Willoughby Seago

Theoretical Physics

Gauge Theories in Particle Physics

January 16, 2023

Course Notes

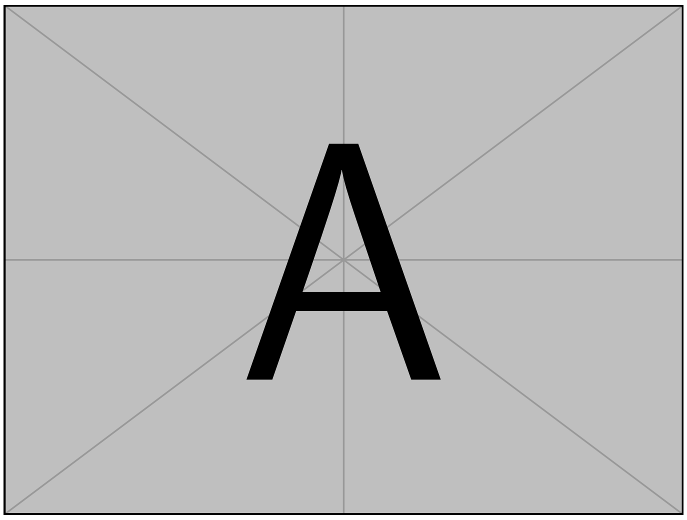
Gauge Theories in Particle Physics

Willoughby Seago

January 16, 2023

These are my notes from the course gauge theories in particle physics. I took this course as a part of the theoretical physics degree at the University of Edinburgh.

These notes were last updated at 23:25 on January 25, 2023. For notes on other topics see https://github.com/WilloughbySeago/Uni-Notes.



Chapters

Contents

Page

List of Figures

Page

One

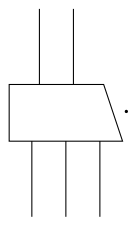
ZX Calculus

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.

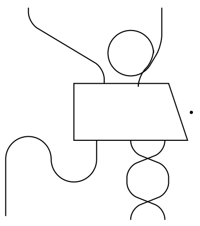
We'll introduce ZX calculus in a seemingly backwards manner, first introducing which sorts of diagrams we can have, then how to manipulate the diagrams then what the diagrams mean.

1.1 Types of Diagrams

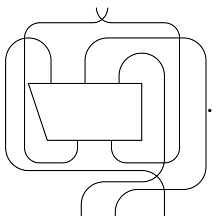
A diagram in ZX calculus is somewhat like a flowchart. The playing field is the two-dimensional page. We imagine that time goes upwards and space extends to the left and right. This means that a process described by a ZX calculus starts by entering the bottom of the diagram and ends when we leave the 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:


. (1.1.1)

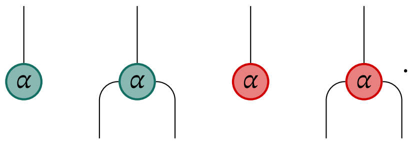
In diagrams it isn't important exactly how we draw the wires, so long as they are connected in the same way, so in the same order both on the box and along the top and bottom, the diagram corresponds to the same equation. For example, the following is equivalent to the previous diagram


. (1.1.2)

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.

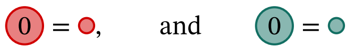

. (1.1.3)

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



. (1.1.4)

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:



(1.1.5)

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 1.1 is a process which takes in two qubits and outputs two qubits.

1.2 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

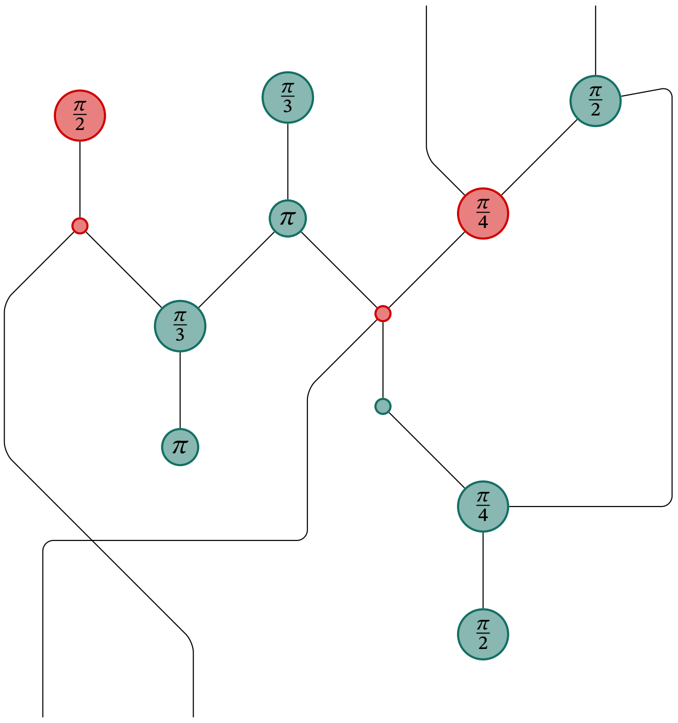
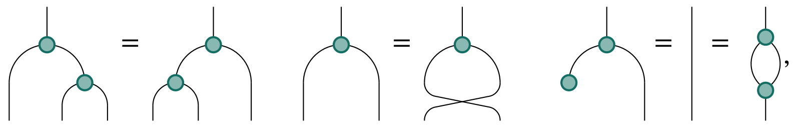


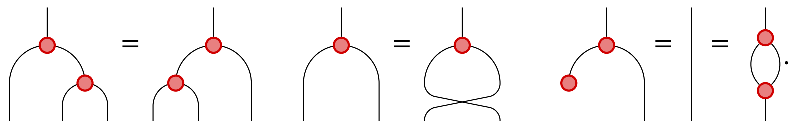
Figure 1.1: A diagram in ZX calculus taking two qubits to two qubits.

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.

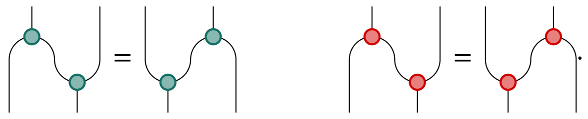
The second type of rule corresponds to specific properties of the basic building blocks. There are quite a few of these, and for now we'll just list them without much explanation. First we have the **monoid rules** which are



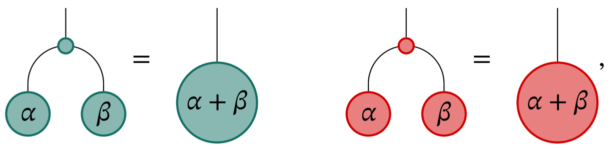
and the same for the other colour:



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

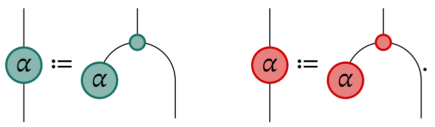

(1.2.1)

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

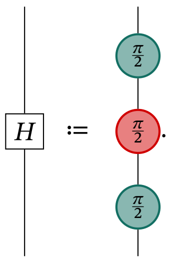
 (1.2.2)

where addition is taken modulo 2π.

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

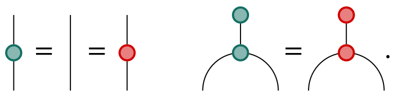
 (1.2.3)

We also define the **Hadamard**:

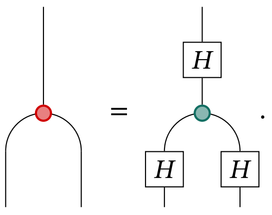
 (1.2.4)

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

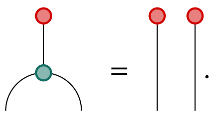
We then have two **identity rules**, the first is just a repeat of one of the monoid rules, but now in this new shorthand, the second is new:

 (1.2.5)

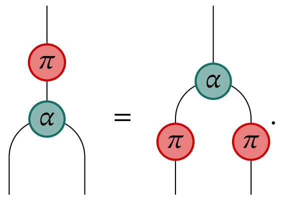
The next rule allows us to change the colour of a node, at the cost of some Hadamards, it is appropriately called the **colour change rule**:

 (1.2.6)

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

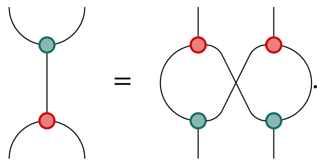
 (1.2.7)

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**:



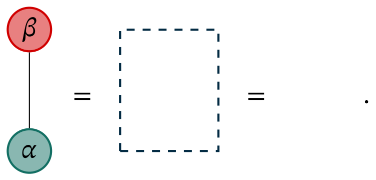
(1.2.8)

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



(1.2.9)

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



(1.2.10)

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.

1.3 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

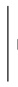
$$|+\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad \text{and} \qquad |-\rangle = H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (1.3.1)$$

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

$$[\![-]\!] : (n\text{-to-}m \text{ qubit ZX diagram}) \to (2^n \times 2^m \text{ complex matrices}). \quad (1.3.2)$$

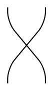
Then a process which takes a single qubit, does nothing to it, and immediately outputs it is represented as



$$\longmapsto \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.3.3}$$

Unsurprisingly doing nothing to a qubit gives the identity.

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

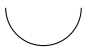


$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.3.4}$$

In terms of matrices this acts as follows:

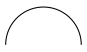
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \left[\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} \right] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix} = \begin{pmatrix} ac \\ bc \\ ad \\ bd \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} \otimes \begin{pmatrix} a \\ b \end{pmatrix}. \tag{1.3.5}$$

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:



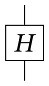
$$\longmapsto \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \tag{1.3.6}$$

Similarly we can destroy qubits:



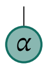
$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}. \tag{1.3.7}$$

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

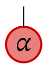


$$\longmapsto \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.3.8}$$

We can create a single qubit:

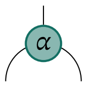


$$\longmapsto \begin{pmatrix} 1 \\ e^{i\alpha} \end{pmatrix}, \quad \text{and}$$

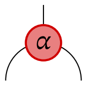


$$\longmapsto \begin{pmatrix} 1 + e^{i\alpha} \\ 1 - e^{i\alpha} \end{pmatrix} \tag{1.3.9}$$

The three-arity spiders give the following matrices

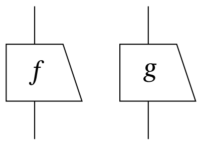


$$\longmapsto \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{pmatrix}, \tag{1.3.10}$$



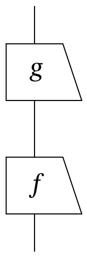
$$\longmapsto \begin{pmatrix} 1 + e^{i\alpha} & 1 - e^{i\alpha} & 1 - e^{i\alpha} & 1 + e^{i\alpha} \\ 1 - e^{i\alpha} & 1 + e^{i\alpha} & 1 + e^{i\alpha} & 1 - e^{i\alpha} \end{pmatrix} \tag{1.3.11}$$

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



$$\mapsto f \otimes g \tag{1.3.12}$$

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:



$$\mapsto g \circ f \tag{1.3.13}$$

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.

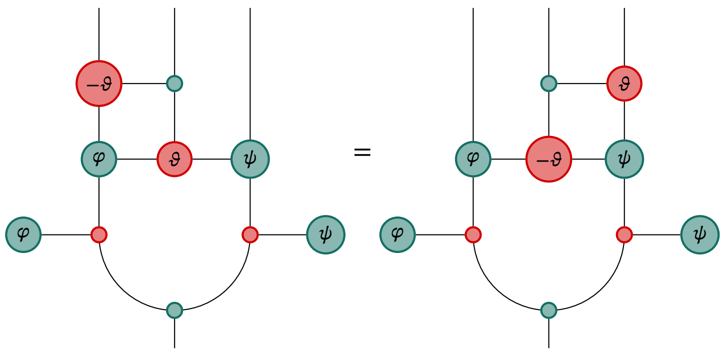
Theorem 1.3.14 — ZX Calculus is Sound. Let D_1 and D_2 be diagrams in ZX calculus. If $D_1 = D_2$ according to the rules of ZX calculus then $[\![D_1]\!] = [\![D_2]\!]$.

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.

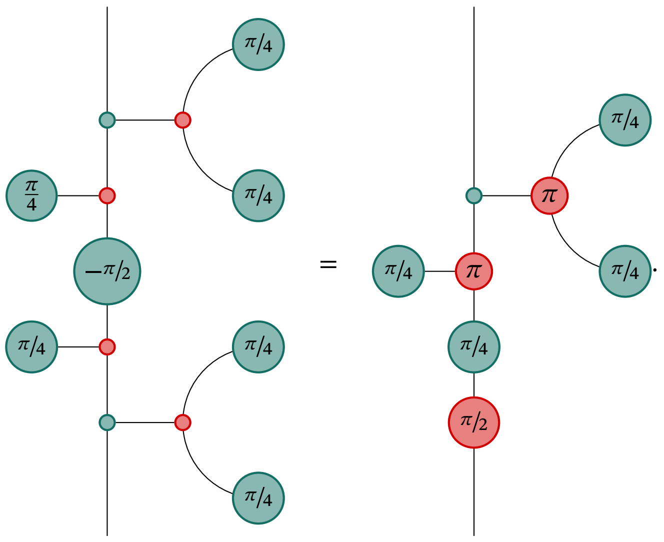
Theorem 1.3.15 — ZX Calculus is Approximately Universal. For any $2^m \times 2^n$ matrix, f, and any error margin, $\varepsilon > 0$, there exists a diagram, D, in ZX calculus built only from terms with phases an integer multiple of $\pi/4$ such that $\|[\![D]\!] - f\| < \varepsilon$ for some appropriate 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:

 (1.3.16)

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

 (1.3.17)

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

Theorem 1.3.18 — $\pi/4$-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,

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}, \tag{1.3.19}$$

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.

Two

Semantics

2.1 Types of Semantics

Consider the two following pseudocode fragments:

$$P = \text{if } 1 = 1 \text{ then F else G}, \tag{2.1.1}$$
$$Q = \text{if } 1 = 0 \text{ then F else F}. \tag{2.1.2}$$

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

$$[\![-]\!] : \text{Programs} \to \text{Mathematical Objects}. \tag{2.1.3}$$

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 $[\![P]\!] = [\![Q]\!] = [\![\mathrm{F}]\!]$.

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

$$[\![\mathrm{F};\ \mathrm{G}]\!] = [\![\mathrm{F}]\!] \circ [\![\mathrm{G}]\!]. \tag{2.1.4}$$

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

$$[\![\ \text{paralell}\,(\mathrm{F},\ \mathrm{G})]\!] = [\![\mathrm{F}]\!] \otimes [\![\mathrm{G}]\!]. \tag{2.1.5}$$

2.2 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.

2.3 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.

Three

Categories

3.1 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

$$[\![-]\!] : \text{programs} \to \text{mathematical objects}. \tag{3.1.1}$$

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 $\lambda\ \mathrm{x}.\ *\ 2\ \mathrm{x}$ which takes an argument, 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.

3.2 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.

3.3 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 3.3.1 — 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

$$h \circ (g \circ f) = (h \circ g) \circ f, \tag{3.3.2}$$

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

$$f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f. \tag{3.3.3}$$

3.3.1 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.

3.4 Categories: The Examples

3.4.1 Set

Definition 3.4.1 — 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 3.4.2 **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

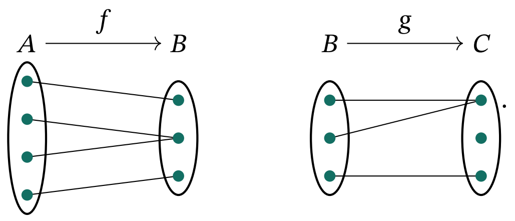
$$(h\circ(g\circ f))(x) = h((g\circ f)(x)) = h(g(f(x))) = (h\circ g)(f(x)) = ((h\circ g)\circ f)(x) \tag{3.4.3}$$

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

$$(f \circ \mathrm{id}_A)(x) = f(\mathrm{id}_A(x)) = f(x) = \mathrm{id}_B(f(x)) = (\mathrm{id}_B \circ f)(x) \tag{3.4.4}$$

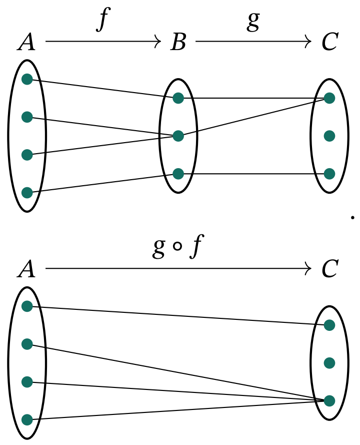
and so $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$. □

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



(3.4.5)

Then composition is just following the lines between sets:



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

(3.4.6)

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.

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.

3.4.2 Rel

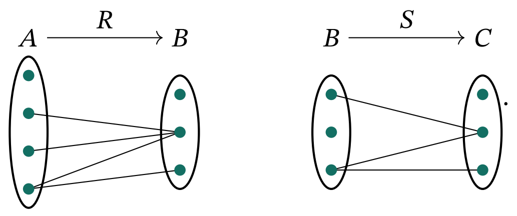
Definition 3.4.7 — Relation Let A and B be sets. A **relation**, R, is a subset of $A \times B$. If $(a, b) \in R$ we write aRb.

Relations are morphisms in a category we will defined shortly (Definition 3.4.15), so for $R \subset A \times B$ we write $R\colon A \to B$. 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

$$R = \{(b, 2), (c, 2), (d, 2), (d, 3)\} \subseteq A \times B, \qquad \text{and} \tag{3.4.8}$$

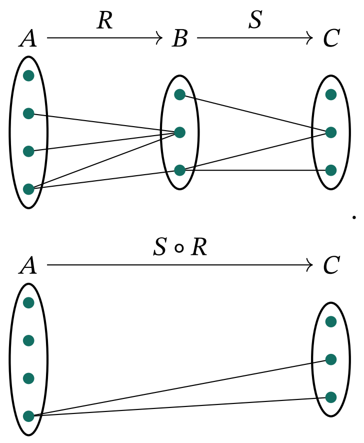
$$S = \{(1, \beta), (3, \beta), (3, \gamma)\} \subseteq B \times C \tag{3.4.9}$$

can be represented pictorially as



(3.4.10)

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



(3.4.11)

That is,

$$S \circ R = \{(d, \beta), (d, \gamma)\} \subseteq A \times C. \tag{3.4.12}$$

This leads to the following definition.

Definition 3.4.13 — 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

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

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

$$\mathrm{id}_A \coloneqq \{(a, a) \in A \times A \mid a \in A\} \subseteq A \times A. \tag{3.4.14}$$

Now we can define a category.

Definition 3.4.15 — 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

$$S \circ R = \{(a, c) \mid \exists b \in B : aRb \wedge bSc\} \subseteq A \times C. \tag{3.4.16}$$

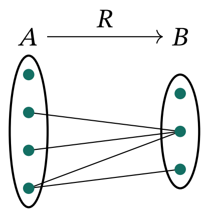
The identity morphism is the identity relation, $\mathrm{id}_A : A \to A$, given by

$$\mathrm{id}_A = \{(a, a) \mid a \in A\} \subseteq A \times A. \tag{3.4.17}$$

Lemma 3.4.18 **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



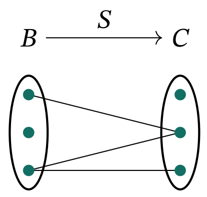
$$\rightsquigarrow M(R) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.4.19}$$

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

$$M\colon \mathcal{P}(A \times B) \to \mathcal{M}_{|B|\times|A|}(\mathbb{Z}_2) \tag{3.4.20}$$

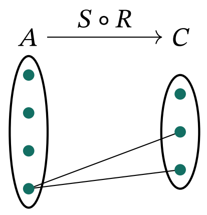
where $\mathbb{Z}_2 = \{0, 1\}$ has addition and multiplication defined mod 2.

As an example notice that we have



$$\rightsquigarrow M(S) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.4.21}$$

and



$$\rightsquigarrow M(S \circ R) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.4.22}$$

The composite can then be calculated as

$$M(S \circ R) = M(S)M(R) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.4.23}$$

This map, M, actually defines a functor ( ) $M\colon$ **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 $\mathbf{C} \to \mathbf{Mat}_{\Bbbk}$ for some appropriate category $\mathbf{C}$ and ring $\Bbbk$. This in turn is more commonly thought of as a map $\mathbf{C} \to \mathbf{FVect}_{\Bbbk}$ identifying matrices with linear maps. This allows for generalisations to infinite dimensional representations, $\mathbf{C} \to \mathbf{Vect}_{\Bbbk}$. We'll define $\mathbf{FVect}_{\Bbbk}$ and $\mathbf{Vect}_{\Bbbk}$ shortly (Definitions 3.4.28 and 3.4.35).

[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 3.4.47). 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.

3.4.3 $\mathbf{Vect}_{\Bbbk}$, $\mathbf{FVect}_{\Bbbk}$, $\mathbf{Hilb}$, and $\mathbf{FHilb}$

3.4.3.1 $\mathbf{Vect}_{\Bbbk}$ and $\mathbf{FVect}_{\Bbbk}$

Definition 3.4.24 — 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 3.4.25 — 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

$$T(u +_V v) = T(u) +_W T(v), \qquad \text{and} \qquad T(\alpha \cdot_V v) = \alpha \cdot_W T(v) \quad (3.4.26)$$

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

$$T(u + v) = T(u) + T(v), \qquad \text{and} \qquad T(\alpha v) = \alpha T(v). \qquad (3.4.27)$$

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

Definition 3.4.28 — $\mathbf{Vect}_\Bbbk$ 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 3.4.29 $\mathbf{Vect}_\Bbbk$ 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

$$\begin{aligned}(S \circ T)(u + v) &= S(T(u + v)) = S(T(u) + T(v)) \\ &= S(T(u)) + S(T(v)) = (S \circ T)(u) + (S \circ T)(v) \qquad (3.4.30)\end{aligned}$$

using the linearity of T and then S, and

$$(S \circ T)(\alpha v) = S(T(\alpha v)) = S(\alpha T(v)) = \alpha S(T(v)) = \alpha(S \circ T)(v) \quad (3.4.31)$$

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

Definition 3.4.32 — 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

$$\alpha_1 e_1 + \cdots + \alpha_n e_n = 0 \qquad (3.4.33)$$

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

$$v = \alpha_1 e_1 + \cdots + \alpha_n e_n \tag{3.4.34}$$

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

Lemma 3.4.36 $\mathbf{FVect}_\Bbbk$ 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 3.4.37 — $\mathbf{Mat}_\Bbbk$ The category $\mathbf{Mat}_\Bbbk$ has natural numbers as objects with a morphism $n \to m$ being an $n \times m$ matrix with entries in $\Bbbk$. Composition of morphisms is matrix multiplication and the identity matrix is the identity morphism.

Lemma 3.4.38 $\mathbf{Mat}_\Bbbk$ is a category.

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

There is an equivalence ( ) $\mathbf{Mat}_\Bbbk \to \mathbf{FVect}_\Bbbk$ 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.

3.4.3.2 **Hilb** and **FHilb**

Definition 3.4.39 — 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 3.4.40 — 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

$$\sum_{i=1}^{\infty} \|v_i\| \tag{3.4.41}$$

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

$$\sum_{i=1}^{\infty} v_i \tag{3.4.42}$$

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

$$\left\| v - \sum_{i=1}^{n} v_i \right\| < \varepsilon, \tag{3.4.43}$$

or equivalently,

$$\lim_n \left\| v - \sum_{i=1}^{n} v_i \right\| = 0. \tag{3.4.44}$$

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 3.4.45 — 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

$$\|T(v)\|_K \leq M\|v\|_H \tag{3.4.46}$$

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

Definition 3.4.47 — 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 3.4.48 **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 3.4.29, so the composite of two bounded linear maps is a bounded linear map. □

3.4.4 More Examples

Example 3.4.49 — 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 $f: (X, \bullet) \to (Y, *)$ is continuous and $f(\bullet) = *$.

See Chapter A for relevant definitions.

Example 3.4.50 — 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 3.4.51 — 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 3.6.1). 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).

3.5 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

$$A \xrightarrow{f} B \xrightarrow{g} C \tag{3.5.1}$$

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,

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & B \arrow[r, "g"] & C \end{tikzcd} . \tag{3.5.2}$$

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

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "h"'] & B \arrow[r, "g"] \arrow[d, "i"'] & C \\ D \arrow[r, "k"'] & E \arrow[ur, "j"'] & \end{tikzcd} \tag{3.5.3}$$

commuting is equivalent to the following requirements:

$$g \circ f = j \circ k \circ h, \qquad i \circ f = k \circ h, \qquad \text{and} \qquad g = j \circ i, \tag{3.5.4}$$

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

$$\begin{tikzcd} A \arrow[loop left, "\mathrm{id}_A"] \arrow[r, "f"] & B \arrow[loop right, "\mathrm{id}_B"] \end{tikzcd} \tag{3.5.5}$$

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:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[rr, bend left, "g \circ f"] & B \arrow[r, "g"] \arrow[rr, bend right, "h \circ g"'] & C \arrow[r, "h"] & D \end{tikzcd} . \tag{3.5.6}$$

3.6 Terminology

We now introduce some terminology which will be helpful.

Definition 3.6.1 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 $g, h: B \to C$;
- we call f a **monomorphism**, or **monic**, if $f \circ g = f \circ h$ implies $g = h$ for all $g, h: C \to A$.

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

Proof. Suppose this wasn't the case, so $f: A \to B$ has two inverses, $g, g': B \to A$, 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

$$g = g \circ \mathrm{id}_B = g \circ (f \circ g') = (g \circ f) \circ g' = \mathrm{id}_A \circ g' = g'. \tag{3.6.3}$$

□

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

$$\begin{tikzcd} A \arrow[r, shift left, "f"] & B \arrow[l, shift left, "f^{-1}"] \end{tikzcd} \tag{3.6.4}$$

commutes.

3.7 Graphical Notation

3.8 Functors

3.9 Natural Transformations

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**.

A.1.2 Two Binary Operations

Definition A.1.6 — Ring A **ring**, $(R, +, -, \cdot, 1, 0)$ is a set, R, equipped with two binary operations, $+ : R \times R \to R$ and $\cdot : R \times R \to R$ such that

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

  $$a \cdot (b + c) = a \cdot b + a \cdot c, \qquad \text{and} \qquad (a + b) \cdot c = a \cdot c + b \cdot c \quad \text{(A.1.7)}$$

  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 $f : R \to S$ 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.1.8 — 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.1.9 — 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 Topological Spaces

Definition A.2.1 — 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.2.2 — 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

$$f^{-1}(U) \coloneqq \{x \in X \mid \exists y \in U \text{ such that } y = f(x)\} \subseteq X. \tag{A.2.3}$$

Index