.
The next corollary provides the same statement for the semantics category:

**Theorem 3.2** (Baez, Pollard). There is a symmetric monoidal category \( \text{Dynam} \) where

- objects are finite sets \( X, Y, \ldots \)
- a morphism \( X \to Y \) is an open dynamical system, i.e. a cospan

\[
\begin{array}{c}
V \\
\uparrow^i \\
X \\
\downarrow^o \\
Y
\end{array}
\]

\( i, o \) together with a smooth vector field on \( \mathbb{R}^V \).

Finally, another result of Baez and Pollard shows the existence of a symmetric monoidal functor \( \Box : \text{Petri} \to \text{Dynam} \) from the syntax to the semantics.

**Theorem 3.3** (Baez, Pollard). There is a symmetric monoidal functor

\( \Box : \text{Petri} \to \text{Dynam} \) sending any open Petri net with rates to the corresponding open dynamical system.

The upshot is that functoriality and monoidality

\[
\Box(f \circ g) = \Box f \circ \Box g
\]

\[
\Box(f \otimes g) = \Box f \otimes \Box g
\]

tell us that if you want to understand the open dynamical systems of the composite (or tensor product) of two open Petri nets, then you just have to find the open dynamical systems of each one and then compose (tensor). This is exactly the **principle of compositionality**: to determine the behavior of a big complicated thing, you need only understand the behaviors of its components, and then assemble them together. And by the way, this works for many other kinds of network graphs, not just Petri nets. It’s all part of Baez’s larger body of work on a general categorical framework for a theory of networks which encompasses electrical circuits, Markov processes, signal-flow graphs in control theory, and more!

**This rapid tour** through chemical reaction networks is only one way that compositionality, functorial semantics, and monoidal categories are being used in applications. The next example gives a second way: natural language processing.
3.2 Natural Language Processing

At long last, we’ve made it to our second application of category theory—natural language processing! It is, simply put, a branch of artificial intelligence that aims to train computers to understand human language. What’s nice is that computers can understand meanings of words (through models like Word2vec, for instance) and computers can understand grammar (through parts of speech tagging, for instance). But what’s not-so-nice is that computers aren’t too good at understanding meanings of sentences and longer bodies of text.

In 2010, Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark sought to address this problem in “Mathematical Foundations for a Compositional Distributional Model of Meaning.” In this paper, the authors rely heavily on the principle of compositionality—the idea that the meaning of a sentence can be determined by the meanings of its individual words together with the grammatical rules for combining them. So if a computer can understand meanings of individual words and if it can understand grammatical rules, then the only thing it needs help with is knowing how to combine them to form a meaningful whole. And that’s where the category theory comes in! Guided by functorial semantics, Coecke et. al. model natural language as a (monoidal) functor between compact closed categories

\[
\text{grammar } \to \text{meanings of words}
\]

This functor assigns a grammar type to a word, and the monoidal structures provide a way to combine the meanings of those words (and their grammar types) to form a sentence, whose meaning can be determined via the principal of compositionality.

In the remaining pages, we’ll dive into the details by answering the following questions:

i. **(the syntax category)** How can we make sense of grammar, mathematically? Specifically, how does grammar form a compact closed category?

ii. **(the semantics category)** How can we make sense of meanings of words, mathematically? That is, how do meanings of words form a compact closed category?

iii. **(the functor)** How is the functor \(\text{grammar } \to \text{meanings of words}\) defined and how does it allow one to determine the meaning of a full sentence?

Let’s start by answering the first question.
The Syntax Category: Pregroup

Following the work of Coecke et. al., we can model grammar algebraically via a pregroup, a construction due to mathematician Joachim Lambek in the early 1990s. Informally, a pregroup is cooked up from the following recipe:

\[
\text{poset} + \text{monoid} + \text{“duals”} = \text{pregroup}
\]

In other words, a pregroup is

- a poset \((P, \leq)\)

- that has a multiplication (we’ll denote it by juxtaposition) that’s compatible with the partial order, i.e. if \(p \leq q\) then \(ap \leq aq\) and \(pa \leq qa\) for all \(a \in P\),

- together with a unit 1 satisfying \(1p = p1 = p\) for all \(p \in P\),

- and moreover each element \(p\) has both a left dual \(p^l\) and a right dual \(p^r\) with maps

\[
\begin{align*}
p^l & \quad \leq 1 \quad \leq pp^l & \quad \text{and} \quad pp^r & \quad \leq 1 \quad \leq p^r p.
\end{align*}
\]

that are required to satisfy the yanking equations in (2).

I’ve referred to the inequalities as maps (and have labeled them as such) because they are actually morphisms in a category! Indeed, every poset \((P, \leq)\) can be viewed as a category: an object is an element in \(P\) and there is an arrow \(p \rightarrow q\) if and only if \(p \leq q\). (In particular, there is at most one arrow between any two elements in a poset.) Composition is given by transitivity: if \(p \leq q\) and \(q \leq r\) then \(p \leq r\), and associativity is immediate. Also, every element has an identity arrow since the partial order is reflexive: \(p \leq p\) for all \(p \in P\). So because a pregroup is a poset, we may also view it as a category. Therefore I’ll now draw an arrow \(\rightarrow\) in lieu of the partial order \(\leq\). Moreover, a pregroup is really a poset-with-extra-structure and therefore we may view it as a category-with-extra-structure. Not surprisingly, given the reappearance of the yanking equations, that
extra structure is compact closure! In summary, a pregroup is an example of a compact closed category. It is, in particular, a compact closed category that is not symmetric. Indeed, those four inequalities above are really the unit and counit maps
\[ \eta^l \eta^r \xrightarrow{1} 1 \xrightarrow{1} \eta^r \eta^l \quad \text{and} \quad \eta^l \eta^r \xrightarrow{1} 1 \xrightarrow{1} 1. \]
discussed in Section 2.1, and the yanking equations (2) amount to the following:
\[ p \eta^l \eta^r \xrightarrow{1} pp \eta^r \eta^l \rightarrow 1 \quad \text{and} \quad pp \eta^r \eta^l \xrightarrow{1} p^r p. \]

Let’s look at two examples of pregroups. The first is an arithmetic example, which will help to get our feet wet. The second is a grammatical example, which is used in the DisCoCat model of Coecke, Sadrzadeh, and Clark.

**Example 3.4.** The set
\[ \{ f : \mathbb{Z} \rightarrow \mathbb{Z} \mid \text{f is monotone and unbounded} \} \]
is a pregroup. The partial order is given pointwise: \( f \leq g \) if and only if \( fn \leq gn \) for all \( n \). The monoid multiplication is given by function composition \( f \cdot g := f \circ g \). The monoidal unit is \( \text{id}_\mathbb{Z} \). Given such a function \( f \), its left and right duals are given by
\[ f^l n := \min \{ m \in \mathbb{Z} \mid n \leq fm \} \quad \text{and} \quad f^r n := \max \{ m \in \mathbb{Z} \mid fm \leq n \}. \]

For example, if \( fm = 2m \), then
\[ f^l n = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even}, \\ \frac{n+1}{2} & \text{if } n \text{ is odd} \end{cases} \quad \text{and} \quad f^r n = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even}, \\ \frac{n+1}{2} & \text{if } n \text{ is odd}. \end{cases} \]

In short, \( f^l n = \lfloor \frac{n+1}{2} \rfloor \) and \( f^r n = \lceil \frac{n}{2} \rceil \).

You can find this example in “Iterated Galois Connections in Arithmetic and Linguistics” by Lambek, which appears in the Springer book *Galois Connections and Applications*. You’ll also find mention of it in the “Mathematical Foundations” paper of Coecke et al.

**Fun fact:** the pair \( (f^l, f) \) forms a special kind of categorical adjunction called a Galois connection since it satisfies
\[ f^l n \leq m \iff n \leq f^r m. \]

Indeed if \( n \) is even, then \( n/2 \leq m \iff n \leq 2m \). And if \( n \) is odd, then \( (n + 1)/2 \leq m \) which means \( n + 1 \leq 2m \) which is true iff \( n \leq 2m \). Similarly, the pair \( (f, f^r) \) forms a Galois connection since
\[ fn \leq m \iff n \leq f^r m. \]

Indeed, if \( n \) is even then \( 2n \leq m \iff n \leq m/2 \). And if \( n \) is odd, then \( 2n \leq m \) means \( n \leq m/2 \) which is true iff \( n \leq (m - 1)/2 \).

For a couple of great introductions to Galois connections (They are super cool and appear in lots of places in math!) take a look at Lecture 4 of John Baez’s online course on applied category theory as well as Section 1.5 of *Seven Sketches* by Fong and Spivak.
While arithmetic is fun, this next example is the one we’re most interested in.

**Example 3.5.** Given any finite poset $X$, we can construct the free pre-group generated by $X$, denoted $\text{Preg} X$. For a simple example, suppose $X = \{n, s\}$ whose elements we’ll think of as basic grammar types: $n$ is the type of a noun and $s$ is the type of a (declarative) sentence. Elements of $\text{Preg}\{n, s\}$ are concatenations of the letters $n$ and $s$ and their left and right duals and iterations of those duals and so on. For example, some grammatical types in $\text{Preg}\{n, s\}$ are:

\[
\begin{align*}
n & = \text{noun} \\
s & = \text{sentence} \\
n^l & = \text{adjective} \\
n^l s n^l & = \text{transitive verb}
\end{align*}
\]

The strings of letters are called **compound types**, and there is a morphism $a \to b$ between compound types if and only if $a$ can reduce to $b$ by application of one or more of the counit maps $\epsilon^r$ and $\epsilon^l$.

Consider a banana, for example. It has type $n$, of course, while the adjective yellow has type $nn^l$. The reason that adjectives have grammar type $nn^l$ is that an adjective can always be paired on the left with a noun, resulting in a new noun—e.g. yellow banana.

\[\begin{array}{c}
\text{yellow} & \text{banana} \\
\end{array}\]

Indeed, to verify that the grammar type of yellow banana is $n$, we start by concatenating the types of the individual words to obtain $nn^l n$.

Then we apply the counit map $\epsilon^l : n^l n \to 1$ together with the identity map $1_n : n \to n$ (this is given to us by the reflexivity axiom of posets: $n \leq n$) to see that $nn^l n$ reduces down to $n$:

\[\begin{array}{c}
\text{"yellow" is an adjective} \\
\text{"banana" is a noun} \\
n n^l \\
\end{array}\]

\[\begin{array}{c}
\text{yellow} \\
\end{array}\]

\[\begin{array}{c}
\text{banana} \\
\end{array}\]

\[\begin{array}{c}
1_n \cdot \epsilon^l \\
n \\
\end{array}\]

\[\begin{array}{c}
\text{"yellow banana"} \\
\text{is a noun} \\
1 \cdot n = n \\
\end{array}\]

The dot in $1_n \cdot \epsilon^l$ is meant to suggest “apply $1_n$ to $n$ while simultaneously applying $\epsilon^l$ to $n^l n$.”
This tells us that the phrase yellow banana has grammar type $n$.
That’s good. A yellow banana is a noun!

In light of this discussion on yellow bananas, you might enjoy taking a few seconds to think about why $n^r s n^l$ represents the grammar type of transitive verb.

\[
\text{<ponder> ... </ponder>}
\]

A transitive verb is a word that accepts a noun on the right and another noun on the left such that the resulting phrase is a full sentence. Since we like bananas, here’s another fruit-based example:

\[
\text{bananas are fruit}
\]

To determine the grammar type of this phrase, we concatenate the grammatical types of the individual words and then apply the counit maps to reduce, as before:

\[
\begin{align*}
\text{Here’s that same reduction written out step-by-step. For clarity, I’ll indicate the concatenation with a dot:} \\
n n^r s n^l n &= nn^r \cdot s \cdot n^l n \\
&\xrightarrow{\epsilon^r \cdot 1_n} 1 \cdot s \cdot n^l n = s \cdot n^l n \\
&\xrightarrow{1_s \cdot \epsilon^l} s \cdot 1 = s
\end{align*}
\]

In words, we’ve used the counit maps to reduce the concatenation of the grammar types for bananas are fruit to the letter $s$, which confirms that “bananas are fruit” is indeed a grammatically correct sentence. More generally, for any $a, b \in \text{Preg}(n, s)$, we draw a morphism $a \rightarrow b$ if and only if $a$ can be reduced to $b$ in a similar fashion.

Alright, that’s (a very condensed version of) the pregroup story!
To summarize, the language model of Coecke et. al. amounts to a structure-preserving functor

\[
\text{syntax} \rightarrow \text{semantics}
\]
In this section, we’ve just shown that the syntax category is taken to be a pregroup freely generated on a finite set of basic grammar types, i.e. \( \text{syntax} = \text{Preg}X \). Let’s move on to semantics now.

**The Semantics Category: Vector Spaces**

As stated in the introduction to Section 3.2, computers are able to understand meanings of individual words pretty well. That’s because computers understand numbers! For example, a great way to inform a computer of the meaning of the word *banana* is to represent *banana* by a number and then give that number to the computer.

**What number?**

Well, it’s not exactly a number. It’s an array of numbers—a vector.

**Okay, what vector?**

The answer is simple, though I’d like to motivate it by sharing the following theorem.

**Theorem** (The Yoneda Lemma for Linguistics). *You shall know a word by the company it keeps.*

**Proof.** John Firth

Okay, so it’s not a theorem. But it is a great quote! Firth’s idea is that *words that appear in similar contexts will have similar meaning.* In the linguistics community, this is referred to as the *distributional hypothesis.* So you might imagine that *apple* is more similar to *banana* than it is to *puppy* since *apples* and *bananas* often occur near words such as *sweet*, *snack*, *green*, *eat*, etc., whereas *puppy* occurs more often near words such as *pet*, *cute*, *furry*, *bark*, and so on. As another example, you might not know what the word *yegg* means (or perhaps you do), but you can probably infer it from this sentence:

> The cops grabbed him and another yegg for a Philadelphia store burglary.

So we can represent the meaning of a word by a vector. This is often called a *distributional model of meaning.* But what, exactly, is the assignment \( \text{word} \rightarrow \text{vector} \)? Suppose we have a fixed corpus—your favorite book, say. Start by choosing a set of so-called *context words* \( \{w_1, \ldots, w_n\} \). This can be every word in the corpus or some subset of it. By representing each \( w_i \) as the \( i \)th standard basis vector

\[
\begin{align*}
\mathbf{w}_i &= (0, \ldots, 1, \ldots, 0) \\
&= \text{ith basis vector}
\end{align*}
\]


---

20 Firth, J. R. A synopsis of linguistic theory, 1930–1955. In Selected Papers of JR Firth, 1952–59 (ed. J. Firth and F. Palmer). Indiana University Press.

Are you wondering why I’ve referred to Firth’s idea as the Yoneda Lemma? To find out why, I recommend reading up on the Yoneda Perspective.

21 James Lardner and Thomas Reppetto, *NYPD: A City and Its Police*, 2000
we obtain a basis \( \{w_1, \ldots, w_n\} \) for a vector space \( V \). Then any word \( w \) in the corpus has a vector representation given by a linear combination of the context words

\[
w = \sum_{i=1}^{n} c_i w_i
\]

The coefficients \( c_i \) are real numbers that indicate the number of times that \( w \) occurs near \( w_i \) in the corpus.

Here’s an example. Suppose we’re reading a book that contains the words\[ \{\text{sweet, green, furry}\} \]

Let’s choose them to be our context words and make the assignment so that

\[
\text{sweet} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{green} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{furry} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\]

Then if \( \text{banana, puppy} \) and \( \text{fruit} \) are also words in our book, we might have something like

\[
\text{banana} = \begin{bmatrix} 21 \\ 9 \\ 0 \end{bmatrix} \quad \text{puppy} = \begin{bmatrix} 8 \\ 1 \\ 32 \end{bmatrix} \quad \text{fruit} = \begin{bmatrix} 43 \\ 19 \\ 0 \end{bmatrix}
\]

In other words, we’ve used data from the corpus to embed these words as vectors inside of a three-dimensional vector space. This prompts us to say that the meaning of the word banana is the vector \((21, 9, 0)\), the meaning of puppy is \((8, 1, 32)\), and the meaning of fruit is \((43, 19, 0)\).

And this works! That is, you can feed distributional models into your computer, and they’ll ace the word-similarity portion of your SAT exam. Or your can compute the dot product between words, and you’ll find that vectors are closer together precisely when the words they represent have the same meaning. It’s all familiar territory for NLP practitioners. The semantics category for Coecke et al. is thus the category of finite dimensional vector spaces over \( \mathbb{R} \). That is,

\[
\text{meanings of words} = \text{FVect}
\]

Unfortunately, the distributional model does not work for sentences. The same sentence rarely occurs twice in a given document, therefore we can’t follow the same procedure above. This is where category theory can help lend a hand. In light of the principle of compositionality, the meaning of a sentence should be able to be computed given the meanings of its individual words and the

---

\[ \text{You can decide what “near” means. That is, the context of } w \text{ is the set of words within } k \text{ words of } w, \text{ where } k = 1 \text{ or 2 or 3 or whatever you like.} \]
rules of grammar for combining them. And we can pair meanings-of-words with grammatical types via a map from syntax (grammar) to semantics (meanings of words), i.e. via a functor

\[ F : \text{Preg}X \rightarrow \text{FVect} \]

where \( X \) is a finite set of basic grammar types. In fact, as we know from Section 2.1, both \( \text{FVect} \) and \( \text{Preg}X \) are compact closed categories, so we’ll ask that \( F \) be a *strong monoidal functor*—one that preserves the compact closed structure. That’s the gist behind the categorical compositional distributional model of Coecke et. al.

But how is \( F \) actually defined? Let’s talk about that next.

**The Functor: Syntax \( \rightarrow \) Semantics**

In this section, we’ll give an explicit description of the functor

\[ F : \text{syntax} \rightarrow \text{semantics} \]

or more specifically,

\[ F : \text{Preg}X \rightarrow \text{FVect} \]

For simplicity, let’s take \( X = \{n,s\} \) as we did before. Now to define a functor, we need simply to say what it does on objects and morphisms. So let’s do that. On objects,

- \( F \) assigns to the noun type \( n \) a vector space \( N := Fn \), which we’ll call a *noun space*
- \( F \) assigns to the sentence type \( s \) a vector space \( S := Fs \), which we’ll call a *sentence space*

and on morphisms

- \( F \) assigns to a type reduction \( a \xrightarrow{r} b \) a linear map \( Fa \xrightarrow{Fr} Fb \) that sends the vector corresponding to a word or phrase of type \( a \) in \( Fa \) to the vector corresponding to a word or phrase of type \( b \) in \( Fb \).

Moreover, asking that \( F \) preserve the compact closed structure means that

- units and counits in \( \text{Preg}\{n,s\} \) map to units and counits in \( \text{FVect} \)

\[
\begin{align*}
\text{e.g.} & \text{ given } n \in \text{Preg}\{n,s\}, \\
F(1 \xrightarrow{\eta_n} n'n) &= F(1 \xrightarrow{\eta_n} nn') = \eta_N \\
\text{ and } & F(nn' \xrightarrow{\epsilon_n} 1) = F(n'n \xrightarrow{\epsilon_n} 1) = \epsilon_N
\end{align*}
\]

where \( \eta_N : \mathbb{R} \rightarrow N \otimes N \) and \( \epsilon_N : N \otimes N \rightarrow \mathbb{R} \) are the linear maps that we defined on p. 17. A similar idea holds if we replace \( n \) by any element of \( \text{Preg}\{n,s\} \).
• duals map to duals
  
  e.g. $Fn^* = Fn^! = N^*$. But our vector spaces are finite dimensional
  and so $N^* \cong N$ and therefore $Fn^* = Fn^! = N$.

• a compound type is assigned to a tensor product of vector spaces.
  
  e.g. $F(n'sn') \cong Fn^! \otimes Fs \otimes Fn^! = N \otimes S \otimes N$

And that’s it!

Except... this might not be very enlightening yet. It’ll surely be
helpful to look at a toy example. So in the next couple of pages, let’s
use the DisCoCat model to compute the meaning of the sentence

$\text{bananas are fruit}$

By “compute the meaning,” I mean the following: we want to be
able to view the sentence $\text{bananas are fruit}$ as a vector, then feed that
vector into the functor $F$ and get an output vector that encodes for
the meaning of the sentence.

$F(\text{bananas are fruit}) = ??$

That output vector will be the “meaning” of the sentence. Our goal is
to find that meaning.

**Goal: Compute the meaning of $\text{bananas are fruit}$.”**

Let’s proceed systematically. I’ll list the computations step-by-
step, starting from the beginning.

**Step 1: Assign each word a grammar type in $\text{Preg}\{n, s\}$.**

That’s easy enough:

- $\text{bananas} \to n$
- $\text{are} \to n'sn'$
- $\text{fruit} \to n$

**Step 2: Fix a noun space $Fn = N$ and a sentence space $Fs = S$.**

Let’s suppose $N$ is the three-dimensional space spanned by the basis
vectors

$\text{sweet, green, furry}$
which we can represent as column vectors

\[
\begin{align*}
\mathbf{w}_1 &= \text{sweet} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, & \mathbf{w}_2 &= \text{green} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, & \mathbf{w}_3 &= \text{furry} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\end{align*}
\]

as before. These basis vectors generate the noun space. But what about the sentence space $S$? For simplicity, let’s define $S$ to be a “true or false” space so that it’s a one-dimensional vector space spanned by a single vector $\vec{1}$. The origin $0 \in S$ corresponds to “false” while $\vec{1}$ corresponds to “true.” What about scalar multiplies of $\vec{1}$? If you like, you’re more than welcome to think of a positive scalar multiple of $\vec{1}$ as the meaning vector for sentence that is super true. The larger the scalar, the more true the sentence!

Finally, note that once we’ve established $N$ and $S$, the verb space comes for free:

\[
F(n^s n^r n^s n^l) = N \otimes S \otimes N
\]

This is a nine-dimensional space spanned by vectors of the form $\mathbf{w}_i \otimes \vec{1} \otimes \mathbf{w}_j$ where $i$ and $j$ range between 1 and 3.

**Step 3: Determine the vector representations of each word in the sentence.**

We’ll simply recycle the vectors we used earlier:

\[
\begin{align*}
\text{bananas} &= \begin{bmatrix} 21 \\ 9 \\ 0 \end{bmatrix}, & \text{fruit} &= \begin{bmatrix} 43 \\ 19 \\ 0 \end{bmatrix}
\end{align*}
\]

Note that both of these vectors live in the noun space $N$ since each word has grammar type $n$. But what about the transitive verb are? By Step 1, we know that are has grammar type $n^s n^r n^s n^l$ and is therefore a vector in the tensor product $N \otimes S \otimes N$. That is, there are coefficients $c_{ij} \in \mathbb{R}$ so that

\[
\text{are} = c_{11}\text{sweet} \otimes \vec{1} \otimes \text{sweet} + c_{12}\text{sweet} \otimes \vec{1} \otimes \text{green} + \cdots + c_{33}\text{furry} \otimes \vec{1} \otimes \text{furry}.
\]

Eek. That looks uncomely.

Fortunately, we learned in Section 2.1 that $F\text{Vect}$ is a compact closed category and therefore it exhibits process-state duality, which is the sophisticated way of saying

\[
\text{every vector in a tensor product can be identified with a linear map},
\]

which is the long way of saying

\[
\text{every vector is really a matrix}!
\]
And that is excellent news, for if we know “what is what,” i.e. if we know that *puppies* are *furry* but not *green* and so on, then we can re-express the vector for *are* as a $3 \times 3$ matrix. The $ij$th entry of this matrix is the coefficient $c_{ij}$ which is

$$ c_{ij} = \begin{cases} 1, & \text{if } w_i = w_j, \\ 0, & \text{otherwise} \end{cases} $$

The upshot is that the transitive verb *are* has matrix representation

$$ \text{are} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

It’s no surprise that we get the identity matrix. *Being* is all about identity. That is, the verb *are* tells you when something IS something else.

**Step 4: Choose a type reduction in $\text{Preg}\{n,s\}$**

In this step, which takes place in the grammar category, we simply perform the type reduction already done on page 39. To recap, we know the grammar types of *bananas* and *are* and *fruit*, and so we concatenate those types to obtain $nn'sn'l'n$. Using the left and right counit maps, this string of letters reduces down to $s$, which confirms that the phrase *bananas are fruit* is a tried-and-true sentence. In Step 4, we simply take that reduction morphism

$$ nn'sn'l'n \xrightarrow{c_{n'sn'l'n}'c_{s'n'\hat{c}n'}} n $$

and hold on to it. We’ll need to use it in Step 5.

Aside: You might wonder about the word “Choose” in “Step 4: Choose a type reduction.” What’s up with that? Incidentally, no choice was needed in this toy example of ours, so the purpose of this aside might be unclear. Indeed, there’s only one way to parse the sentence *bananas are fruit*. But there exist sentences that can be parsed in more than one way. Consequently, the grammar type of such sentences may reduce down to type $s$ via more than one reduction morphism. In Step 4, we are required to choose one. As an illustration, here is a nice sentence:

$I saw a man with a telescope.$

How did you parse it? Perhaps

$I saw (a man with a telescope).$

or perhaps

$I saw (a man) with a telescope.$
Those are two parsings of the same sentence, each of which corresponds to a different type reduction in the pregroup. In turn, this gives rise to different meaning vectors! And rightly so. Those two sentences have different meanings! Step 4 is simply reminding us of this fact.

### Step 5: Apply $F$!

This is the fun part! We have a morphism in the pregroup

$$nm^n sn^n \xrightarrow{e_n^1 e^n_1} n$$

and we can apply $F$ to get a linear map of vector spaces

$$N \otimes N \otimes S \otimes N \otimes N \xrightarrow{\epsilon_N \otimes 1_S \otimes \epsilon_N} S$$

where $\epsilon: N \otimes N \rightarrow \mathbb{R}$ is the linear map given on page 17 and $1_S: S \rightarrow S$ denotes the identity map on $S$. Finally, apply this linear map to the vector corresponding to the sentence

$$\epsilon_N \otimes 1_S \otimes \epsilon_N(\text{bananas} \otimes \text{are} \otimes \text{fruit})$$

which amounts to a simple matrix multiplication

$$\epsilon_N \otimes 1_S \otimes \epsilon_N(\text{bananas} \otimes \text{are} \otimes \text{fruit}) = \begin{bmatrix} 21 & 9 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 43 \\ 19 \\ 0 \end{bmatrix} = 1074$$

Conclusion? The meaning of the sentence $\text{bananas are fruit}$ is

$$1074\vec{1}$$

which is super true. Voila!
Some Closing Remarks

This functor described above is somewhat reminiscent of a topological quantum field theory, which is a functor from the category of cobordisms (another compact closed category) to the category of complex Hilbert spaces. But in 2014 Anne Preller showed that the only functors from a pregroup $P$ freely generated on a finite set of basic types to $\text{FVect}$ are those mapping to one-dimensional spaces. The key to her proof is the fact that $P$ is a poset and hence there is at most one morphism between any two objects. In particular, any morphism from an object to itself must be the identity. As a consequence, if $a$ is in $P$ then the morphism $(1_a \cdot \eta_a) \circ (\epsilon_a \cdot 1_a)$ from $aa'a \to aa'a$ must equal the identity on $aa'a$. Graphically:

Now consider a functor $P \to \text{FVect}$. It assigns $a$ in $P$ to a vector space $A$ in $\text{FVect}$, and it assigns $(1_a \cdot \eta_a) \circ (\epsilon_a \cdot 1_a)$ to the corresponding linear map, $(1_A \cdot \eta_A) \circ (\epsilon_A \cdot 1_A): A \otimes A^* \otimes A \to A \otimes A^* \otimes A$, which we’ll just denote by $f$,

and which must be an isomorphism. Now if the dimension of $A$ is at least 2, then we can choose orthogonal basis vectors $e_1$ and $e_2$ so that $e_1 \otimes e_2^* \otimes e_1 \in A \otimes A^* \otimes A$. And since $\epsilon_A$ computes the inner product between $e_1$ and $e_2^*$, we have $f(e_1 \otimes e_2^* \otimes e_1) = 0$. Therefore $f$ is not injective, and so it cannot be an isomorphism.

The intuition is, perhaps, that pregroups have too few morphisms to capture the semantics. In particular, pregroups do not allow us to distinguish different parsings of strings of types. One string may reduce in several ways—e.g. (Men and women) whom I like vs. Men and (women whom I like)—and the morphisms in a pregroup do not account for this. So in some sense, there isn’t enough “wiggle room” for meaning in pregroup syntax, so the output can only be a one-dimensional vector space. But all is not lost! As Preller showed, the problem can be fixed by replacing a free pregroup with a free compact closed category. For more details, see her paper “From Logical to Distributional Models.”
3.3 Further Reading

For more on the work of Baez and Pollard:

- Certainly take a look at their paper, “A Compositional Framework for Reaction Networks”. The n-Category Café also contains expositions, including a post by Baez with the same title and “Dynamical Systems and Their Steady States” by Maru Sarazola.

- The examples in Section 3.1 can be found in “The Mathematics of Networks”, a wonderfully accessible talk by Baez on YouTube.

- All of the decorated cospan formalism can be found in Brendan Fong’s thesis, “The Algebra of Open and Interconnected Systems.” As part of the ACT workshop, Jonathan Lorand and Fabrizio Genovese wrote about Fong’s thesis in an article titled “Hypergraph Categories of Cospans” on the n-Category Café.

- The framework of chemical reaction networks is also being used to model ATP coupling! This project was birthed during the ACT workshop. For more, take a look at “Coupling Through Emergent Conservation Laws” on Baez’s Azimuth blog.
For more on the work of Coecke, Sadrzadeh, and Clark:

• There is a delightful 5-minute YouTube video (made in the style of Minute Physics!) called “How Quantum Theory Can Help Understanding Natural Language” that does an excellent job of explaining the ideas behind DisCoCat in a non-technical way.

• The yellow banana example of the previous section was just a toy example meant to showcase the functor of the DisCoCat model of meaning. But this is a document on applied category theory, and so you’d surely like to see some applications! For empirical data arising from actual implementations of the DisCoCat model, take a look at:
  – “Experimental support for a categorical compositional distributional model of meaning” by Edward Grefenstette and Mehrnoosh Sadrzadeh
  – “Prior disambiguation of word tensors for constructing sentence vectors” by Dimitri Kartsaklis and Mehrnoosh Sadrzadeh
  – “Quantization, Frobenius and bi-algebras from the categorical framework of quantum mechanics to natural language semantics” by Mehrnoosh Sadrzadeh

• The DisCoCat model was also featured on the n-Category Café as part of the Applied Category Theory Workshop. In fact, it was featured twice! The two blog posts are:
  – “Linguistics Using Category Theory” by Corey Griffith and Jade Master. This wonderfully written article gives another recap of the DisCoCat model, as well as another toy example. Check out the comment section, too. There are some nice discussion going on there!
  – “Cognition, Convexity, and Category Theory” by Brad Theilman and me. This blog post is a summary of “Interacting Conceptual Spaces I,” a paper in which the authors change the semantics category of the DisCoCat model from FVec to something that attempts to model human cognition more closely, namely convex spaces! This is a great example of tweaking the semantics part of functorial semantics.

• The last suggested resource is unrelated to DisCoCat, but it’s still in the vein of machine learning + applied category theory and so I thought I’d share: Did you know that backpropagation is a functor?
4 But Wait! There’s More...

There’s much more to applied category theory—I’ve only presented a very tiny subset of hand-selected ideas. But there’s so much more to see, learn, and do! So to close out these notes, I’ll leave you with a few more links where you can discover other themes, constructions, and examples of applied category theory.

• To start, there’s the main Applied Category Theory webpage, which has
  – a description of the 2018 workshop that took place at the Lorentz Center in Leiden, Netherlands
  – a call for the 2019 workshop, to be hosted by Bob Coecke in Oxford.

• Jelle Herold and the folks at Statebox filmed most of the 2018 workshop talks, and you can watch them here: https://statebox.org/events/act-leiden.html. Speakers include Samson Abramsky, John Baez, Bob Coecke, Kathryn Hess, Aleks Kissinger, Tom Leinster, David Spivak, and many more!

• Back in March 2018, there was an applied category theory workshop hosted at the National Institute of Standards and Technology. Slides and videos of the talk can be found here: http://www.appliedcategorytheory.org/nist-workshop-slides/

• There is also Seven Sketches in Compositionality (subtitle: “An Invitation to Applied Category Theory”) by Brendan Fong and David Spivak. I’ve referenced this book several times already, but that’s because it’s such a gem! (I was sold just after reading the preface.) It’s a delightful and insightful introduction to more themes, more constructions, and more examples within applied category theory. Even better, no prior knowledge of category theory is assumed. The book is based on a MIT course the authors taught. You can find videos of their lectures here: http://math.mit.edu/dspivak/teaching/sp18/

• And as if all of these great resources weren’t enough, John Baez is running a free online course on applied category theory. Participants have been working through the Seven Sketches book. The lectures and ensuing discussions are a treasure trove of exciting mathematics: https://forum.azimuthproject.org/categories/applied-category-theory-course. Also be sure to take a look at the “applied category” tag on his blog, Azimuth.