Willoughby Seago

Theoretical Physics

Categories and Quantum Informatics

January 17, 2023

COURSE NOTES

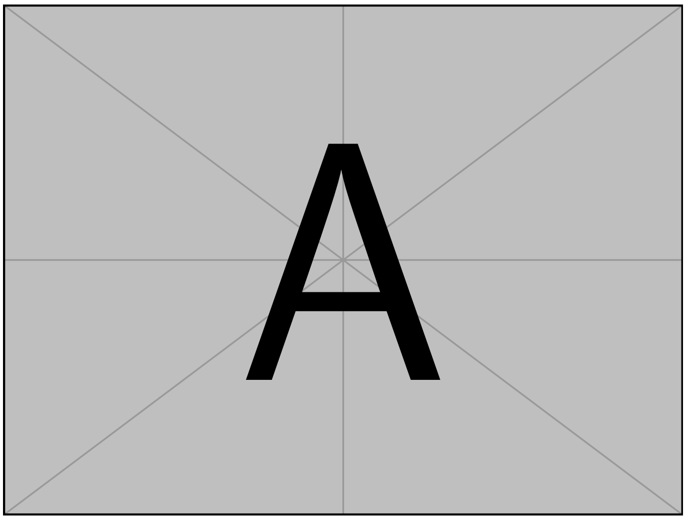
Categories and Quantum Informatics

Willoughby Seago

January 17, 2023

These are my notes from the course categories and quantum informatics. I took this course as a part of the theoretical physics degree at the University of Edinburgh.

These notes were last updated at 23:56 on January 27, 2023. For notes on other topics see https://github.com/WilloughbySeago/Uni-Notes.



Chapters

Page

Contents

Page

List of Figures

Page

One

Introduction

The material delivered in the lectures is included in these notes, but so is extra material pulled from the textbook [1], as well as other sources, such as [2]. I include various additional examples, some which require some mathematical background to understand. Some definitions are included in the appendices, but those unfamiliar with the material in these examples should feel free to just skip them. I've also included pointers to notes from other courses where appropriate, particularly the courses *Symmetries of Quantum Mechanics* and *Symmetries of Particles and Fields*, which both cover the areas of representation theory and Lie theory. The notes from these courses and others can be found at `https://github.com/WilloughbySeago/Uni-Notes`. Again, any unfamiliar material from these courses can be skipped. A fair few of the examples simply come from relevant Wikipedia pages, and I haven't performed detailed checks of all the facts in these cases.

There are a few notational things which don't align with the course. A big one is leaving out brackets for functors, writing FA and Ff instead of $F(A)$ and $F(f)$. The use of $-$ as a blank to be filled in with some object is also not used in the course.

Part I

Introduction

Two

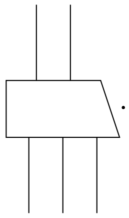
ZX Calculus

In this chapter we will introduce **ZX calculus**. This is a diagramatic notation for performing calculations. ZX calculus is mathematically rigorous, and developing the maths explaining this is a large part of this course. ZX calculus provides a higher level of abstraction that a quantum circuit, focusing less on implementation and more on what the circuit is doing. ZX calculus is built from a relatively small number of building blocks. It is the freedom we have in combining these that makes ZX calculus so powerful.

We'll introduce ZX calculus in a seemingly backwards manner, first introducing which sorts of diagrams we can have, then how to manipulate the diagrams then what the diagrams mean.

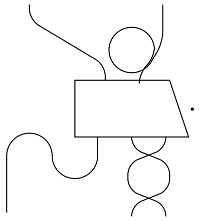
2.1 Types of Diagrams

A diagram in ZX calculus is somewhat like a flowchart. The playing field is the two-dimensional page. We imagine that time goes upwards and space extends to the left and right. This means that a process described by a ZX calculus starts by entering the bottom of the diagram and ends when we leave the top of the diagram. Qubits are represented by wires, which are just lines. Processes are represented by boxes, for now we won't focus on what the process might be. The following diagram represents a process which takes in three qubits and produces two qubits:



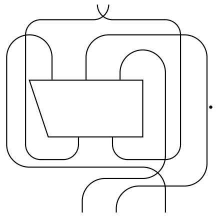
. (2.1.1)

In diagrams it isn't important exactly how we draw the wires, so long as they are connected in the same way, so in the same order both on the box and along the top and bottom, the diagram corresponds to the same equation. For example, the following is equivalent to the previous diagram



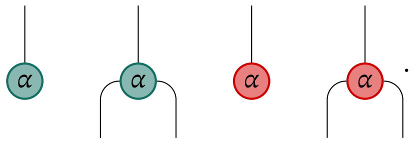
(2.1.2)

We are also free to change the orientation of the box, so long as the the connectivity stays the same. This is why we draw the box as a trapezium without rotational symmetry. For example, the following diagram is equivalent to both of the previous diagrams.



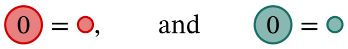
(2.1.3)

A sensible question to ask now is what process does this box represent. We'll get to this. For now we'll just say that the process can be built up of fundamental process. There are four processes which we use to build any diagram in ZX calculus. They are



(2.1.4)

Actually, α can take any value in $[0, 2\pi)$, so there are really an uncountable number of these building blocks. For short if the phase is zero then we omit the label:



(2.1.5)

We call these **spiders**. In particular, the green is a Z spider and the red is an X spider.

Combining these pieces we can quickly build up fairly complex diagrams. For example, the diagram in Figure 2.1 is a process which takes in two qubits and outputs two qubits.

2.2 Simplifying Diagrams

There are two types of rules by which we might manipulate diagrams. The first, which we've already seen, are **graphical rules** which allow us to move different pieces around so long as we don't change the connectivity. More formally two diagrams are equivalent if the are isotopic as graphs, a concept we'll make precise later, but for now two diagrams are isotopic if fixing all of the inputs and outputs

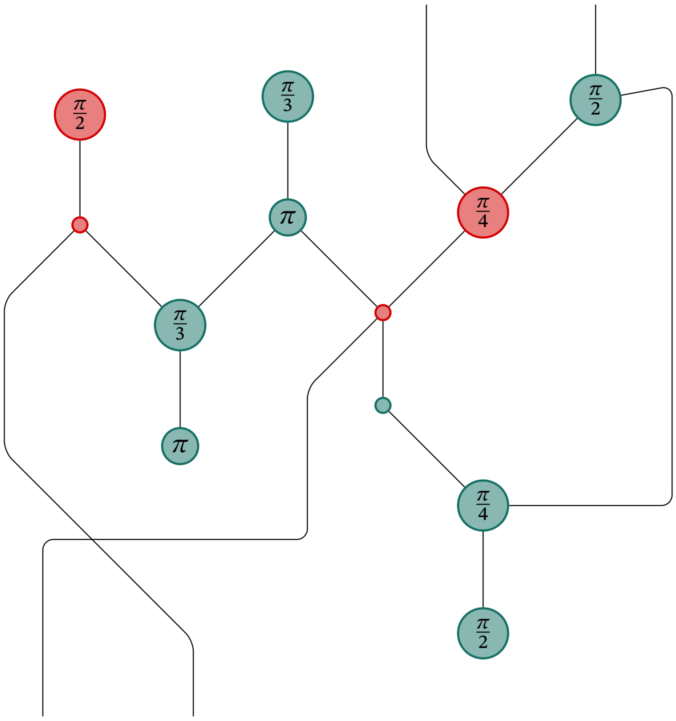
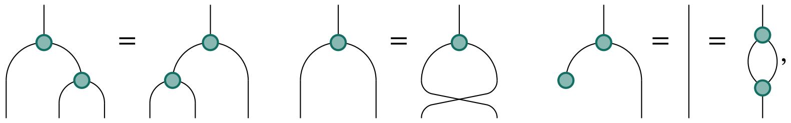


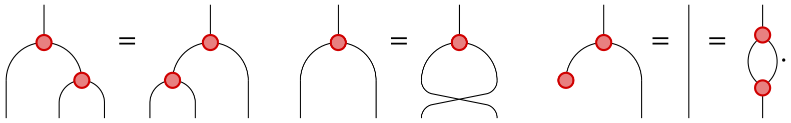
Figure 2.1: A diagram in ZX calculus taking two qubits to two qubits.

as well as the points at which they connect it is possible to continuously morph one into the other. We allow the wires to pass through each other in this process.

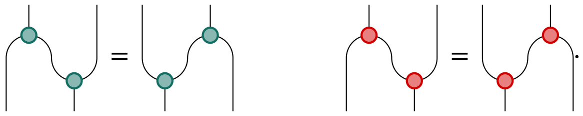
The second type of rule corresponds to specific properties of the basic building blocks. There are quite a few of these, and for now we'll just list them without much explanation. First we have the **monoid rules** which are



and the same for the other colour:

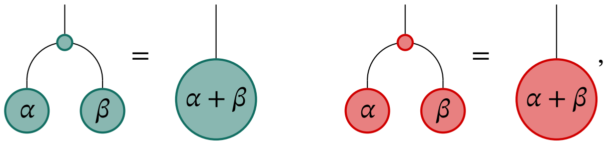


The next set of rules are called the **Frobenius rules**, they are



(2.2.1)

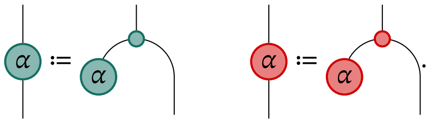
The **fusion rules** allows us to combine multiple nodes of the same colour in some circumstances:



(2.2.2)

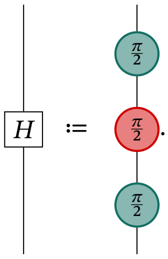
where addition is taken modulo 2π.

Before introducing the next set of rules we introduce some shorthand notation:



(2.2.3)

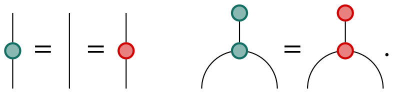
We also define the **Hadamard**:



(2.2.4)

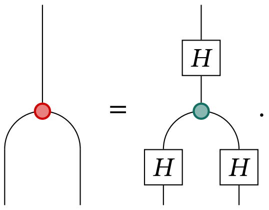
It's safe to give this a square symbol, with rotational symmetry, since we can see from the definition that it is rotationally symmetric.

We then have two **identity rules**, the first is just a repeat of one of the monoid rules, but now in this new shorthand, the second is new:



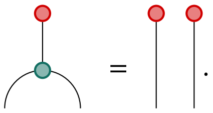
(2.2.5)

The next rule allows us to change the colour of a node, at the cost of some Hadamards, it is appropriately called the **colour change rule**:



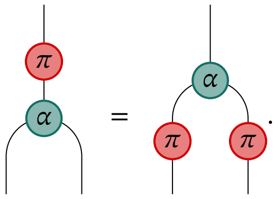
(2.2.6)

The next rule is called the **copy rule**, since it allows us to make two diagrams out of one:



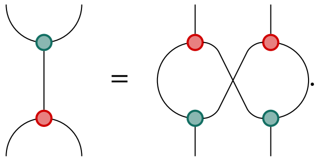
(2.2.7)

Our next rule allows for an X spider with a phase of π to be copied pulling it through a Z spider. It is called the **π-copy rule**:



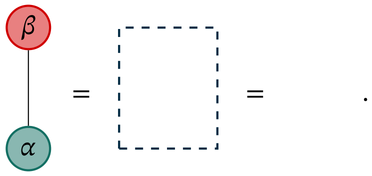
(2.2.8)

The next rule is called the **bialgebra rule**:



(2.2.9)

The final rule is rather simple, it's simply that we can ignore overall phases, called the **scalar rule**, it corresponds to the following:



(2.2.10)

Here the dashed box as well as the empty space both represent the empty diagram, which is simply the trivial identity process taking in no qubits, doing nothing, and outputting no qubits.

2.3 Interpretation

We'll see in more detail what these rules mean, where they come from, and why they have the names they do. For now it is enough to know that combined the monoid rules, Frobenius rules, fusion rules, and identity rules tell us that it doesn't matter how the dots of the same colour are connected, so long as the phases in the dots add to the same value modulo 2π.

We can represent a qubit as an element of $\mathbb{C}^2$. Then the rules about Z spiders tell us how to multiply matrices which are diagonal in the computational basis, $\{|0\rangle, |1\rangle\}$, with eigenvalues $e^{i\alpha}$, and the rules about X spiders tell us how to multiply matrices which are diagonal in the Hadamard transformed basis formed from

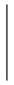
$$|+\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad \text{and} \qquad |-\rangle = H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{2.3.1}$$

The colour change rule tells us how to convert one basis to the other. The bialgebra rule tells us that these bases are complimentary, that they are at the maximal angle to each other. The copy and π-copy rules are just artefacts of nicely chosen bases.

Using this we can develop the **standard model** of ZX calculus, which represents each diagram as a matrix acting on the input qubits. More formally we can define a map

$$[\![-]\!] : (n\text{-to-}m \text{ qubit ZX diagram}) \to (2^n \times 2^m \text{ complex matrices}). \tag{2.3.2}$$

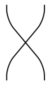
Then a process which takes a single qubit, does nothing to it, and immediately outputs it is represented as



$$\longmapsto \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.3.3}$$

Unsurprisingly doing nothing to a qubit gives the identity.

The following diagram takes in two qubits, swaps them, and then returns them:

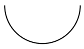


$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.3.4}$$

In terms of matrices this acts as follows:

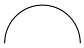
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \left[\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} \right] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix} = \begin{pmatrix} ac \\ bc \\ ad \\ bd \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} \otimes \begin{pmatrix} a \\ b \end{pmatrix}. \tag{2.3.5}$$

It is possible to have diagrams which create qubits from nothing. In this case we should take the input to simply be 1. The following diagram creates a pair of qubits:



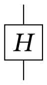
$$\longmapsto \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{2.3.6}$$

Similarly we can destroy qubits:



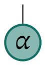
$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}. \tag{2.3.7}$$

The Hadamard gives the **Hadamard matrix**, note that we're ignoring an overall scalar, there's usually a factor of $1/\sqrt{2}$:



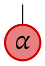
$$\longmapsto \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.3.8}$$

We can create a single qubit:



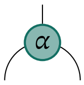
$$\longmapsto \begin{pmatrix} 1 \\ e^{i\alpha} \end{pmatrix},$$

and

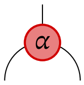


$$\longmapsto \begin{pmatrix} 1 + e^{i\alpha} \\ 1 - e^{i\alpha} \end{pmatrix} \tag{2.3.9}$$

The three-arity spiders give the following matrices

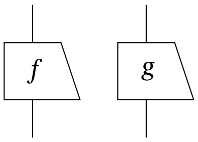


$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{pmatrix}, \tag{2.3.10}$$



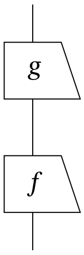
$$\longmapsto \begin{pmatrix} 1 + e^{i\alpha} & 1 - e^{i\alpha} & 1 - e^{i\alpha} & 1 + e^{i\alpha} \\ 1 - e^{i\alpha} & 1 + e^{i\alpha} & 1 + e^{i\alpha} & 1 - e^{i\alpha} \end{pmatrix} \tag{2.3.11}$$

Writing two processes next to each other gives their tensor product:



$$\longmapsto f \otimes g \tag{2.3.12}$$

Writing two processes one after the other connected up represents doing them in the order they are connected from bottom to top, which is composition:



$$\longmapsto g \circ f \tag{2.3.13}$$

Note that for processes represented by matrices composition is just matrix multiplication.

This mapping makes precise what any one ZX diagram represents. What is important is that this isn't changed when we apply the rules of ZX calculus. This is the crux of the following theorem.

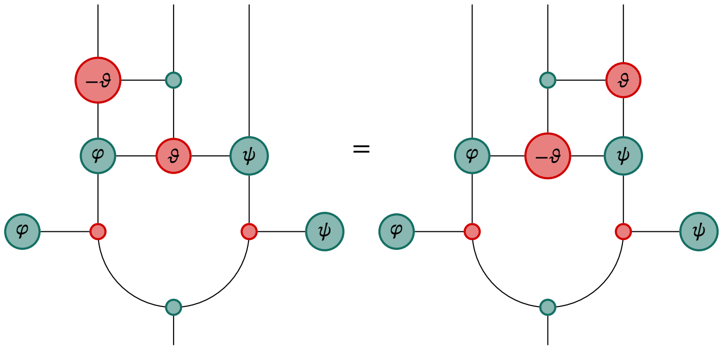
Theorem 2.3.14 — ZX Calculus is Sound. Let D_1 and D_2 be diagrams in ZX calculus. If $D_1 = D_2$ according to the rules of ZX calculus then $[\![D_1]\!] = [\![D_2]\!]$.

As well as being a rigorous way to manipulate objects ZX calculus can also approximate any process from m qubits to n qubits to arbitrary precision.

Theorem 2.3.15 — ZX Calculus is Approximately Universal. For any $2^m \times 2^n$ matrix, f, and any error margin, $\varepsilon > 0$, there exists a diagram, D, in ZX calculus built only from terms with phases an integer multiple of $\pi/4$ such that $\|[\![D]\!] - f\| < \varepsilon$ for some appropriate matrix norm $\|-\|$.

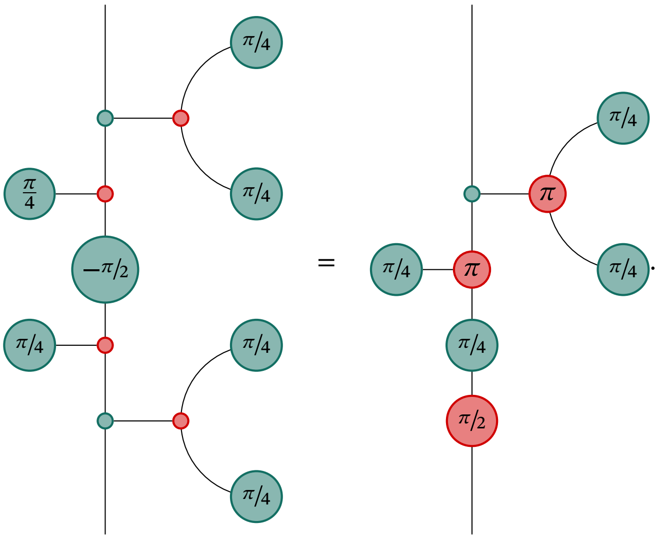
This is one of the reasons that ZX calculus is so powerful.

Another desirable quality for a notation like ZX calculus is that it be complete. By this we mean that if two matrices are equal and both given by some ZX diagram then there should be a graphical proof of this using only the rules of ZX calculus. This is the case if we assume the following two axioms, which are sound under the standard interpretation:



(2.3.16)

for any phases φ, ψ, and ϑ which are integer multiples of $\pi/4$, and the second axiom



(2.3.17)

Call ZX calculus with these rules added **π/4-ZX calculus**.

Theorem 2.3.18 — $\pi/4$-ZX Calculus is Complete. Let D_1 and D_2 be diagrams in $\pi/4$-ZX calculus. If $[\![D_1]\!] = [\![D_2]\!]$ then $D_1 = D_2$ under the axioms of $\pi/4$-ZX calculus.

The third thing making ZX calculus powerful is how well it can be automated. All a ZX calculation is is a finite labelled graph. Once you've implemented a way of applying the rules this can then be done very efficiently on a computer.

A common use of ZX calculus is in quantum circuit optimisation. Given some quantum algorithm as a quantum circuit it is often possible to optimise the circuit. For example, the T gate,

$$T = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{-i\pi/4} \end{pmatrix}, \tag{2.3.19}$$

is typically expensive to implement, so reducing the number of T gates in a circuit is usually desirable. Given some circuit making use of T gates we can use the universality of ZX calculus to convert the circuit into a ZX diagram, then manipulate the ZX diagram and then convert it back to a, hopefully, more optimised circuit with fewer *T* gates.

Three

Semantics

3.1 Types of Semantics

Consider the two following pseudocode fragments:

$$P = \text{if } 1 = 1 \text{ then F else G}, \tag{3.1.1}$$

$$Q = \text{if } 1 = 0 \text{ then F else F}. \tag{3.1.2}$$

Are P and Q the same program? There are two schools of thought:

- No. Clearly looking at them both programs are implemented differently, P makes reference to G, Q makes reference to 0.
- Yes. Both programs take no input and output F.

Whether or not we count these as the same program depends on what we are interested in.

To aid in our analysis we assign the code fragments their meanings, encoded in some appropriate mathematical object. This gives a mapping

$$[\![-]\!] : \text{Programs} \to \text{Mathematical Objects}. \tag{3.1.3}$$

For now we'll leave the details of exactly what mathematical objects alone. If we are interested in implementation details then we assign P and Q to objects encoding these details. This is called **operational semantics**. If we aren't interested in implementation details then we assign P and Q to objects which treat them as black boxes with inputs and outputs. This is called **denotational semantics**. If this is what we do then we find that $[\![P]\!] = [\![Q]\!] = [\![\text{F}]\!]$.

We want this mapping to preserve the structure of our programs. For example, suppose that we have two processes, F and G, which can be composed by running them one after another. We might write this as F; G in a language using semicolons to terminate a line. In order to reason about our program, regardless of which type of semantics we are interested in, we want the result to be the same if we compose the programs and then look at the semantics or look at the semantics and then compose the programs. That is we want

$$[\![\text{F; G}]\!] = [\![\text{F}]\!] \circ [\![\text{G}]\!]. \tag{3.1.4}$$

Here $\circ$ is some method of composing the semantics of two programs. Another structure which we may want to preserve is the ability to compute things in parallel, for example

$$[\![\text{ paralell (F, G)}]\!] = [\![\text{F}]\!] \otimes [\![\text{G}]\!]. \tag{3.1.5}$$

3.2 Motivation

Why might we care about this sort of analysis? This reasoning can be used to ground assumptions and correct erroneous assumptions. We can also use semantics to justify transformations of programs, for example, to demonstrate that a compiled program does the same thing as the original program. It is also often the case that it is easier to reason about the mathematical objects encoding the programs, rather than the programs themselves, in fact sometimes it isn't possible to reason directly about the programs, such as in quantum computing where many operations are like black boxes, even if we can't look at the operational semantics we can still consider the denotational semantics and the flow of information through the program to compare programs.

Choosing different semantics also allows us to focus on different details. Operational semantics focus on implementation details, such as memory usage and running time, which is useful if we want to improve the efficiency of our programs. Denotational semantics focus on results, which is useful if we want to check that our program does what we want.

3.3 Mathematical Objects

There are many possible mathematical objects to encode programs. Simple programs can be modelled as set theoretical functions from some set of possible inputs to some set of possible outputs. More specifically we can use something like λ-calculus to represent programs. In this way we can represent both operational semantics, by writing our functions as expressions in the input variables, and denotational semantics, by focussing on the input and output values.

In quantum computing operational semantics are often not an option. So we will mostly stick to denotational semantics. The objects we choose to represent programs are categories. Category theory supports combing programs as we saw in the examples above and gives us a powerful graphical language for computations.

Part II

Categories

Four

Categories

4.1 Motivation

We want an abstract mathematical formalism in which the meaning of a program lives, allowing us to abstract away implementation details and reason about computations. Such a formalism provides a map

$$[\![-]\!] : \text{programs} \to \text{mathematical objects}. \tag{4.1.1}$$

There are several features which are desirable of such a formalism, including but not limited to,

- a notion of composition, if F and G are programs and $\mathrm{F};\ \mathrm{G}$ is running F then G then we should have $[\![\mathrm{F};\ \mathrm{G}]\!] = [\![\mathrm{G}]\!] \circ [\![\mathrm{F}]\!]$ where $\circ$ represents composition in this mathematical formalism;
- a notion of concurrency, if $\mathrm{F}\ \mathrm{par}\ \mathrm{G}$ corresponds to running F and G at the same time then we should have $[\![\mathrm{F}\ \mathrm{par}\ \mathrm{G}]\!] = [\![\mathrm{F}]\!] \otimes [\![\mathrm{G}]\!]$;
- a notion of calling programs recursively, if X is some code calling F then we should be able to compute $\mathrm{F}(\mathrm{X})$, and this requires our mathematical formalism to have structures of the form $[\![\mathrm{F}(\mathrm{X})]\!] = [\![\mathrm{F}]\!]([\![\mathrm{X}]\!])$.

More concisely, our formalism should preserve composition, concurrency, and function application.

The question we have to ask is what sort of mathematical objects we're considering. There are multiple options, each with their own advantages and disadvantages. Some common options are

- λ-calculus is an algebraic notation for functions. It is similar in syntax to *Haskell*. It works with anonymous functions, or lambda functions, such as the function $\lambda\ \mathtt{x.}\ *\ 2\ \mathtt{x}$ which takes an argument, $\mathtt{x}$, and multiplies it by 2, using prefix notation. Compare this to the *Haskell* function

  ```
  timesThree x = (*) 2 x
  ```

  This choice is good for analysing implementation details and is simply a precise notation for applying functions, a common mathematical operation.

- Partially ordered sets, or posets, where the elements of the poset are partially completed calculations, any two elements are comparable if they are the same calculation at, potentially, different levels of completion, and the more complete calculation is greater. This choice is good for analysing computations step by step without worrying about how each step is implemented.

- Categories, which is what we'll use. This option subsumes both λ-calculus, as a method of defining functions, and posets, which can be regarded as a special case of a category.

Our goal will be to work in a general category to develop theory, imposing only the required restrictions for things to work out. Then we can pick a particular category to analyse our work in, with the interpretation depending on the category we pick. Often we can work in a generic category and then specialise the result to a category to perform either classical or quantum computations.

4.2 Categories: The Idea

A category consists of two pieces of data:

- Objects, $A, B, C, \ldots$;
- Morphisms, $f : A \to B$, between objects.

Given some specific category there are various ways to think of computations occurring in this category. Some examples are given here:

- We can think of the objects as physical systems and the morphisms as processes. For example,
  - Two objects may be a full cup and an empty cup and a process may be drinking the drink or making a new drink.
  - Two objects might be a plate and pieces of broken pottery and a process may be dropping the plate on the floor.
- We can think of objects as data types, and morphisms as functions between these types. Borrowing *Haskell* notation some examples are
  - One object might be Int, and a morphism f :: Int →Int defined by f n = 2 ∗ n.
  - Another object might be String, and a morphism len :: String –> Int defined by

```
len [] = 0
len (x : xs) = 1 + len xs
```

  - Another object might be Num a ⇒[a], and a morphism

```
mySum :: (Num a) ⇒ [a] → a
mySum [] = 0
mySum (x : xs) = x + mySum xs
```

- Objects are algebraic structures and morphisms are structure preserving maps. For example,
  - Objects are sets and morphisms are functions.

  - Objects are groups and morphisms are homomorphisms.
  - Objects are topological spaces and morphisms are continuous functions.
  - Objects are vector spaces and morphisms are linear maps.

- Objects are logical propositions and morphisms are implications between them. For example, we could take the objects "it rains" and "I get wet" and then we might have a morphism "it rains" $\implies$ "I get wet". But what if we're indoors? We might introduce another object, "I am outside", and then we may have a morphism "it rains" $\wedge$ "I am outside" $\implies$ "I get wet". Here we've implicitly defined another object "it rains" $\wedge$ "I am outside" using logical conjunction, $\wedge$. This is actually an example of a product in this category, we'll see what this means later.

It turns out that the second and fourth examples, programs/algorithms and propositions/implications are actually the same! This is known as the Curry–Howard isomorphism, and allows us to write proofs as programs and vice versa.

The mindset that one should have when doing category theory is

> Morphisms are more important than objects.

This might seem backwards at first, for example we spend a lot of time thinking about groups, and homomorphisms are only one aspect that we consider, but it turns out that we can learn a lot about groups by studying how they relate to other groups, and this is done through homomorphisms. This mindset is particularly useful for cases such as quantum computing where we *can't* look at internal structure, and can only look at how systems relate to each other.

4.3 Categories: The Definition

Categories are objects and morphisms. The definition of a category simply states what we mean by this and the properties that morphisms are expected to have.

Definition 4.3.1 — Category A **category**, $\mathbf{C}$, consists of the following data

- a collection of **objects**, $\mathrm{Ob}(\mathbf{C})$ (often denoted $\mathrm{Obj}(\mathbf{C})$ or simply $\mathbf{C}$);
- for every pair of objects, $A, B \in \mathrm{Ob}(\mathbf{C})$ a collection of **morphisms** (also known as **maps** or **arrows**), $\mathbf{C}(A, B)$ (often denoted $\hom_{\mathbf{C}}(A, B)$, $\mathrm{Mor}_{\mathbf{C}}(A, B)$, possibly without the subscript $\mathbf{C}$ when the category is clear, this collection is often called a **hom set**), where for $f \in \mathbf{C}(A, B)$ we write $f : A \to B$ or $A \xrightarrow{f} B$;
- a map $\circ : \mathbf{C}(B, C) \times \mathbf{C}(A, B) \to \mathbf{C}(A, C)$ which assigns to each $f : A \to B$ and $g : B \to C$ some **composite** $(g \circ f) : A \to C$;
- for every object $A \in \mathrm{Ob}(\mathbf{C})$ a morphism $\mathrm{id}_A : A \to A$, that is $\mathrm{id}_A \in \mathbf{C}(A, A)$, called the **identity morphism**.

This data is subject to the following conditions:

- **associativity** of $\circ$: for all objects $A, B, C, D \in \mathrm{Ob}(\mathbf{C})$ and for all morphisms $f : A \to B$, $g : B \to C$, and $h : C \to D$ we have

  $$h \circ (g \circ f) = (h \circ g) \circ f, \tag{4.3.2}$$

  so we can unambiguously write $h \circ g \circ f$ for both of these;
- **identity** law: for all objects $A, B \in \mathrm{Ob}(\mathbf{C})$ and for all morphisms $f : A \to B$ we have

  $$f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f. \tag{4.3.3}$$

4.3.1 Technicality

Notice that in the definition we use the word “collection”. It is tempting to replace this with “set”, but this can cause issues. For example, the set of all sets is not a set, due to Russell’s paradox. However, we will shortly see that we have categories where the objects are all sets, and so $\mathrm{Ob}(\mathbf{C})$ cannot be a set in this case. We aren’t going to worry too much about these types of issues, and may erroneously refer to these collections as sets. A category is **small** if both $\mathrm{Ob}(\mathbf{C})$ and the collection of all morphisms between any two objects are sets. A category is **locally small** if for all objects A and B $\mathbf{C}(A, B)$ is a set. Many statements we make throughout will apply only to small, or more likely locally small categories.

4.4 Categories: The Examples

4.4.1 Set

Definition 4.4.1 — Set The category **Set** has sets as objects. A morphism $A \to B$ is simply a function from A to B. Composition of morphisms is composition of functions, defined for $f : A \to B$ and $g : B \to C$ by $(g \circ f) : A \to C$ given by $(g \circ f)(a) = g(f(a))$ for all $a \in A$. The identity morphism is the identity function, $\mathrm{id}_A : A \to A$, given by $\mathrm{id}_A(a) = a$ for all $a \in A$.

Lemma 4.4.2 **Set** is a category.

Proof. The collection $\mathbf{Set}(A, B)$ of functions $A \to B$ exists for all sets A and B. It will be empty if $B = \varnothing$ and $A \neq \varnothing$ which is allowed, if $A = \varnothing$ and $B = \varnothing$ then there is a unique function $f : \varnothing \to \varnothing$ which is also the identity on the empty set. Function composition is associative. Take sets A, B, C, and D, and morphisms $f : A \to B$, $g : B \to C$, and $h : C \to D$. Then

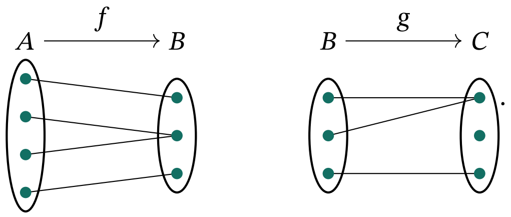
$$(h\circ(g\circ f))(x) = h((g\circ f)(x)) = h(g(f(x))) = (h\circ g)(f(x)) = ((h\circ g)\circ f)(x) \tag{4.4.3}$$

for all $x \in A$, so $h \circ (g \circ f) = (h \circ g) \circ f$. The identity function is then such that

$$(f \circ \mathrm{id}_A)(x) = f(\mathrm{id}_A(x)) = f(x) = \mathrm{id}_B(f(x)) = (\mathrm{id}_B \circ f)(x) \qquad (4.4.4)$$

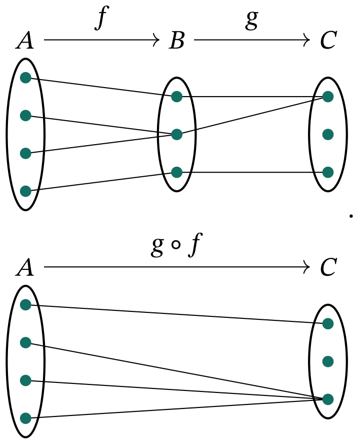
and so $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$. □

We can think of a function $f : A \to B$ as dynamically indicating how elements of A transform into elements of B. Pictorially, we can represent $f : A \to B$ and $g : B \to C$ as



(4.4.5)

Then composition is just following the lines between sets:


$A \xrightarrow{f} B \xrightarrow{g} C$
$A \xrightarrow{g \circ f} C$

(4.4.6)

As a process composition in set is just doing one thing and then the other.

Set is the prototypical category. It is often tempting to think of other examples of categories, which we'll meet shortly, as simply sets with extra structure, and indeed this is often, but not always, the case. A **concrete category** is a category in which all objects can be mapped to sets and morphisms can be mapped to functions between these sets. This mapping is done with something called a functor, which we'll define later (Definition 4.8.1).

We'll list some properties of **Set** here, some of which won't make sense until later.

- **Set** is a concrete category.
- **Set** has the empty set as an initial object, and the singleton as a terminal object.
- **Set** is complete and co-complete, in particular the product is the Cartesian product and the coproduct is the disjoint union.

- **Set** is a monoidal category with the monoidal product given by the Cartesian product.

4.4.2 Rel

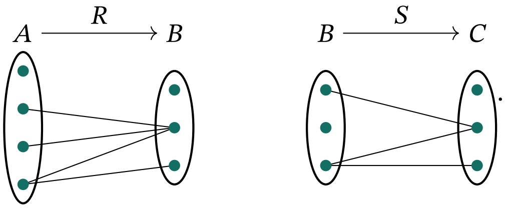
> **Definition 4.4.7 — Relation** Let A and B be sets. A **relation**, R, is a subset of $A \times B$. If $(a, b) \in R$ we write aRb.

Relations are morphisms in a category we will defined shortly (Definition 4.4.15), so for $R \subset A \times B$ we write $R : A \to B$. In a similar manner to functions between sets being dynamic transformations we can think of relations as being non-deterministic transformations, where each element can transform into multiple objects, or possibly don't map across at all. For example, if we take $A = \{a, b, c, d\}$, $B = \{1, 2, 3\}$, and $C = \{\alpha, \beta, \gamma\}$ then the relations

$$R = \{(b, 2), (c, 2), (d, 2), (d, 3)\} \subseteq A \times B, \qquad \text{and} \tag{4.4.8}$$

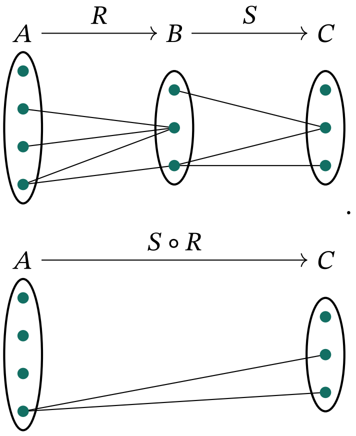
$$S = \{(1, \beta), (3, \beta), (3, \gamma)\} \subseteq B \times C \tag{4.4.9}$$

can be represented pictorially as



(4.4.10)

As with **Set** we then compose relations by joining up these lines:



(4.4.11)

That is,

$$S \circ R = \{(d, \beta), (d, \gamma)\} \subseteq A \times C. \tag{4.4.12}$$

This leads to the following definition.

Definition 4.4.13 — Relation Composition Let A, B, and C be sets with relations $R \subseteq A \times B$ and $S \subseteq B \times C$. Then the **composite relation** $S \circ R$ is the set

$$S \circ R := \{(a,c) \in A \times C \mid \exists b \in B \text{ such that } (a,b) \in R \text{ and } (b,c) \in S\}.$$

Now that we have composition we just need an identity, and after some playing around with the definition of composition one quickly comes to the identity

$$\mathrm{id}_A := \{(a,a) \in A \times A \mid a \in A\} \subseteq A \times A. \tag{4.4.14}$$

Now we can define a category.

Definition 4.4.15 — Rel The category **Rel** has sets as objects. A morphism $R\colon A \to B$ is a relation $R \subseteq A \times B$. Composition of morphisms is composition of relations, defined for $R\colon A \to B$ and $g\colon B \to C$ by

$$S \circ R = \{(a,c) \mid \exists b \in B : aRb \wedge bSc\} \subseteq A \times C. \tag{4.4.16}$$

The identity morphism is the identity relation, $\mathrm{id}_A : A \to A$, given by

$$\mathrm{id}_A = \{(a,a) \mid a \in A\} \subseteq A \times A. \tag{4.4.17}$$

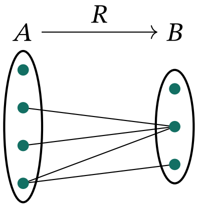
Lemma 4.4.18 **Rel** is a category.

Proof. The collection $\mathbf{Rel}(A,B)$ is simply the power set of $A \times B$. Importantly the empty set is a relation on any pair of sets, including the empty set, and the empty set is the identity relation on the empty set. So consider nonempty sets.

Composition of relations is associative. Let $R\colon A \to B$, $S\colon B \to C$, and $T\colon C \to D$ be relations. We want to show that $T \circ (S \circ R) = (T \circ S) \circ R$. To do so we will prove that each pair $(a,b) \in T \circ (S \circ R)$ is also an element of $(T \circ S) \circ R$. The converse, that each pair $(a,b) \in (T \circ S) \circ R$ is an element of $T \circ (S \circ R)$, follows by the same logic in reverse. So take some $(a,b) \in T \circ (S \circ R)$. By definition there exists some x such that $(x,b) \in T$ and $(a,x) \in S \circ R$. Thus, there exists some y such that $(y,x) \in S$ and $(a,y) \in S$. Since $(y,x) \in S$ and $(x,b) \in T$ we have that $(y,b) \in T \circ S$. Since $(a,y) \in R$ we have $(a,b) \in (T \circ S) \circ R$.

Let $R\colon A \to B$ be a relation and $\mathrm{id}_A = \{(a,a) \mid a \in A\}$ the identity relation. Then for all $(a,b) \in R$ we have $(a,a) \in \mathrm{id}_A$, meaning that $(a,b) \in R \circ \mathrm{id}_A$. Similarly for all $(a,b) \in R \circ \mathrm{id}_A$ we must have x such that $(x,b) \in R$, and $(x,a) \in \mathrm{id}_A$, which means that $x = a$ and so $(a,b) \in R$. Thus $R \circ \mathrm{id}_A = R$. Similarly $\mathrm{id}_B \circ R = R$. Hence the identity law is satisfied. □

We can represent relations as binary $|B| \times |A|$ matrices with a 1 in the (i,j) slot if $(a,b) \in R$, where a is the jth element of A and b the ith element of B in some arbitrary fixed ordering of A and B. Further this mapping is one-to-one, meaning each binary matrix also defines a representation. For example, indexing top to bottom we have



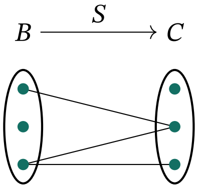
$$\rightsquigarrow M(R) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4.19}$$

Composition of relations is then given by matrix multiplication taking everything mod 2. More formally we have a map

$$M\colon \mathcal{P}(A \times B) \to \mathcal{M}_{|B|\times|A|}(\mathbb{Z}_2) \tag{4.4.20}$$

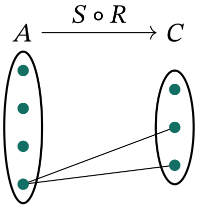
where $\mathbb{Z}_2 = \{0, 1\}$ has addition and multiplication defined mod 2.

As an example notice that we have



$$\rightsquigarrow M(S) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.4.21}$$

and



$$\rightsquigarrow M(S \circ R) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4.22}$$

The composite can then be calculated as

$$M(S\circ R) = M(S)M(R) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4.23}$$

This map, M, actually defines a functor (Definition 4.8.1) $M\colon \mathbf{Rel} \to \mathbf{Mat}_{\mathbb{Z}_2}$, where $\mathbf{Mat}_{\mathbb{Z}_2}$ is the category of binary matrices, whose objects are natural numbers and a morphism $n \to m$ is an $n \times m$ matrix. The functor M then maps sets to their size, $A \mapsto M(A) = |A|$, and relations to matrices as described above.

This is actually an example of a more general idea of a representation[1], where we replace objects with matrices preserving the structure of the original space. These ideas can all be understood as functors $\mathbf{C} \to \mathbf{Mat}_{\mathbb{k}}$ for some appropriate category $\mathbf{C}$ and ring $\mathbb{k}$. This in turn is more commonly thought of as a map $\mathbf{C} \to \mathbf{FVect}_{\mathbb{k}}$ identifying matrices with linear maps. This allows for generalisations to infinite dimensional representations, $\mathbf{C} \to \mathbf{Vect}_{\mathbb{k}}$. We'll define $\mathbf{FVect}_{\mathbb{k}}$ and $\mathbf{Vect}_{\mathbb{k}}$ shortly (Definitions 4.4.28 and 4.4.35).

[1] see *Symmetries of Quantum Mechanics* or *Symmetries of Particles and Fields*

On the surface **Rel** seems quite similar to **Set**, after all the objects of both are the same. However, it's really the morphisms which are important, and these are quite different. The ability to replace relations with matrices means that **Rel** is actually quite similar to another category, **Hilb** (Definition 4.4.47). This makes **Rel** a nice in between for **Set** and **Hilb**, which turn out to be the categories in which we think of most classical and quantum computing as occurring in respectively.

We now list some properties of **Rel**:

- **Rel** is a concrete category.
- **Rel** is a dagger category.
- **Rel** has both products and coproducts given by disjoint union.
- **Rel** is a monoidal category with the monoidal product given by the Cartesian product.

4.4.3 $\mathbf{Vect}_{\Bbbk}$, $\mathbf{FVect}_{\Bbbk}$, **Hilb**, and **FHilb**

4.4.3.1 $\mathbf{Vect}_{\Bbbk}$ and $\mathbf{FVect}_{\Bbbk}$

Definition 4.4.24 — Vector Space A **vector space**, $(V, \Bbbk, +, \cdot)$, is a set, V, a field[a], $\Bbbk$, and two operations, **vector addition**, $+\colon V \times V \to V$, and **scalar multiplication**, $\cdot\colon \Bbbk \times V \to V$, such that

- $(V, +)$ is an Abelian group, that is
  - vector addition is associative: $u + (v + w) = (u + v) + w$ for all $u, v, w \in V$;
  - vector addition is commutative: $u + v = v + u$ for all $u, v \in V$;
  - additive identity: there exists $0 \in V$ such that $0 + v = v$ for all $v \in V$;
  - additive inverse: for every $v \in V$ there exists $-v \in V$ such that $v + (-v) = v - v = 0$;
- scalar multiplication distributes over vector addition: $\alpha \cdot (u + v) = (\alpha \cdot v) + (\alpha \cdot v)$ for all $\alpha \in \Bbbk$ and $u, v \in V$;
- field identity acts as the identity: $1 \cdot v = v$ for all $v \in V$ where 1 is the multiplicative identity in $\Bbbk$;
- field addition distributes over scalar multiplication: $(\alpha + \beta) \cdot v = (\alpha \cdot v) + (\beta \cdot v)$;
- compatibility of field and scalar multiplication: $\alpha \cdot (b \cdot v) = (a \cdot_{\Bbbk} b) \cdot v$ for all $\alpha, \beta \in \Bbbk$ and $v \in V$ where $\cdot_{\Bbbk}$ is multiplication in the field.

[a] if you don't know what this is just replace it with $\mathbb{R}$ or $\mathbb{C}$ and don't worry about it

Definition 4.4.25 — Linear Map Let $(V, \Bbbk, +_V, \cdot_V)$ and $(W, \Bbbk, +_W, \cdot_W)$ be vector spaces over the same field, $\Bbbk$. A **linear map** between these vector spaces is a function $T\colon V \to W$ such that

$$T(u +_V v) = T(u) +_W T(v), \qquad \text{and} \qquad T(\alpha \cdot_V v) = \alpha \cdot_W T(v) \quad (4.4.26)$$

for all $\alpha \in \Bbbk$ and $u, v \in V$.

From now on we drop the explicit symbol for scalar multiplication, writing αv for $\alpha \cdot v$, as well as dropping labels differentiating which vector space an operation is defined on. So linearity is expressed as

$$T(u + v) = T(u) + T(v), \qquad \text{and} \qquad T(\alpha v) = \alpha T(v). \qquad (4.4.27)$$

We also refer to V and W as vector spaces (over $\Bbbk$) leaving the operations (and potentially the field) implicit.

Definition 4.4.28 — $\mathbf{Vect}_\Bbbk$ Fix some field $\Bbbk$. The category $\mathbf{Vect}_\Bbbk$ has vector spaces over $\Bbbk$ as objects and linear maps as morphisms. Composition of morphisms is composition of linear maps, which is composition of the underlying functions. The identity morphisms are the identity linear maps, which are the underlying identity functions.

Lemma 4.4.29 $\mathbf{Vect}_\Bbbk$ is a category.

Proof. Composition of linear maps inherits associativity from the underlying functions and similarly the identity laws follow from the identity laws of the underlying functions. We therefore only need to show that the composite of two linear maps is again linear. Let $T\colon U \to V$ and $S\colon V \to W$ be linear maps between vector spaces U, V, and W over some field $\Bbbk$. Then for all $u, v \in V$ and $\alpha \in \Bbbk$ we have

$$\begin{aligned}(S \circ T)(u + v) &= S(T(u + v)) = S(T(u) + T(v)) \\ &= S(T(u)) + S(T(v)) = (S \circ T)(u) + (S \circ T)(v) \end{aligned} \quad (4.4.30)$$

using the linearity of T and then S, and

$$(S \circ T)(\alpha v) = S(T(\alpha v)) = S(\alpha T(v)) = \alpha S(T(v)) = \alpha(S \circ T)(v) \quad (4.4.31)$$

again using linearity of T and then S. Hence the composite of two linear maps is again a linear map. □

Definition 4.4.32 — Basis Let V be a vector space. A subset $\mathcal{B} \subset V$ is **linearly independent** if for every finite subset of $\{e_1, \dots, e_n\} \subseteq \mathcal{B}$ satisfies

$$\alpha_1 e_1 + \dots + \alpha_n e_n = 0 \qquad (4.4.33)$$

only for the trivial case of $\alpha_i = 0$.

A subset $\mathcal{B} \subset V$ spans V if every $v \in V$ can be written as

$$v = \alpha_1 e_1 + \cdots + \alpha_n e_n \tag{4.4.34}$$

for some finite subset $\{e_1, \dots, e_n\} \subseteq \mathcal{B}$. We call α_i the **components** of v. If a subset $\mathcal{B} \subset V$ is linearly independent and spans V then we call it a **basis**.

Every vector space has a basis (assuming the axiom of choice). It is a fact that any two bases have the same cardinality, which we call the **dimension** of the space. A vector space is **finite-dimensional** if it's dimension is finite.

Definition 4.4.35 — $\mathbf{FVect}_{\Bbbk}$ The category $\mathbf{FVect}_{\Bbbk}$ has finite-dimensional vector spaces as objects and linear maps as morphisms.

Lemma 4.4.36 $\mathbf{FVect}_{\Bbbk}$ is a category.

Proof. This follows from $\mathbf{Vect}_{\Bbbk}$ being a category. □

Linear maps are often thought of as matrices, although this only works in the finite-dimensional case. Take two vector spaces V and W with bases $\{v_i\}$ and $\{w_i\}$ respectively. Then a linear map $T\colon V \to W$ defines a matrix with components T_{ij} given by $T(v_i)_j$, where $T(v_i)_j$ is the jth component of $T(v_i)$, which is what we get evaluating the linear map at the ith basis vector, v_i. A matrix, T_{ij}, defines a linear map $T\colon V \to W$ similarly, by defining $T(v_i)$ to be formed from components $T(v_i)_j = T_{ij}$, and then using linearity to extend this definition to any vector in V.

Definition 4.4.37 — $\mathbf{Mat}_{\Bbbk}$ The category $\mathbf{Mat}_{\Bbbk}$ has natural numbers as objects with a morphism $n \to m$ being an $m \times n$ matrix with entries in $\Bbbk$. Composition of morphisms is matrix multiplication and the identity matrix is the identity morphism.

Lemma 4.4.38 $\mathbf{Mat}_{\Bbbk}$ is a category.

Proof. The product of an $m \times n$ matrix and a $n \times \ell$ matrix is an $m \times \ell$ matrix, reflecting the fact we can compose morphisms $\ell \to n$ and $n \to m$ to get a morphism $\ell \to m$. Associativity follows from associativity of matrix multiplication and the identity matrix is clearly the identity. □

There is an equivalence (Definition 4.8.5) $\mathbf{Mat}_{\Bbbk} \to \mathbf{FVect}_{\Bbbk}$ given by sending $n \to \Bbbk^n$ and sending a matrix to a linear map as described above. This formalises the idea that linear maps and matrices are equivalent ways of doing linear algebra in finite dimensions.

4.4.3.2 **Hilb** and **FHilb**

Definition 4.4.39 — Inner Product Space An **inner product space**, $(V, \langle - | - \rangle)$, is a vector space, V, over the field $\Bbbk = \mathbb{R}, \mathbb{C}$ equipped with an **inner product** $\langle - | - \rangle : V \times V \to \Bbbk$ such that the inner product is

- conjugate symmetric: $\langle u|v\rangle = \langle v|u\rangle^*$ for all $u, v \in V$;
- linear in the *second* argument: $\langle u|\alpha v\rangle = \alpha\langle u|v\rangle$ for all $\alpha \in \Bbbk$ and $u, v \in V$, this implies antilinearity in the first argument: $\langle \alpha u|v\rangle = \alpha^*\langle u|v\rangle$;
- positive-definite: $\langle v|v\rangle > 0$ for all $v \in V$ with $v \neq 0$ and $\langle 0|0\rangle = 0$, note that conjugate symmetry implies $\langle v|v\rangle = \langle v|v\rangle^*$ so $\langle v|v\rangle$ is real.

Note that for the $\Bbbk = \mathbb{R}$ case we can simply ignore the complex conjugates.

Definition 4.4.40 — Hilbert Space A **Hilbert space** is an inner product space $(H, \langle - | - \rangle)$ such that H is complete with respect to the norm $\| - \| : H \to \mathbb{R}$ defined by $\|v\| := \sqrt{\langle v|v\rangle}$, we say that this is the norm induced by the inner product. Being complete means that if the sequence $\{v_i\} \subseteq H$ is such that

$$\sum_{i=1}^{\infty} \|v_i\| \tag{4.4.41}$$

converges (as a series in $\mathbb{R}$) then

$$\sum_{i=1}^{\infty} v_i \tag{4.4.42}$$

converges to some $v \in H$, in the sense that for all $\varepsilon > 0$ there exists some $N \in \mathbb{N}$ such that for all $n > N$ we have

$$\left\| v - \sum_{i=1}^{n} v_i \right\| < \varepsilon, \tag{4.4.43}$$

or equivalently,

$$\lim_{n} \left\| v - \sum_{i=1}^{n} v_i \right\| = 0. \tag{4.4.44}$$

We will assume that Hilbert spaces are inner product spaces over $\mathbb{C}$ unless stated otherwise, although most facts will hold for real Hilbert spaces as well. Complex Hilbert spaces are where all quantum mechanics takes place, so we don't lose much by making this assumption.

This requirement of completeness is really just a technical requirement to make things well defined in certain circumstances, and we won't ever have reason to make use of it explicitly. For our purposes inner product space and Hilbert space are basically synonyms, but we're slightly safer working in Hilbert spaces due to this extra condition.

Definition 4.4.45 — Bounded Linear Map Let H and K be Hilbert spaces with norms $\|-\|_H$ and $\|-\|_K$ respectively, both induced by the inner product. A **bounded linear map** is a map $T\colon H \to K$ such that

$$\|T(v)\|_K \leq M\|v\|_H \tag{4.4.46}$$

for some $M \in \mathbb{R}$ and all $v \in H$.

Definition 4.4.47 — Hilb and FHilb Fix some field $\Bbbk = \mathbb{R}, \mathbb{C}$. The category **Hilb** has Hilbert spaces as objects and bounded linear maps as morphisms. The category **FHilb** has finite-dimensional Hilbert spaces as objects and bounded linear maps as morphisms.

Lemma 4.4.48 **Hilb** and **FHilb** are categories.

Proof. Associativity and identities follow from the underlying functions. We need only show that the composite of two bounded linear maps is again a bounded linear map. Let $T\colon H \to K$ and $S\colon K \to J$ be bounded linear maps between the Hilbert spaces H, K, and J. Then there exist some $M, N \in \mathbb{R}$ such that $\|T(v)\|_K \leq M\|v\|_H$ and $\|S(u)\|_J \leq N\|u\|$ for all $v \in H$ and $u \in K$. Then we have $\|(S \circ T)(v)\|_J = \|S(T(v))\|_J \leq N\|T(v)\|_K \leq NM\|v\|_H$ for all $v \in V$, and $NM \in \mathbb{R}$, so the map is bounded again. The composite of two linear maps is linear, as shown in Lemma 4.4.29, so the composite of two bounded linear maps is a bounded linear map. □

We now make a few basic definitions.

Definition 4.4.49 — Orthonormal Let H be a Hilbert space and $\{e_i\}$ a basis of H, then we say that this basis is **orthogonal** if $\langle e_i|e_j\rangle = 0$ for $i \neq j$. If $\langle e_i|e_j\rangle = \delta_{ij}$ we say the basis is **orthonormal**.

Definition 4.4.50 — Adjoint If H and K are Hilbert spaces and $T\colon H \to K$ is a linear map then we define the **adjoint** to be the linear map $T^\dagger\colon K \to H$ such that $\langle T(v)|w\rangle = \langle v|T^\dagger(w)\rangle$.

In terms of matrices representing linear maps the adjoint corresponds to the conjugate transpose matrix.

Definition 4.4.51 — Bras and Kets Let H be a Hilbert space. Given some $v \in H$ its **ket** is the map $|v\rangle\colon \mathbb{C} \to H$ defined by $z \mapsto zv$ and its bra is the map $\langle v|\colon H \to \mathbb{C}$ defined bg $w \mapsto \langle v|w\rangle$.

This definition is rather formal and doesn't correspond to how we usually think about bras and kets. Typically we think about elements of H as being kets. This works since each linear map $\mathbb{C} \to H$ is completely defined by where it sends 1, so really linear maps of this form just pick out elements of H, and $|v\rangle$ is such that $1 \mapsto 1v = v$. So we often write $|v\rangle$ when we might actually mean v by this definition. We then think of bras similarly as being their own vectors in some other space, which we'll define in the next definition, and then we think of the mapping $w \mapsto \langle v|w\rangle$ as simply performing multiplication $\langle v||w\rangle = \langle v|w\rangle$.

Definition 4.4.52 — Dual Space Let V be a vector space. Then $V^* := \mathbf{Vect}_{\Bbbk}(V, \Bbbk)$ is the **dual space**.

This definition means that V^* is the space of linear maps $V \to \Bbbk$, which in the case of a Hilbert space we can recognise as the space of bras. Note that we are equipping $\mathbf{Vect}_{\Bbbk}(V, \Bbbk)$ with the obvious vector space (or Hilbert space) structure given by adding and scaling linear maps pointwise.

4.4.4 More Examples

Example 4.4.53 — Sets with Structure Many categories can be described as having objects formed from sets with structure, and morphisms being structure preserving functions. These are all concrete categories, in the sense that we can forget the structure and just think of them as sets and functions. Some examples of these sets with structure are

- The category **Mon** has monoids as objects and monoid homomorphisms as morphisms.
- The category **Grp** has groups as objects and group homomorphisms as morphisms.
- The category **Ring** has rings as objects and ring homomorphisms as morphisms.
- The category **CRing** has commutative rings as objects and ring homomorphisms as morphisms.
- The category **Field** has fields as objects and field homomorphisms as morphisms.
- The category R–**Mod** for a fixed ring R has left modules over R as objects and module homomorphisms as morphisms. Note that the special case where R is a field is $\mathbf{Vect}_R$.
- The category **Top** has topological spaces as objects and continuos maps as morphisms.
- The category $\mathbf{Top}_{\bullet}$ has pointed topological spaces, $(X, \bullet)$, as objects, that is topological spaces with some special point, $\bullet$, and based maps as morphisms, that is continuous maps preserving this special point, that is $f\colon (X, \bullet) \to (Y, *)$ is continuous and $f(\bullet) = *$.

- The category **Pos** has posets as objects and monotone functions as morphisms.

See Chapter A for relevant definitions.

Example 4.4.54 — Posets Given a poset, $(P, \leq)$, we can define a category whose objects are elements of P and there is a unique morphism $a \to b$ if $a \leq b$. Since this morphism is unique and $a \leq a$ the identity is simply the unique morphism $a \to a$. Since $a \leq b$ and $b \leq c$ implies $a \leq c$ there is a unique morphism $a \to c$ in this case, which must then be the morphism formed by composing the morphisms $a \to b$ and $b \to c$.

Example 4.4.55 — Single Object Categories Given a monoid, M, we can interpret it as a category with a single object, which we call $\bullet$, and then the elements of M are morphisms $\bullet \to \bullet$ with composition of morphisms given by the monoid product. The identity of the monoid is the identity morphism. In a sense categories just generalise monoids to have more objects. The process of defining a quantity, then mapping the definition to a particular category with a single object, and then allowing there to be multiple objects is called **oidification**, and the resulting object is suffixed with -oid. So a monoidoid is a category.
Given a group, G, we can interpret it as a single object category where all morphisms are isomorphisms (Definition 4.6.1). This extra condition is simply reflecting the condition that a group has all inverses. A **groupoid** is then a category in which all morphisms are isomorphisms.
Note that we can proceed in the opposite direction, given a category with a single object (with all morphisms being isomorphisms) this can be interpreted as a monoid (group).

4.5 Diagrams

Composition of morphisms can be quite confusing. There is lots of data to specify, which objects are involved, what are the morphisms called, do the morphisms have any other properties we might be interested in and so on. It doesn't help that the order in which we write composition of functions is the reverse of the order in which the functions are applied. The solution to this is to draw pictures with all of the objects as nodes and the morphisms as arrows between them. Composition of morphisms is then given by reading the arrows backwards. A simple example is

$$A \xrightarrow{f} B \xrightarrow{g} C \tag{4.5.1}$$

which represents two morphisms, $f : A \to B$ and $g : B \to C$, which can be composed to give $(g \circ f) : A \to C$. We might add this morphism to our diagram,

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g\circ f"'] & B \arrow[r, "g"] & C \end{tikzcd} . \tag{4.5.2}$$

although we don't have to since it's existence is ensured by the definition of a category. Both paths taken in this diagram give the same result.

In general if all paths in a diagram between two fixed objects give the same result we say that the diagram **commutes**, or is a **commutative diagram**. This can be useful to specify a lot of algebraic relations in a single commutative diagram. For example, the diagram

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[r, "g"] \arrow[d, "i"'] & C \\
D \arrow[r, "k"'] & E \arrow[ur, "j"'] &
\end{tikzcd} \tag{4.5.3}$$

commuting is equivalent to the following requirements:

$$g \circ f = j \circ k \circ h, \qquad i \circ f = k \circ h, \qquad \text{and} \qquad g = j \circ i, \tag{4.5.4}$$

which are given by requiring that the outer parallelogram commutes, the square commutes, and the triangle commutes respectively.

Since the definition of a category means every object has an identity morphism and that these identity morphisms don't really affect composition we leave the identity morphisms implicit in the diagram. We could write them in, for example

$$\begin{tikzcd}
A \arrow[loop left, "\mathrm{id}_A"] \arrow[r, "f"] & B \arrow[loop right, "\mathrm{id}_B"]
\end{tikzcd} \tag{4.5.5}$$

commuting is simply the identity law, telling us that, among other things, $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$. We can also state the associativity law as the commutativity of the following:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[rr, bend left, "g\circ f"] & B \arrow[r, "g"] \arrow[rr, bend right, "h\circ g"'] & C \arrow[r, "h"] & D.
\end{tikzcd} \tag{4.5.6}$$

4.6 Terminology

We now introduce some terminology which will be helpful.

Definition 4.6.1 For a morphisms $f : A \to B$

- we call A the **domain**, or **source**, of f;
- we call B the **codomain**, or **target**, of f;
- we call f an **endomorphism** if $A = B$;
- we call f an **isomorphism** if there exists $f^{-1} : B \to A$ such that $f^{-1} \circ f = \mathrm{id}_A$ and $f \circ f^{-1} = \mathrm{id}_B$;
- we call f an **automorphism** if it is an isomorphism and endomorphism;

- we call A **isomorphic** to B if f is an isomorphism, and write $A \cong B$;
- we call f an **epimorphism**, or **epic**, if $g \circ f = h \circ f$ implies $g = h$ for all $g, h \colon B \to C$;
- we call f a **monomorphism**, or **monic**, if $f \circ g = f \circ h$ implies $g = h$ for all $g, h \colon C \to A$.

The most important definition here is isomorphisms. Often equality is too strict a requirement for comparing two objects. Instead isomorphism is usually the correct level of similarity. In particular in a concrete category objects are isomorphic if there is an invertible structure preserving map between them, giving a one-to-one pairing of elements. This allows for trivial differences between objects, like renaming of elements or operations, to be ignored. For example, the groups $(\{0, 1\}, +_2)$ and $(\{1, -1\}, \cdot)$ are not equal, but they are isomorphic.

Lemma 4.6.2 Let $\mathbf{C}$ be a category with objects A and B. If $f \colon A \to B$ is an isomorphism then the inverse is unique.

Proof. Suppose this wasn't the case, so $f \colon A \to B$ has two inverses, $g, g' \colon B \to A$, which are both such that $g \circ f = g' \circ f = \mathrm{id}_A$ and $f \circ g = f \circ g' = \mathrm{id}_B$. Then we have

$$g = g \circ \mathrm{id}_B = g \circ (f \circ g') = (g \circ f) \circ g' = \mathrm{id}_A \circ g' = g'. \tag{4.6.3}$$

□

The condition that f and f^{-1} are inverses is equivalent to requiring that

$$\begin{tikzcd} A \arrow[r, shift left, "f"] & B \arrow[l, shift left, "f^{-1}"] \end{tikzcd} \tag{4.6.4}$$

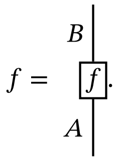
commutes.

4.7 Graphical Notation

In this section we will introduce a graphical notation which can be used to reason about categories. This notation treats categories as processes, taking some input and producing some output. We read the notation from bottom to top, which we can imagine is the progression of "time" through the process. This notation obeys the rules, and spirit, of category theory. In particular, we don't write objects. Instead we represent each object through it's identity morphism, so A is represented by id_A. An identity morphism is then represented by a line, labelled by the object, representing a process in which nothing happens, the input at the bottom of the page, is just passed directly to the output at the top of the page:

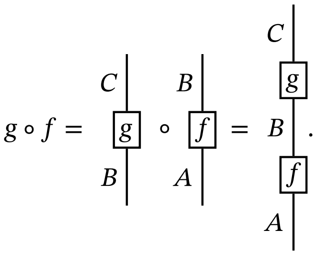
$$\mathrm{id}_A = A\,\Bigg|. \tag{4.7.1}$$

A general morphism, $f : A \to B$, is represented in this graphical notation by placing a box on the wire, representing some process occurring, and labelling the wire either side with the appropriate object:



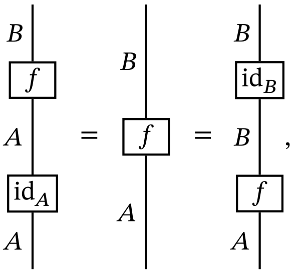
(4.7.2)

Composition of morphisms is represented by doing one process after the other, which in this notation just means writing one box after the other and joining them by a wire of the appropriate type. If we introduce a second morphism $g : B \to C$ then we have



(4.7.3)

This graphical notation makes the axioms of a category implicit. For the identity law since identity morphisms are just drawn as wires clearly the following holds:



(4.7.4)

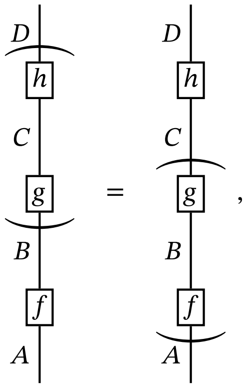
and this is just the identity law

$$f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f. \quad (4.7.5)$$

Similarly we don't bother drawing brackets around composites, since associativity makes them unnecessary. If we did draw brackets then considering the objects and morphisms

$$A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D \quad (4.7.6)$$

we can write the associativity law as



, (4.7.7)

which is just

$$(h \circ g) \circ f = h \circ (g \circ f). \tag{4.7.8}$$

So by manipulating this graphical notation naturally we apply the category axioms. This makes this notation very powerful. While this notation looks new if we were to instead draw it horizontally and replace wires with $\circ$ and id_X then it would just be the usual algebraic notation for morphism composition. The real power of this notation comes when we introduce new concepts, such as monoidal products, which extend the graphical notation into a two-dimensional notation which works in much the same way, while the usual algebraic notation quickly becomes cluttered and hard to use.

4.8 Functors

The spirit of category theory is that it is not objects that are important but morphisms between them. This suggests that whenever we see a mathematical object we should look for maps between them preserving the relevant structure. Well, categories are objects, so we should look for maps between them. Such maps should preserve the category's structure, which is

- the collection of objects;
- the collections of morphisms;
- the composition of morphisms;
- the identity morphisms.

For example, if we have a morphism $f : A \to B$ in one category then our map should send this to another morphism in the new category, and the domain and codomain of that morphism should somehow be related to A and B. It shouldn't matter whether we do composition of morphisms and then map, or map and the compose morphisms. Identities should map to identities. To this end we make the following definition of a map between categories preserving this structure.

Definition 4.8.1 — Functor Let $\mathbf{C}$ and $\mathbf{D}$ be categories. A **functor**, $F\colon \mathbf{C} \to \mathbf{D}$, is composed of two mappings:

- each object $A \in \mathrm{Ob}(\mathbf{C})$ is mapped to some object $F(A) \in \mathrm{Ob}(\mathbf{D})$;
- each morphism $f \in \mathbf{C}(A, B)$ is mapped to some morphism $F(f) \in \mathbf{D}(F(A), F(B))$;

such that

- composition is preserved: $F(g \circ f) = F(g) \circ F(f)$ for $f\colon A \to B$ and $g\colon B \to C$ morphisms in $\mathbf{C}$;
- identities are preserved: $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$ for $A \in \mathrm{Ob}(\mathbf{C})$.

Notation 4.8.2 It is common to drop the brackets when applying a functor, a bit like we may apply a linear map, T, to some vector, v, by writing Tv and thinking of it as matrix multiplication. In this case a functor $F\colon \mathbf{C} \to \mathbf{D}$ maps each $A \in \mathrm{Ob}(\mathbf{C})$ to some object $FA \in \mathrm{Ob}(\mathbf{D})$, each morphism $f \in \mathbf{C}(A, B)$ to some morphism $Ff \in \mathbf{D}(FA, FB)$, such that $F(g \circ f) = Fg \circ Ff$ and $F\,\mathrm{id}_A = \mathrm{id}_{FA}$.

Note that what we have defined above is a **covariant functor**. It is also possible to define a **contravariant functor** which reverses the direction of morphisms. The difference in the definition is that each $f \in \mathbf{C}(A, B)$ is assigned to some $Ff \in \mathbf{D}(FB, FA)$ and $F(g \circ f) = Ff \circ Fg$. We'll assume that all functors are covariant.

Example 4.8.3

- The constant functor, $C_X\colon \mathbf{C} \to \mathbf{D}$, maps every object of $\mathbf{C}$ to $X \in \mathrm{Ob}(\mathbf{D})$ and every morphism $f\colon A \to B$ in $\mathbf{C}$ to the identity morphism $\mathrm{id}_X\colon X \to X$ in $\mathbf{D}$.
- The identity functor, $\mathrm{id}_{\mathbf{C}}\colon \mathbf{C} \to \mathbf{C}$ maps every object and morphism in $\mathbf{C}$ to itself.
- The power set can be regarded as a functor $\mathcal{P}\colon \mathbf{Set} \to \mathbf{Set}$, sending each set to its power set and each function $f\colon A \to B$ to the map sending $U \in \mathcal{P}(A)$ to its image
$$f(U) \coloneqq \{f(u) \mid u \in U\}. \tag{4.8.4}$$
- The map $V \mapsto V^*$ defines a functor $\mathbf{Vect}_{\Bbbk} \to \mathbf{Vect}_{\Bbbk}$.
- If G is a group viewed as a one object category then a functor $G \to \mathbf{Set}$ defines a group action, sending the single object of G to the set, S, upon which G acts, and sending each morphism, g (recall morphisms are just elements of the group) to the mapping on that set $s \mapsto g \,.\, s$.

- If G is a group viewed as a one object category then a functor $G \to \mathbf{Vect}_{\Bbbk}$ defines a group action on a vector space, which is to say this functor defines a representation.

- If G and H are groups viewed as one object categories then a functor $G \to H$ is simply a group homomorphism, sending the single object of G to the single object of H and preserving composition, which is the group operation.

- The map sending each complex Lie group to its Lie algebra is a functor **LieGrp** → **LieAlg**[a].

- A **forgetful functor**, $\mathbf{C} \to \mathbf{Set}$ maps a category to **Set** by "forgetting" the structure of the objects in **C**. A category, **C**, is called **concrete** if there exists a forgetful functor $\mathbf{C} \to \mathbf{Set}$.

  - The forgetful functor $U\colon \mathbf{Grp} \to \mathbf{Set}$ sends each group to its underlying set and each group homomorphism to its underlying function.
  - The forgetful functor $U\colon \mathbf{Vect}_{\Bbbk} \to \mathbf{Set}$ sends each vector space to its underlying set and each linear transformation to its underlying function.
  - The forgetful functor $U\colon \mathbf{Top} \to \mathbf{Set}$ sends each topological space to its underlying set and each continuous function to its underlying function.

- A partially forgetful functor $\mathbf{C} \to \mathbf{D}$ generalises forgetful functors by forgetting some, but not necessarily all, structure of the objects in **C**.

  - The partially forgetful functor $\mathbf{Grp} \to \mathbf{Mon}$ sends each group to itself, but now viewed as a monoid, essentially forgetting that inverses exist.
  - The partially forgetful functor $\mathbf{Hilb} \to \mathbf{Vect}_{\Bbbk}$ sends each Hilbert space to itself, but forgets the inner product structure.
  - The partially forgetful functor $\mathbf{CRing} \to \mathbf{Ring}$ sends each commutative ring to itself, forgetting that multiplication is commutative.
  - The partially forgetful functor $\mathbf{Ring} \to \mathbf{Ab}$ sends each ring to its additive group, forgetting how to multiply.

- A **free functor** is, in a sense[b] the reverse of a forgetful functor, sending a set to the most general object of the appropriate type.

  - The free functor $\mathbf{Set} \to \mathbf{Grp}$ sends each set the free group it generates, which is the set of words generated by concatenating elements of the set and elements declared to be their inverses, with the group operation being concatenation and the identity the empty string. No other relations between elements of the set exist.

- The free functor **Set** → $\mathbf{Vect}_{\Bbbk}$ takes a set and declares that all the elements in it are linearly independent and form a basis for a vector space whose elements are formal linear combinations of these elements.

[a] See *Symmetries of Quantum Mechanics* or *Symmetries of Particles and Fields*

[b] This sense can be made formal through the notion of an adjoint, an important concept in category theory but one we won't explore.

The next definition introduces some terminology used to talk about functors.

Definition 4.8.5 A functor $F\colon \mathbf{C} \to \mathbf{D}$ is

- **full** when the mapping $f \mapsto Ff$ defines a surjection $\mathbf{C}(A, B) \to \mathbf{D}(FA, FB)$;
- **faithful** when the mapping $f \mapsto Ff$ defines an injection $\mathbf{C}(A, B) \to \mathbf{D}(FA, FB)$;
- **essentially surjective on objects** when for each $B \in \mathrm{Ob}(\mathbf{D})$ there is some $A \in \mathrm{Ob}(\mathbf{C})$ such that $FA \cong B$, essentially when the mapping $A \mapsto FA$ is surjective, but replacing equality with isomorphism in the definition of surjectivity;
- an **equivalence** when it is full, faithful, and essentially surjective on objects;
- an **endofunctor** if $\mathbf{C} = \mathbf{D}$.

The most important definition here is that of an equivalence. Equality of categories is far too strong a condition. We can also define a notion of isomorphism between categories, but this is also too strong. Relaxing the requirements for isomorphism, by changing surjective on objects to essentially surjective on objects, we get the notion of equivalence, which is the sweet spot where a functor preserves enough structure for the two categories to be the same for all intents and purposes, without being too restrictive.

It should be noted that there are multiple, equivalent, definitions of equivalence. The one we've given here is probably the easiest to understand, but not always the easiest to apply.

One example we've already seen of an equivalence is the functor $\mathbf{Mat}_{\Bbbk} \to \mathbf{FVect}_{\Bbbk}$ sending $n \mapsto \Bbbk^n$ and a matrix to the associated linear map on the vector space with some fixed basis. This is an equivalence since all finite dimensional vector spaces of the same dimension are isomorphic, which means that this functor is essentially surjective on objects.

Now that we have categories and maps between them it is natural to ask if this forms a category, and indeed it does!

Definition 4.8.6 — Cat The category **Cat** has (small) categories as its objects and functors as its morphisms. Composition of functors is composition of the two mappings $\mathrm{Ob}(\mathbf{C}) \to \mathrm{Ob}(\mathbf{D})$ and $\mathbf{C}(A, B) \to \mathbf{D}(FA, FB)$, and the identity is the identity functor.

To be clear, if $F\colon \mathbf{C} \to \mathbf{D}$ and $G\colon \mathbf{D} \to \mathbf{E}$ are functors then the functor $(G \circ F)\colon \mathbf{C} \to \mathbf{E}$ sends $A \in \mathrm{Ob}(\mathbf{C})$ to $(G \circ F)(A) = (G \circ F)A = G(F(A)) = GFA$, and sends $f \in \mathbf{C}(A, B)$ to $(G \circ F)(f) = (G \circ F)f = G(F(f)) = GFf \in \mathbf{E}(GFA, GFB)$. Associativity is guaranteed by associativity of the underlying mappings and the identity functor is clearly an identity morphism.

4.9 Natural Transformations

We have now defined functors as mathematical objects. Following the spirit of category theory we should look for maps between functors preserving their structure. A map between functors should preserve the functor structure. This means that it shouldn't matter if we map to a different functor and then apply the functor, or apply a functor and then map to the other functor. This leads to the following definition.

Definition 4.9.1 — Natural Transformation Let $\mathbf{C}$ and $\mathbf{D}$ be categories and $F, G\colon \mathbf{C} \to \mathbf{D}$ functors. For every object $A \in \mathrm{Ob}(\mathbf{C})$ a **natural transformation**, $\zeta\colon F \Rightarrow G$, assigns a morphism $\zeta_A\colon FA \to GA$ in $\mathbf{D}$ such that for every morphism $f\colon A \to B$ in $\mathbf{C}$ the following diagram commutes:

$$\begin{tikzcd}
FA \arrow[r, "\zeta_A"] \arrow[d, "Ff"'] & GA \arrow[d, "Gf"] \\
FB \arrow[r, "\zeta_B"'] & GB.
\end{tikzcd} \tag{4.9.2}$$

If every component, ζ_A, is an isomorphism then ζ is called a **natural isomorphism** and F and G are said to be **naturally isomorphic**, written $F \cong G$.

This leads to an alternative, equivalent, definition of an equivalence. A functor $F\colon \mathbf{C} \to \mathbf{D}$ is an equivalence if and only if there is a functor $G\colon \mathbf{D} \to \mathbf{C}$ and natural isomorphisms $G \circ F \Rightarrow \mathrm{id}_{\mathbf{C}}$ and $F \circ G \Rightarrow \mathrm{id}_{\mathbf{D}}$. Intuitively, an equivalence is a function which is invertible up to natural isomorphism.

Note that it is common to write all of the data needed to define a natural transformation as

$$\begin{tikzcd}
\mathbf{C} \arrow[r, bend left=50, "F"{name=U}] \arrow[r, bend right=50, "G"'{name=D}] & \mathbf{D}. \arrow[Rightarrow, from=U, to=D]
\end{tikzcd} \tag{4.9.3}$$

Example 4.9.4 — Natural Transformations These examples are taken from [2] and make use of the notation used there representing a natural transformation as a collection of morphisms indexed by objects, $(\zeta_A)_{A \in \mathrm{Ob}(\mathbf{C})}$.

- A **discrete category** is a category in which the only maps are the identity maps. If $\mathbf{C}$ is a discrete category then a functor $F\colon \mathbf{C} \to \mathbf{D}$ is simply a family of objects $(FA)_{A\in\mathrm{Ob}(\mathbf{C})}$. In this case a natural transformation $\zeta\colon F \Rightarrow G$ is just a family of maps $(\zeta_A : FA \to GA)_{A\in\mathrm{Ob}(\mathbf{C})}$. The naturality axiom is automatically fulfilled since it holds trivially for identities.

- Fix some natural number n. For any commutative ring, R, the $n \times n$ matrices with entries in R form a monoid, $M_n(R)$, under matrix multiplication. Any ring homomorphism $R \to S$ induces a monoid homomorphism $M_n(R) \to M_n(S)$ by acting elementwise on the entries of the matrix with the homomorphism. This defines a functor $M_n : \mathbf{CRing} \to \mathbf{Mon}$.

  The elements of any ring, R, form a monoid, $U(R)$, under multiplication, this is an example of a forgetful functor $U\colon \mathbf{CRing} \to \mathbf{Mon}$.

  Every $n \times n$ matrix, X, over a commutative ring R has a determinant $\det_R(X) \in R$. The properties

  $$\det_R(XY) = \det_R(X)\det_R(Y), \qquad \text{and} \qquad \det_R(I) = 1 \quad (4.9.5)$$

  tell us that for each commutative ring R the function $\det_R : M_n(R) \to U(R)$ is a monoid homomorphism. Thus we have a family of maps

  $$(\det_R : M_n(R) \to U(R))_{R\in\mathrm{Ob}(\mathbf{CRing})}. \qquad (4.9.6)$$

  This family of maps defines a natural transformation $\det\colon M_n \Rightarrow U$.

- Consider the category $\mathbf{FVect}_{\mathbb{k}}$. The map $(-)^* : \mathbf{FVect}_{\mathbb{k}} \to \mathbf{FVect}_{\mathbb{k}}$ sending each vector space to its dual is a contravariant functor. Thus the map $(-)^{**} : \mathbf{FVect}_{\mathbb{k}} \to \mathbf{FVect}_{\mathbb{k}}$ sending each vector space to its double dual is also a covariant functor. For each $V \in \mathrm{Ob}(\mathbf{FVect}_{\mathbb{k}})$ we have a canonical isomorphism $\zeta_V\colon V \to V^{**}$. Given $v \in V$ the element $\zeta_V(v) \in V^{**}$ is evaluation at v, that is $\zeta_V(v)\colon V^* \to \mathbb{k}$ maps $\varphi \in V^*$ to $\varphi(v) \in \mathbb{k}$. This defines a natural transformation $\zeta\colon 1_{\mathbf{FVect}_{\mathbb{k}}} \Rightarrow (-)^{**}$ meaning that each vector space is naturally isomorphic to its double dual. Note that given $F, G\colon \mathbf{C} \to \mathbf{D}$ we say that $FA \cong GA$ naturally in A if F and G are naturally isomorphic. In this case $V \cong V^{**}$ naturally in V.

Functors are objects, and natural transformations are maps between them, so we should try to define a category.

Definition 4.9.7 — Functor Category Let $\mathbf{C}$ and $\mathbf{D}$ be categories. The **functor category** $[\mathbf{C}, \mathbf{D}]$, also written $\mathbf{D}^{\mathbf{C}}$, has functors $F, G\colon \mathbf{C} \to \mathbf{D}$ as objects and natural transformations $F \Rightarrow G$ as morphisms. Composition of natural transformations is done component wise and the identity is the identity natural transformation which assigns the to each $A \in \mathrm{Ob}(\mathbf{C})$ the identity morphism id_{FA} in $\mathbf{D}$.

An example of a functor category is the category of G-sets[2] for some group G, which is $[G, \mathbf{Set}]$, where G is viewed as a one object category. Similarly $[G, \mathbf{Vect}_{\Bbbk}]$ is the category of $\Bbbk$-linear representations of G. In both of these cases morphisms are so called equivariant maps. Given sets (vector spaces) X and Y upon which we have a G action (representation) defined an **equivariant map** is a function (linear map) $f : X \to Y$ such that $f(g \,.\, x) = g \,.\, f(x)$, that is a map which commutes with the group action, so

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g."'] & Y \arrow[d, "g."] \\
X \arrow[r, "f"'] & Y
\end{tikzcd} \tag{4.9.8}$$

commutes. This diagram is exactly what we get if we specialise the diagram defining a natural transformation to functors $G \to \mathbf{Set}$ ($G \to \mathbf{Vect}_{\Bbbk}$).

4.10 Products

We want to generalise the Cartesian product of sets, which is defined as

$$A \times B \coloneqq \{(a, b) \mid a \in A \text{ and } b \in B\}. \tag{4.10.1}$$

To do so we need to remove reference to elements, since this isn't a concept that we have in an arbitrary category. Clearly this definition defines a new set, so we have a new object in **Set**. In order to do away with reference to elements notice that we can define two functions $p_A : A \times B \to A$ and $p_B : A \times B \to B$ which project out the elements of a pair. That is $p_A(a, b) = a$ and $p_B(a, b) = b$. We can summarise this as

$$A \xleftarrow{p_A} A \times B \xrightarrow{p_B} B. \tag{4.10.2}$$

The key insight comes in requiring that $A \times B$ is, in a sense, the most general[3] set satisfying this property, since the Cartesian product doesn't add in any unnecessary restrictions. Thus, if we have another set, X, which is a candidate for defining the product then there must be morphisms $f : X \to A$ and $g : X \to B$ which are candidates for p_A and p_B respectively. We can now make "most general" precise. Given $A \times B$ with X, f, and g we should be able to recreate f and g by first mapping to $A \times B$ and then projecting out. This map to $A \times B$ is what we call the product of f and g, written $f \times g$, (f, g), or $\binom{f}{g}$. We can summarise this definition by requiring that the following diagram commutes:

$$\begin{tikzcd}
& X \arrow[dl, "f"'] \arrow[d, dashed, "!" ', "f \times g"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "p_A"] \arrow[r, "p_B"'] & B.
\end{tikzcd} \tag{4.10.3}$$

Here we use a dashed arrow to denote that this diagram is telling us that this arrow exists, and the ! tells us that this arrow is unique.

This is an example of a more general concept in category theory, called a **limit**. The general idea is that we can define some prototype, here $A \times B$ with the maps

[2] that is sets upon which G acts

[3] this concept, both for products and more general (co)limits, is explained well in [3], which is where I'm replicating this argument from.

p_A and p_B. Then we posit a candidate object which factors through the prototype, then we look for an object satisfying this property.

Notice that we now have done away with all references to elements of $A \times B$, so we can generalise this definition to arbitrary categories.

Definition 4.10.4 — Product Let $\mathbf{C}$ be a category with objects A and B. The product of A and B, if it exists, is the object $A \times B$ and the morphisms $p_A : A \times B \to A$ and $p_B : A \times B \to B$ such that

$$\begin{tikzcd}
& X \arrow[dl, "f"'] \arrow[d, dashed, "! \, f \times g"] \arrow[dr, "g"] & \\
A & A \times B \arrow[l, "p_A"] \arrow[r, "p_B"'] & B
\end{tikzcd} \tag{4.10.5}$$

commutes for all X with morphisms to $f : X \to A$ and $g : X \to B$.

Example 4.10.6 — Products

- In **Top** the product of two topological spaces is the topological space formed from the Cartesian product of the two topological spaces equipped with the product topology, which is the coarsest topology for which all projections are continuous.
- In R–**Mod** the product is the Cartesian product of the modules with addition defined componentwise and multiplication defined to be distributive.
- In **Grp** the product is the direct product of groups, that is the Cartesian product of the underlying sets and then componentwise multiplication.
- In **Rel** products are disjoint unions.
- In a poset viewed as a single object category the product of two elements is the greatest lower bound.

Definition 4.10.7 — Product Category Let $\mathbf{C}$ and $\mathbf{D}$ be categories. The **product category $\mathbf{C} \times \mathbf{D}$** has

- as objects pairs of objects (A, B) with $A \in \mathrm{Ob}(\mathbf{C})$ and $B \in \mathrm{Ob}(\mathbf{D})$;
- as morphisms $(A_1, B_1) \to (A_2, B_2)$ pairs of morphisms (f, g) with $f : A_1 \to A_2$ and $g : B_1 \to B_2$ that is, $f \in \mathbf{C}(A_1, A_2)$ and $g \in \mathbf{D}(B_1, B_2)$;
- as composition componentwise composition from the two categories, that is $(f_2, g_2) \circ (f_1, g_1) = (f_2 \circ f_1, g_2 \circ g_1)$;

- as identities pairs of identities from the two categories, that is $\mathrm{id}_{(A,B)} = (\mathrm{id}_A, \mathrm{id}_B)$.

For the case where **C** and **D** are small the product category $\mathbf{C} \times \mathbf{D}$ is exactly the categorical product of **C** and **D** in **Cat**.

The main use of product categories is to define bifunctors, which are the generalisation of functors to two variables. A **bifunctor** is exactly a functor $\mathbf{C} \times \mathbf{D} \to \mathbf{E}$, but it's usually easier to think of **C** and **D** as being separate, just like how we might treat the horizontal and vertical axes as being independent in a function $\mathbb{R}^2 \to \mathbb{R}$.

Five

Monoidal Categories

5.1 Tensor Products

Definition 5.1.1 — Bilinear Let U, V and W be vector spaces over $\Bbbk$. A **bilinear map** is a function $T\colon U \times V \to W$ which is linear in each variable. That is, the maps $T(-, v)\colon U \to W$ defined by $u \mapsto T(u, v)$ and $T(u, -)\colon V \to W$ defined by $v \mapsto T(u, v)$ are linear.

The tensor product provides a way to combine two vector spaces into a new vector space. The simplest definition is that given vector spaces U and V over some field $\Bbbk$ the tensor product $U \otimes V$ consists of all linear combinations of elements of the form $u \otimes v$ with $u \in U$ and $v \in V$. This definition makes direct reference to elements of the vector spaces, so isn't compatible with the spirit of category theory. The useful thing about the tensor product is it combines two vector spaces into one in such a way that we can define linear maps on the new space in terms of linear maps on the original spaces. For example, the linear map sending $u \in U$ to $2u$ automatically extends to a linear map on $U \otimes V$ sending $u \otimes v$ to $2u \otimes v = (2u) \otimes v = 2(u \otimes v)$. This inspires the following definition.

Definition 5.1.2 — Tensor Product Let U, V, and W be vector spaces over $\Bbbk$. The **tensor product** of U and V is a vector space, which we call $U \otimes V$, equipped with a bilinear map $f\colon U \times V \to U \otimes V$ such that for all bilinear maps $g\colon U \times V \to W$ there exists a unique linear map $h\colon U \otimes V \to W$ such that $g = h \circ f$. That is, such that

$$\begin{tikzcd}
U \times V \arrow[r, "f\text{ (bilinear)}"] \arrow[dr, "g\text{ (bilinear)}"'] & U \otimes V \arrow[d, dashed, "h\text{ (linear)}"] \\
& W
\end{tikzcd} \tag{5.1.3}$$

commutes.

The important idea here is that a linear map $U \otimes V \to W$ is just as good as a bilinear map $U \times V \to W$, and since linear maps are much nicer to work with, and since $U \times V$ is not a vector space, we usually prefer to work with $U \otimes V$.

Note that not all elements of $U \otimes V$ are of the form $u \otimes v$. For example, there is no way to write $e_1 \otimes e_1 + e_2 \otimes e_2 \in V \otimes V$ in this form. This will be important later.

The tensor product also gives us a natural way to combine two linear maps, we simply act on the relevant part of the tensor product with the relevant function.

> **Definition 5.1.4 — Tensor Product of Linear Maps** Let U, U', V and V' be vector spaces over $\Bbbk$. Let $f\colon U \to U'$ and $g\colon V \to V'$. Then the **tensor product** of f and g is defined to be the map $f \otimes g\colon U \otimes V \to U' \otimes V'$ given by defining $(f \otimes g)(u \otimes v) = f(u) \otimes g(v)$ and extending this to all of $U \otimes V$ through linearity.

The tensor product of vector spaces can be extended to any inner product space in a straight forward manner, the inner product simply factorises.

> **Definition 5.1.5 — Tensor Product of Inner Product Spaces** Let $(U, \langle - | - \rangle_U)$ and $(V, \langle - | - \rangle_W)$ be inner product spaces (Hilbert spaces) over $\Bbbk$. Then $(U \otimes V, \langle - | - \rangle_{U \otimes V})$ is an inner product space (Hilbert space) with the inner product
>
> $$\langle u \otimes v | u' \otimes v' \rangle_{U \otimes V} = \langle u | u' \rangle_U \langle v | v' \rangle_V. \tag{5.1.6}$$

5.2 Monoidal Categories: The Idea

Monoidal categories generalise the tensor product to other categories. The idea being that tensor products allow us to work with multiple objects at the same time, or in parallel. Recall that we've seen four ways to consider categories:

- physical systems and the processes occurring;
- data types and the algorithms manipulating them;
- algebraic structures and structure preserving functions;
- logical propositions and implications between them.

We can consider the idea of a monoidal category in each of these frameworks:

- independent systems evolving separately;
- running algorithms in parallel;
- products or sums of geometric structures;
- using separate proofs of P and Q to prove $P \wedge Q$.

5.3 Monoidal Categories: The Need for Specificity

Consider processes A, B, and C with some notion of parallel composition, $\otimes$. So, $A \otimes B$ is what we get by doing both processes A and B at the same time. After playing around with this idea for a while one may come upon the question of what the relationship between

$$(A \otimes B) \otimes C \qquad \text{and} \qquad A \otimes (B \otimes C) \tag{5.3.1}$$

should be. It's not right for them to be equal, since this isn't the case for vector spaces with the tensor product or for sets with the Cartesian product, which is the equivalent in **Set** as we'll see later.

For example, viewed as sets we have $((a, b), c)$ as an element of $(A \times B) \times C$, but $(a, (b, c))$ as an element of $A \times (B \times C)$. Clearly there is a map $(A \times B) \times C \to A \times (B \times C)$ given by $((a, b), c) \mapsto (a, (b, c))$. We say that this map associates these distinct objects. This map is also clearly invertible, the inverse being $(a, (b, c)) \mapsto ((a, b), c)$. So, we have $(A \times B) \times C \cong A \times (B \times C)$.

More questions arise when we think about what the processes A, B, and C are. For example there is often a concept of a trivial system where we don't do anything. We want it to be such that running this trivial system in parallel with another system is as if we weren't doing the trivial thing at all. However this is clearly not the case in, for example, a computer running a trivial algorithm, it still has to compile and run this trivial algorithm. So again we want the computation of A and the trivial system in parallel to be isomorphic to the computation of A alone.

Another question, which we won't see an answer too until later, is does order matter? That is, should we treat $A \otimes B$ and $B \otimes A$ as the equal? What about isomorphic? It is also possible that they aren't even isomorphic, although in many of the cases we're interested in they will be.

To answer these questions we have to be careful when defining a monoidal category, the task of the next section.

5.4 Monoidal Categories: The Definition

Definition 5.4.1 — Monoidal Category A **monoidal category** is formed from the following data:

1. a category, $\mathbf{C}$;
2. a **tensor product functor**

   $$- \otimes - : \mathbf{C} \times \mathbf{C} \to \mathbf{C}, \tag{5.4.2}$$

   also called the **monoidal product**;
3. a **unit object** $I \in \mathrm{Ob}(\mathbf{C})$;
4. a natural isomorphism

   $$\alpha : (- \otimes -) \otimes - \Rightarrow - \otimes (- \otimes -) \tag{5.4.3}$$

called the **associator** which has components

$$\alpha_{A,B,C} \colon (A \otimes B) \otimes C \to A \otimes (B \otimes C) \tag{5.4.4}$$

for $A, B, C \in \mathrm{Ob}(\mathbf{C})$;

5. a natural isomorphism

$$\lambda \colon I \otimes - \Rightarrow - \tag{5.4.5}$$

called the **left unitor** which has components

$$\lambda_A \colon I \otimes A \to A \tag{5.4.6}$$

for $A \in \mathrm{Ob}(\mathbf{C})$;

6. a natural isomorphism

$$\rho \colon - \otimes I \Rightarrow - \tag{5.4.7}$$

called the **right unitor** which has components

$$\rho_A \colon A \otimes A \to A \tag{5.4.8}$$

for $A \in \mathrm{Ob}(\mathbf{C})$.

This data must be such that the **triangle equation** holds, which is that

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[rr, "\alpha_{A,I,B}"] \arrow[dr, "\rho_A \otimes \mathrm{id}_B"'] & & A \otimes (I \otimes B) \arrow[dl, "\mathrm{id}_A \otimes \lambda_B"] \\
& A \otimes B &
\end{tikzcd} \tag{5.4.9}$$

commutes for all $A, B \in \mathrm{Ob}(\mathbf{C})$, and the **pentagon equation** holds, which is that

$$\begin{tikzcd}
& (A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B \otimes C,D}"] & & A \otimes ((B \otimes C) \otimes D) \arrow[dr, "\mathrm{id}_A \otimes \alpha_{B,C,D}"] & \\
((A \otimes B) \otimes C) \otimes D \arrow[ur, "\alpha_{A,B,C} \otimes \mathrm{id}_D"] \arrow[drr, "\alpha_{A \otimes B,C,D}"'] & & & & A \otimes (B \otimes (C \otimes D)) \\
& & (A \otimes B) \otimes (C \otimes D) \arrow[urr, "\alpha_{A,B,C \otimes D}"'] & &
\end{tikzcd} \tag{5.4.10}$$

commutes for all $A, B, C, D \in \mathrm{Ob}(\mathbf{C})$.

This is quite a large definition so we'll break it down and hopefully this will make it less daunting. First lets look at what the data of a monoidal category tells us:

1. the category tells us the type of objects and morphisms we are considering;

2. the tensor product functor tells us how to combine two objects to get a new object, and two morphisms to get a new morphism;

3. the unit object represents a trivial process in which nothing happens;

4. the associator allows us to move brackets around and only change things by an isomorphism, which means no important change occurs;

5. the unitors allow us to remove the unit object when it is involved in a product, again only changing things by an isomorphism.

Now lets look at the triangle equation. It tells us how the unitors and associator combine. Starting at the top left, with $(A \otimes I) \otimes B$ there are two ways to get to $A \otimes B$. We can either remove I by applying ρ_A to the $A \otimes I$ bit, which means applying $\rho_A \otimes \mathrm{id}_B$ to $(A \otimes I) \otimes B$, or we can reassociate, by applying $\alpha_{A,I,B}$ to get $A \otimes (I \otimes B)$ then remove the unit by applying λ_B to the $I \otimes B$ bit, which means applying $\mathrm{id}_A \otimes \lambda_B$ to $A \otimes (I \otimes B)$. The triangle equality says that both of these are actually the same, that is

$$\rho_A \otimes \mathrm{id}_B = (\mathrm{id}_A \otimes \lambda_A) \circ \alpha_{A,I,B}. \tag{5.4.11}$$

Finally, lets look at the pentagon equation. This tells us how the associator can be applied to reassociate products of four objects, and it turns out that this is enough to completely specify how the associator reassociates products of any number of elements. Starting on the left we have the completely left associated $((A \otimes B) \otimes C) \otimes D$. One path we can take to reassociate is to ignore D and reassociate $(A \otimes B) \otimes C$ to $(A \otimes (B \otimes C))$, this is done using $\alpha_{A,B,C} \otimes \mathrm{id}_D$. Next we can ignore the fact that $B \otimes C$ is the product of two objects and think of it as a single object, X, so we have $(A \otimes X) \otimes D$, which we can reassociate using $\alpha_{A,X,D} = \alpha_{A,B\otimes C,D}$ to get $A \otimes ((B \otimes C) \otimes D)$. Finally we can ignore A and reassociate $(B \otimes C) \otimes D$ using $\mathrm{id}_A \otimes \alpha_{B,C,D}$ to get $A \otimes (B \otimes (C \otimes D))$. Alternatively, if we start with $((A \otimes B) \otimes C) \otimes D$ we can treat $A \otimes B$ as a single object, Y, and reassociate $(Y \otimes C) \otimes D$ with $\alpha_{Y,C,D} = \alpha_{A\otimes B,C,D}$ to get $(A \otimes B) \otimes (C \otimes D)$. Then we treat $C \otimes D$ as a single object, Z, and reassociate $(A \otimes B) \otimes Z$ using $\alpha_{A,B,Z} = \alpha_{A,B,C\otimes D}$ to get $A \otimes (B \otimes (C \otimes D))$. The pentagon equation tells us that both of these ways of reassociating from left to right give are the same, that is

$$(\mathrm{id}_A \otimes \alpha_{B,C,D}) \circ (\alpha_{A,B\otimes C,D}) \circ (\alpha_{A,B,C} \otimes \mathrm{id}_D) = \alpha_{A,B,C\otimes D} \circ \alpha_{A\otimes B,C,D}. \tag{5.4.12}$$

To summarise, the triangle equation says that all ways of removing I from $(A \otimes I) \otimes B$ to get $A \otimes B$ are the same, and the pentagon equation says that all ways of reassociating $((A \otimes B) \otimes C) \otimes D$ to get $A \otimes (B \otimes (C \otimes D))$ are the same.

While these requirements seem quite complex they are incredibly powerful, and fairly easy to work with in practice, because they are so restrictive. It turns out that the monoidal product works exactly how our intuition says it does, and this is the crux of the next theorem, which we shan't prove.

Theorem 5.4.13 — Coherence Theorem for Monoidal Categories. In a monoidal category and well-typed equation built from associators, unitors, and their inverses holds.

By well-typed we simply mean that both sides of the equation must be morphisms with matching domain and codomain, and that any compositions are defined, so morphisms in the composition match domain to codomain. This means that we don't have to actually use the triangle and pentagon equations directly, as long as they hold we can write down pretty much anything we like as long as it makes sense and it will be true.

5.5 Monoidal Categories: The Examples

5.5.1 Set

The obvious choice for a monoidal product on **Set** is the Cartesian product. We then need to look for a unit, something which we can take a Cartesian product with but not really change anything. One way we might come to an answer is to recognise that the unit behaves just like 1 in a product, hang on, 1 is something that we can represent as a set, a singleton, and there's our answer. Given some set A and some singleton[1] $\{\bullet\}$ the Cartesian product $A \times \{\bullet\}$ consists of elements $(a, \bullet)$ with $a \in A$, and $\{\bullet\} \times A$ consists of elements $(\bullet, a)$. From both of these we can easily recover A by just ignoring the $\bullet$. This gives us our unitors. We've already seen in Section 5.3 how $((a,b),c)) \mapsto (a,(b,c))$ defines the associator for the Cartesian product. Thus we can make the following definition.

Definition 5.5.1 — Set as a Monoidal Category The category **Set** can be promoted to a monoidal category by defining

1. the monoidal product to be the Cartesian product;
2. the unit to be some singleton, $I = \{\bullet\}$;
3. the associator to have components $\alpha_{A,B,C} : (A\times B)\times C \to A\times(B\times C)$ defined by $((a,b),c) \mapsto (a,(b,c))$;
4. the left unitor at A to be the function $\lambda_A : \{\bullet\} \times A \to A$ defined by $(\bullet, a) \mapsto a$;
5. the right unitor at A to be the function $\rho_A : A \times \{\bullet\} \to A$ defined by $(a, \bullet) \mapsto a$.

! This is not the only way we can make **Set** into a monoidal category, but it is the only one that we'll use and it is usually what people are talking about when they say **Set** is a monoidal category. Another ways of making **Set** into a monoidal category involve taking $A \otimes B = A \sqcup B$ to be the disjoint union with $I = \varnothing$. Yet another choice is $A \otimes B = A \sqcup B \sqcup (A \times B)$ with $I = \varnothing$.

This common definition of **Set** as a monoidal category is an example of a more general idea. In some category **C** a **terminal object**, $T \in \mathrm{Ob}(\mathbf{C})$, is an object such that for any object $A \in \mathrm{Ob}(\mathbf{C})$ there exists exactly one morphism $A \to T$. Similarly a **initial object**, $I \in \mathrm{Ob}(\mathbf{C})$, is an object such that for any object $A \in \mathrm{Ob}(\mathbf{C})$ there exists exactly one morphism $I \to A$. Any category, **C**, with terminal objects and products can be made into a monoidal category by taking the monoidal product to be the categorical product and the unit to be the terminal object. Similarly and category with initial objects and coproducts[2] can be made into a monoidal category

[1] Since an isomorphism in **Set** is a bijection all sets of the same size are isomorphic, so all singletons are the same for the purpose of category theory.

[2] A **coproduct** is defined similarly to a categorical product, but with the arrows reversed.

by taking the tensor product to be the coproduct and the unit to be the initial object. An example of a coproduct/initial object monoidal category is **Set** with disjoint union as a monoidal product.

Lemma 5.5.2 **Set** equipped with the Cartesian product is a monoidal category.

Proof. We first demonstrate that the triangle equation holds. To do so we modify the diagram defining the triangle equation to show the elements being mapped, instead of the objects, and commutativity should be clear from this

$$\begin{tikzcd}
((a,\bullet),b) \arrow[rr, mapsto, "\alpha_{A,I,B}"] \arrow[dr, mapsto, "\rho_A \times \mathrm{id}_B"'] & & (a,(\bullet,b) \arrow[dl, mapsto, "\mathrm{id}_A \times \lambda_B"] \\
& (\rho_A(a,\bullet), \mathrm{id}_B(b)) = (a,b) = (\mathrm{id}_A(a), \lambda_B(\bullet,b)). &
\end{tikzcd} \tag{5.5.3}$$

Similarly, we can demonstrate the commutativity of the pentagon:

$$\begin{tikzcd}
\begin{array}{c}(\alpha_{A,B,C}((a,b),c), \mathrm{id}_D(D)) \\ = ((a,(b,c)),d)\end{array} \arrow[r, mapsto, "\alpha_{A,B\times C,D}"] & \begin{array}{c}\alpha_{A,B\times C,D}((a,(b,c)),d) \\ = (a,((b,c),d))\end{array} \arrow[dd, mapsto, "\mathrm{id}_A \times \alpha_{B,C,D}"] \\
(((a,b),c),d) \arrow[u, mapsto, "\alpha_{A,B,C} \times \mathrm{id}_D"] \arrow[d, mapsto, "\alpha_{A\times B,C,D}"'] & \\
\begin{array}{c}\alpha_{A\times B,C,D}(((a,b),c),d) \\ = ((a,b),(c,d))\end{array} \arrow[r, mapsto, "\alpha_{A,B,C\times D}"'] & \begin{array}{c}(\mathrm{id}_A(a), \alpha_{B,C,D}((b,c),d)) \\ = (a,(b,(c,d))) = \\ \alpha_{A,B,C\times D}((a,b),(c,d)).\end{array}
\end{tikzcd} \tag{5.5.4}$$

□

Appendices

A

Maths Definitions

A.1 Algebraic Structures

A.1.1 One Binary Operation

Definition A.1.1 — Magma A **magma**, $(M, \cdot)$, is a set, M, equipped with a binary operation $\cdot : M \times M \to M$.
A **magma homomorphism**, $(M, \cdot_M) \to (N, \cdot_N)$, is a function $f : M \to N$ such that $f(m \cdot_M m') = f(m) \cdot_N f(m')$ for all $m, m' \in M$.

Definition A.1.2 — Semigroup A **semigroup**, $(S, \cdot)$, is an associative magma, meaning that

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \tag{A.1.3}$$

for all $a, b, c \in S$.
A **semigroup homomorphism** is a magma homomorphism between semigroups.

Definition A.1.4 — Monoid A **monoid**, $(M, \cdot, e)$, is a semigroup with an identity element, $e \in M$, such that $e \cdot m = m \cdot e = m$ for all $m \in M$.
A **monoid homomorphism**, $(M, \cdot_M, e_M) \to (N, \cdot_N, e_N)$, is a function $f : M \to N$ such that $f(m \cdot_M m') = f(m) \cdot_N f(m')$ for all $m, m' \in M$ and $f(e_M) = e_N$.

Definition A.1.5 — Group A **group**, $(G, \cdot, e, -^{-1})$, is a monoid with inverses, meaning for each $g \in G$ there exists $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = e$.
A **group homomorphism**, $(G, \cdot_G, e_G, -^{-1}) \to (H, \cdot_H, e_H, -^{-1})$, is a function $f : G \to H$ such that $f(g \cdot_G g') = f(g) \cdot_H f(g')$ for all $g, g' \in G$.

Note that the definition of a group homomorphism implies that $f(e_G) = e_H$ and $f(g^{-1}) = f(g)^{-1}$ for all $g \in G$, unlike in the monoid case.

For magmas, semigroups, and monoids if $x \cdot y = y \cdot x$ we say it is a **commutative magma/semigroup/monoid**. For groups we call a commutative group an **Abelian group**.

Definition A.1.6 — Group Action Let G be a group and S a set. Then a **left group action** is a map $\varphi\colon G \times S \to S$ such that

- the identity acts as the identity: $\varphi(e, s) = s$;
- group multiplication is preserved: $\varphi(g, \varphi(h, s)) = \varphi(gh, s)$.

Often we write $g \,.\, s$ or gs in place of $\varphi(g, s)$, in which case the axioms are $es = s$ and $g(hs) = (gh)s$, which look very similar to the group identity and associativity laws. Note that a right action is a map $\psi\colon S \times G \to S$ such that $\psi(s, e) = s$ and $\psi(\psi(s, g), h) = \psi(s, gh)$, or $se = s$ and $(sg)h = s(gh)$.

Definition A.1.7 — Group Representation A **representation** of a group G is a pair (ρ, V) consisting of a vector space V and a group action $\rho\colon G \times V \to V$. Alternatively, $\rho\colon V \to \mathrm{GL}(V)$ is a homomorphism and G acts on V through $g \,.\, v = \rho(g)v$.

Note that

$$\begin{aligned}\mathrm{GL}(V) &\coloneqq \{\text{invertible linear maps on } V\}\\ &\cong \{n \times n \text{ matrices with entries in } \Bbbk\} \eqqcolon \mathrm{GL}(n, \Bbbk) \qquad \text{(A.1.8)}\end{aligned}$$

where $n = \dim V$ and V is a vector space over $\Bbbk$.

A.1.2 Two Binary Operations

Definition A.1.9 — Ring A **ring**, $(R, +, -, \cdot, 1, 0)$ is a set, R, equipped with two binary operations, $+\colon R \times R \to R$ and $\cdot\colon R \times R \to R$ such that

- $(R, +, 0, -)$ is an Abelian group.
- $(R, \cdot, 1)$ is a monoid.
- Multiplication distributes over addition:

  $$a \cdot (b + c) = a \cdot b + a \cdot c, \qquad \text{and} \qquad (a + b) \cdot c = a \cdot c + b \cdot c \quad \text{(A.1.10)}$$

  where we take multiplication to have higher operator precedence than addition.

A **ring homomorphism**, $(R, +_R, -_R, \cdot_R, 1_R, 0_R) \to (S, +_S, -_S, \cdot_S, 1_S, 0_S)$ is a function $f\colon R \to S$ such that $f(a +_R b) = f(a) +_S f(b)$, $f(a \cdot_R b) = f(a) \cdot_S f(b)$, and $f(1_R) = 1_S$.
If $x \cdot y = y \cdot x$ for all $x, y \in R$ we call R a **commutative ring**.

Note that some sources only require that $(R, \cdot)$ is a semigroup, in which case what we define here is called a **ring with identity** or **ring with unity**. Sometimes a ring without identity is called a **rng**, with i taken out as there is no **i**dentity.

It is also common to define **rig** or **semiring|seerig** by only requiring that $(R,+)$ is a commutative monoid, meaning we drop inverses, which for an additive group would be **n**egatives, $-g$ being the additive inverse of g. A **rg** is then defined by dropping the requirements for multiplicative identities and additive inverses.

Definition A.1.11 — Integral Domain Let R be a ring. A **zero-divisor** is some element $a \in R$ such that $ab = 0$ and/or $ba = 0$ for some $b \in R$ with $b \neq 0$. A ring with *no* zero-divisors is called a **domain**. A commutative domain is an **integral domain**.

Definition A.1.12 — Field Let R be a ring. A **unit** is some element $a \in R$ with $a \neq 0$ such that there exists $a^{-1} \in R$ with $a \cdot a^{-1} = a^{-1} \cdot a = 1$. A division ring is a ring in which all nonzero elements are units. That is, $(R^{\times}, \cdot, e, -^{-1})$ is a group, where $R^{\times} = R \setminus \{0\}$.
A commutative division ring is a **field**.

A.2 Orderings

Definition A.2.1 — Partial Order A (non-strict) **partially ordered set**, or **poset**, $(P, \leq)$, is a set, P, equipped with a (non-strict) **partial order**, which is a reflexive, antisymmetric, transitive relation. That is,

- **reflexivity**: for all $a \in P$ we have $a \leq a$;
- **antisymmetric**: for $a, b \in P$ if $a \leq b$ and $b \leq a$ then $a = b$;
- **transitivity**: for $a, b, c \in P$ if $a \leq b$ and $b \leq c$ then $a \leq c$.

Definition A.2.2 — Total Order A (non-strict) **totally ordered set**, $(X, \leq)$, is a poset with the added condition of totality, that is for all $a, b \in X$ we have either $a \leq b$ or $b \leq a$ (or both, in which case $a = b$ by antisymmetry).

Definition A.2.3 — Monotone Function Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be partially ordered sets. A **monotone function**, or **order-preserving function** $(P, \leq_P) \to (Q, \leq_Q)$, is a function $f : P \to Q$ such that if $a \leq_P b$ for $a, b \in P$ then $f(a) \leq_Q f(b)$.

A.3 Topological Spaces

Definition A.3.1 — Topological Space A **topological space**, $(X, \mathcal{T})$ is a set, X, equipped with a **topology**, $\mathcal{T}$, which is a subset of $\mathcal{P}(X)$ such that

- $\varnothing \in \mathcal{T}$;

- $X \in \mathcal{T}$;
- an arbitrary (potentially infinite) union of elements of $\mathcal{T}$ is again an element of $\mathcal{T}$;
- a finite intersection of elements of $\mathcal{T}$ is again an element of $\mathcal{T}$.

Elements of $\mathcal{T}$ are called open sets.

Definition A.3.2 — Continuous Function Let $(X, \mathcal{T}_X)$ and $(Y, \mathcal{T}_Y)$ be topological spaces. A **continuous function**, $(X, \mathcal{T}_X) \to (Y, \mathcal{T}_Y)$, is a function $f : X \to Y$ such that for all $U \in \mathcal{T}_Y$ we have $f^{-1}(U) \in \mathcal{T}_X$ where

$$f^{-1}(U) := \{x \in X \mid \exists y \in U \text{ such that } y = f(x)\} \subseteq X. \tag{A.3.3}$$

Bibliography

[1] C. Heunen and J. Vicary. *Categories for Quantum Theory*. Oxford: Oxford University Press, 2019.

[2] T. Leinster. *Basic Category Theory*. Cambridge: Cambridge University Press, 2014. DOI: 10.48550/ARXIV.1612.09375.

[3] B. Milewski. *Category Theory for Programmers*. 2019. URL: https://github.com/hmemcpy/milewski-ctfp-pdf.

Index